Policy Specification Using Sequence Diagrams

Applied to Trust Management

Bjørnar Solhaug

Dissertation for the degree philosophiae doctor (PhD)

at the University of Bergen

October 2009

Scientific Environment

The work leading up to this thesis has been conducted as part of the ENFORCE project which was funded by the Research Council of Norway. The ENFORCE project was a joint initiative between Department of Information Science and Media Studies, University of Bergen; Norwegian Research Center for Computers and Law, University of Oslo; Department of Informatics, University of Oslo; and SINTEF ICT in Oslo.

The work was conducted in affiliation with the Department of Information Science and Media Studies at the Faculty of Social Sciences, University of Bergen, and with SINTEF ICT. In partial fulfillment of the requirements to the degree of PhD, the doctoral training program of the Faculty of Social Sciences was completed during the project.

UNIVERSITY OF BERGEN

SINTEF

UNIVERSITY
OF OSLO

Acknowledgments

First and foremost I want to express profound thanks to my supervisors Dag Elgesem and Ketil Stølen for their willingness to and interest in spending time and effort in guiding me in the work that has led to this thesis. The many discussions on my work, as well as their suggestions, criticism and encouragement have been decisive. Ketil Stølen's dedication and drive as leader of the ENFORCE project has also been invaluable and cannot be appreciated enough.

I am grateful to Atle Refsdal and Tobias Mahler for our cooperation in the ENFORCE project, and for their suggestions and ideas for how to improve my work. Many thanks to Jon Bing for his involvement in and contribution to the ENFORCE project, and for his comments on and suggestions to my work.

Many thanks to Fabio Massacci, Jakka Sairamesh, Peter Wahlgren and Christian Damsgaard Jensen for regularly visiting the ENFORCE project and giving their valuable advices and suggestions from their outsider's perspective. Thanks also to the ENFORCE advisory board for giving their recommendations to the project.

I am thankful to Fredrik Seehusen and Tom Lysemose for our joint efforts and fruitful cooperation. I am in debt to Mass Soldal Lund, Ragnhild Kobro Runde, Judith Rossebø, Gyrd Brændeland and Ida Hogganvik for their advice and criticism.

Thanks to Tor Hjalmar Johannessen and Cathrine Holst for their comments on and suggestions to parts of my work. I am grateful also Arild Waaler who inspired and encouraged me to begin as a doctoral research fellow in the first place.

I also want to thank the people at the department of Cooperative and Trusted Systems at SINTEF ICT where I have my office and have been carrying out the daily work on the thesis. Finally, I owe my thanks to the academic staff and the administrative staff at Department of Information Science and Media Studies and at the Faculty of Social Sciences at the University of Bergen.

The work on this thesis has been conducted within the IKT SOS project ENFORCE (164382/V30) which was funded by the Research Council of Norway.

Abstract

With the ever increasing importance of computer networks such as the Internet, and the today almost ubiquitous online services, the needs for the management of these networks and services, as well as the management of the associated security, risk and trust are growing correspondingly.

Policy based management of information systems has the last decade emerged as an adaptive and flexible approach to this challenge. Policies are rules governing the choices in the behavior of systems, the enforcement of which ensures that the system meets the relevant requirements. This thesis addresses the problem of capturing, specifying and developing policies. We propose a language suitable for the specification of policies across domains, at various abstraction levels, and that facilitates human interpretation. At the same time the language is underpinned by a formal semantics, allowing precise and rigorous analysis.

Abstraction allows details about system functionality and architecture to be ignored, thus facilitating analysis and supporting understanding, which is beneficial and useful particularly during the initial phases of policy development. From the initial, abstract levels, a policy specification is typically further developed by adding details, making it more concrete and closer to implementation and enforcement. This thesis proposes a notion of policy refinement that relates policy specifications of different abstraction levels, precisely defining what it means that a low-level, concrete policy specification is a correct representation of a high-level, abstract specification. Refinement allows policy specifications to be developed in a stepwise and incremental manner, and ensures that the enforcement of the final, concrete specification implies the enforcement of the previous, more abstract specifications.

The applicability of the approach is demonstrated within the domain of policy based trust management. The thesis proposes a method for the development of trust management policies that facilitates the modeling and analysis of trust within systems, and the evaluation of the risks and opportunities to which the system is exposed as a consequence of trust-based decisions. The method is supported by designated languages for the appropriate modeling and analyses, and aims at the capturing and formalization of policies the enforcement of which optimizes the trust-based decisions by minimizing risks and maximizing opportunities.

List of Original Publications

1. Bjørnar Solhaug, Dag Elgesem and Ketil Stølen. Why trust is not proportional to risk. In *Proceedings of the 2nd International Conference on Availability, Reliability and Security (ARES'07)*, pages 11–18. IEEE Computer Society, 2007.

2. Bjørnar Solhaug, Dag Elgesem and Ketil Stølen. Specifying policies using UML sequence diagrams – An evaluation based on a case study. In *Proceedings of the 8th International Workshop on Policies for Distributed Systems and Networks (POLICY'07)*, pages 19–28. IEEE Computer Society, 2007.

3. Bjørnar Solhaug and Tor Hjalmar Johannessen. Specification of policies using UML sequence diagrams. *Telektronikk*, 105(1):90–97, 2009.

4. Bjørnar Solhaug and Ketil Stølen. Compositional refinement of policies in UML – Exemplified for access control. In *Proceedings of the 13th European Symposium on Research in Computer Security (ESORICS'08)*, volume 5283 of *LNCS*, pages 300–316. Springer, 2008.

5. Fredrik Seehusen, Bjørnar Solhaug and Ketil Stølen. Adherence preserving refinement of trace-set properties in STAIRS: Exemplified for information flow properties and policies. *Software and Systems Modeling*, 8(1):45–65, 2009.

6. Bjørnar Solhaug and Ketil Stølen. Preservation of policy adherence under refinement. Technical report A11358, SINTEF ICT, 2009.

7. Atle Refsdal, Bjørnar Solhaug and Ketil Stølen. A UML-based method for the development of policies to support trust management. In *Trust Management II – Proceedings of the 2nd Joint iTrust and PST Conference on Privacy, Trust Management and Security (IFIPTM'08)*, volume 263 of *IFIP*, pages 33–49. Springer, 2008.

The publications are available as Chapter 9 through Chapter 15 in Part II of the thesis. In case of publication 2 and 4 we have included the full technical reports rather than the published papers.

Contents

Part I

Context and Overview

Chapter 1

Introduction

During the work that led up to this thesis we developed a number of related artifacts that contribute to the domain of policy based management of information systems, with particular focus on policy based trust management. In this chapter we give some background and motivation for our work, briefly present the artifacts, and give an overview of the thesis.

1.1 Background

The steady growth of the information society is strongly evident in all parts of the world. Individuals, commercial enterprises, governmental institutions and other organizations increasingly offer, consume and depend on electronic services provided over computerized networks such as the Internet.

Statistics of Internet usage in the world [48] show that more than 20% of the world's population were in 2008 users of the Internet, which is an increase of over 300% since 2000. 48% of the population in Europe are users of Internet, which is an increase of 266% since 2000. In North-America and Oceania/Australia, the usage is even more widespread. Statistics delivered by the European Commission [20] show that in 2007, 30% of individuals aged 16 to 74 and 65% of enterprises in the EU interact with public authorities over the Internet. Furthermore, over 25% of individuals have the last three months in 2008 ordered/bought goods or services for private use over the Internet; in 2007, 4.2% of enterprises' total turnover were from e-commerce via the Internet, which is an increase of 100% since 2004; the same year, 59% of 20 basic public services were available online.

The increasing importance of computer networks and Internet services imposes a corresponding increase in the challenge of managing systems and networks. Enterprise information systems must integrate local networks and inter-organizational networks with Internet-based services [23], and organizations and individuals need to cope with challenges related to security, trust, privacy and risk. Statistics show that in 2005, 1.3% of individuals with Internet access have encountered fraudulent payment card use within the last year, which is an increase of 44% in one year. The corresponding number for encountering abuse of personal information sent over the Internet is 3.8%, and for encountering computer virus resulting in loss of data or time is 34.4%. As for enterprises, 29% encountered a security problem, where computer virus attack were most prominent [20].

Policy based management of information systems [16, 22, 45] has the last decade emerged as an adaptive and flexible approach to this challenge. A policy is commonly defined as a set of rules governing the choices in the behavior of a system [102]. By separating the policy from the implementation of the system, the policy can be modified for dynamically changing the behavior of the system, without changing the underlying implementation [102]. Whatever the domain of management, e.g. security, access control, trust or services, the correct enforcement of the policy rules ensures that the system meets the relevant requirements.

Policy based management must be supported by adequate languages, methods and tools for the specification, development, analysis and enforcement of policies. Policies are derived from business goals, service level agreements, or trust relationships [103], and during these initial phases of policy capturing there is a need to ensure communication and common understanding between the involved stakeholders, including decision makers, users, security officers and developers. A policy specification language aimed at human interpretation and that support abstraction may fulfill such a need.

Abstraction allows details about system functionality and architecture to be ignored, thus facilitating policy capturing at a high-level where the most important and general requirements are taken into account. Abstraction is desirable also because detection and corrections of errors, as well as policy analysis, are cheaper and easier at an abstract and high level [113].

From the initial, abstract level, a policy specification is further developed by adding details, making it more concrete and closer to enforcement. Policy refinement has been recognized as an important feature of policy development [80], and refers to the process of transforming a high-level, abstract policy specification into a low-level, concrete one, ensuring that the latter fulfills the requirements of the former.

In addition to refinement, decomposition has been recognized as one of the most important features of system design and development [113]. Decomposition allows individual parts of a specification to be developed separately, thus breaking the development problem down into manageable pieces. In order to facilitate human understanding and communication, to facilitate analysis, and to support the development process from policy capturing to policy enforcement, we believe that refinement and decomposition should be features of the policy development process.

A precise characterization of the notions of abstraction and refinement, as well as the possibility of precise and rigorous analysis require a formal basis of approaches to policy based management [43, 44, 63]. Such a formal basis can be provided by a formal semantics for the policy specification language. A formal semantics furthermore allows computerized tool support to be developed, and allows a precise characterization of what it means that a system fulfills a policy.

Another response to the challenges we are faced with in the information society is that of trust management [56, 97]. The online environments of today for interactions and transactions that traditionally have taken place in face-to-face situations are increasingly also environments in which traditional social relations are established. Trust is very often a prerequisite for interactions to take place, and is relevant for open, dynamic and distributed systems, where entities collaborate, exchange information and rely on other entities in situations where the identity, intentions, capabilities, etc. of others may be uncertain. Risk is an inherent aspect of trust since there is always a chance of deception or betrayal [19, 72]. The motivation to trust others is the opportunities involved in successful trust-based transactions.

Policy based trust management seeks to capture and enforce rules that govern trust-based decisions. These rules are intended to ensure that risks are minimized and opportunities maximized in the trust-based interactions and transactions. Methods for the capturing and specification of trust management policies must support the precise modeling, analysis and assessment of trust, as well as the evaluation of the involved risks and opportunities.

This thesis addresses the challenges described in this section by proposing a policy specification language that supports the formalization of policies at various abstraction levels with the aim of facilitating human understanding and communication. Notions of policy refinement are introduced that precisely capture what it means that one policy specification is more concrete, i.e. less abstract, than another policy specification. The policy refinement relations support a modular and stepwise policy development process where individual parts of a policy specification can be refined separately and under any number of refinement steps.

The proposed language is generic in the sense that it allows specification of policies for various domains, such as security, access control, services and trust. The thesis demonstrates the applicability of the language by addressing the domain of policy based trust management. A method for trust policy development is proposed for supporting the capturing of policies that ensure optimization of the trust-based decisions within systems. As a part of this method, the resulting trust policies are formalized using the policy specification language put forward in this thesis.

1.2 Contribution

The main contribution of this thesis is a policy specification language with support for policy capturing, policy abstraction and refinement, policy analysis, and analysis of the relation between a policy specification and systems for which the policy applies. This contribution is manifested in three separate, yet strongly related artifacts, namely the policy specification language, the policy adherence relations and the policy refinement relations.

The fourth main artifact of the thesis is the method for the development of trust management policies. This artifact is put into close interplay with the other artifacts of this thesis by serving as a domain in which the applicability of the policy specification language is demonstrated. However, the method also represents a free-standing contribution to the domain of trust management.

In addition to the four artifacts that constitute the main contribution of the thesis, important work has been carried out in order to establish a conceptual foundation for trust management. In the context of this thesis and the method for trust policy development we have clarified what we mean by trust management and precisely defined the notion of trust. We have furthermore analyzed and precisely defined the relation between the notion of trust on the one hand and the notions of risk and opportunity on the other hand. Such a clarification is decisive for the establishment of well-defined languages and methods for trust modeling and analysis, and for communicating the analysis results.

In the remainder of this section we briefly present each of the four artifacts proposed in this thesis and explain how they are related.

1.2.1 The Policy Specification Language

A key feature of policies is that they "define choices in behavior in terms of the conditions under which predefined operations or actions can be invoked rather than changing the actual operations themselves" [103]. This means that the capabilities or potential behavior of the system generally span wider than what is prescribed by the policy, i.e. the system can potentially violate the policy. A policy can therefore be understood as a set of conditional normative rules about system behavior, characterizing the ideal, desirable or acceptable behavior of the system with respect to the domain of management.

In this thesis we have taken a deontic approach (pertaining to norms of duty or obligation) to policy based management, where each rule is classified as a permission, an obligation or a prohibition. We propose a policy specification language, called Deontic STAIRS, which is a notation supporting this approach by being equipped with language constructs customized for the specification of each type of rule.

The deontic approach to policy specification is motivated by the normative character of the requirements imposed by the policy rules. The classification of rules into permissions, obligations and prohibitions is based on Standard Deontic Logic [79] in which these notions and the relations between them have been formalized. Other approaches to policy specification have language construct of such deontic type [1, 61, 70], and the categorization is furthermore implemented in the ISO/IEC standard for open distributed processing [51].

In addition to allowing the specification of normative constraints on behavior, Deontic STAIRS supports the specification of the conditions under which a policy rule applies through a so called policy trigger. The policy trigger specifies system behavior the occurrence of which triggers the given constraint.

Taken together, Deontic STAIRS supports the specification of policy rules that consists of two parts, namely the triggering behavior and the behavior that is constrained by the rule together with the type of constraint. A policy specification is then given as a set of such policy rules.

The relevant behaviors are specified as scenarios that describe the entities involved in the behavior and how these entities interact. By means of abstraction, several parts of the system can be represented as one entity, thus hiding details and internal aspects that are irrelevant from a certain perspective or for a certain purpose.

Deontic STAIRS is based on STAIRS [41, 95], which is a formal approach to system development with UML sequence diagrams, and is underpinned by a formal semantics. The semantics explains the meaning of the policy specifications in precise, mathematical terms, and also supports formal analysis. The formal semantics furthermore allows the development of tool support for the specification, development and analysis of policy specifications.

Deontic STAIRS is not tailored for a specific type of policy, thus allowing the specification of policies for various purposes, such as security, trust, access control, service level and network management. A special purpose policy language will typically have tailored constructs for its domain, but a general purpose language is advantageous as it applicable across domains and at various abstraction levels. Section 5.2 gives a presentation of Deontic STAIRS.

1.2.2 The Policy Adherence Relations

Adherence of a system to a policy specification means that the system (implementation) satisfies the policy. In order to ensure that a policy specification is correctly interpreted and unambiguously understood, the notion of adherence should be well-defined. This is ensured by the formalization of the notion of adherence as a relation between a policy specification and a system to which the policy is applied.

This is illustrated in Fig. 1.1, where the policy specification P is depicted to the left and the system S is depicted to the right. Adherence relates the two, where $P \rightarrow_a S$ denotes that S adheres to P.

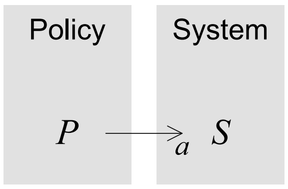

Figure 1.1: Adherence of system S to policy specification P

The formalization of the notion of adherence also yields a notion of policy consistency or policy conflict; a policy specification is inconsistent or conflicting if there is no system that adheres to the policy specification. This is the case if, for example, there is one rule that prohibits a given behavior while another rule permits the same behavior under the same conditions.

In the development of a system to which a given policy applies, the requirements imposed by the policy can be taken into account during or even after system implementation. This may, however, be costly and time consuming, and may also require parts of the system to be redesigned. Alternatively, the policy requirements can be taken into account throughout the system development process by integrating these requirements with the requirements to the system design and functionality. Such an approach requires the relation between a policy specification and a system specification to be well understood. The relation is formalized by a notion of adherence that defines what it means that a system specification satisfies a policy specification, where both specifications may be at any level of abstraction. Section 5.3 gives a presentation of the adherence relations.

1.2.3 The Policy Refinement Relations

Allowing policy specifications to be represented at various levels of abstraction requires a precise characterization of what it means for one specification to be more or less abstract than another specification. This is captured through the notion of policy refinement.

Refinement of a policy specification means to strengthen the specification, taking into account more details about the system to which the policy applies, making the specification more concrete, and bringing it closer to implementation and enforcement. The formalization of refinement as a relation between policy specifications precisely defines what it means that one policy specification is a correct refinement of another.

This is illustrated at the left hand side of Fig. 1.2 with the relations between the policy specifications P_1, P_2 and P_3. The policy specification

P_1 is at a high-level of abstraction, and the squiggly arrow from P_1 to P_2 denotes that the latter is a refinement of the former.

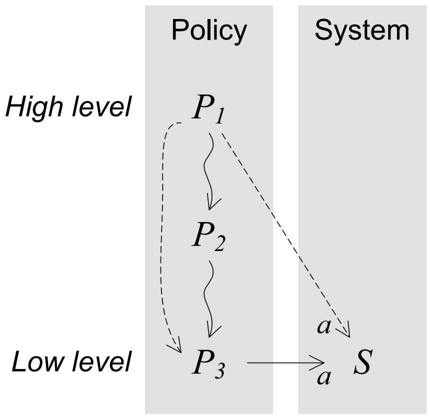

Figure 1.2: Refinement of policy specification

Policy refinement supports the policy development process by allowing the development of a high-level, initial policy specification to a low-level, finalized policy specification to be conducted under any number of refinement steps. This is illustrated by the dashed, squiggly arrow relating the policy specifications P_1 and P_3; since P_2 is a valid refinement of P_1 and P_3 is a valid refinement of P_2, the refinement relation ensures that P_3 is a valid refinement of P_1. Fig. 1.2 illustrates this property with only two refinement steps, but the property holds for any number of steps.

The refinement relation furthermore ensures that if a system adheres to a concrete policy specification, it also adheres to all the previous, more abstract specifications from earlier development phases. This is illustrated in Fig. 1.2 with the dashed arrow from the abstract policy specification P_1 to the system S; since S adheres to the low level policy specification P_3 it also adheres to the abstract specifications P_2 and P_1. This means that the correct enforcement of the final specification correctly enforces the requirements that were captured and formalized during the very initial phases.

The notion of adherence relating policy specifications to system specifications allows the requirements imposed by the policy to be integrated with the process of system development. Since this notion of adherence is defined for policy specifications and system specifications at any abstraction level, policy adherence can be taken into account from the very initial phases of the development of both the policy and the system. For analysis of abstract specifications with respect to adherence to be meaningful, however, the analysis results must be preserved under refinement. Otherwise the analysis must be conducted again after each refinement step. The integration of policy development with system development is supported by results of preservation of adherence under refinement. Section 5.4 gives a presentation of the policy refinement relations.

1.2.4 The Method for the Development of Trust Management Policies

The method for the development of trust management policies aims at identifying and assessing the trust-based decisions within a system, and evaluating the impact of these decisions in terms of the risks and opportunities towards the system. On the basis of these assessments and evaluations the most beneficial decisions are identified. Finally, a trust policy is captured and formalized the enforcement of which ensures that precisely the most beneficial trust-based decisions are made.

As illustrated in Fig. 1.3, the method is divided into three main stages, namely system modeling, trust analysis and trust policy specification. The activities of each stage are furthermore supported by adequate modeling languages.

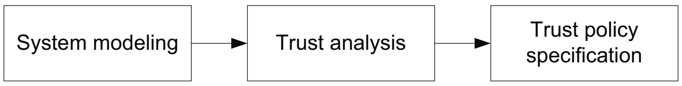

Figure 1.3: Method for developing trust management policies

In order to analyze something it is necessary to obtain a good understanding of what this something is. The goal of the system modeling stage is to describe the behavior of actors and entities within the system that results from trust-based decisions. This modeling includes the basis of these decisions, i.e. the evidence on which the actors base their estimation of the trustworthiness of other entities. Additionally, the system modeling documents the requirements the actors make on the trustworthiness of other entities for accepting a trust-based transaction, i.e. the trust thresholds.

The information documented in the system models obtained through the first stage serves as input to the activities of the second stage, namely the trust analysis. Since trust may be ill-founded, i.e. the evidence on which trust-based decisions are made may be incomplete or even partially false, these decisions must be evaluated with respect to this. Furthermore, the impact of the trust-based decisions on the system in terms of risks and opportunities must be evaluated. Additionally, the impact of alternative decisions should be evaluated since it may be that the current system behavior is not the optimal one. The goal of the second stage of the method is to identify the system decisions that yield the most optimal system behavior by minimizing risks and maximizing opportunities.

Having identified the optimal choices of behavior, these choices are captured in the form of a trust policy which is formalized during the third stage of the method. This policy should describe the conditions under which trust-based decisions should be made and also the requirements to the trust levels for accepting or rejecting a trust-based transaction.

The latter stage of the method is supported by Deontic STAIRS and demonstrates the suitability of applying Deontic STAIRS to the domain of trust management. Section 5.5 gives a presentation of the method.

1.3 Thesis Overview

This thesis is based on a collection of 7 research papers and divided into two parts. Part I presents the context and the overview of the work of the thesis, whereas Part II contains the research papers.

In Chapter 2 we give a characterization of and motivation for the problems addressed by this thesis. We identify needs that the artifacts developed as a part of this thesis intend to meet, and present a set of criteria that the artifacts should fulfill for the successful accomplishment of our research objectives. Chapter 3 presents a method for technology research and explains how we have applied the method in the work of this thesis. In Chapter 4 we present the state of the art of the research within the domain of this thesis, and in Chapter 5 we present each of the artifacts of the thesis. An overview of the research papers and their content is given in Chapter 6. In the discussion of Chapter 7 we evaluate the artifacts with respect to the success criteria of Chapter 2, and discuss related work. Part I is concluded in Chapter 8, where we also discuss directions for future work.

The research papers of Part II can be read independently of each other, but the recommended order is the order in which they are organized. Readers that are only interested in the policy specification language can read paper 3, and consult papers 4, 5 and 6 for the formal foundations and the notions of refinement and adherence, as well as results of preservation of adherence under refinement. Readers that are only interested in trust and trust management can read paper 1 and paper 7.

Chapter 2

Problem Characterization

In this chapter we motivate and characterize the problems addressed by this thesis. In Section 2.1 through Section 2.3 we present the background to this thesis, thereby identifying needs that the artifacts presented in Section 1.2 aim to meet, as well as identifying requirements to these artifacts. In Section 2.4 we summarize by giving for each artifact a set of criteria that should be fulfilled for a successful accomplishment of our research objectives.

2.1 Policy Based Management

Policy based management of information systems [16, 22, 45] has the last decade or so emerged as an adaptive and flexible approach to administer and control distributed systems with respect to issues such as security, access control, services, networks and trust. An international policy community has continued to evolve, and much of the key activities has been centered around the annual Policy Workshop [45] which recently has been lifted to an IEEE Symposium and is arranged for the 10th time in 2009.

An important motivation for the use of policies for systems management is that they allow systems to be dynamically changed in order to meet new or different requirements, without stopping the system and without changing the underlying implementation. This is reflected in a widely adopted definition of policies, namely that a policy is a set of rules governing the choices in the behavior of a system [102].

As mentioned in the previous section, policies govern choices in behavior by specifying the conditions under which predefined operations can be invoked rather than changing the actual operations themselves [103]. This means that the potential behavior of the system generally span wider than what is prescribed by the policy and that the system can potentially violate the policy. A policy can therefore be understood as imposing normative constraints on system behavior, characterizing the ideal, desirable or acceptable behavior of the system with respect to the domain of management.

The deontic approach to policy specification of this thesis, in which policy rules are classified as permissions, obligations or prohibitions, is based on Standard Deontic Logic [79]. The classification is furthermore implemented in the ISO/IEC standard for open distributed processing [51].

The deontic approach is suitable for specification and reasoning about policies because of the normative character of policy rules. The formalization of the deontic notions in Standard Deontic Logic is for reasoning about normative expressions, the relation between them, and what can be deduced from them. The deontic approach is furthermore generic in the sense that it allows the specification of policy rules and the reasoning about them for various domains of policy based management, such as security, access control and trust.

Policy based management should be supported by adequate languages and methods that facilitate the various phases of policy development, analysis and enforcement. With a deontic approach to policy specification, the language must have the expressiveness to capture the modalities of permission, obligation and prohibition. Additionally, since policy rules are conditional rules, the language must support the specification of the conditions under which a given rule applies in the form of a triggering construct.

Policies may be derived from business goals, service level agreements, trust relationships, security requirements, risk analyses, etc. During the initial capturing of a policy, various stakeholders are typically involved, such as chief executive officers and other decision makers, data security officers, system users and system developers. At this point, a policy language suitable for capturing policies at the appropriate abstraction level, that is intuitively understandable to all stakeholders, and that supports communication between stakeholders is desirable.

Abstraction means to identify and represent the essential aspects and characteristics of the system under consideration, omitting details that are unimportant or irrelevant from a certain perspective [94]. The higher the level of abstraction, the less detail included. A policy language facilitating abstraction addresses the problem of “closing the gap between business an IT” [58], thereby avoiding misunderstandings and miscommunication between stakeholders.

From the initial and abstract phases, the policy specification is typically further developed by making it more concrete; details and information are added, specifics about system functionality and architecture are taken into account, and the policy specification is brought closer to implementation and enforcement. During such a process of lowering the abstraction level it is important to ensure that the resulting policy specification is a correct representation of the initial specification, i.e. that the concrete specification fulfills the requirements of the abstract specification. This can be ensured by a well-defined notion of policy refinement.

Although recognized as an important issue already during the very initial research on policy based management [80], policy refinement still remains poorly explored in the literature [6, 92]. Policy refinement is in [80] referred to as the process of transforming a high-level, abstract policy specification into a low-level, concrete one. By formalizing refinement as a relation between policy specifications, we obtain a precise characterization of what it means that a concrete specification correctly represents or fulfills an abstract specification.

In order to support the development process and make it more manageable, the refinement relation should be transitive. Transitivity ensures that the result of any number of refinements is a valid refinement of the initial, most abstract specification, thereby supporting a stepwise and incremental development process. Furthermore, the refinement relation should ensure that the enforcement of the final, most concrete policy specification implies the enforcement of the initial, most abstract specification.

The correct enforcement of a policy specification means that the system under management satisfies the policy, which we refer to as policy adherence. In order to ensure a precise and unambiguous interpretation of policy specification and to support verification of systems against policy specifications, the notion of policy adherence should be precisely defined. By creating a model of the system under consideration, policy adherence can be defined as a relation between a policy specification and the system model.

The deontic notions of permission, obligation and prohibition and the relations between them have been formalized with Standard Deontic Logic [79]. When adopting a deontic approach to policy specification and proposing a language with support for the capturing of such deontic constraints, the interpretation of these constructs should be validated against the axiomatization of Standard Deontic Logic. The adherence relation should therefore capture the properties of these notions as formalized in the logic.

For several reasons, and as indicated by the above discussion, a policy specification language should be underpinned by a formal semantics [44, 63, 86]. A formal semantics yields a precise and unambiguous definition of the meaning of a policy specification, thus avoiding misunderstandings between stakeholders, and avoiding further development or enforcement that deviates from the intended interpretation. A formal semantics also facilitates reasoning about and analysis of policies, for example to ensure that a refined policy specification fulfills the requirements expressed during the initial policy capturing, to detect and resolve policy conflicts or inconsistencies, or to monitor or analyze a system with respect to policy adherence. A formal semantics furthermore facilitates the development of tool support for automated development and analysis of policy specifications.

Decomposition is held as an important feature of system design and development [113] since it allows specifications to be divided into several separate modules that are analyzed or developed in isolation. We believe that modularity of specifications may play an equally important role in the development of policy specifications. The formal semantics of a policy specification language should therefore be compositional, such that the semantics of a diagram can be determined from the semantics of the sub-diagrams and the composition operators.

Modularity may also substantially facilitate the development process by supporting modular refinement. This means that the policy specification can be divided into manageable pieces that are refined separately, and that the policy development can be divided between several developers or development teams.

2.2 Policy Specification

Since its introduction in 1997, the use of the UML [83] has steadily grown, and it is currently the *de facto* modeling standard in the industry. The widespread use of and knowledge about the UML alone, as well as its intuitive and graphical appearance, makes it an interesting starting point in the search for or development of a notation suitable for specification of policies or aspects of policies.

More importantly, however, the UML is interesting for its support for raising the level of abstraction and for being platform independent. The latter means that the UML system and software models do not make any assumptions about the hardware, programming language, operating systems, networks, etc. that implement them. The UML can furthermore be used for the modeling of business processes, work processes and other non-software systems, which is desirable for a general purpose policy specification language. Finally, a policy specification based on the UML facilitates the analysis of the relation between a policy specification and a system specification, where the latter is given in standard UML.

The UML has been evaluated and utilized in various work on policies and policy based management [1, 4, 12, 63, 70], but, as these works indicate, the language offers little specialized support for the specification and analysis of policies. There is therefore a need for extensions of the UML customized for the domain of policy based management.

Policies are rules about system behavior, and since UML sequence diagrams describe dynamic/behavioral aspects of systems they can be considered for policy specification. As opposed to e.g. UML state machine diagrams which describe the complete behavior of single system entities, sequence diagrams describe system behavior by showing how system entities interact. The interaction perspective is suitable for expressing constraints on the behavior in distributed systems.

Sequence diagrams are furthermore the most popular UML notation for specifying and modeling dynamic aspects of systems [30, 111]. A conjecture is that the popularity stems from the intuitive and easy to understand representations of interactions with sequence diagrams. Even with several entities taking part in the interaction, a good sequence diagram facilitates understanding and communication between various stakeholders such as developers, decision makers and security personnel.

Recall from the previous section that we define policy rules as conditional deontic constraints in the form of permissions, obligations and prohibitions. This means on the one hand that we must be able to specify the scenario that triggers the rule, i.e. the condition for the rule to apply, and on the other hand to specify the scenario that is constrained by the rule along with the deontic modality.

Intuitively, a permission rule states that in case the triggering behavior is fulfilled, the constrained behavior should be allowed. This means that the permitted behavior must be offered as a potential choice, yet allowing alternative behavior to be conducted instead. An obligation rule states that if the triggering behavior is fulfilled, the obliged behavior must be conducted. A prohibition rule states that if the triggering behavior is fulfilled, the prohibited behavior must not be conducted.

In order to illustrate the required expressiveness and compare this with the expressiveness of standard UML, we consider in the following some basic, simplified examples of access control of users to an application, where user access to services is granted on the basis of credentials presented to the application by the users.

Assume a permission rule stating that a user U is authorized to access services provided the user has presented a valid credential to the application A. A proposal for capturing this rule is depicted in the diagram *permit* in Fig. 2.1.

The entities participating in the interaction specified by a UML sequence diagram are represented by lifelines with a head containing the name and/or type of the entity. Messages are represented by arrows, where the arrow tail represents the event of sending a message on a lifeline and the arrow head represents the event of receiving a message on a lifeline. The events are ordered from top to bottom along each lifeline, and the sending of a message is ordered before the reception of the same message.

The annotations to the left of the diagram *permit* show which part that constitutes the triggering behavior and which part that constitutes the constrained behavior. In standard UML there is no support for the specification of the conditional relation between the two scenarios. Instead this diagram specifies an example scenario, describing behavior that the system should be able to conduct. Furthermore, a policy rule does not require that the triggering behavior must occur. Instead it requires that *if* the triggering behavior occurs, the rule imposes a corresponding constraint on the continuation of the system behavior. In the case of a permission, there is also no support in the UML to specify that the permitted behavior is behavior that must be offered as a potential choice only.

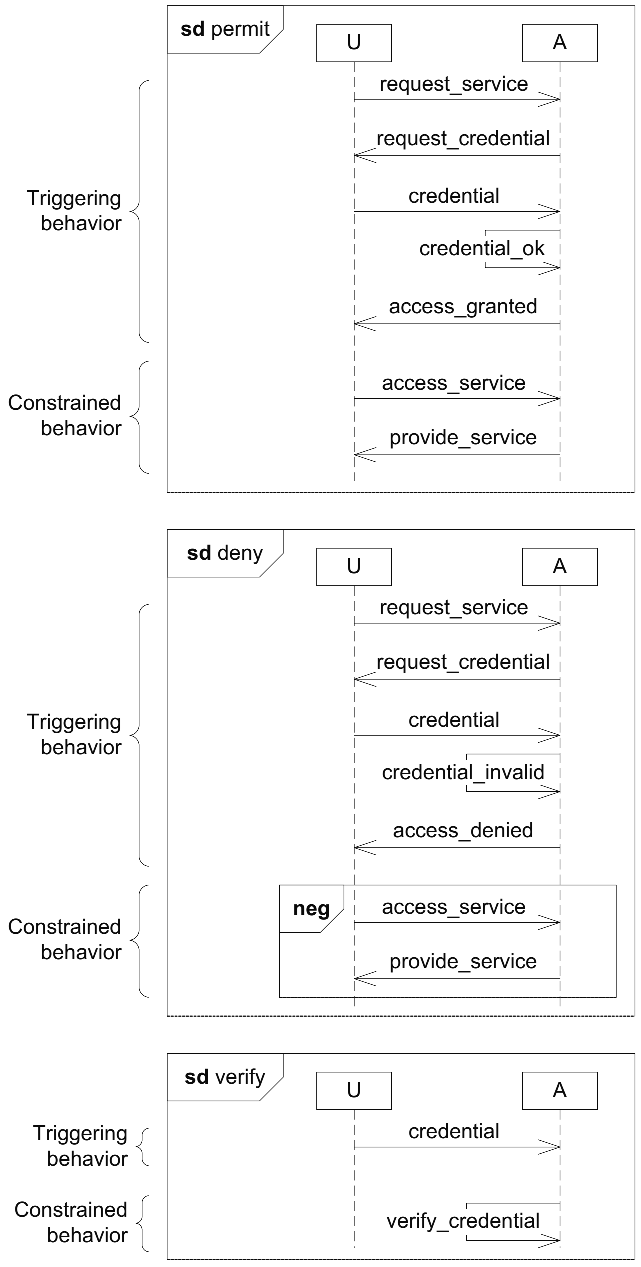

Figure 2.1: Capturing policy rules for access control using standard UML sequence diagrams

Assume next a prohibition rule stating that a user is unauthorized to access services in case the credential presented to the application is invalid. A proposal for capturing this rule is depicted in the diagram *deny* in Fig. 2.1. The expressiveness of UML sequence diagrams to specify illegal behavior can be utilized, e.g. by using the `neg` construct as shown in the diagram. The diagram then expresses that the triggering behavior followed by the constrained behavior is illegal. The triggering behavior in isolation is, however, legal, which is not implied by the prohibition rule we try to specify: The rule only requires that if the triggering behavior occurs, the constrained behavior is not allowed to occur in the continuation.

Assume finally an obligation rule stating that if the application receives a credential from a user, the application is obliged to verify, i.e. to check the validity of, the credential. A sequence diagram specification of this rule is proposed in the diagram *verify* in Fig. 2.1. As before there is no support for capturing the conditional relation between the two scenarios. More importantly, there is also no support to properly distinguish obligations from permissions. Standard UML sequence diagrams distinguishes between positive and negative behavior, but lack the expressiveness to distinguish between behavior that must occur (obligation) and behavior that must be offered as a potential choice while accepting other choices to be made instead (permission).

A further issue to consider is that UML sequence diagrams are normally interpreted as describing exactly the behavior a system may or may not conduct. Consider, for example, the diagram *permit* of Fig. 2.1. With respect to this diagram, system behavior in which for example the user sends several credentials or the application interacts with other entities in parallel is inconclusive. This means that behavior that is not part of the diagram is neither legal nor illegal. When specifying a policy, we should only have to focus on the system behavior and entities of relevance for the policy, e.g. security, trust or service level. This means that for a system to fulfill behavior as described in e.g. the diagram *permit* of Fig. 2.1, any additional system behavior can be interleaved, including behavior involving other system entities. With the standard interpretation of sequence diagrams, the policy specification would have to explicitly specify all the potential system behaviors that fulfill the relevant behavior. Such an approach would not only be tedious and user-unfriendly. It would also result in policy specifications containing much irrelevant information blurring the important aspects, and it would furthermore prevent the capturing and specification of policies without detailed knowledge about the system architecture, entities and functionality.

A final, nevertheless important, concern is of a pragmatic nature, i.e. related to the use and users of the language. In order to support and facilitate use of the language, both for specification and interpretation, a policy specification language should be equipped with intuitive and easily recognizable syntactic constructs for capturing the various parts of a policy. This means that a sequence diagram specifying a policy rule should have a keyword that distinguishes it from other diagrams, there should be constructs that identifies and isolates the triggering behavior and the constrained behavior, and there should be constructs for specifying each of the three deontic modalities. Such an extension of the UML should, however, be conservative and as modest as possible to ensure that people that are familiar with the UML can easily understand and use the notation, and to avoid that the notation is used erroneously.

2.3 Trust Management

When introduced in 1996 [14], the trust management problem was identified as an aspect of security in distributed systems, comprising collectively the management of security policies, security credentials and trust relationships in relation to access control. Since then, trust management has been subject to increased attention and is addressed in a more general setting than security alone. More recently, trust management has been described as an activity “in the intersection between sociology, commerce, law and computer science” [56], focusing on the need for a basis upon which trust can be established and assessed in a setting of electronic interactions over a distributed network such as the Internet.

Trust management involves making assessments and decisions regarding transactions that involve risk [37, 56]. In a situation of trust there is always a possibility of deception or betrayal, which means that there is an inevitable relation between trust and risk [19]. In the management of trust, the level of involved risk must be understood in order to make a well-founded decision with respect to the transaction in question. It is, however, not enough to evaluate the risk alone. The decision to accept the risk involved in trust-based decisions is motivated by the potential reward, i.e. the opportunities involved in the transaction. The management of trust must therefore support trust-based decision making by assessing trust and correctly estimating the involved risks and opportunities.

For a system in which actors make decisions between choices of behavior on the basis of trust, the trust relations have a direct impact on the level of risk and opportunity to which the system as a whole is exposed. A method for analyzing, evaluating and optimizing the trust-based decisions within a system must be supported by adequate techniques and languages for specifying and modeling the relevant parts and aspects of the system. In particular there must be support for the specification of the potential trust relations, the trust levels involved in these relations, as well as the involved levels of risk and opportunity.

The identification and evaluation of all the potential trust-based transactions in which a system can take part should determine the optimal choices of behavior, i.e. the system behavior that yields the minimum level of risks and the maximum level of opportunities. A policy is a set of rules governing the choices in the behavior of a system. In the setting of trust management, a policy can be specified the enforcement of which ensures the optimization of the trust-based decision within the system.

A language supporting the specification of trust management policies must support the specification of the circumstances under which a trust-based decision should be made. Since the level of trust is the decisive factor, the language must also support the specification of the level of trust that is required for a particular transaction to be accepted. Finally, the language must support the specification of the transaction under consideration.

2.4 Success Criteria

The presentation in the above sections demonstrates the need for the artifacts presented in Section 1.2. In this section give a set of success criteria that should be fulfilled by these artifacts, and in Chapter 7 we discuss to which extent these criteria are fulfilled.

In order to identify and understand the requirements to the artifacts, we need to understand who the intended user groups of the artifacts are. For the four artifacts presented in this thesis we distinguish between two main user groups.

On the one hand we have the user group we refer to as the intended end-users. For the policy specification language together with the notions of policy adherence and policy refinement, the typical end-users are personnel responsible for the capturing, specification, development, analysis and/or enforcement of policy specifications. These end-users will typically also need to be able to communicate the content of policy specifications as well as the results of development and analysis processes to other stakeholders such as decision makers and clients. In the same way, these other stakeholders need to be able to communicate their requirements and wishes to the developers. The underlying formalism of the language, as well as the formalization of the adherence and policy refinement relations should be hidden from end-users. Instead, understanding the basic meaning of the policy specifications and what it intuitively means to adhere to and refine policy specifications should be sufficient. In order to ensure that policy specifications are correctly developed and enforced, the end-users should be supported by methods and tools. These may include development guidelines for language use and automated tools for verification of correctness of refinement and detection of conflicts and other errors.

For the method for the development of trust management policies, the typical end-users are trust and risk analysts. The analysts need a method that is understandable, that is applicable, and that is supported by adequate description techniques. Any underlying formalisms of the method and the description techniques should be hidden from the analysts. Instead there should be available tool support for activities such as creating models, deriving analysis results and checking consistency. The analysts furthermore need support for interacting with clients and other stakeholders in gathering the required information for the analysis and in documenting and communicating the analysis results.

On the other hand we have the user group that develop methods and tools for supporting end-users in applying the artifacts. The underlying formalisms of the artifacts are intended for this user group. The formal foundation allows the soundness of any rules of policy development methods and guidelines to be proved, and to verify that automated tool support, e.g. for checking adherence and refinement, works correctly. Developers of tool support for the method for the development of trust management policies may prove that automation of analysis tasks such as risk and opportunity estimation and consistency checking is correct.

The success criteria for each of the four artifacts are presented in the following.

2.4.1 The Policy Specification Language

1. *The language should have the expressiveness to capture the deontic modalities of permission, obligation and prohibition.*

   A policy is a set of normative rules about system behavior, specifying behavior that may, should or should not occur, and the deontic modalities are appropriate for the specification of such constraints. For most domains of management, such as security, access control, services and networks, there is need for rules of all these three modalities.

   For the domain of trust management, trust policies determine the trust levels under which a trust-based transaction should be accepted or rejected, which can be captured by obligations and prohibitions, respectively. In cases where the decision has no impact on the overall level of risk and opportunity, permission rules are suitable for specifying that both acceptance and rejection should be available as allowed choices.

2. *The language should have the expressiveness to capture conditional scenarios.*

   A policy specifies the conditions under which operations or actions can be invoked, and a policy language must therefore support the specification of these conditions. In particular, there should be support for the specification of a triggering scenario that characterizes the circumstances under which the corresponding deontic constraint applies.

3. *The language should be intuitively understandable to end-users such as decision makers, clients, developers and other stakeholders, including personnel with little technical background.*

   During the process of capturing, developing and formalizing a policy many different stakeholders from various backgrounds may be involved. An intuitively understandable notation facilitates the establishment of a common understanding of scenarios and requirements, and supports communication between stakeholders.

4. *The language should allow policy specifications to ignore behavior that is not relevant for the policy.*

   Policy specifications refer only to system entities and system behavior of relevance to the policy, e.g. issues in relation to security, trust or services. This means that the system to which the policy applies generally have entities and behavior that go beyond those mentioned in the policy specification. As such, the scenarios described in a policy specification can be understood as an abstraction of potential system scenarios, thereby facilitating specification and interpretation of policies.

   The language should allow for this in the interpretation of what it means for the system to fulfill behavior as specified in the policy by allowing any irrelevant behavior to be interleaved. Without this requirement the policy would have to explicitly specify all the various ways in which a system may fulfill a triggering behavior or a constrained behavior.

5. *The language should be a conservative extension of standard UML.*

   By a conservative extension we mean that the use of standard UML sequence diagram constructs within the policy specification language complies with the standard, both syntactically and semantically. This is to ensure that the language is easily understandable for people that are familiar with the UML and to avoid that the language is used erroneously.

   Aligning the language with standard UML is desirable also because it facilitates the analysis of the relation between a policy specification and a system specification, when the latter is represented in UML.

6. *The language should be underpinned by a semantics that is unambiguous, allows formal analysis of policy specifications, and allows the development of tool support.*

   An unambiguous semantics is crucial in order to ensure that the implementation and enforcement of a policy indeed is an enforcement of the intended requirements, as different personnel may be involved during the different phases of policy capturing, development and implementation.

Formal analysis of specifications is beneficial for several reasons, such as the detection of inconsistencies or errors, and the verification of policy adherence. Tool support is desirable for facilitating the specification and development process and to do automated analysis of policy specifications.

7. *The semantics should be compositional, meaning that the semantics of a composed diagram can be determined from the semantics of the sub-diagrams and the composition operators.*

   A compositional semantics facilitates the parsing and interpretation of specifications, allowing a specification to be understood by focusing on individual parts separately. A compositional semantics furthermore supports reuse of sub-diagrams and substitution of one sub-diagram by another, without requiring all parts of the specification to be considered simultaneously.

2.4.2 The Policy Adherence Relations

8. *The notion of policy adherence should be intuitively understandable to end-users such as decision makers, clients, developers and other stakeholders, including personnel with little technical background.*

   For end-users to correctly use the policy specification language and correctly specify the desired requirements, the notion of policy adherence must be understood. As the end-users are not supposed to know or understand the underlying formalism the notion of adherence should be understandable at an intuitive level and with reference to the policy specification language. The formalization of the adherence relations allows tool support to be developed for analyzing the relation between policy specifications and systems, aiding the end-users in detecting policy breaches.

9. *The adherence relations should capture the properties of the deontic modalities as axiomatized in Standard Deontic Logic.*

   The deontic modalities have been studied and formalized in the context of Standard Deontic Logic [79]. The adherence relations characterize what it means to satisfy the deontic constraints of a policy specification, and this interpretation should be validated against the axiomatization of Standard Deontic Logic.

10. *The notion of adherence of a system implementation to a policy specification should be independent of the system platform.*

Sequence diagrams and Deontic STAIRS specifications do not assume any particular platform or programming language for the system in question, and the formalization of the adherence relation should therefore allow for this.

2.4.3 The Policy Refinement Relations

11. *The notions of policy refinement should be intuitively understandable to end-users such as decision makers, clients, developers and other stakeholders, including personnel with little technical background.*

    The correct development of policy specifications under the refinement paradigm requires that the notion of refinement is correctly understood by the end-users. As for the adherence relation, the notions of policy refinement should be understandable without knowing the underlying formalization. In order to ensure the correctness of policy refinement steps, the end-users should be supported by development methods and guidelines that ensure refinement, as well as automated tool support for verifying that refinement steps are correct.

12. *The policy refinement relations should ensure that all requirements from the abstract policy specification are preserved in the refined policy specification.*

    The very underlying idea of policy refinement is that the low-level, concrete policy specification that is developed from the high-level abstract one gives a more detailed representation of the abstract specification in the sense that it is closer to implementation and enforcement, while guaranteeing that all requirements are preserved. This means that adherence of a given system to a concrete policy specification should imply adherence of the same system to the abstract specification.

13. *The policy refinement relations should support the stepwise development of policy specifications.*

    For feasibility reasons it should be possible to conduct the development of a policy specification under any number of refinement steps. For policy development under the refinement paradigm to support this, the refinement relations must ensure that the result of any number of refinement steps is a valid refinement of the initial, most abstract specification.

14. *The policy refinement relations should support modular development of policy specifications.*

    Modularity of refinement is desirable as it facilitates the process of developing a policy specification by allowing the specification to be divided into manageable pieces that are refined separately. Modularity furthermore allows the development of a policy specification to be separated between several developers or development teams.

2.4.4 The Method for the Development of Trust Management Policies

15. *The method should be applicable and understandable to end-users.*

    The value of the method relies on the feasibility of applying the method for trust analysis and trust policy capturing. Analysts, i.e. the intended end-users, must therefore understand each stage of the method and how to conduct the tasks of the stages.

16. *The method should offer description techniques that are understandable to all relevant stakeholders, including end-users, decision makers, engineers and analysts.*

    In order to develop a good and adequate policy, it is essential that decision makers, developers, analysts, etc. have a clear and shared understanding of the system, the relevant scenarios and the alternative policy rules. Moreover, the policy rules must be understandable for those who are supposed to adhere to or implement them.

17. *The method should support the modeling of the trust-based decisions within a system, including the entity that makes the decision (the trustor), the level of trust held by the trustor, and the basis upon which the trustor determines this level of trust.*

    In order to understand the impact of trust-based decisions on the overall behavior of the system, the points in which these decisions are made must be identified and described. Next, since the basis of the decisions is trust, the trustor and the level of trust must be explicitly described in order to explain the decision. Finally, in order to explain the trust level and evaluate the well-foundedness of the trust, the evidence used by the trustor for the determination of the trust level must be described.

18. *The method should support the evaluation of the well-foundedness of trust.*

    Trust is a subjective notion, i.e. it is a belief held by the trustor. If the evidence behind this belief is weak or false, the trust is ill-founded, which means that critical decisions are made on failed assumptions. In order to evaluate the well-foundedness of the trust-based decisions, the extent to which trust is well-founded must be determined.

19. *The method should support the evaluation of the risks and opportunities associated with the trust-based decisions.*

Trust-based transactions involve both risks and opportunities. In order to identify the most beneficial trust-based decisions, the levels of risk and opportunity must be evaluated for each potential trust-based transaction.

20. *The method should support the capturing and formalization of trust policies.*

    Having identified the most beneficial decisions, these choices of behavior can be ensured by capturing a policy the enforcement of which minimizes risks and maximizes opportunities. This requires that there is language support for the specification of trust policies, i.e. policies that specify the conditions under which a given trust-based transaction is acceptable. The condition must include the levels of trust under which the rule applies.

21. *The method should be based on well-defined notions of trust, risk and opportunity, and the relations between the notions should be precisely defined.*

    Trust is a complex and compound notion that may mean different things in different contexts. As a result there exists a variety of definitions of and approaches to trust. A trust management method must be based on a precisely defined notion of trust in order to clarify what the method addresses and evaluates. Since risk and opportunity are inherent aspects of trust, these notions must also be precisely accounted for. In particular, the impact of trust on the levels of risk and opportunity must be precisely defined.

Chapter 3

Research Method

Computer science is a relatively young discipline, and it is even debated whether computer science at all can be qualified as a science [2, 17, 28]. In [2] the authors claim that computer science is not a science, and exemplify this by contrasting computation with classical mathematics. The latter, they claim, provides a framework for dealing precisely with notions of "what is", whereas the former provides a framework for dealing precisely with notions of "how to". A similar standpoint is found in [17], where it is argued that science is concerned with the discovery of facts and laws, as opposed to computer science which is an engineering discipline concerned with building things.

The standpoint that computer science indeed is a science is, however, widespread. The examination of computer science as a science in [28] points out that the activities within the field of computer science is a blend of science, engineering and mathematics, and that there are numerous examples of computer science research that is settled in the scientific paradigm. The scientific paradigm is the process of forming and testing hypotheses, where successful hypotheses become models that explain and predict phenomena in the world. Computer science follows this paradigm, the author claims, in studying information processes.

In this chapter we present a method for technology research that rests on the scientific paradigm. Subsequently we describe how we have applied this method in the work that has led to this thesis.

3.1 A Technology Research Method

The scientific paradigm of forming and testing hypotheses is in [108] referred to as the classical research method, and the basic question for the researcher is *What is the real world like?* This is distinguished from technology research, which is "research for the purpose of producing new and better artifacts" [108]. Artifacts are objects manufactured by human beings, and a technology researcher aims at creating new and better artifacts. The basic question leading the technology researcher is *How to produce a new/improved artifact?*

Despite the difference between the basic questions leading the classical researcher on the one hand and the technology researcher on the other hand, the authors assert that the research process in both cases follows the same principal phases, and that technology research should be founded on the classical research paradigm. In both cases, the starting point is an overall hypothesis of the form "B solves the problem A". In classical research A is the need for a new theory, whereas in technology research A is the need for a new artifact.

Having identified the need for a new artifact, the technology researcher identifies requirements that the artifacts should fulfill in order to satisfy the need. The requirements may for example be gathered from the intended users of the artifacts or by considering the environment in which the artifact is to be deployed.

The innovative phase follows the requirements gathering, where the researcher goes about to invent the artifact that is intended to fulfill the requirements. When the artifact is finalized, the researcher must substantiate that the artifact indeed fulfills the identified requirements. The overall hypothesis to be evaluated is the proposition that the artifact satisfies the need that was initially identified.

In order to verify or substantiate that the overall hypothesis is satisfied, a set of more concrete sub-hypotheses must be formulated, the joint satisfaction of which implies that the overall hypothesis is satisfied. By deriving a set of explicit predictions from the sub-hypotheses the falsification of which discredits the overall hypothesis, the researcher is served a basis for gathering evidence about the validity of the sub-hypotheses. Through systematic evaluation, the predictions are tested whereby the researcher builds argumentation for the validity of the overall hypothesis.

In case of a successful evaluation, the researcher may conclude that the invented artifact does satisfy the identified need, and that something new or improved has been created. If, however, the overall hypothesis is falsified, the researcher must go back to the invention phase and improve the artifact or build a new artifact after which the evaluation is conducted again. The innovation and evaluation phase may also lead to new insight that in turn leads to a reformulation of the requirements to the artifact. The technology research process is therefore an iterative process.

Figure 3.1 is adapted from [108] and summarizes the three main steps of the iterative process:

1. *Problem analysis:* The need for a new artifact is identified and requirements to the artifact are gathered.
2. *Innovation:* An artifact intended to meet the identified need is created.

3. *Evaluation:* Predictions about the artifact are formulated based on the hypothesis that the artifact fulfills the identified need. If the predictions come true, the researcher has substantiated that the artifact fulfills the need.

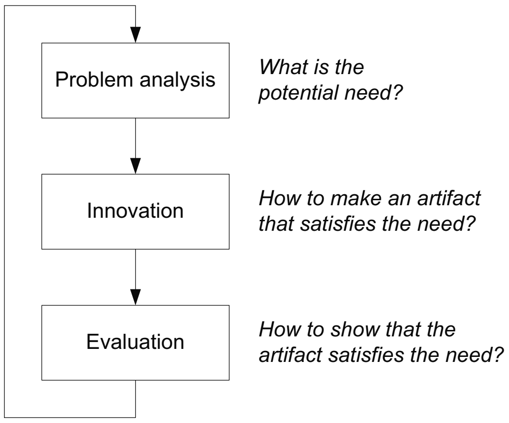
Problem analysis
What is the potential need?
Innovation
How to make an artifact that satisfies the need?
Evaluation
How to show that the artifact satisfies the need?

Figure 3.1: Method for technology research – main steps

Technology development is closely related to technology research since the one often includes activities or aspects of the other. According to [108], technology research is distinguished from technology development by the former giving rise to new knowledge of general interest. More precisely, this means that to qualify as technology research, the invented artifact must represent new knowledge, this knowledge must be of interest to others, and the results and the new knowledge must be documented in a way that it enables examination and evaluation by others.

3.2 How We Have Applied the Research Method

The method applied to the work leading up to this thesis is based on the technology research method described in the previous section. The work has been conducted as an iterative process in which the artifacts and the requirements to them have been changed and improved as new insight was gained while the work progressed.

Part I of this thesis documents the three phases of the research process as illustrated in Fig. 3.1. Chapter 2 documents the problem analysis and the requirements to the artifacts. Chapter 5 documents the innovation phase by presenting the invented artifacts. Finally, Chapter 7 documents the evaluation of the artifacts. In the following we explain in more detail how the method was applied in the development of each of the four artifacts.

3.2.1 Developing the Policy Specification Language

Success criteria 1 through 7 in Section 2.4 summarize the requirements to the Deontic STAIRS language. These requirements were quite persistent throughout the work, but it was initially no matter of course that UML should serve as a basis for the language.

In the search for a suitable notation for our purposes the decisive criteria were the expressiveness, the support for human understanding and communication, and the formal semantics. A natural starting point was to do a broad survey of state of the art policy languages and evaluate these with respect to our criteria. Common to most of the established and commonly used policy specification languages, such as e.g. Policy Description Language [71], Ponder [23], Rei [61] and XACML [82], is that they are purely textual and generally require some formal or technical background for specification or interpretation by humans.

The choice of the UML as a basis for the development of Deontic STAIRS followed by the conjecture that policy development can benefit from established techniques from system development. In particular we aimed to use abstraction to facilitate representation of policies at levels of detail suitable for all the various phases of the development process. As such, abstraction provides support for understanding and communication. We evaluated UML sequence diagrams as a policy language with the STAIRS semantics as a formal foundation. STAIRS [41, 95] formalizes the semantics that is only informally described in the UML standard [83] and is also supported with notions of refinement that we carefully considered as inspiration and basis for policy refinement.

The evaluation of UML sequence diagrams were conducted against our identified requirements to a policy specification language and were done by testing the notation on various examples from the literature and on case studies. As such, the sequence diagram notation were evaluated as one of our artifacts, which helped to precisely identify the needs that this notation does not fulfill. Through iterations of innovation and evaluation, customized extensions were defined in the search for an appropriate language.

In addition to evaluating the Deontic STAIRS language by testing it on examples and case studies, the language has been presented to and tested on potential users of the language from various backgrounds, and it has been presented to and commented on by researchers from various relevant backgrounds. The evaluation of the policy specification language against the success criteria is presented in Section 7.1.1.

3.2.2 Developing the Policy Adherence Relations

The development of the artifact of the policy adherence relations basically amounted to propose and evaluate formal definitions. The application of the research method on the development of this artifact is illustrated in Fig. 3.2, which is adapted from [87].

The process started with the identification of the requirements, which are summarized as success criteria 8 through 10 in Section 2.4. After the identification of the requirements, definitions intended to fulfill these requirements were proposed. The notion of adherence were formalized by searching for a definition aligned with the axiomatization of the properties of the deontic modalities in Standard Deontic Logic [79].

An important part of the evaluation of these definitions was to explore their mathematical properties. This is illustrated in the right hand branch of the diagram in Fig. 3.2. Partly, the evaluation was conducted through precise mathematical proofs of properties we needed to verify, for example that the adherence relation respect the axiomatization of Standard Deontic Logic and that adherence to a refined, concrete policy specification implies adherence to the previous, abstract specification. The proofs were structured following the proof method of Lamport [67], which minimizes the chance of doing errors. In case properties do not hold, i.e. there exists a counter example, the proof method simplifies the identification of the counter example and the identification of the proposed definition that is the source of the failure of the property.

In addition to the mathematical exploration of the proposed definitions, they were tested and evaluated against examples and against our own understanding and intuitions about what should be the desired properties. This is illustrated in the left hand fork of the diagram in Fig. 3.2.

As the diagram further shows, the evaluation triggered new iterations in two ways. On the one hand, evaluation could yield new insight to the problem addressed, in which case the requirements from the initial phase of problem analysis had to be considered again. On the other hand, the evaluation could prove that the identified requirements were not fulfilled, in which case the proposed formal definitions needed to be revised and re-evaluated. The evaluation of the adherence relations against the success criteria is presented in Section 7.1.2.

3.2.3 Developing the Policy Refinement Relations

The development and evaluation of the policy refinement relations followed a strategy similar to the development and evaluation of the policy adherence relations. The identification of the requirements are summarized as success criteria 11 through 14 in Section 2.4. Existing literature on policy refinement were surveyed in the search for an adequate definition of policy refinement, and the refinement relations of STAIRS served as inspiration and basis. The latter was based on our hypothesis that policy development can benefit from established methods for system development, together with the fact that policy refinement is not an established feature of policy based management

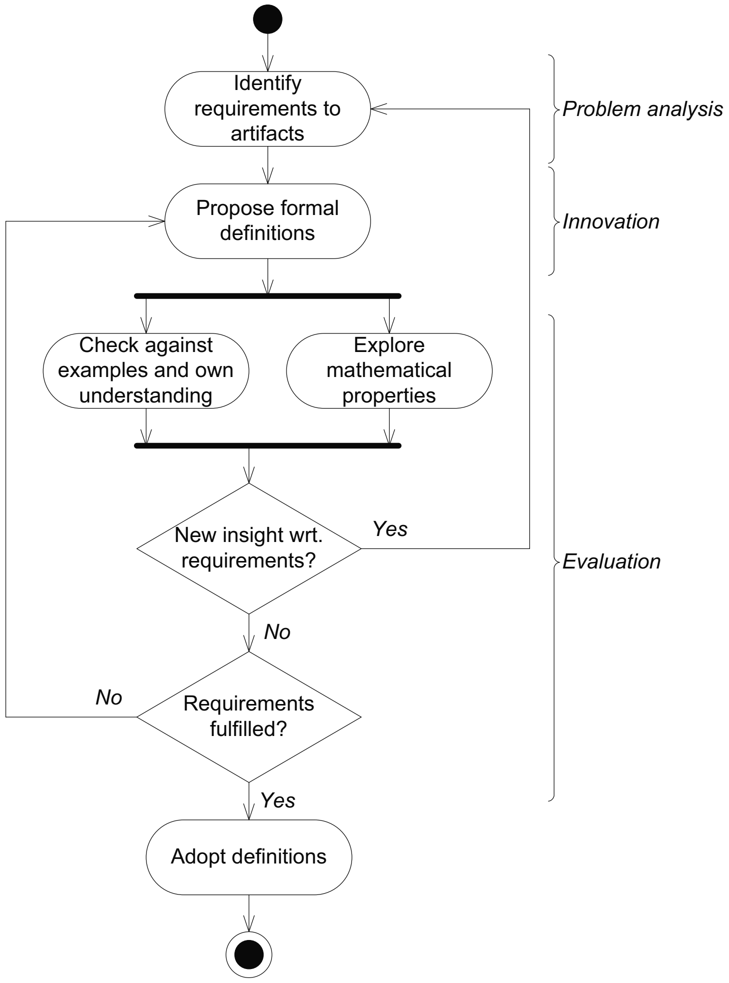
Identify requirements to artifacts
Propose formal definitions
Check against examples and own understanding
Explore mathematical properties
New insight wrt. requirements?
Yes
No
Requirements fulfilled?
No
Yes
Adopt definitions
Problem analysis
Innovation
Evaluation

Figure 3.2: Research method for policy adherence and policy refinement

and is not well addressed in the state of the art [6, 16, 92].

Similar to the development of the policy adherence relations, the iterative process of developing the policy refinement relations followed the pattern illustrated in Fig. 3.2. Mathematical properties such as transitivity and modularity were explored by means of structured proofs and identification of counter examples, and the definitions were tested on examples and our own understanding. The evaluation of the refinement relations against the success criteria is presented in Section 7.1.3.

3.2.4 Developing the Method for the Development of Trust Management Policies

The requirements to the method for the development of trust management policies are summarized as success criteria 15 through 21 in Section 2.4. The proposed method consists of two main parts, namely the description of the three stages of the method, and the specification languages supporting the activities of each of the stages.

The innovation phase consisted of proposing a method intended to fulfill the requirements. In order to demonstrate and evaluate the method, an example case was constructed to which the method was applied and evaluated against. The example is the bank case described in Chapter 15. This particular case was chosen since it represents many key aspects of trust management, such as the modeling and assessment of trust, risk and opportunity. In addition to the proposed method, modeling languages were chosen that we hypothesized would adequately support the activities of the various phases of the method.

The evaluation of the method and its languages was done by testing it on the bank case and on potential users of the language. The most important issue to address here was the usability of the method and its support for assessing and evaluating trust, evaluating risks and opportunities, and capturing a trust policy. The languages used, Subjective STAIRS [89] and Deontic STAIRS, were evaluated with respect to expressiveness and support for the activities of the method, as well as their support for human understanding and communication. Subjective STAIRS is a language for trust modeling and has been developed as part of other work where it already has been evaluated for its suitability to model and assess trust [87].

The first finalized version of the method with its modeling support is documented as part of this thesis. Since then is has been tested in a field study involving an industrial partner and an industrial case in which trust and trust relations are the core issues to understand. The evaluation of the method against the success criteria is presented in Section 7.1.4.

Chapter 4

State of the Art

This chapter presents state of the art within policy based management. Section 4.1 addresses frameworks for and approaches to policy based management, including strategies for policy refinement and policy conflict analysis. Section 4.2 reviews some of the most commonly used policy specification languages, and also presents the formalization of notations that are similar to the policy specification language proposed in this thesis. Section 4.3 addresses trust management, and reviews approaches to trust management, trust analysis and trust policy specification.

4.1 Policy Based Management

The governing of system behavior by means of policy enforcement may fulfill various purposes, and different frameworks and models have been developed in order to support and facilitate the various activities involved in policy based management. In this section we present some common approaches to and strategies for policy based management.

4.1.1 Security Policies and Management Policies

Policies are commonly categorized into security policies and management policies [103], and most approaches reside in only one of the categories, although there also exist general purpose approaches [23, 103].

Security policies specify rules for authorization or access control, permitting only authorized users to access services or resources. The rules commonly describe the conditions under which a subject (the accessor) can access an object (the resource), and also the type of access, e.g. read, write or execute. A well known framework is the access control matrix [11, 68] in which the rights of a set of subjects S on a set of objects O are defined by an access control matrix A. Each entry $A(s, o)$, $s \in S$ and $o \in O$, yields the set of rights of s on o. Discretionary access control [11] is based on the identity of the subject and object. Access rights are constrained by the owner of the object and cannot be passed by one subject to another subject. With mandatory access control [11], access is controlled by system mechanisms and individual users cannot alter the access rules. The Bell-LaPadula model [8] is based on the assigning of security classification to subjects and objects. Subjects can only access objects of equal or lower classification, so the Bell-LaPadula model ensures confidentiality by preventing information to flow from higher to lower levels of security. Biba's model [10] is similar to the Bell-LaPadula model, but is designed to ensure integrity of information by restricting access to do data modification.

Management policies specify rules for management actions that are to be taken in response to changing circumstances. They are used to, for example, administer networks and services for quality of service, determine when to make storage backups and determine when to do software configurations [103]. The rules of a management policy is commonly of the event(s)-condition(s)-action(s) form, requiring the performance of the action(s) by the occurrence of the events(s) provided the condition(s) is fulfilled.

The Policy Core Information Model (PCIM) [81] is joint work between the IETF [49] Policy Framework Working Group and the DMTF [29], and provides a policy model in which each policy rule consists of a set of conditions and a set of actions. The condition set can be expressed in disjunctive or conjunctive normal form, and constraints on the ordering of the action set can be imposed.

Support for various structuring techniques to facilitate the specification of policies has been suggested as a requirement to policy languages [23]. The PCIM policy model has several features that contribute to the fulfillment of this requirement, such as the aggregation of policy rules into a group of rules all pertaining to e.g. a specific department or set of related activities. Furthermore, conditions and actions can be stored in a repository for reuse by multiple rules, and resources can be assigned roles such that single policy rules can be specified for several resources simultaneously.

Role-based access control (RBAC) [33] is a security policy framework in which permissions are related to a position in an organization, rather than to individual people. Separating the specification of policies from the assignment of roles has the obvious advantage that policies do not have to be changed when people are reassigned to other positions. Furthermore, the structuring of roles into hierarchies allows permissions to be inherited by senior roles (e.g. chief executive officer) from junior roles (e.g. employee).

In addition to roles, the use of domains has been proposed as a structuring technique for facilitating policy specification [23]. A domain is in [112] defined as a collection of elements and services that are administered in a coordinated fashion. Single policy rules can then be defined for such collections, e.g. the specification of access for all subjects in one domain to all objects in another domain. Roles and domains typically reflect structures of systems and organizations and may facilitate both specification and interpretation of policies.

The issue of enforcement of security policies is addressed in [98], where a security policy is defined as a specification of system executions as unacceptable. The purpose of the work is to provide a precise characterization of the class of security policies that can be enforced by so called execution monitoring (EM). EM enforcement mechanisms work by monitoring execution steps of a system, and terminating the execution if it is about to violate the security policy being enforced. Mechanisms that use more information than is available only from observing system execution steps are by definition not EM mechanisms.

The problem of observability is addressed in [42] in the setting of specification and enforcement of policies for data protection requirements. Data protection requirements impose constraints on how data may be used in the future, e.g. the obligation that data must be deleted after some period of time. The adherence to such an obligation may be impossible to observe, and therefore impossible to enforce by EM mechanisms. The proposed solution in [42] is to transform non-observable obligations into observable access control requirements and observable obligations. Examples of such transformations are to demand the right to pull evidence from the user about adherence, e.g. through audit, and to impose the duty on the user of pushing evidence. The authors stress, however, that the transformed policy may weaken the goals of the original one, which is the cost of observability. The notion of enforcement is also less strict than in [98] where an execution that violates the policy is terminated; since violations of obligations may not be prevented, enforcement includes the application of compensating actions, such as penalties, to which the user has agreed. This notion of enforcement is also used in [9].

Usage control [84] is an extension of access control by constraining not only access to resources but also the subsequent usage of the resources. Usage control policies are more general than policies for data protection as addressed in [42]. The Obligation Specification Language (OSL) presented in [43] for expressing usage control requirements is underpinned by a formal semantics, and a framework is proposed for specifying, analyzing and enforcing usage control requirements.

4.1.2 Policy Refinement

Policy refinement has been recognized as an important research issue since the initial research on policy based management [80], where it is referred to as the process of transforming a high-level policy specification into a low-level policy specification. A framework for policy refinement is proposed in which policies are organized in a hierarchy in which each level in the hierarchy is derived from the policy in the above level. The policy at the lower level represents a refinement of the policy at the higher level, and the former should fulfill the latter. The relation between the hierarchy levels is, however, not formalized and the judgment of whether a low-level policy correctly or appropriately represents a high-level policy must be left to human managers.

One aspect of policy refinement as proposed in [80] is that of goal refinement, where the set of low-level goals derived from a high-level goal intends to fulfill the latter. Goal refinement has been further elaborated within the area of requirements engineering and has served as basis for more recent approaches to policy refinement. The KAOS method [25] is based on identifying refinement patterns, where each pattern represents a possible decomposition of a high-level goal into a set of low level-goals. The fulfillment of the low-level goals of a pattern implies the fulfillment of the high-level goal. Such patterns are proved correct once and for all, and can then form a library that serve as basis for guiding the refinement process.

Goal refinement and the KAOS method have been adopted by several approaches to policy refinement [4, 6, 91, 92, 93]. The approach presented in [4, 6] identifies the policy refinement problem as composed of two parts. The first part is the refinement of high-level goals into operations supported by the concrete objects/devices, such that when performed will achieve the high-level goal. The proposed solution to this problem is to combine the KAOS goal elaboration method with techniques for deriving so-called strategies, which are mechanisms by which a given system can achieve a particular goal. A strategy is a relation between a formal description of the system for which the policy applies and the goal, and is derived from the latter two by abductive reasoning. The second part of the policy refinement problem is the refinement of abstract entities into concrete objects/devices. The proposed solution to this problem is a formal representation of object relationships based on domain hierarchies with rules for deducing concrete objects/devices for abstract ones. The formalism of the approach is implemented in event calculus [65] which is held as suitable as it allows formal reasoning and fits the event-driven nature of the systems that are addressed by the approach.

The policy refinement framework presented in [93] also uses the KAOS method for goal elaboration to derive low-level goals from high-level ones. The strategy for identifying the system execution traces that fulfill the low-level goals is, however, based on linear temporal logic (LTL) formulas and model checking. The requirements described by a set of low-level goals are encoded into LTL formulas that describe the absence of the fulfillment of the low-level goals. Through model checking, in which a formal system model and the LTL formulas serve as input, a counterexample to the LTL formula is identified, where the counterexample is a system trace. Since the LTL formula describes the absence of the fulfillment of the desired goals, the identified trace indicates system behavior that will achieve the desired goal. The work in [92] extends [93] by introducing a mechanism for abstracting policies from system trace executions in a systematic manner.

The investigation of policy refinement has gained interest only recently [16], and the literature on further approaches to policy refinement is scarce. The work in [109] proposes an approach to the refinement of access control policies by decomposition. The idea is that instead of enforcing a policy for a distributed system in a centralized manner, the enforcement should be local and distributed in order to increase performance. At the abstract level a policy is specified for controlling the access to a resource that represents the resources of the distributed system. Based on a resource hierarchy that describes how resource instances combine into abstract resources, the high-level policy is decomposed (refined) so as to control access to specific, concrete resources such as servers and printers. The combined and distributed enforcement of the low-level policies should then represent the enforcement of the initial, high-level policy for the overall system. In [26] a framework is presented in which low-level policies can be derived from high-level policies based on models of network systems at various abstraction levels. The uppermost level of the network model offers a business oriented view of the network, whereas the lowest level gives a technical view. A high-level policy specified for the uppermost level is then subject to automatic refinement when descending through the abstraction levels of the model. The formalization of OSL [43] defines an ordering of events by a notion of refinement. An event is a pair of an event name and a set of parameters, and an event e_1 is refined by an event e_2 iff the events have the same name and the set of parameters of e_1 is a subset of the set of parameters of e_2. The refined event is therefore more specific and low-level. This notion of refinement allows the specification of requirements on an abstract event that apply to all its refinements.

4.1.3 Policy Conflict Analysis

Policy analysis for the detection and resolution of conflicts or inconsistencies has been recognized as an important research issue within policy based management [16, 103]. Policy conflicts are inherent to policy based management of large distributed systems since different parts of a policy may be specified by different managers, since policies are enforced in a distributed manner, and since various rules may apply to the same entities [74]. A policy conflict occurs, for example, when an obligation requires an activity to be conducted for which there is no authorization. In [74] such conflicts are referred to as modality conflicts and are inconsistencies that arise when two or more rules with opposite modalities refer to the same entities and actions.

Some of the potential policy conflicts can be detected syntactically. An example from [74] gives a pair of conflicting rules in which the first states that users are prohibited from rebooting workstations, and the second states that system administrators, which are also users, are permitted to reboot workstations. Such a conflict can be resolved by rewriting the first rule or by excluding the system administrators from the user domain such that the first rule does not apply to them. The authors argue, however, that rewriting and reimplementing rules may be inconvenient since it possibly is resource consuming and a lengthy activity. Excluding the system administrators from the user domain is also not desirable since there may be several other rules that apply to the users that also should apply to system administrators.

The two rules should therefore be allowed to coexist, and strategies must be implemented to ensure that the first rule does not apply to system administrators. The suggested strategy is to impose a precedence relationship, for which there are several principles. One such strategy is to have prohibitions to take precedence over permissions. Such a strategy may be appropriate for many situations, but may also be too inflexible. In the given example it means that system administrators will not be permitted to reboot workstations, which is an undesirable interpretation. A second strategy is to assign priorities to rules, thus defining a precedence ordering. This is implemented in the policy model in [81]. A problem that may arise, however, is that the specified priorities do not reflect the actual importance of the rules since priorities may be assigned by different managers at different locations. The third proposed strategy is to determine priority based on the distance between a policy rule and the managed objects, as suggested in [69] regarding authorization policies for object-oriented databases. A policy rule for a subclass will take precedence over a conflicting rule defined for its superclass since the former rule is closer to the entities for which the rule applies than the latter rule is, i.e. the former is more specific. The fourth and final principle for defining a precedence relationship proposed in [74] is similar to the previous one. The principle is that rules for a sub-domain should have precedence over its ancestor domain since the former is more specific. A rule specified for a system administrator, for example, will then take precedence over a rule specified for a user, since the system administrator domain is a sub-domain of the user domain. In [5] policy specifications are formalized using event calculus, which supports policy analysis for conflict detection.

4.2 Policy Specification

In this section we first present some of the most established languages that have been developed for policy specification. Subsequently we present languages that are comparable to the policy specification language proposed in this thesis.

4.2.1 Policy Specification Languages

The Policy Description Language (PDL) [71] is a declarative language that was developed for network management. It is based on the event-condition-action format, which are rules that are triggered when the event occurs and require the execution of the action provided the condition is true. Events can be primitive, or they can be composed by e.g. disjunction, conjunction (simultaneous occurrence) and sequencing. PDL is underpinned by an operational semantics and has been implemented and demonstrated in use for Lucent switching products [110].

Ponder [23] is a declarative, object oriented language with support for the specification of both security policies and management policies. Security policies are specified as positive or negative authorizations and define activities that subjects can or cannot perform on targets, where a target is a service or a resource. Management policies are specified as event-condition-action rules, similar to PDL, and specified as positive or negative obligations. Both authorization rules and obligation rules can optionally be specified with a constraint that defines the conditions under which the rule applies.

An important feature of Ponder is that of domains, which are the grouping of subjects or targets to reflect geographical or organizational boundaries, object type, management responsibility, etc. for the convenience of human managers. Domains are similar to directories, and can be organized in hierarchies with sub-domains. Instead of specifying rules for individual subjects and targets, rules are specified for domains, which may substantially facilitate policy specification for large-scale systems. Ponder has also support for the specification of information filtering policies, refrain policies and delegation policies. Information filtering means to allow access only to partial information, filtering away a subset of information that is held as more sensitive. Refrain policies are similar to negative authorizations and specify actions that subjects must not perform. The difference is that whereas the former are enforced on the target side, the latter are enforced by subjects. Refrain policies are relevant in situations in which the target cannot be trusted to enforce policies. Delegation policies specify transfer of access rights and are often relevant for access control systems. Finally, Ponder supports the grouping and structuring of policies to facilitate management, as well as the specification of roles and role hierarchies.

The Policy Definition Language [64] (not to be confused with PDL described above) precedes Ponder, and has a very similar syntax. Rules are specified as obligations or authorizations, and define constraints on actions performed by a subject on a target. As for Ponder, rules are event triggered and may be specified with a condition. Additionally, there is support for specifying domains of subjects and targets.

Rei [61] is a policy framework aimed at supporting the specification and analysis of policies in pervasive computing environments. The policy specification language is based on deontic concepts, and includes constructs for rights, prohibitions, obligations and dispensations (deferred obligations). Additionally, there are constructs for specifying meta policies that are invoked to resolve occurrences of policy conflicts. Policy rules apply to subjects that can be specified as individuals, roles or groups, or combinations of the three.

LaSCO [44] is a graphical approach to the specification of security constraints on objects, where the constraints restrict access to system resources. In this framework, a policy consists of two parts, namely a domain and a requirement. The domain is a specification of assumptions about the system, whereas the requirement is a specification of what is allowed assuming the domain is satisfied. The system is represented by a set of objects that interact through events, and the policy puts constraints on these events.

Syntactically, objects are represented by nodes and events relating objects are represented by directed edges. Both the entities whose access are constrained and the entities representing the resources are represented as objects. A user object can, for example, be related to a file object with a read event. The objects and events together form the domain specification. The requirement is specified as an annotation on the edge and is a predicate that must evaluate to true for the event to be allowed. Such a predicate may, for example, be that the security classification of the user must be equal to or higher than the security classification of the file. The graphical representation of policies is limited to nodes for specifying objects and edges for specifying events that relate objects, and the notation relies heavily on textual annotations. The approach is limited to access control since there is no support for the specification of obligations.

XACML [82] is an OASIS standard for the specification of policies for information access over the Internet. The language is an XML notation that specifies authorizations to objects, themselves also represented in XML. The policy rules in XACML specify the actions a subject can perform on a resource. A subject can be represented as an identity, a group or a role, and the resource can be specified at any granularity down to single elements within the document. XACML also includes a request/response language for specifying queries to check whether a given action should be allowed, and to interpret the result. The notations are quite verbose and low level, and not aimed at human interpretation.

The XACML framework assumes a policy architecture featuring a policy decision point (PDP), a policy enforcement point (PEP) and a policy repository. An access request is presented by the subject to the PEP which is the entity that protects the resource. The PEP formulates the request in the XACML request language and forwards it to the PDP which compares it with the XACML policies in the repository. The response, which may be permit, deny, not applicable or indeterminate, is returned to the PEP which, based on the response from the PDP, authorizes or denies access.

4.2.2 Interaction Specifications

The policy specification language proposed in this thesis is based on UML sequence diagrams [83] and its formalization with STAIRS [41, 95]. UML sequence diagrams specify interactions between system entities by describing the exchange of messages between the entities. The sequence diagram notation is largely based on the ITU recommendation message sequence charts (MSCs) [47]. In the following we present various approaches to the specification of interactions and their formalization.

The MSC notation is equipped with operators for sequential, parallel and alternative composition, as well as operators for specifying loop and optional interactions. The intuitive interpretation of these constructs is similar to that of standard UML and STAIRS. There are, however, no constructs for specifying illegal or prohibited behavior. MSCs are also not supported by a well-defined notion of refinement. An operational semantics for MSCs is provided by ITU [46], which is based on work by Mauw and Reniers [77, 78].

In [62], a compositional denotational semantics for MSCs based on the notion of partial-order multi-sets is proposed. This denotational semantics complements the standardized operational semantics, and does not aim to introduce new expressiveness. Refinement is also not addressed.

The variant of MSCs presented in [66] is underpinned by a formal semantics and supported by formal notions of refinement. Whereas the ITU operational semantics is based on process algebra, the MSC semantics in [66] is defined in terms of streams. Streams consist of a sequence of system channel valuations and a sequence of state valuations. A specification is represented by a set of streams, and the existence of more than one element in the set indicates nondeterminism. The proposed notions of refinement include the reduction of possible behavior and component decomposition.

A feature of this variant of MSCs that go beyond that of the ITU recommendation is a trigger construct that supports the specification of one scenario causing the occurrence of another scenario, thus facilitating the capturing of liveness properties. Additionally, the approach in [66] operates with four different interpretations of MSCs, namely existential, universal, exact and negated. The existential interpretation means that the behavior described by the MSC should be possible, in the sense that the behavior cannot be prohibited in all executions. The universal interpretation means that the specified behavior must occur in all executions. The exact interpretation is a strengthening of the universal one, and means that all behaviors that are not explicitly described by the MSC in question are prohibited. The negated interpretation means that the specified behavior is unwanted and should not occur.

Live sequence charts (LSCs) [24, 40] are an extension of MSCs that particularly address the issue of expressing liveness properties. LSCs support the specification of two types of diagrams, namely existential and universal.

An existential diagram describes an example scenario that must be satisfied by at least one system run, whereas a universal diagram describes a scenario that must be satisfied by all system runs. Universal charts can furthermore be specified as conditional scenarios by the specification of a prechart that, if successfully executed by a system run, requires the fulfillment of the scenario described in the chart body.

The universal/existential distinction is a distinction between mandatory and provisional behavior, respectively. Such a distinction is also made between elements of a single LSC by characterizing these as hot or cold, where a hot element is mandatory and a cold element is provisional. LSCs furthermore have the expressiveness to specify forbidden scenarios by placing a hot condition that evaluates to false immediately after the relevant scenario. The condition construct of LSCs and MSCs corresponds to UML 2.1 state invariants and is a condition that must evaluate to true when a given state is active. If and when the system fulfills the given scenario, it is then required to satisfy the false condition, which is impossible.

An operational semantics for LSCs is provided in [40]. The semantics is tailored for a tool for requirements capturing and the execution of LSCs. Whereas the semantics of MSCs is that of weak sequencing, which yields a partial ordering of events, the semantics of LSCs requires the beginning of sub-diagrams to be synchronization points between lifelines. As a result the semantics of a loop, for example, is generally not the same as the semantics of a diagram in which the iterations are written out as a series of interactions.

Modal sequence diagrams (MSDs) [39] are defined as a UML 2.0 profile. The notation is an extension of UML sequence diagrams, and is based on the universal/existential distinction of LSCs. The main motivation for the development of the MSD language is the problematic definitions of the assert and negate constructs of UML sequence diagrams. The authors observe that the UML 2.0 specification is contradictory in the definition of these constructs, and also claim that the UML trace semantics of valid and invalid traces is inadequate for properly supporting an effective use of the constructs.

The semantics for MSDs is basically the same as for LSCs. The main difference is that the LSC prechart construct is left out. Instead, a more general approach is adopted in which cold fragments inside universal diagrams serve the purpose of a prechart. A cold fragment is not required to be satisfied by all runs, but if it is satisfied, it requires the satisfaction of the subsequent hot fragment. As for LSCs, MSDs are not supported by a well defined notion of refinement.

Triggered message sequence charts (TMSCs) [101] are an approach in the family of MSCs the development of which was motivated by the fact that MSCs do not have the expressiveness to specify conditional scenarios, i.e. that one interaction (the triggering scenario) requires the execution of another (the action scenario). The work was also motivated by the observation that MSCs are not supported by a formal notion of refinement, and that MSCs lack structuring mechanisms for properly grouping scenarios together.

The triggering scenarios of TMSCs are similar to the trigger construct of the MSC variant in [66] and the precharts of LSCs. An important semantic difference, however, is that whereas the former are synchronized at the beginning of precharts and main charts, TMSCs are based on weak sequencing in the spirit of the MSC recommendation of ITU.

The semantics of TMSCs is defined in terms of so-called acceptance trees. Acceptance trees record the traces that are defined by a specification, and also distinguish between required and optional behavior. Similar to STAIRS and Deontic STAIRS, TMSCs are supported by a formal notion of refinement that is transitive and modular.

4.3 Trust Management

Trust management was first addressed as a separate problem in [14], where it is referred to as the problem of authentication and authorization in distributed systems. The underlying idea of the approach is that a system does not need to verify the identity of entities accessing system resources, only trust them to do so. Rather than authenticating entities, the approach is based on verification of credentials and the assignment of access rights to these. Given an access request, the decision of whether access should be granted or not depends on a policy and a set of credentials. The approach is implemented in PolicyMaker [15] and further developed in KeyNote [13].

Trust negotiation, see e.g. [114], is an approach to automated trust establishment between entities that are previously unknown to each other, and that communicate over an electronic network. In this approach trust is gradually built through the exchange of credentials and access control policies. The negotiation begins with an entity A requesting access to a resource controlled by an entity B. B may then reply by requesting A to present a particular credential, as specified by B's access control policy regarding the resource in question. If the credential is held as sensitive by A, A will reply by requesting a credential from B. Gradually, trust can be established by such exchange of credentials and access control policies. In practice the trust negotiation process is conducted by automated agents and is transparent to users.

The above approaches to trust management address the preservation of information security in relation to trust. As the interest in trust management has evolved and increased the last decade, the scope of trust management has also widened. Interactions that traditionally have taken place in face-to-face situations increasingly occur over electronic networks such as the Internet. This includes commerce, entertainment, social relations, public services, etc. The approach presented in [85] suggests that what needs to be understood in relation to trust is the involved risks, and that trust management is "to identify the circumstances under which we are prepared to accept risks that may be exposed by relying on certain entities". Such a broader scope is also proposed in [56], where trust management is described as an activity "in the intersection between sociology, commerce, law and computed science", and is "to make assessments and decisions regarding the dependability of potential transactions involving risk".

The works presented in [37, 38] take the view that trust management is "the activity of collecting, codifying, analysing and presenting evidence relating to competence, honesty, security or dependability with the purpose of making assessments and decisions regarding trust relationships for Internet applications". The evidence may be certificates, risk assessments, usage experience or recommendations. The SULTAN framework presented in these papers supports the specification and analysis of trust relations, as well as the evaluation of associated risks. Trust is defined as a quantified belief, where the quantification can be in a numerical range or classified as e.g. low, medium and high.

The SULTAN language specifies trust relations as policy rules of the following form: *PolicyName:* **trust***(Tr,Te,As,L)* $\leftarrow$ *Cs;* , where *PolicyName* is the name which is unique, *Tr* denotes the trustor, *Te* the trustee, *As* the action set, *L* the trust level, and *Cs* the constraints that must be satisfied for the trust relationship to be established. Notice that *L* ranges from complete distrust to complete trust, so the rules may also specify the conditions under which an entity should not be trusted.

SULTAN also captures positive and negative recommendations in the following form: *PolicyName:* **recommend***(Rr,Re,As,L)* $\leftarrow$ *Cs;* , where *Rr* is the recommendor, *Re* is the recommendee, and *L* is the recommendation level, i.e. the level of confidence in the recommendation issued by *Rr*.

Recommendations and trust relations can in the SULTAN framework be combined such that a recommendation serves as the basis for trust or vice versa. Using this language a system manager can specify rules and conditions for trust based decisions. The SULTAN framework also consists of an analysis tool that can be queried about specifications, for example to detect inconsistencies. The SULTAN risk service calculates risk levels by returning the probability of an unwanted incident to occur, along with the loss of asset value in case of the occurrence. The probabilities and degrees of loss must be fed into the service, e.g. based on monitoring, studies and statistics.

Recommendations can be gathered from known peers directly in order to estimate the trustworthiness of other entities. In [57] this is brought a step further where recommendations are passed from peer to peer and form a basis for trust to propagate over networks. The problem addressed is that of trust transitivity which implies that an entity A may trust an entity C based on the fact that A already trusts an entity B that in turn trusts

C. A framework is proposed for modeling and reasoning about transitive trust in chains and networks, and for identifying the conditions under which transitive trust may occur.

The approach in [57] uses subjective logic [53] for expressing trust and computing trust transitivity. Subjective logic is a belief calculus which is related to probability theory, but where the sum of the possible outcomes not necessarily add up to 1. When rolling a die there are six possible outcomes, the probabilities of which add up to 1. Subjective logic models the belief held by an agent in the truth of a statement by a triple (b, d, u) where b denotes the belief, d denotes the disbelief and u denotes the uncertainty, such that $b, d, u \in [0, 1]$ and $b + d + u = 1$. Subjective logic can then be used to represent trust as the belief held by the trustor in the trustworthiness of the trustee with respect to a given transaction.

The work in [54] shows how trust modeling using subjective logic can be combined with reputation systems to support users in assessing quality of web-based services. Reputation is global and objective in the sense that all users will see the same reputation score for a particular agent. Trust, on the other hand, is local and subjective in the sense that different users generally have different trust in the same agent. By combining the reputation systems with trust analysis, the authors claim that we are provided a more powerful and flexible framework for online trust and reputation management.

A method for belief-based risk analysis supported by the belief calculus of subjective logic is presented in [55]. Traditional methods for risk analysis, e.g. [3, 21, 27], follow a pattern of identifying assets, vulnerabilities and threats, as well as evaluating the levels of risk the assets are exposed to as a result of the threats and vulnerabilities. The claim in [55] is that such methods can be more general and flexible by using subjective logic. The idea is that the opinions gathered regarding threats and vulnerabilities may include a degree of ignorance, which should be taken into account when making assessments. The resulting risk assessments will then document the degree of ignorance or uncertainty regarding the estimations, and thereby give a more true picture than traditional methods are capable of.

The trust management method presented in [35] uses requirements engineering methods that support the modeling of organizations and actors, as well as the modeling of trust and trust relationships. The work is motivated by the observation that existing requirements engineering methods treat non-functional requirements, such as security and trust, at a computer system level, i.e. at the level of software, hardware and networks. The claim is that this level is generally not appropriate for trust management, which requires an understanding at an organizational level. The method offers modeling languages for specifying issues such as dependencies, trust and delegation. A dependency means that one actor depends on another actor to accomplish a goal, execute a task or deliver a service. Trust is a relation between two agents, where the trustor trusts the trustee with respect to the accomplishment of a goal. Delegation means that one actor delegates to another actor a goal, the permission to execute a task, or access to a resource. The models and their formalizations allow the analysis of trust at early phases of requirements engineering with the advantage that organization oriented requirements with respect to trust and security can be captured, while ignoring details about the realization of these with, for example, digital certificates or access control mechanisms. The method is supported by automatic verification of requirements, and the authors claim that the model can be enhanced to capture degrees of trust.

The method presented in [35] is further developed in [36] to take into account two different levels of analysis, namely a social level and an individual level. The former addresses the structure of organizations, whereas the latter addresses individual agents. This distinction is important, the author states, where an organizational role that normally is trusted is played by an untrusted individual. The approach offers techniques for the modeling and analysis of the two levels, and the automatic detection of conflicts that may arise.

Chapter 5

Overview of Invented Artifacts

In this chapter we give an overview of the artifacts that are proposed in this thesis. We begin by presenting the overall picture where we explain how the artifacts are related and how they combine into a larger framework for policy based management. Subsequently, each of the artifacts is presented separately in more detail. We also give references to the chapters in Part II for further details.

5.1 Overall Picture

Policy based management of information systems basically amounts to enforcing a set of rules for the purpose of fulfilling some critical requirements to the systems. The process from the initial requirements capture down to policy enforcement comprises a variety of activities, however, each of which should be supported by adequate methods.

We refer collectively to the three first artifacts of this thesis, namely the policy specification language, the policy adherence relations and the policy refinement relations, as Deontic STAIRS. Deontic STAIRS is a domain independent approach to policy based management, with support for policy capturing, formalization, development and analysis.

During the initial policy capturing, the policy rules are derived from business goals, security requirements, trust relations, service level agreements, etc. This activity typically involves personnel of various backgrounds, and it is important to ensure communication between the participants and to establish a common understanding of the requirements. The requirements must furthermore be correctly communicated to the technical personnel responsible for developing and implementing the policy.

Deontic STAIRS facilitates this high-level phase of policy capturing by the support for abstraction. Abstraction involves the perspective or purpose of the viewer, and different purposes result in different abstractions [94]. During the initial activity of policy capturing, any details about system entities, architecture and functionality that are irrelevant or unimportant from the viewpoint of the involved personnel can be ignored.

The representation of policy specifications at various levels of abstraction is depicted at the left hand side of Fig. 5.1. P_1 denotes the initial, most abstract specification that results from the policy capturing phase.

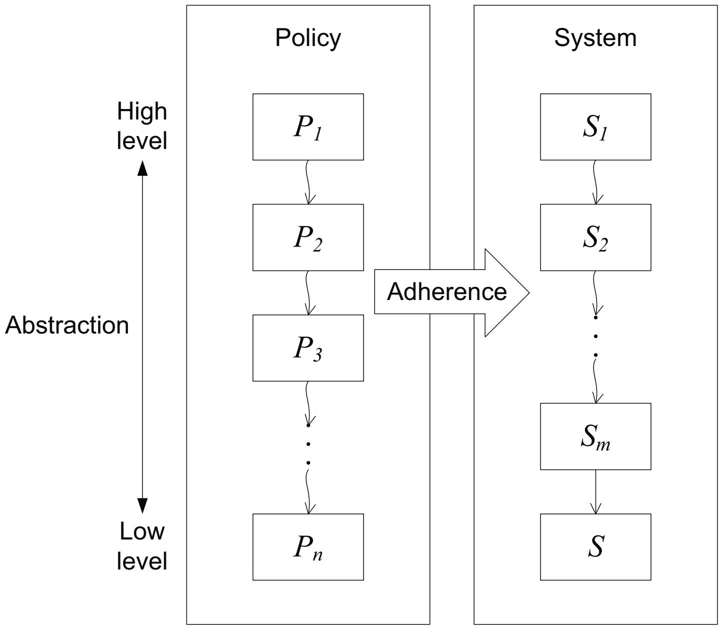

Figure 5.1: Overview of artifacts

The notion of abstraction also pertain to the relation between specifications [94]; in order to be able to represent policy specifications from different viewpoints and at different levels of detail, it is necessary to precisely characterize what it means that one policy specification is an abstraction of another policy specification. This characterization is in Deontic STAIRS formally captured through the notions of policy refinement. Policy refinement is defined as a binary relation between policy specifications that precisely captures what it means that one policy specification is more concrete and low-level than another policy specification. In Fig. 5.1, the policy refinement relation is represented by the squiggled arrows between the policy specifications P_1 and P_2, P_2 and P_3, etc., down to the most concrete and low-level policy specification P_n.

The policy refinement relations of Deontic STAIRS not only capture the notion of abstraction. They also support the process of developing a policy specification from the initial and high-level to the final and low-level policy specification to be implemented. Firstly, the policy refinement relations support an incremental development process where the policy specification can be refined under any number of steps. Secondly, the policy refinement relations support a modular development process where a policy specification can be decomposed into several manageable pieces that are developed separately, possibly by several development teams.

The correct formalization, development and enforcement of policies require that policy specifications are correctly and unambiguously understood. It must therefore be precisely described what it means that a system satisfies a policy specification. In Deontic STAIRS this is formalized through the notion of policy adherence. This is illustrated in Fig. 5.1 by the arrow from the policy representations on the left hand side to the system representations at the right hand side. By S_1 through S_m, we denote various system specifications, whereas S denotes a system that implements these specifications.

The artifact of the adherence relation is directly related to the artifact of the policy specification language since it defines how to satisfy specifications in the language. It is, however, also related to the artifact of policy refinement. Policy refinement means to bring a policy specification to a lower level of abstraction, thus bringing it closer to implementation and enforcement. A refined specification should fulfill the requirements represented at the abstract level, which means that the enforcement of the concrete specification should imply the enforcement of the abstract specification. The fulfillment of this property is demonstrated by showing that adherence to a refined policy specification implies adherence to the previous, more abstract policy specification. With reference to Fig. 5.1, if it is established that the system S adheres to the concrete policy specification P_n, it follows that S adheres to all of the more abstract policy specifications up to P_1.

Through the adherence relation we also capture the notion of consistency of policy specifications. Inconsistency of policy specifications arise from conflicting policy rules, in which case the set of systems that adhere to the policy specification is empty. The adherence relation therefore facilitates not only the analysis of the relation between policy specifications and systems, but also the analysis of policy specifications for detection of inconsistencies.

Deontic STAIRS furthermore allows analysis of the relation between policy specifications and system specifications by characterizing what it means that a system specification adheres to a policy specification. Such a characterization is important for situations in which a system is developed and an existing policy imposes requirements to the system under development. In Fig. 5.1, this means that for each of the policy specifications P_i to the left and each of the system specifications S_j to the right, the adherence relation characterizes what it means that S_j satisfies P_i.

The squiggled arrows from the system specification S_1 to the system specification S_2 and down to the system specification S_m to the right in Fig. 5.1 illustrate the process of system development under refinement. The combined view of policy refinement and system refinement allows analysis of the relation between a policy specification and a system specification at all phases of the respective development processes. However, for analysis of the relation between abstract specifications P_i and S_j to be meaningful, the analysis results must be preserved under refinement. Otherwise the analysis must be conducted from scratch after each refinement step [76]. In this thesis we have addressed the issue of preservation of policy adherence under refinement. By identifying the conditions under which policy adherence is preserved under refinement of policy specifications and under refinement of system specifications, analysis can be conducted at abstract phases when it generally is easier and more efficient. Early analysis is beneficial also because changes are generally quicker and cheaper to implement during the early phases of the development process.

Policy refinement is not well addressed in the research on policy-based management, which means that the problem of preservation of policy adherence under refinement is also scarcely addressed. The problem of integrating policy requirements with the requirements to system design and functionality, and the preservation of these under the development process is, however, recognized and well addressed. This is particularly the case for security requirements.

In [76] it is argued that security requirements should be taken into account during the system development process. The reason is that enforcing security only at the end of the development process "by preventing certain behaviors. . . may result in a so useless system that the complete development effort would be wasted" [76]. A further argument stated in [59] is that "it would be desirable to consider security aspects already in the design phase, before a system is actually implemented, since removing security flaws in the design phase saves cost and time".

The problem of security preservation under refinement addressed in e.g. [52, 76] assumes that the security properties to be enforced are given already at the beginning of the system development process. The challenge is therefore to ensure that these properties are preserved under refinement. The capturing and specification of the desired security requirements is in this case held separate from the system development in which these requirements are integrated. In the context of policy-based management this would correspond to a development case in which the policy is captured and developed before the system to which the policy applies is developed while ensuring that the system adheres to the policy.

The approach proposed in [7], on the other hand, is an integration of the security model with the system model in a combined development process. The system model specifies the system design and functionality, whereas the security model specifies the access control policy for the system. A similar approach is proposed in [59, 60] for providing support for various security aspects to be taken into account during the overall system development.

Fig. 5.2 illustrates these two development cases of taking the policy specification into account during the system development. Development case (a) shows a separate development process in which the initial, most abstract policy specification P_1 is developed under refinement into the resulting, concrete policy specification P_3. Thereafter, the system is developed from the initial, abstract system specification S_1 down to the resulting, concrete system specification S_3. The solid, straight arrow denotes that S_1 adheres to the final policy specification P_3, i.e. $P_3 \rightarrow_a S_1$. We show that the refinement relation of STAIRS is adherence preserving, which means that if $P_3 \rightarrow_a S_1$ holds, so do $P_3 \rightarrow_a S_3$, which is denoted by the dashed arrow of development case (a) in Fig. 5.2.

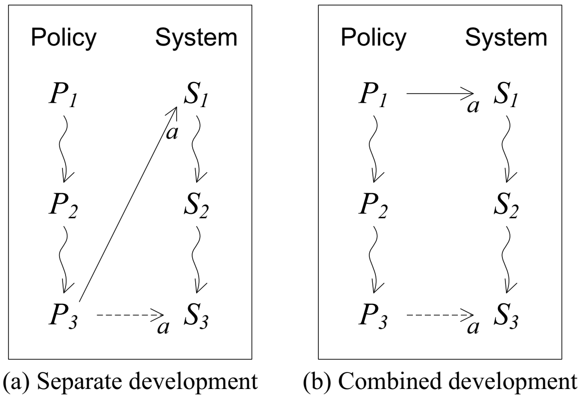

Figure 5.2: Development cases

In development case (a) the policy specification is fixed under refinement of the system specification. Development case (b) in Fig. 5.2 shows a combined development process in which both the policy specification and the system specification may undergo refinement. As for development case (a), if adherence is established at an abstract level the result of this analysis effort should be preserved under refinement. This is illustrated with $P_1 \rightarrow_a S_1$ at the abstract level in Fig 5.2 and the dashed arrow relating P_3 and S_3 denoting preservation of adherence at the refined level. Since policy refinement generally involves a strengthening of the requirements imposed by the policy specification, adherence is not preserved in the general case. The important thing, however, is to preserve the results of the analysis that was conducted at the abstract level such that the same analysis need not be conducted again after each refinement step. New requirements imposed by the policy specification after a step of policy refinement yield a proof obligation that needs to be solved at the refined level. We show that under a set of identified rules for the combined refinement of policy specifications and system specifications, adherence results that are established at the abstract level are preserved under refinement.

In summary, the three artifacts of the policy specification language, the policy adherence relations and the policy refinement relations provide a framework for the formalization, development and analysis of policies. The framework is generic in the sense that it is applicable to various domains, such as the management of security, trust, access control, and services. In order to demonstrate the applicability of Deontic STAIRS we have in this thesis addressed trust management as a particular case. As part of this, we have proposed a method for the development of trust management policies, which constitutes the fourth artifact of this thesis. The method is supported by appropriate modeling languages, including Deontic STAIRS. As such, the method demonstrates the suitability of using our policy specification language for the formalization of trust management policies.

5.2 The Policy Specification Language

The Deontic STAIRS language is a defined as an extension of the UML 2.1 sequence diagram notation [83] allowing the specification of conditional rules in the form of permissions, obligations and prohibitions. With this extension, we are providing support for the specification and development of policies in the setting of UML.

Sequence diagrams capture dynamic/behavioral aspects of information systems, and since policies express constraints on behavior, sequence diagrams are a suitable candidate for policy specification. This type of diagram is also, together with use case diagrams, the most widespread UML notation for specifying dynamic aspects of systems [30, 111]. Since the policy specification language is defined as a modest and conservative extension of the UML, people that are familiar with the UML should be able to use and understand the notation.

Deontic STAIRS is based on STAIRS [41, 95], which is a formal approach to system development with UML sequence diagrams, and is underpinned by a formal semantics. A formal semantics explains the meaning of the policy specifications in precise, mathematical terms, and also supports formal analysis. A formal semantics furthermore allows the development of tool support for the specification, development and analysis of policy specifications.

Deontic STAIRS is not tailored for a specific type of policy, thus allowing the specification of policies for various domains. A special purpose policy language may have tailored constructs for its domain, but a general purpose language is advantageous as it applicable across domains and at various abstraction levels.

5.2.1 Examples of Deontic STAIRS Specifications

We begin our presentation of the Deontic STAIRS language by giving an example of policy rule specifications. Fig. 5.3 shows the specification of the three rules for access control that we addressed in the setting of standard UML in Fig. 2.1 of Section 2.2.

A policy rule is specified as a sequence diagram that consists of two parts, namely the triggering behavior and the rule body. The keyword rule in the upper left corner indicates the type of diagram and is followed by a chosen name for the rule. The triggering behavior specifies the conditions under which the rule applies and is captured with the keyword trigger. The rule body specifies the behavior that is constrained by the rule and is captured by one of the keywords permission, obligation or prohibition indicating the modality of the rule.

The rule *permit* in Fig. 5.3 specifies that a user U is authorized to access services on the application A provided that the user has presented a valid credential to the application. As indicated by the keyword, this is a permission rule. The interpretation is that whenever a system for which the rule applies fulfills the triggering behavior, the permitted behavior must be offered as a potential choice of behavior while allowing other behavior to be conducted instead.

The rule *prohibit* addresses the case in which the user fails to authenticate to the application. The authentication failure is the behavior that triggers the rule, and the constraint is, as indicated by the keyword, that the user is prohibited to access services on the application. The interpretation is that whenever a system for which the rule applies fulfills the triggering behavior, the prohibited behavior must not be conducted.

Finally, the rule *verify* is an obligation rule stating that whenever the application receives a credential from the user, the application is obliged to check the validity of the credential. The interpretation is that whenever a system for which the rule applies fulfills the triggering behavior, the obliged behavior must be conducted.

Deontic STAIRS is a quite modest extension of the standard UML sequence diagram notation; the only new constructs are that of the policy trigger and the deontic modalities. The extension is, however, significant for providing the UML sequence diagram notation with the expressiveness required for policy specification. Firstly, the triggering construct allows the specification of conditional scenarios, i.e. scenarios that should, may or should not occur only if a certain condition given as a scenario is fulfilled. Secondly, whereas standard UML sequence diagrams cannot distinguish between behavior that must occur and behavior that must be offered as a potential choice, these two forms of valid behavior is with Deontic STAIRS precisely captured by the deontic modalities of obligation and permission, respectively.

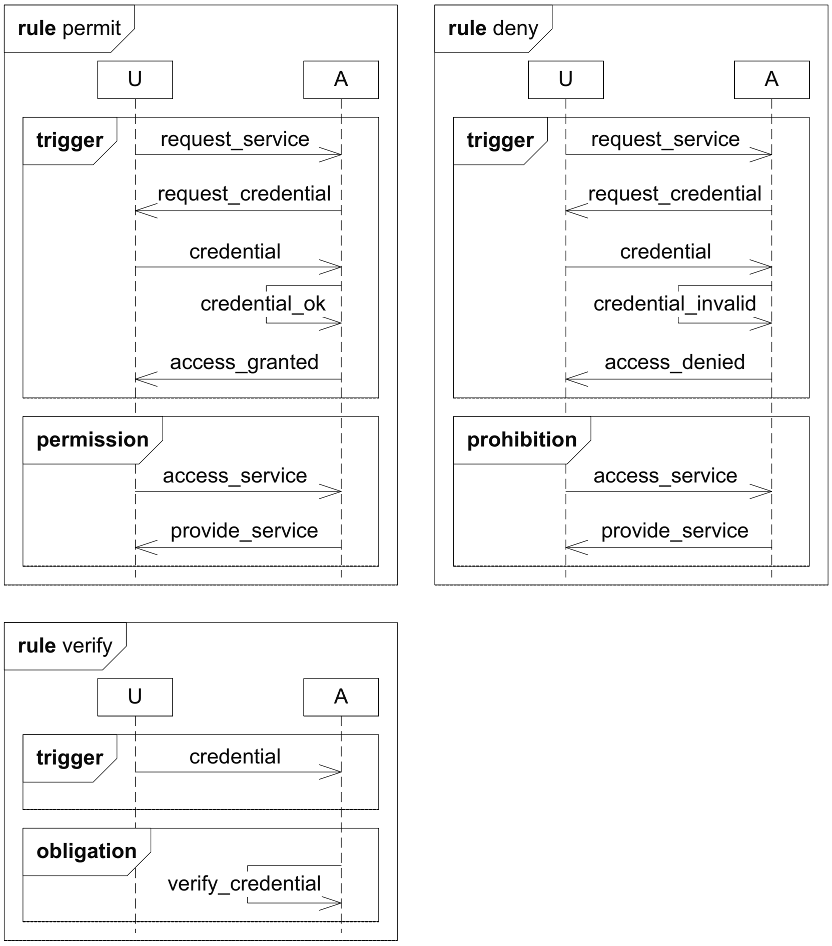

Figure 5.3: Capturing policy rules for access control using Deontic STAIRS

The extension of the UML sequence diagram notation is conservative in the sense that to the extent that UML constructs are part of the Deontic STAIRS language, the constructs are used in accordance with the UML standard [83]. This is important to ensure that people that are familiar with the UML should not be prone to use Deontic STAIRS erroneously.

As shown in Fig. 5.3, each diagram in the Deontic STAIRS language specifies a single policy rule. By definition, a policy is a set of rules, so a policy specification is in Deontic STAIRS therefore given as a set of diagrams. The policy specification $P = \{permit, deny, verify\}$, for example, is the set that consists of the three rules depicted in Fig. 5.3.

5.2.2 Syntax and Semantics

In the following we introduce the textual syntax of the Deontic STAIRS language that precisely defines the legal expressions in the language. We furthermore explain the semantics. Since policy rules are specified using sequence diagrams we first give the definition of the textual syntax of sequence diagrams as formalized in the STAIRS approach.

The set of legal expressions is defined using a textual representation of specifications, where d denotes a sequence diagram. By $\mathcal{E}$ we denote the set of all events, where an event occurs on a lifeline and is either the transmission of a message or the reception of a message. Formally, the set of syntactically correct sequence diagrams $\mathcal{D}$ is defined as the least set such that

$$\begin{aligned}
& & & \mathsf{skip} \in \mathcal{D} \wedge \mathcal{E} \subset \mathcal{D} \\
d \in \mathcal{D} & & \Rightarrow & \quad \mathsf{opt}\ d \in \mathcal{D} \\
d_1, d_2 \in \mathcal{D} & & \Rightarrow & \quad d_1\ \mathsf{seq}\ d_2 \in \mathcal{D} \wedge d_1\ \mathsf{par}\ d_2 \in \mathcal{D} \wedge d_1\ \mathsf{alt}\ d_2 \in \mathcal{D} \\
d \in \mathcal{D} \wedge S \subseteq \mathbb{N} \cup \{0, \infty\} & & \Rightarrow & \quad \mathsf{loop}\ S\ d \in \mathcal{D}
\end{aligned}$$

The set of syntactically correct Deontic STAIRS diagrams $\mathcal{R}$ is then defined as the least set such that

$$\begin{aligned}
d_1, d_2 \in \mathcal{D} \quad \Rightarrow \quad & \mathsf{trigger}\ d_1\ \mathsf{permission}\ d_2 \in \mathcal{R} \wedge \\
& \mathsf{trigger}\ d_1\ \mathsf{obligation}\ d_2 \in \mathcal{R} \wedge \\
& \mathsf{trigger}\ d_1\ \mathsf{prohibition}\ d_2 \in \mathcal{R}
\end{aligned}$$

The inductive definition of the sequence diagram syntax shows that sequence diagrams are built by combining sub-diagrams using various composition operators. The base case shows that a single event $e \in \mathcal{E}$ is a valid sequence diagrams, as is also the diagram skip, which is a STAIRS extension of the UML standard for expressing that nothing happens. The opt operator specifies that the operand is optional, the seq operator specifies the sequential composition of its operands, and the par operator specifies the parallel composition of its operands. The alt operator is for specifying alternatives, and the loop construct is for specifying several iterations of the same diagram. Observe that the number of iterations may be underspecified since S is given as a set rather than a single value.

The basic composition operators are seq, par and alt, since opt and loop are syntactic sugar. The specification opt d is defined by d alt skip, whereas loop is defined by alt and seq. The specification loop $\{1, 2\}$ d, for example, is shorthand for d alt (d seq d).

Semantically, a sequence diagram is represented by a set of traces, where a trace is a sequence of event occurrences ordered by time. As mentioned above, an event occurs on a lifeline, and is either the transmission of an message (represented by an arrow tail) or the reception of a message (represented by an arrow head). Events are ordered from top to bottom along each lifeline, and the transmission of a message is ordered before the corresponding reception of the message.

The event of sending a message m is represented by the pair $(!, m)$, whereas the event of receiving a message m is represented by the pair $(?, m)$. The message m is a triple (s, tr, re) that encodes the message content in the form of a signal s, as well as the transmitter tr and the receiver re. Both tr and re are lifelines.

As a basic example, consider the sequence diagram d of Fig. 5.4 showing the transmission of the two messages m and n from lifeline L_1 to lifeline L_2. The first event to occur is the transmission of the message m on lifeline L_1 which we abbreviate $!m$. Next, either the reception of the message m occurs on lifeline L_2, abbreviated $?m$, or the transmission of the message n occurs on lifeline L_1, abbreviated $!n$. There are therefore two traces representing the interaction specified in Fig. 5.4, namely $\langle !m, ?m, !n, ?n\rangle$ and $\langle !m, !n, ?m, ?n\rangle$.

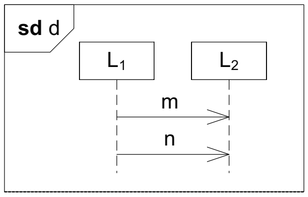

Figure 5.4: Sequence diagram

Formally, the semantics of a sequence diagram d is defined by the function $[\![\ ]\!]$ that yields the set of traces $[\![d]\!]$ representing the behavior specified by the diagram. For an event $e \in \mathcal{E}$ and for the diagram skip the semantics is defined as follows, where $\langle\rangle$ denotes the empty trace.

$$\begin{aligned} [\![e]\!] &\overset{\mathsf{def}}{=} \{\langle e\rangle\} \\ [\![\mathsf{skip}]\!] &\overset{\mathsf{def}}{=} \{\langle\rangle\} \end{aligned}$$

These definitions serve as the base cases in the inductive definition of the semantics of composed diagrams. Since opt and loop are syntactic sugar it remains to define the semantics with respect to the operators par, seq and alt. The reader is referred to Chapter 12 for these definitions.

In the semantics of the Deontic STAIRS language we need to capture the triggering behavior, the constrained behavior and the deontic modality. The specification of a policy rule is therefore semantically represented by a triple as defined in the following.

$$\begin{aligned}
[\![\textsf{trigger } d_1 \textsf{ permission } d_2]\!] &\stackrel{\textsf{def}}{=} (pe, [\![d_1]\!], [\![d_2]\!]) \\
[\![\textsf{trigger } d_1 \textsf{ obligation } d_2]\!] &\stackrel{\textsf{def}}{=} (ob, [\![d_1]\!], [\![d_2]\!]) \\
[\![\textsf{trigger } d_1 \textsf{ prohibition } d_2]\!] &\stackrel{\textsf{def}}{=} (pr, [\![d_1]\!], [\![d_2]\!])
\end{aligned}$$

The semantics $[\![P]\!]$ of a policy specification P, i.e. a set of policy rule specifications, is defined as the set of the semantic representation of each of the rules in P, i.e. $[\![P]\!] = \{[\![r]\!] \mid r \in P\}$.

The semantic representation of a policy specification as a set of semantic representations of policy rule specifications facilitates the reasoning about policies by consulting individual rules separately. Similarly, when analyzing the relation between policy specifications, or between a policy specification and a system for which the policy applies, this can be done by addressing individual rules. However, the reasoning about all rules of a policy specification in combination may not be straightforward. This is unfortunate for cases in which there is a need to represent and understand the requirements imposed by all rules simultaneously, for example to check whether one policy specification is implied by another. We have therefore defined a combined semantics for policy specifications which is presented in Chapter 14.

The presentation of the policy adherence relations and the policy refinement relations below refers to the semantics presented above. The reader is referred to Chapter 14 for details about the combined semantics, the corresponding definitions of policy adherence and policy refinement, as well as results of the relations between the two approaches.

5.3 The Policy Adherence Relations

Adherence to a policy means to satisfy or fulfill the policy. In this thesis we have captured a notion of policy adherence both with respect to systems and with respect to system specifications. In the former case, the question is what it means that a running system for which the policy applies satisfies the policy specification. In the latter case, the question is whether the specification of a system under development satisfies a given policy specification. In the following we present each of these adherence relations in turn.

5.3.1 Relating Policy Specifications to Systems

In order to formalize and reason about the relation between a system and a policy specification we need to establish a representation of the system. Such a representation can be the set of traces representing the possible system executions. A given policy rule applies to a system trace if the triggering scenario is fulfilled by the trace. In that case, the rule imposes a constraint on the continuation of the system trace after the point in which the rule is triggered. Adherence to the policy rule then depends on the continuation of the system trace, the behavior that is constrained by the rule and the modality of the rule.

Intuitively, adherence to a permission rule means that after the point in which the rule is triggered, there must exist a possible continuation that fulfills the permitted behavior. Adherence to an obligation rule means that all possible continuations fulfill the obliged behavior, whereas adherence to a prohibition rule means that none of the possible continuations fulfill the prohibited behavior.

This can be illustrated by structuring the system traces into a forward branching tree depicting how the system executions evolve. Assume, for example, a system with three possible executions represented by the trace set $S = \{h_1, h_2, h_3\}$ and that h_1 and h_2 have a common prefix h_a. The structured representation of S is depicted in Fig. 5.5.

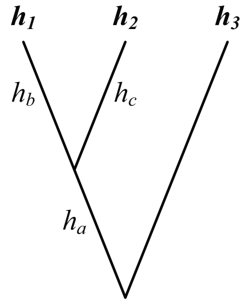

Figure 5.5: Structured traces

Recall that semantically, a policy rule is represented by a triple (dm, T, B), where $dm \in \{pe, ob, pr\}$ denotes the deontic modality, T is the trace set representing the triggering behavior and B is the trace set representing the constrained behavior. The variation over traces in T and B represents the various way of conducting the respective behaviors. Assuming that the system trace $h_3 \in S$ does not fulfill any of the triggering traces T, the rule (dm, T, B) does not apply and is trivially adhered to by the system execution.

In case the prefix h_a fulfills a trace in T, the rule (dm, T, B) imposes a constraint on the continuations h_b and h_c. For a permission, at least one of h_b and h_c must fulfill a trace in the set B. For an obligation, both h_b and h_c must fulfill a trace in B, and for a prohibition, none of these continuations may fulfill any of the traces in B.

Adherence of a system S to a policy rule r is denoted $r \rightarrow_a S$. For a formal definition, the reader is referred to Chapter 12. For a policy specification P, adherence is denoted $P \rightarrow_a S$ and defined $\forall r \in P : r \rightarrow_a S$.

An important feature of the adherence relation is that it takes into account that a policy specification refers only to the system entities and system behavior of relevance to the domain of management. The rules in Fig. 5.3, for example, refer only to issues in relation to authentication and authorization. In general, a system for which these rules apply will have entities and functionality that go beyond what is specified in the rules. The user may, for example, interact with the application for other purposes than service consumption, the application may interact with other entities, and there may also be system behavior in which the user and application are not involved.

For a system trace $h \in S$ to fulfill a triggering trace $t \in T$, t must be a sub-trace of h, which is denoted $t \lhd h$. This means that sequence of events t must occur in the same order in h, however allowing other events to be interleaved. We have, for example that $\langle a, b, c\rangle \lhd \langle e, a, b, e, f, c\rangle$, but none of $\langle a, b, c\rangle \lhd \langle e, a, c, e, f, b\rangle$ and $\langle a, b, c\rangle \lhd \langle e, b, e, f, g, b\rangle$. The sub-trace relation $\lhd$ is formally defined in Chapter 12.

As a more specific example, consider again the rules in Fig. 5.3. The triggering behavior of the rule *permit* is represented by the singleton set that consist of the trace $\langle !rs, ?rs, !rc, ?rc, !c, ?c, !co, ?co, !ag, ?ag\rangle$. The rule body is represented by the singleton set consisting of the trace $\langle !as, ?as, !ps, ?ps\rangle$. Assume, now that at some point during a system execution the following sequence of events occurs.

$$\langle \dots, !rs, !e, ?rs, !rc, ?e, ?rc, !c, ?c, !f, !vc, !g, ?vc, !co, ?co, !ag, ?ag,\\ ?f, !as, ?as, !ps, ?g, ?ps, \dots\rangle$$

It is easily verified that this system trace triggers the rule *permit* immediately after the occurrence of the event $?ag$, and that the continuation of the trace fulfills the rule body. Incidentally, the system trace furthermore triggers the obligation rule *verify* by the events $!c$ and $?c$ and fulfills the obliged behavior in the continuation by the events $!vc$ and $?vc$.

The formalization of the notion of policy adherence also yields a notion of policy consistency or policy conflict. A policy is conflicting if, for example, one rule specifies a certain behavior as permitted while the same behavior is prohibited by another rule. In that case no system adheres to the policy specification, and we say that the policy is inconsistent.

The problem of detecting and resolving policy inconsistencies is important in policy based management of distributed systems [74], since policy conflicts are likely to occur. Different policy rules may be specified by different managers and enforced in a distributed manner, and multiple policy rules may apply to the same system entities. With a precise notion of policy consistency, tools and methods may be developed for the detection and resolution of conflicts.

5.3.2 Relating Policy Specifications to System Specifications

Adherence of a system specification S to a policy specification P is defined as a generalization of the adherence relation for systems. The important difference between the representation of a system and the representation of a system specification is that a system execution is represented by a trace in the former case and by a trace set in the latter case. The representation of a system execution as a trace set in the case of system specifications captures a notion of underspecification in the sense that each trace is considered as a valid fulfillment of a particular behavior.

In this thesis we use the STAIRS [41, 95] formalization of UML sequence diagrams for representing system specifications. An important feature of STAIRS is that it has the expressiveness to distinguish between underspecification and inherent nondeterminism. Inherent nondeterminism captures alternative behaviors each of which must be offered as a potential choice. In the STAIRS syntax the alt operator is used for underspecification, whereas the xalt operator is introduced for capturing inherent nondeterminism.

As an example, consider the specification of a beverage machine that should offer both coffee and tea, where coffee can be offered as americano or espresso. If this is specified by (*americano* alt *espresso*) xalt *tea*, the machine must always offer the choice between coffee and tea since it is represented by inherent nondeterminism. A machine that can only serve espresso if coffee is chosen fulfills the specification since this alternative is represented by underspecification. The reader is referred to Chapter 13 for further details and for the formal semantics of the xalt operator.

The STAIRS expressiveness to capture inherent nondeterminism is crucial in relation to permission rules of policy specifications, since permissions specify choices that must be offered as a potential alternative.

In the following we explain adherence of a STAIRS system specification to a Deontic STAIRS policy specification by giving an example. The sequence diagram *application* in Fig. 5.6 specifies the interaction between a user U and an application A. After the user requesting a service and sending a credential for verification, there are two alternatives. The first alternative describes the situation in which the credential is invalid and the user accessing services is negative. The second alternative describes the situation in which the credential is valid. In that case, three choices of behavior should be available. The first one is simply skip, capturing that the user may refrain from doing anything. The second specifies the user accessing services, and

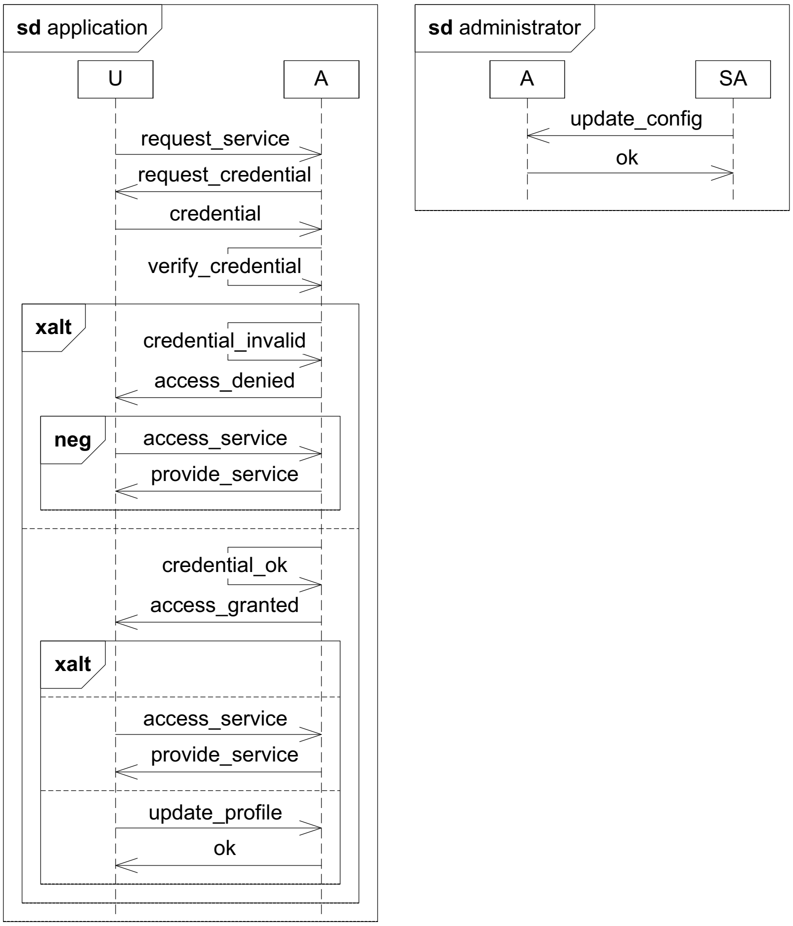

Figure 5.6: System specification

the third specifies the user updating its user profile on the application.

A full system specification will typically describe other interactions between the user and the application, and interactions between other system entities. To keep the example small we have added only one diagram, namely *administrator*, which is depicted to the right in Fig. 5.6. This shows the security administrator SA updating the configurations of the application A. We let the complete system specification S be given by *application* par *administrator*, i.e. the parallel composition of the two diagrams. Semantically, this is represented by the interleaving of the traces from each of the operands.

Consider again the policy specification P given by the three rules in Fig 5.3. It is easily verified that the first alternative of the diagram *application* triggers the prohibition rule *deny*. Adherence of the specification

S to the rule *deny* then requires that the body of the prohibition rule is not fulfilled by the alternative in question in the system specification. The rule body describes the user accessing services, which is specified as negative using the neg operator on the relevant part of the system specification. The system specification therefore adheres to the prohibition rule, which is denoted $deny \rightarrow_a S$.

In the same way, the second alternative of the system specification triggers the permission rule *permit*. Adherence requires that after the message *access_granted* in the system specification, the body of the permission rule must be given as a potential alternative. This is indeed the case by the second operand of the xalt operator that follows the message in question. System adherence to the permission rule is denoted $permit \rightarrow_a S$.

Finally, the system specification S adheres to the obligation rule *verify* since the triggering message *credential* is followed by the obliged message *verify_credential*. This is denoted $verify \rightarrow_a S$.

Since the system specification S adheres to each of the rules, S adheres to the policy specification P, which is denoted $P \rightarrow_a S$. For more details and the formal definition of adherence of system specifications to policy specifications, the reader is referred to Chapter 13. In Chapter 14 we show that the adherence relation that is formalized on the basis of the combined semantics of policy rules is equivalent to the adherence relation described in this subsection.

5.4 The Policy Refinement Relations

Refinement of a policy specification means to strengthen the specification, taking into account more details about the system for which the policy applies, making the specification more concrete, and bringing it closer to implementation and enforcement. The formalization of policy refinement as a relation between policy specifications precisely defines what it means that one policy specification is a correct refinement of another.

A policy specification is refined by refining individual rules in the specification or by adding rules to the specification. An individual rule is refined by weakening the trigger or strengthening the rule body. Weakening the trigger means to increase the set of traces that trigger the rule, thus making the rule applicable under a wider set of circumstances. For permissions and obligations, the rule body is strengthened by reducing the set of traces representing the behavior, thus reducing underspecification. Since prohibitions specify illegal behavior, the body of these rules are strengthened by increasing the set of traces representing the rule body.

Refinement of policy rules by weakening the trigger or strengthening the rule body is related to refinement in the context of assumption/guarantee specifications [18] in which the assumption specifies properties of the environment in which a component is supposed to execute, and the guarantee specifies requirements to the component that must be fulfilled whenever the component is executed in an environment that satisfies the assumption. An assumption/guarantee specification can be refined by weakening the assumption or strengthening the guarantee [18].

In the following we explain policy refinement by giving examples. The reader is referred to Chapter 12 and Chapter 14 for the formal definitions and for results on transitivity and modularity of refinement.

Let $P_1 = \{\mathit{permit}, \mathit{deny}, \mathit{verify}\}$ be the initial, most abstract specification, where each of the three rules are as specified in Fig. 5.3. The permission rule *permit* specifies that the user U is permitted to access services from the application A in case the user has presented a valid credential to the application. Through refinement we may weaken the trigger and accept also the presentation of a user id (e.g. a username and password) as sufficient for getting access to services. This is specified in the rule *permit2* in Fig. 5.7. Semantically, the trace set representing the trigger of *permit* is a subset of the trace set representing the trigger of *permit2*. The latter is therefore a refinement of the former, which we denote $\mathit{permit} \rightsquigarrow \mathit{permit2}$.

Let $P_2 = \{\mathit{permit2}, \mathit{deny}, \mathit{verify}\}$. Since *permit2* is a refinement of *permit*, we also have that P_2 is a refinement of P_1, which we denote $P_1 \rightsquigarrow P_2$. It is easily verified that for any system S, if S adheres to P_2 it also adheres to the more abstract specification P_1.

In this case we have refined P_1 by refining a single rule only, but a policy specification can be refined by refining any number of the rules and by adding any number of rules.

Next, we further refine the specification P_2 by adding the obligation rule *disable* of Fig. 5.7. This rule specifies that in case of three consecutive user login failures, the application must disable the user, log the incident and alert the user. Notice that the three obliged activities in the rule body are composed using the par operator for parallel composition. This means that there are no constraints on the ordering of these activities.

Let, now, $P_3 = \{\mathit{permit2}, \mathit{deny}, \mathit{verify}, \mathit{disable}\}$. Since the adding of rules is a valid refinement, we have that $P_2 \rightsquigarrow P_3$. By transitivity we also have that $P_1 \rightsquigarrow P_3$. Moreover, adherence to P_3 implies adherence to both P_2 and P_1.

Finally, we illustrate refinement by strengthening the body of the obligation rule *disable*. By replacing the par construct with the seq construct for weak sequencing, stronger constraints are placed on the ordering of the three obliged activities. Semantically, the trace set of the sequential composition of sequence diagrams is a subset of the trace set of the parallel composition of the same sequence diagrams. The specification *disable2* in which par is replaced by seq is therefore a valid refinement of the specification *disable*, i.e. $\mathit{disable} \rightsquigarrow \mathit{disable2}$.

Let $P_4 = \{\mathit{permit2}, \mathit{deny}, \mathit{verify}, \mathit{disable2}\}$. Since $\mathit{disable} \rightsquigarrow \mathit{disable2}$ we

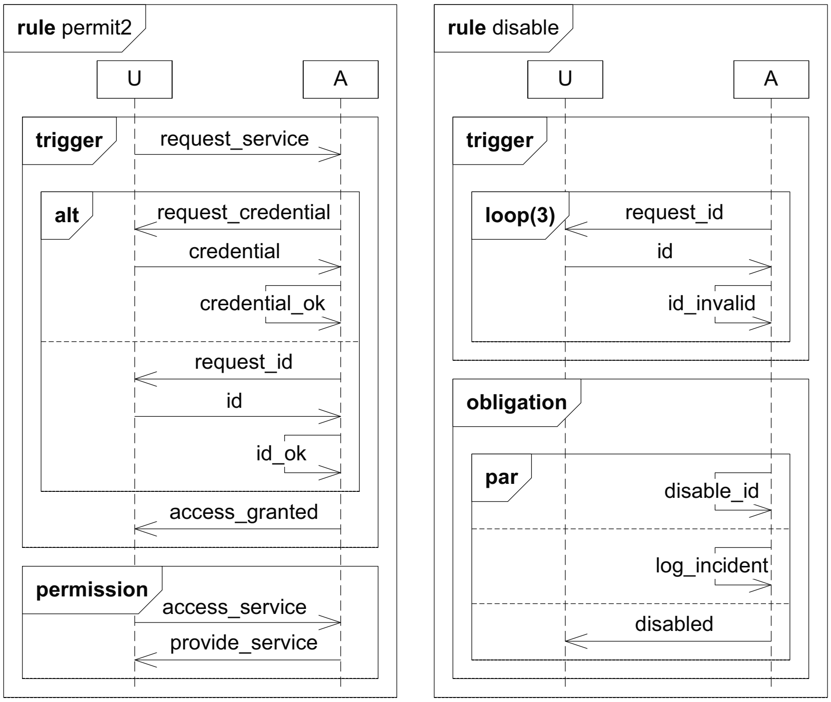

Figure 5.7: Policy refinement

have that $P_3 \rightsquigarrow P_4$. By transitivity P_4 also refines P_2 and P_1. As before, system adherence to P_4 implies adherence to all the previous, more abstract specifications.

Policy specifications can also be refined by decomposing lifelines, referred to as detailing [41]. At an abstract level several entities can be represented by one lifeline, thereby ignoring details about internal behavior and about architecture that are irrelevant for the given level of abstraction. By lifeline decomposition, such details can be taken into account at later development phases of lower levels of abstraction. The reader is referred to Chapter 12 for further details and examples.

The notion of policy refinement presented in this section is based on a notion of refinement of policy rules, and the formalization refers to the semantics of policy rules presented in Section 5.2.2. In Chapter 14 we define a combined semantics for policy specifications in which the semantics of individual rules are composed into one representation. The advantage of the latter approach is that specifications that characterize the same requirements to systems are also semantically equivalent. The notion of policy refinement presented in Chapter 14 is based on the combined semantics, which yields a more general notion of policy refinement than the one presented in this section. The motivation for the generalization and the relation between the two notions of refinement are discussed in detail in Chapter 14.

5.5 The Method for the Development of Trust Management Policies

The method for the development of trust management policies aims at understanding and constraining trust-based decisions within systems where these decisions have direct impact on the risks and opportunities towards the system. The application of the method should result in a trust policy that governs the trust-based decisions, thus ensuring the most beneficial overall level of risk and opportunity.

Such a method requires on the one hand that the notions of trust, risk and opportunity are precisely defined and that the relations between these notions are accounted for. On the other hand, such a method requires adequate techniques for the modeling of the relevant aspects of the systems under analysis.

The most basic concepts are defined in Chapter 15 where the method is presented in detail. A more elaborate conceptual analysis and clarification is given in Chapter 9 which is partially based on previous work [75]. In the following we define and explain the most central concepts of relevance.

Our notion of trust is based on the definition proposed by Gambetta [34] and defined as the subjective probability by which an actor (the trustor) expects that another entity (the trustee) performs a given action on which the welfare of the trustor depends. By this definition, trust is a probability ranging from 0 (complete distrust) to 1 (complete trust). It is subjective, which means that it is a belief held by the trustor about the trustee. The welfare of the trustor refers to one or more assets of the trustor that is affected; in case the trustee performs as expected it may have a positive effect on the welfare of the trustor, otherwise it may have a negative effect.

The positive and negative outcomes are related to the aspects of opportunity and risk, respectively. In a situation of trust there is always a possibility of deception or betrayal, which means that there is an inevitable relation between trust and risk [19, 72]. In the same way, trust is always related to opportunity, which is the dual to risk. In a trust based transaction, the trustor may be willing to accept the risk considering the involved opportunities.

A risk is defined as the probability of the occurrence of a harmful event [50], i.e. an event with a negative impact on an asset. The level of risk is given as a function from the consequence (loss) of the harmful event and the probability of its occurrence [3]. The dual notion of opportunity is then defined as the probability of the occurrence of a beneficial event, and the level of opportunity is given as a function from the consequence (gain) of the beneficial event and the probability of its occurrence.

As an example, assume a situation in which a person (the trustor) considers to lend the amount of \$80 to another (the trustee) with the promise of being repaid the amount with 50% interest, i.e. with a gain of \$40. The trust level is 0.9, i.e. the trustor believes there is a 0.9 probability that the trustee will pay the \$120, and that there is a 0.1 probability that the \$80 will be lost. Using multiplicity as the risk and opportunity function, the opportunity level is $0.9 \cdot 40 = 36$ and the risk level is $0.1 \cdot 80 = 8$. Since the opportunity outweighs the risk, the trustor should accept the transaction.

Importantly, however, trust is only a belief held by the trustor, so the estimated trust level may be wrong, and therefore also the estimated levels of risk and opportunity. Trust is important precisely in situations in which decisions must or should be made, even if there is lack of evidence about the future behavior of the trustee. In order to precisely assess and evaluate trust-based decisions, the belief of the trustor and the basis for this belief must be considered.

We say that trust is well-founded if the trust held by the trustor equals the trustworthiness of the trustee. By trustworthiness we mean the objective probability by which the trustee performs a given action on which the welfare of the trustor depends. It is only in case of well-founded trust that the trustor can correctly estimate the involved risks and opportunities.

To continue the example, assume the trustworthiness of the trustor is only 0.65. The actual opportunity level is then $0.65 \cdot 40 = 26$, and the actual risk level is $0.35 \cdot 80 = 28$, i.e. the risk is higher than the opportunity.

The method for the capturing and development of policies for trust man-

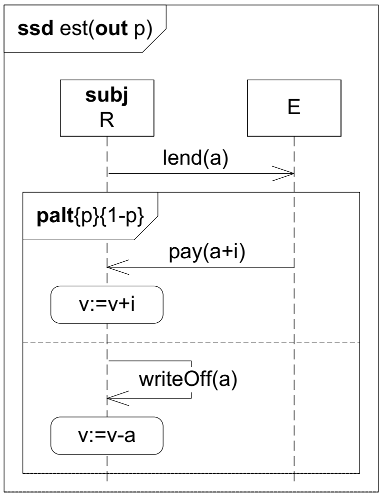

Figure 5.8: Modeling trust

agement is divided into three main stages, namely system modeling, trust analysis and trust policy specification. The system models serve as a basis for the trust analysis and describe the behavior of system actors, the decisions they make and the basis for these decisions. In particular, the modeling describes decisions based on trust as well as the considerations behind these decisions.

Subjective STAIRS [89] was selected for the purpose of system modeling. The notation allows us to specify subjective beliefs about scenarios, as well as actual (objective) scenarios, and also to show how the subjective beliefs influence the choices made by actors.

Fig. 5.8 shows the modeling of a subjective belief of an entity R. The keyword ssd in the upper left corner denotes the type of diagram (subjective sequence diagram), and *est* is the chosen diagram name. Exactly one of the lifelines in the diagram is annotated with the keyword subj, indicating the actor whose beliefs are modeled.

The diagram models the belief held by R of the outcome of lending the amount a of money to the actor E. The palt (probabilistic alternative) operator specifies the probabilities of the various outcomes as subjectively estimated by R. In this case R believes there is a probability p that E will repay the loan amount a in addition to an interest i, as specified in the first operand of the palt operator. Furthermore, R believes there is a probability $1 - p$ that E fails to repay, in which case the amount a must be written off.

The outcomes of the scenario that is modeled affect an asset of R, which is specified as R's monetary values v. In the first outcome the value v increases by the interest i, and in the second outcome the value decreases

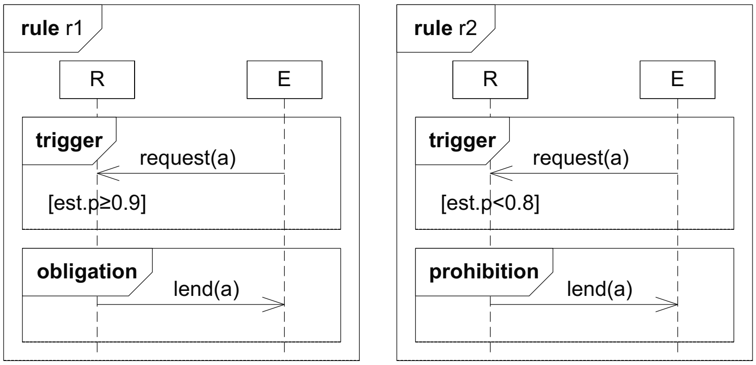

Figure 5.9: Formalizing a trust policy

by the loan amount a.

The diagram in Fig. 5.8 specifies the trust held by the trustor R in the trustee E. The value p is the trust level, and the statement out p means that the value can be referred to from other diagrams. Specifically, the subjective diagrams and their trust values are used as input to objective models describing the actual behavior of the system under analysis. Using the objective models, the trust based decisions made by entities within the system are described, as well as the impact of these decisions on the overall behavior of the system and the risks and opportunities to which the system is exposed. The reader is referred to Chapter 15 for further details about the system modeling.

During the analysis stage, the well-foundedness of the trust relations are evaluated and the involved risks and opportunities are assessed. Potential alternatives to the current choices of behavior are also evaluated in order to identify the optimal system behavior. The activities and output of the analysis stage are described in detail in Chapter 15.

The optimal system behavior is the one that yields a minimized overall level of risks and a maximized overall level of opportunities. Having identified the optimal choices of behavior during the second stage of the method, these choices are formalized as a trust policy during the third and final stage. The trust policy governs trust-based choices of behavior where the decisive factor for each rule is the level of trust.

The trust policy is specified using Deontic STAIRS as exemplified in Fig. 5.9. The trigger specifies the scenario under which the rule applies, including the relevant levels of trust. The rule $r1$ states that if the actor E (the trustee) requests a loan of amount a and the trust level is 0.9 or higher, the actor R (the trustor) is obliged to lend the money. Notice that the use of a guard in the trigger, where the expression *est.p* refers to the relevant trust levels as specified in a subjective sequence diagram. The reason for formalizing such an obligation rule is that the trust analysis stage concluded that the lending of the amount a under these levels of trust yields an opportunity level that outweighs the risk level.

The rule $r2$ states that it is prohibited for R to lend the amount a to E in case the trust level is lower than 0.8. This is because the risk in this case is higher that the opportunity.

For the trust levels for which risks and opportunities even out, it may be left to the trustor to make the decisions. In the example this can be formalized by permission rules for the trust levels between 0.8 and 0.9 stating that both lending the amount and refraining from lending the amount are permitted. The use of such permission rules in trust policies is described and exemplified in Chapter 15.

Chapter 6

Overview of Papers

The main results of the work presented in this thesis are documented in the papers in Part II. In the following we give an overview of these research papers, describing the topics of each paper and indicating how much of the results are credited the author of this thesis.

6.1 Paper 1: Why Trust is not Proportional to Risk

Authors: Bjørnar Solhaug, Dag Elgesem and Ketil Stølen.

Publication status: Published as [105].

Contribution: Bjørnar Solhaug was the main author, responsible for about 90% of the work.

Main topics: The paper gives a conceptual analysis of the notion of trust and clarifies its relation to the notions of risk and opportunity. The paper discusses notions of trust as presented within the disciplines of economics and sociology and relates these notions to the domain of trust management.

6.2 Paper 2: Specifying Policies Using UML Sequence Diagrams – An Evaluation Based on a Case Study

Authors: Bjørnar Solhaug, Dag Elgesem and Ketil Stølen.

Publication status: Technical report A1230, SINTEF ICT, 2007. The report presented in this thesis is the full version of the paper published as [104].

Contribution: Bjørnar Solhaug was the main author, responsible for about 90% of the work.

Main topics: The paper presents an evaluation of standard UML sequence diagrams as a notation for policy specification. Policy rules in the form of conditional permissions, obligations and prohibition were captured on the basis of a case study. The evaluation of sequence diagrams as a notion for formalizing these rules focuses on requirements to expressiveness and human interpretation.

Note: A few misprints in the published technical report have been corrected in the version presented in this thesis.

6.3 Paper 3: Specification of Policies Using UML Sequence Diagrams

Authors: Bjørnar Solhaug and Tor Hjalmar Johannessen.

Publication status: Published as [106].

Contribution: Bjørnar Solhaug was the main author, responsible for about 95% of the work.

Main topics: The paper gives a presentation of policy based management of access control in the context of remote electronic authentication. A conceptualization of policy rules is given, and the paper demonstrates the informal capturing of relevant policy rules that are subsequently formalized using Deontic STAIRS.

6.4 Paper 4: Compositional Refinement of Policies in UML – Exemplified for Access Control

Authors: Bjørnar Solhaug and Ketil Stølen.

Publication status: Technical report A11359, SINTEF ICT, 2009. The report presented in this thesis is the full version of the paper published as [107].

Contribution: Bjørnar Solhaug was the main author, responsible for about 90% of the work.

Main topics: The paper presents the syntax and semantics of Deontic STAIRS, the policy specification language developed as a part of this thesis. The notion of policy adherence is formalized, precisely defining what it means that a system satisfies a policy specification. A notion of policy refinement is formally defined, and the properties of transitivity and modularity of refinement are proved.

6.5 Paper 5: Adherence Preserving Refinement of Trace-set Properties in STAIRS: Exemplified for Information Flow Properties and Policies

Authors: Fredrik Seehusen, Bjørnar Solhaug and Ketil Stølen.

Publication status: Published as [99].

Contribution: The paper was written in close collaboration between the authors. Bjørnar Solhaug was responsible for about 45% of the work.

Main topics: The paper gives a presentation of the STAIRS approach to system development with UML sequence diagrams and shows that trace-set properties are preserved under refinement in STAIRS. Trace properties are properties that can be falsified on single traces, and include safety and liveness properties. Trace-set properties are properties that can only be falsified on sets of traces, and include information flow properties and permission rules of policies. It is known that trace-set properties are not preserved under the standard notion of refinement. The paper demonstrates the potential of STAIRS in this respect by relating STAIRS specifications to information flow properties and policies.

6.6 Paper 6: Preservation of Policy Adherence under Refinement

Authors: Bjørnar Solhaug and Ketil Stølen.

Publication status: Technical report A11358, SINTEF ICT, 2009.

Contribution: Bjørnar Solhaug was the main author, responsible for about 90% of the work.

Main topics: The paper addresses the challenge of integrating the development of policies with the development of systems to which the policy applies. Such an integrated process is beneficial since it allows the requirements imposed by a policy to be taken into account throughout the development process. Fulfillment of policy requirements may be established at any stage of the integrated development process, and at any level of abstraction, by the verification of adherence. The paper characterizes the conditions under which adherence is preserved under the combined development of policies and systems, and presents development rules that guarantee adherence preservation under refinement.

6.7 Paper 7: A UML-based Method for the Development of Policies to Support Trust Management

Authors: Atle Refsdal, Bjørnar Solhaug and Ketil Stølen.

Publication status: Published as [88].

Contribution: Bjørnar Solhaug was one of two main authors, responsible for about 45% of the work.

Main topics: The paper presents a method for capturing and formalizing trust policies the enforcement of which ensures that trust-based decisions within systems minimizes the overall level of risks and maximizes the overall level of opportunities. The method is supported by modeling languages for describing trust relations and their impact on the behavior of systems. The models facilitate the evaluation of well-foundedness of trust and of risks and opportunities. Based on the evaluation, a trust policy is captured and formalized with an adequate policy specification language.

Note: The published paper models the triggering scenario of policy rules as the occurrence of an event. In the version presented in this thesis, the specification of an event is replaced with the specification of an interaction. This revision was done in order to align the notation with final definition of the syntax of the policy specification language as presented in this thesis, with no loss of or changes in the results. A few misprints have also been corrected.

Chapter 7

Discussion

In this chapter we discuss and evaluate the contributions of this thesis. In Section 7.1 we evaluate each of the invented artifacts against the success criteria formulated in Chapter 2, and in Section 7.2 we discuss related work.

7.1 Fulfillment of the Success Criteria

7.1.1 The Policy Specification Language

1. *The language should have the expressiveness to capture the deontic modalities of permission, obligation and prohibition.*

   The fulfillment of this requirement is ensured by equipping the policy specification language with an explicit construct for each of the deontic modalities. This is shown in the definition of the syntax in Section 5.2.2 and in the presentation of Deontic STAIRS in Chapter 12.

2. *The language should have the expressiveness to capture conditional scenarios.*

   This requirement is fulfilled since the trigger construct of the policy specification language allows the specification of conditional scenarios. The syntax is defined in Section 5.2.2 and presented in more detail in Chapter 12.

3. *The language should be intuitively understandable to end-users such as decision makers, clients, developers and other stakeholders, including personnel with little technical background.*

   The support for abstraction allows policies to be represented at various levels of detail, taking into account the viewpoint of particular stakeholders. As such, abstraction contributes to understandability, also for personnel with little technical background. The popularity of UML sequence diagrams for various uses [30, 111] may also indicate that this type of notation is easy to use for most stakeholders. A further conjecture is that a graphical notation is more intuitively understandable than purely textual notations.

A proper evaluation of the requirement to understandability requires thorough empirical investigations which have not been conducted during the work on this thesis. The fulfillment of this success criterion is, however, substantiated by demonstrating the formalization of high-level, informal policy specifications in Chapter 11. The chapter describes a policy development case for access control based on electronic authentication. The case addressed in Chapter 12 demonstrates the formalization of access control policies and their development by policy refinement.

The evaluation of standard UML sequence diagrams for policy specification presented in Chapter 10 showed that sequence diagrams as a policy notation may not be easily understandable to end-users. The identified weaknesses have been mitigated by language constructs of Deontic STAIRS.

4. *The language should allow policy specifications to ignore behavior that is not relevant for the policy.*

   This requirement is satisfied by defining the fulfillment of policy scenarios in terms of the sub-trace relation. This allows the scenarios relevant for the policy to be specified as abstractions of potential system scenarios. This is explained in Section 5.3. The details and formalizations are presented in Chapter 12 and Chapter 14.

5. *The language should be a conservative extension of standard UML.*

   The language constructs of the policy specification language that go beyond the UML sequence diagram notation are the triggering construct and the deontic modalities as shown in Section 5.2.2 in the definition of the syntax. To the extent that standard UML constructs are part of the syntax, they are used in accordance with the UML. Deontic STAIRS is therefore a modest and conservative extension of standard UML.

   Details about the language and its relation to standard UML are presented in Chapter 11 through Chapter 13.

6. *The language should be underpinned by a semantics that is unambiguous, allows formal analysis of policy specifications, and allows the development of tool support.*

   The formal semantics of the policy specification language presented in Chapter 12 through Chapter 14 ensures its unambiguity. As demonstrated by the formal proofs in these chapters, the denotational semantics allows formal verification of properties such as transitivity and modularity of refinement.

The development of tool support for policy specification and analysis has been outside the scope of this thesis, but a denotational semantics may nevertheless facilitate this objective. An operational semantics may, however, be more suitable for tool developers [86]. Such a semantics and tool support has been developed for STAIRS [73], and we believe that this work, or at least parts of it, can be adapted to Deontic STAIRS.

7. *The semantics should be compositional, meaning that the semantics of a composed diagram can be determined from the semantics of the sub-diagrams and the composition operators.*

   This requirement is fulfilled since, for each composition operator, the semantics of a composed specification is derived as a function from the semantics of the sub-specification and the composition operator. This is explained in Section 5.2.2 and presented in more detail in Chapter 12 through Chapter 14.

7.1.2 The Policy Adherence Relations

8. *The notion of policy adherence should be intuitively understandable to end-users such as decision makers, clients, developers and other stakeholders, including personnel with little technical background.*

   A proper evaluation of this criterion requires thorough empirical investigations which have not been conducted during the work on this thesis. Experience from presentations of the work to several potential end-users from various backgrounds, however, indicates that the specifications of conditional scenarios as permitted, obliged and prohibited and what it means to satisfy these are intuitively understood.

   In a practical setting of policy development, analysis and enforcement, the end-users should be supported by tools and methods facilitating the testing and verification of adherence. The use of Deontic STAIRS should therefore not depend on the end-users understanding the formalization of the notion of policy adherence.

9. *The adherence relations should capture the properties of the deontic modalities as axiomatized in Standard Deontic Logic.*

   The fulfillment of this requirement follows from the definition of adherence and is shown by formal proofs in Chapter 12 of the relation between the deontic modalities of the policy specification language.

10. *The notion of adherence of a system implementation to a policy specification should be independent of the system platform.*

    The fulfillment of this requirement is ensured by defining adherence of a system to a policy specification by referring only to the traces the system may potentially produce, not assuming any other knowledge about the system. This is explained in Section 5.3, whereas the formalization of the adherence relations is presented in Chapter 12 through Chapter 14.

7.1.3 The Policy Refinement Relations

11. *The notions of policy refinement should be intuitively understandable to end-users such as decision makers, clients, developers and other stakeholders, including personnel with little technical background.*

    As for the notion of policy adherence, a thorough empirical investigation of the fulfillment of this criterion has not been conducted and is left for future work. We believe, however, that the idea of policy refinement as a process of strengthening the policy specification and making it more low-level is intuitively understandable. The development of policies under the refinement paradigm is demonstrated for an access control case in Chapter 12.

    In a practical setting of stepwise and modular policy development, the end-users should be supported by pragmatic methods that guide the development process and ensure correctness of refinement. Tool support should also be provided for testing and verification of policy refinement. Policy development using Deontic STAIRS should therefore not depend on the end-users understanding the formalization of the notion of policy refinement.

12. *The policy refinement relations should ensure that all requirements from the abstract policy specification are preserved in the refined policy specification.*

    The fulfillment of this requirement is formally proved in Chapter 12 and Chapter 14 by showing that adherence to a refined policy specification implies adherence to all more abstract policy specifications.

13. *The policy refinement relations should support the stepwise development of policy specifications.*

    Support for stepwise development of policy specifications is ensured by transitivity of the policy refinement relation. The property of transitivity is formally proved in Chapter 12 and Chapter 14.

14. *The policy refinement relations should support modular development of policy specifications.*

The fulfillment of this requirement is shown in Chapter 12 by formal proofs of monotonicity of the composition operators with respect to policy refinement. Monotonicity of a composition operator with respect to refinement implies that when the operands are refined separately, the composition of the result is a valid refinement of the composition of the abstract operands. Modularity properties for the notion of policy refinement presented in Chapter 14 are also established.

7.1.4 The Method for the Development of Trust Management Policies

15. *The method should be applicable and understandable to end-users.*

    For end-users to be able to easily understand the method, the description of the method should be refined, and guidelines for how to operationalize its various tasks should be provided.

    The application of the method as presented in this thesis is, however, demonstrated for a banking case in Chapter 15. The method has furthermore been applied in a real setting for analyzing and evaluating trust in a validation authority service for digital certificates. The analysts, i.e. the end-users of our method, had some background in risk analysis. The experiences from the trial are at the time of writing under evaluation.

16. *The method should offer description techniques that are understandable to all relevant stakeholders, including end-users, decision makers, engineers and analysts.*

    The languages supporting the method are Subjective STAIRS for system modeling and Deontic STAIRS for trust policy specification. Deontic STAIRS is discussed with respect to this requirement under success criterion 3 above.

    The method and its specification languages were presented to potential end-users and stakeholders as part of the evaluation. An evident finding was that even relatively small and simplified models using Subjective STAIRS were hard to understand without thorough explanations and guidance. This indicates that other and more intuitive description techniques should be developed for a better support for the system modeling stage, possibly in combination with Subjective STAIRS.

17. *The method should support the modeling of the trust-based decisions within a system, including the entity that makes the decision (the trustor), the level of trust held by the trustor, and the basis upon which the trustor determines this level of trust.*

Subjective STAIRS has the expressiveness to model the potential choices of system behavior and their probabilities as shown in [89] and in Chapter 15 of this thesis. Furthermore, when system behavior results from trust-based decisions, this can be explicitly specified along with the trustor, the trust level and the basis upon which the trustor makes its estimations. This requirement is therefore fulfilled.

18. *The method should support the evaluation of the well-foundedness of trust.*

    The trust modeling conducted as part of the method specifies the subjective probability estimates made by actors within the system, i.e. the trust levels as explained in Chapter 15. The modeling of the system behavior specifies the actual (objective) probability for scenarios to occur, and well-foundedness of trust can be evaluated by comparing the subjective and objective probabilities. This requirement is therefore fulfilled by the method.

19. *The method should support the evaluation of the risks and opportunities associated with the trust-based decisions.*

    This criterion is fulfilled since the method involves the identification of both the beneficial events and the harmful events associated with trust relations, as well as the probabilities for these events to occur and their impact on assets in terms of gain and loss, respectively. The levels of risk and opportunity are derived functionally from these data.

20. *The method should support the capturing and formalization of trust policies.*

    A trust policy is captured by comparing alternative choices of trust-based behavior and identifying the choices that are most beneficial with respect to risks and opportunities. The formalization of the trust policy is supported by Deontic STAIRS as explained and demonstrated in Section 5.5 and in Chapter 15. This requirement is therefore fulfilled.

21. *The method should be based on well-defined notions of trust, risk and opportunity, and the relations between the notions should be precisely defined.*

    This requirement is fulfilled by the establishment in Chapter 9 of a conceptual foundation in which all the relevant notions and the relations between them are clarified and precisely defined.

7.2 Related Work

This section discusses work that is related to the work presented in this thesis. In Section 7.2.1 we address various approaches to system specification and development using notation comparable to Deontic STAIRS. In Section 7.2.2 we address existing approaches to refinement of policy specifications. The reader is also referred to the related work section of the papers presented in Part II of the thesis.

7.2.1 Specifications Using Interactions

UML sequence diagrams extends the ITU recommendation message sequence charts (MSCs) [47], and both MSCs and a family of approaches that have emerged from them, e.g. [39, 40, 62, 66, 101], could be considered as alternatives to notations for policy specification using interactions.

An MSC describes a scenario by showing how components or instances interact in a system by the exchange of messages. Messages in MSCs are, as for UML sequence diagrams, ordered by weak sequencing, which yields a partial ordering of events. Several operators are defined for the composition of MSCs, such as weak sequencing, parallel composition, and alternative executions. ITU has also provided a formal operational semantics of MSCs [46] based on work by Mauw and Reniers [77, 78].

The specification of policy rules as defined in this thesis requires the expressiveness to distinguish between legal and illegal behavior, and to distinguish between behavior that must be conducted and behavior that should be offered as a potential alternative. Illegal behavior can with MSCs be captured by the specification of a guard that evaluates to false, but there are no explicit operators for specifying behavior as illegal, such as the UML `neg` operator. More importantly, the distinction between obliged and permitted behavior go beyond the standard MSC language. There is furthermore no support for the specification of conditional scenarios which means that triggering scenarios cannot be specified. MSCs are also not supported by a well defined notion of refinement.

In [66], a variant of MSCs is provided a formal semantics and is supported by a formal notion of refinement. MSCs are interpreted as existential, universal or negated (illegal) scenarios, which is related to the specification of permissions, obligations and prohibitions, respectively, in Deontic STAIRS. There are, however, no explicit constructs in the syntax for distinguishing between these interpretations. Conditional scenarios with a triggering construct are supported, facilitating the specification of liveness properties, but the composition of the triggering scenario and the triggered scenario is that of strong sequencing. This can be unfortunate in the specification of distributed systems in which entities behave locally and interact with other entities asynchronously.

Four different refinement relations for MSCs are defined in [66]. The first is binding of references which allows a reference to an empty MSC (representing complete underspecification) at the abstract level to be replaced with a reference to a filled in, and thus more specific, MSC at the refined level. The second is property refinement which removes traces, i.e. refinement by reduction of underspecification. The third is message refinement which means to substitute an interaction for a single message. The fourth is structural refinement which means to replace a single instance (lifeline) with a set of instances. The second and fourth notions of refinement are similar to the notions of refinement that are proposed in STAIRS and Deontic STAIRS. In Chapter 13 we show that policy adherence is preserved under refinement of STAIRS specifications. The corresponding property does not hold in the work presented in [66], where it is shown that fulfillment of an MSC under the existential interpretation is not preserved under property refinement of system specifications.

The expressiveness of live sequence charts (LSCs) [24, 40] to capture existential, universal and forbidden scenarios can be utilized for specifying permissions, obligations and prohibitions, respectively. The lack of explicit constructs for specifying forbidden scenarios is, however, unfortunate with respect to user friendliness. Conditionality is, moreover, not supported for existential diagrams, which means that diagrams corresponding to our permissions cannot be specified with triggers. LSCs are also not supported by a precise or formal notion of refinement.

The semantics of LSCs is that of partial ordering of events defined for MSCs [47]. However, the semantics of LSCs defines the beginning of precharts and the beginning of main charts as synchronization points, meaning that all lifelines enter the prechart simultaneously and that the main chart is entered only after all lifelines have completed their respective activities in the prechart. As for the work on MSCs in [66], this yields a strong sequencing between the prechart and main chart, which is not in accordance with the weak sequencing of MSCs and UML sequence diagrams.

The discussion of LSCs with respect to policy specification carries over to modal sequence diagrams (MSDs) [39]. As for LSCs the notation is based on the universal/existential distinction, and the semantics is basically the same.

The triggering scenarios of triggered message sequence charts (TMSCs) [101] can be utilized for the specification of policy rules. There is, however, no support for distinguishing between permitted, obligated and prohibited scenarios; a system specification defines a set of valid traces, and all other traces are invalid.

7.2.2 Policy Refinement

Although a variety of languages and frameworks for policy based management has been proposed the last decade or so, policy refinement is still in its initial phase and little work has been done on this issue. After being introduced in [4] the goal-based approach to policy refinement has emerged as a possible approach and has also later been further elaborated [6, 92, 93].

In the approach described in [4], system requirements that eventually are fulfilled by low-level policy enforcement are captured through goal refinement. Initially, the requirements are defined by high-level, abstract policies, and so called strategies that describe the mechanisms by which the system can achieve a set of goals are formally derived from a system description and a description of the goals. Formal representation and reasoning are supported by the formalization of all specifications in event calculus.

Policy refinement is supported by the refinement of goals, system entities and strategies, allowing low-level, enforceable policies to be derived from high-level, abstract ones. Once the eventual strategies are identified, these are specified as policies the enforcement of which ensures the fulfillment of the abstract goals. As opposed to our approach, there is no refinement of policy *specifications*. Instead, the final polices are specified with Ponder [23], which does not support the specification of abstract policies that can be subject to refinement. The goal-based approach to policy refinement hence focus on refinement of policy requirements rather than policy specifications.

The same observations hold for the goal-based approaches described in [6, 92, 93], where the difference between [4, 6] and [92, 93] mainly is on the strategies for how to derive the policies to ensure the achievement of a given goal. The former two use event calculus and abduction in order to derive the appropriate strategies, whereas the latter two use automated state exploration for obtaining the appropriate system executions. All approaches are, however, based on requirements capturing through goal refinement, and Ponder is used as the notation for the eventual policy specification.

In [6] a policy analysis and refinement tool supporting the proposed formal approach is described. In [4], the authors furthermore show that the formal specifications and results can be presented with UML diagrams to facilitate usability. The UML is, however, used to specify goals, strategies, etc., and not the policies *per se* as in our approach. In our evaluation of the UML as a notation for specifying policies [104] we found that sequence diagrams to a large extent have the required expressiveness, but that the lack of a customized syntax and semantics makes them unsuitable for this purpose. The same observation is made in attempts to formalize policy concepts from the reference model for open distributed processes [51] using the UML [1, 70]. Nevertheless, in this thesis we have demonstrated that with modest extensions, policy specification and refinement can be supported.

Chapter 8

Conclusion

This chapter concludes Part I of the thesis by summarizing the results and pointing out directions for future work.

8.1 Summary

The steady growth of the information society throughout the world imposes new and increased challenges with respect to the management of electronic networks and services, as well as the management of the related risk and security issues. Policy based management of information systems has emerged as an approach to these challenges, in which requirements to systems are fulfilled through the enforcement of adequate policy rules. An important feature of policies is that they allow systems to be dynamically changed by modifying the policy, leaving the underlying implementation of the system unchanged [102].

This thesis contributes to the domain of policy based management by proposing a policy specification language with support for policy capturing, policy abstraction and refinement, and policy analysis. The contribution is manifested in the three artifacts of a policy specification language, formal notions of policy adherence, and formal notions of policy refinement.

The policy specification language, called Deontic STAIRS, is defined as a modest and conservative extension of the UML sequence diagram notation, and its formalization is based on the STAIRS approach to system development using sequence diagrams. The extension of the UML consists of a triggering construct which allows the specification of conditional scenarios, and constructs for specifying permissions, obligations and prohibitions. As demonstrated in this thesis, this extension provides the UML with customized support for the specification of policies.

The policy specification language proposed in this thesis has been developed with the aim of supporting human interpretation and communication. Policies are derived from business goals, service level agreements, trust relationships, security requirements, etc. During the phases of policy capturing, formalization and implementation, various stakeholders with various backgrounds are involved, and measures should be taken to avoid misunderstandings and miscommunications. An important feature of Deontic STAIRS for facilitating human interpretation is the support for abstraction. Abstraction allows details about system functionality and architecture that are irrelevant from a certain viewpoint to be ignored. The proposed notions of policy refinement precisely characterize what it means that one specification is a correct representation of another specification at a different level of abstraction.

The policy refinement relations also facilitate the development of policy specifications by supporting a stepwise and modular development process. Stepwise development means that a policy specification can be carried from the initial, abstract and high-level to the final, concrete and low-level under any number of refinement steps. Modularity means that a policy specification can be refined by refining separate parts of the specification individually. These two features together allow the problem of policy development to be broken down into small and manageable problems.

A further advantage of abstraction is that it may facilitate efficiency of analysis and verification, as well as early discovery of design flaws. However, for analysis to be meaningful at an abstract level, the results must be preserved under refinement. Otherwise, the analysis must be conducted again after each step of refinement. This problem has been addressed in this thesis by establishing results of preservation of policy adherence under refinement.

The formalization of the notion of policy adherence yields a precise characterization of what it means that a system or a system specification satisfies a policy specification. This is important since it precisely and unambiguously characterizes the correct enforcement of policy specifications. The notion of policy adherence also yields a precise notion of policy consistency; a policy specification is inconsistent if the set of systems that adhere to the policy specification is empty.

Finally, the policy specification is underpinned by a formal, denotational semantics. This is important since it defines a precise and unambiguous meaning of policy specifications, and it also allows the formalization of notions such as policy adherence and policy refinement. The formal semantics furthermore allows rigorous analysis of policy specifications and verification of properties of the policy specification language, and it facilitates the development of tool support.

The policy specification language proposed in this thesis is generic in the sense that it is applicable for various domains, such as security, access control, services and trust. In order to demonstrate the applicability of the language we have particularly addressed the domain of policy based trust management. A method for the development of policies to support trust management has been developed, in which the policy specification language is applied in the formalization of trust policies.

8.2 Future Work

This thesis is concerned with the development of languages to support policy capturing, specification and analysis, as well as facilitating these activities by proposing notions of abstraction and refinement. An interesting topic for future work is to address the issue of policy implementation and enforcement.

In [100], a method is presented for the specification of high-level security policies that can be enforced by run-time monitoring mechanisms. The only type of policy rules relevant in this approach is prohibition rules, and there is also no notion of policy trigger. The approach is therefore less general than Deontic STAIRS, and the rules can be specified using standard UML sequence diagrams. The method can nevertheless be considered as a starting point for the development of enforcement mechanisms for Deontic STAIRS specifications.

Policy conflicts or inconsistencies are inherent to policy based management of large distributed systems since rules may be specified by various managers, policies are implemented and enforced in a distributed manner, several rules may apply to the same set of entities and behavior, etc. [74]. The notion of policy consistency is precisely captured in this thesis, and a direction for future work is to investigate and develop methods for conflict detection and resolution. Existing approaches to this problem, such as [31, 74], should be considered.

Semantically the sequence diagrams specifying the trigger and the body of a policy rules are each represented by a set of traces. In the UML standard, however, a sequence diagram is explained in terms of a pair of trace sets, one set of allowed traces and one set of forbidden traces. This bisection is also formalized in the STAIRS approach, and should be considered for Deontic STAIRS. This would yield a more precise characterization of the behavior specified by policy rules.

As an example, assume a permission rule is specified for defining user access to a given service provided the user is authenticated through a secure protocol. The specification of the protocol will then serve as the policy trigger. Assume further that there are three different authentication protocols under consideration, namely a_1, a_2 and a_3. At the initial, abstract level it may be decided that a_1 should be acceptable for authentication. At this level, no decision has been made regarding a_2 and a_3. With support for the bisection of traces into two sets, we can then categorize the protocol a_2 as unacceptable for triggering the permission. Still, no decision has been made regarding a_3, but through refinement a_3 can be specified as acceptable or unacceptable. The protocol a_2 should never be redefined as acceptable since it has been discarded as unacceptable.

Notice, importantly, that operating with this bisection of traces in the specification of policy rules does not imply the specification of forbidden behavior. Specifying an authentication protocol a_2 as negative in the trigger of a permission rule, for example, does not mean that this behavior should never be conducted. It only means that a_2 is not acceptable as a condition for allowing the behavior specified as permitted in this particular rule. Customized constructs should therefore be introduced in order to avoid confusion with standard UML in which negative traces are inadmissible.

The refinement of a policy rule is in this thesis defined as reduction of underspecification in terms of trace set inclusion. Enriching the expressiveness to allow the bisection between the two trace sets opens for richer notions of refinement as investigated in the context of STAIRS [96]. Such an increased expressiveness and richer notions of refinement may be useful, but possibly at the cost of user friendliness and support for human understanding and communication between stakeholders. This potential clash of interests must therefore be carefully evaluated.

An important issue of development methods that has been outside the scope of this thesis is that of complexity. The complexity metrics for modeling languages and the methods the languages support presented in [90] can be used for evaluating the complexity of the framework for policy development proposed in this thesis. The authors claim that the metrics can be used to compare methods, and substantiate that one method is preferred over another, since "the relative complexity of methods and techniques... is expected to affect the learnability and ease of use of a method" [90]. In addition to usage, the complexity of methods may affect the development costs [32]. The complexity of verification or testing of refinement, adherence and consistency is also an important and interesting direction for future work.

The development of tool support is interesting as tools facilitate policy specification, development and analysis, thereby increasing the practical applicability of the approach. A prototype tool has been developed for STAIRS for generating the semantics of sequence diagrams, checking the correctness of refinement steps and generating tests [73]. A similar tool should be developed for supporting specification and development of policy specifications.

Bibliography

[1] Jan Øyvind Aagedal and Zoran Milošević. ODP enterprise language: UML perspective. In *Proceedings of the 3rd International Conference on Enterprise Distributed Object Computing (EDOC'99)*, pages 60–71. IEEE CS Press, 1999.

[2] Harold Abelson and Gerald Jay Sussman. *Structure and Interpretation of Computer Programs.* MIT Press, second edition, 1996.

[3] AS/NZS. *Australian/New Zealand Standard, AS/NZS 4360:2004, Risk Management*, 2004.

[4] Arosha K. Bandara, Emil C. Lupu, Jonathan Moffet, and Alessandra Russo. A goal-based approach to policy refinement. In *Proceedings of the 5th International Workshop on Policies for Distributed Systems and Networks (POLICY'04)*, pages 229–239. IEEE Computer Society, 2004.

[5] Arosha K. Bandara, Emil C. Lupu, and Alessandra Russo. Using event calculus to formalise policy specification and analysis. In *Proceedings of the 4th International Workshop on Policies for Distributed Systems and Networks (POLICY'03)*, pages 26–39. IEEE Computer Society, 2003.

[6] Arosha K. Bandara, Emil C. Lupu, Alessandra Russo, Naranker Dulay, Morris Sloman, Paris Flegkas, Marinos Charalambides, and George Pavlou. Policy refinement for DiffServ quality of service management. In *Proceedings of the 9th IFIP/IEEE International Symposium on Integrated Network Management (IM'05)*, pages 469–482, 2005.

[7] David Basin, Jürgen Doser, and Torsten Lodderstedt. Model driven security: From UML models to access control infrastructures. *ACM Transactions on Software Engineering and Methodology*, 15(1):39–91, 2006.

[8] D. Elliott Bell and Leonard J. LaPadula. Secure computer systems: Mathematical foundations. Technical Report MTR-2547, MITRE Corporation, 1973.

[9] Claudio Bettini, Sushil Jajodia, X. Sean Wang, and Duminda Wijesekera. Provisions and obligations in policy rule management. *Journal of Network and Systems Management*, 11(3):351–372, 2003.

[10] Kenneth J. Biba. Integrity considerations for secure computer systems. Technical Report MTR-3153, MITRE Corporation, 1977.

[11] Matt Bishop. *Computer Security: Art and Science.* Addison-Wesley, 2003.

[12] Xavier Blanc, Marie-Pierre Geravis, and Raymonde Le-Delliou. Using the UML language to express the ODP enterprise concepts. In *Proceedings of the 3rd International Conference on Enterprise Distributed Object Computing (EDOC'99)*, pages 50–59. IEEE CS Press, 1999.

[13] Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis. Keynote: Trust management for public-key infrastructures. In *Proceedings of the 6th International Workshop on Security Protocols*, volume 1550 of *LNCS*, pages 59–63. Springer, 1999.

[14] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proceedings of the IEEE Conference on Security and Privacy (SP'96)*, pages 164–173. IEEE Computer Society, 1996.

[15] Matt Blaze, Joan Feigenbaum, and Martin Strauss. Compliance checking in the policymaker trust management system. In *Proceedings of the 2nd International Conference on Financial Cryptography (FC'98)*, volume 1465 of *LNCS*, pages 254–274. Springer, 1998.

[16] Raouf Boutaba and Issam Aib. Policy-based management: A historical perspective. *Journal of Network and Systems Management*, 15(4):447–480, 2007.

[17] Frederick P. Brooks. The computer scientist as a toolsmith II. *Communications of the ACM*, 39(3):61–68, 1996.

[18] Manfred Broy and Ketil Stølen. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces and Refinement.* Springer, 2001.

[19] Cristiano Castelfranci and Rino Falcone. Social trust: A cognitive approach. In *Trust and Deception in Virtual Societies*, pages 55–90. Kluwer Academic Publishers, 2001.

[20] European Commission. Eurostat. http://epp.eurostat.ec.europa.eu.

[21] The UK Central Computer and Telecommunications Agency. *CRAMM User Guide, issue 5.1*, 2005.

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

[23] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The ponder policy specification language. In *Proceedings of the 2nd International Workshop on Policies for Distributed Systems and Networks (POLICY'01)*, volume 1995 of *LNCS*, pages 18–38. Springer, 2001.

[24] Werner Damm and David Harel. LSCs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19(1):45–80, 2001.

[25] Robert Darimont and Axel van Lamsweerde. Formal refinement patterns for goal-driven requirements elaboration. In *Proceedings of the 4th ACM Symposium on the Foundations of Software Engineering (FSE4)*, pages 179–190, 1996.

[26] João Porto de Albuquerque, Heiko Krumm, and Paulo Lício de Geus. Policy modeling and refinement for network security systems. In *Proceedings of the 6th International Workshop on Policies for Distributed Systems and Networks (POLICY'05)*, pages 24–33. IEEE Computer Society, 2005.

[27] Folker den Braber, Ida Hogganvik, Mass Soldal Lund, Ketil Stølen, and Fredrik Vraalsen. Model-based security analysis in seven steps – a guided tour to the CORAS method. *BT Techology Journal*, 25(1):101–117, 2007.

[28] Peter J. Denning. Is computer science science? *Communications of the ACM*, 48(4):27–31, 2005.

[29] The Distributed Management Task Force. www.dmtf.org.

[30] Brian Dobing and Jeffrey Parsons. How UML is used. *Communications of the ACM*, 49(5):109–113, 2006.

[31] Nicole Dunlop, Jadwiga Indulska, and Kerry Raymond. Dynamic conflict detection in policy-based management systems. In *Proceedings of the 6th International Enterprise Distributed Object Computing Conference (EDOC'02)*, pages 15–26. IEEE Computer Society, 2002.

[32] John Erickson and Keng Siau. Theoretical and practical complexity of modeling methods. *Communications of the ACM*, 50(8):46–51, 2007.

[33] David Ferraiolo and Richard Kuhn. Role-based access controls. In *Proceedings of the 15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[34] Diego Gambetta. Can we trust trust? In *Trust: Making and Breaking Cooperative Relations*, chapter 13, pages 213–237. Department of Sociology, University of Oxford, 2000. Electronic edition.

[35] Paolo Giorgini, Fabio Massacci, John Mylopoulos, and Nicola Zannone. Requirements engineering meets trust management: Model, methodology, and reasoning. In *iTrust 2004*, volume 2995 of *LNCS*, pages 176–190. Springer, 2004.

[36] Paolo Giorgini, Fabio Massacci, John Mylopoulos, and Nicola Zannone. Modelling social and individual trust in requirements engineering methodologies. In *iTrust 2005*, volume 3477 of *LNCS*, pages 161–176. Springer, 2005.

[37] Tyrone Grandison and Morris Sloman. Specifying and analysing trust for Internet applications. In *Proceedings of the Second IFIP Conference on E-Commerce, E-Business, E-Government (I3E'02)*, pages 145–157. Kluwer, 2002.

[38] Tyrone Grandison and Morris Sloman. Trust management tools for internet applications. In *iTrust 2003*, volume 2692 of *LNCS*, pages 91–107. Springer, 2003.

[39] David Harel and Shahar Maoz. Assert and negate revisited: Modal semantics for UML sequence diagrams. *Software and Systems Modeling*, 7(2):237–252, 2007.

[40] David Harel and Rami Marelly. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine.* Springer, 2003.

[41] Øystein Haugen, Knut Eilif Husa, Ragnhild Kobro Runde, and Ketil Stølen. STAIRS towards formal design with sequence diagrams. *Software and Systems Modeling*, 4(4):355–367, 2005.

[42] Manuel Hilty, David Basin, and Alexander Pretschner. On obligations. In *Proceedings of the 10th European Symposium on Research in Computer Security (ESORICS'05)*, volume 3679 of *LNCS*, pages 98–117. Springer, 2005.

[43] Manuel Hilty, Alexander Pretschner, David Basin, Christian Schaefer, and Thomas Walter. A policy language for distributed usage control. In *Proceedings of the 12th European Symposium on Research in Computer Security (ESORICS'07)*, volume 4734 of *LNCS*, pages 531–546. Springer, 2007.

[44] James A. Hoagland, Raju Pandey, and Karl N. Levitt. Security policy specification using a graphical approach. Technical Report CSE-98-

3, Department of Computer Science, University of California, Davis, 1998.

[45] IEEE International Symposium on Policies for Distributed Systems and Networks. www.ieee-policy.org/.

[46] International Telecommunication Union. *Recommendation Z.120 Annex B – Semantics of Message Sequence Chart (MSC)*, 1998.

[47] International Telecommunication Union. *Recommendation Z.120 – Message Sequence Chart (MSC)*, 1999.

[48] Internet world stats. www.internetworldstats.com/.

[49] The Internet Engineering Task Force. www.ietf.org.

[50] ISO/IEC. *ISO/IEC 13335, Information technology – Guidelines for management of IT security*, 1996–2000.

[51] ISO/IEC. *FCD 15414, Information Technology - Open Distributed Processing - Reference Model - Enterprise Viewpoint*, 2000.

[52] Jeremy Jacob. On the derivation of secure components. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'89)*, pages 242–247. IEEE Computer Society, 1989.

[53] Audun Jøsang. A logic for uncertain probabilities. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 9(3):279–311, 2001.

[54] Audun Jøsang, Touhid Bhuiyan, Yue Xu, and Clive Cox. Combining trust and reputation management for web-based services. In *Proceedings of the 5th International Conference on Trust, Privacy and Security in Digital Business (TrustBus'08)*, volume 5185 of *LNCS*, pages 90–99. Springer, 2008.

[55] Audun Jøsang, Daniel Bradley, and Svein J. Knapskog. Belief-based risk analysis. In *Proceedings of the 2nd Workshop on Australasian Information Security, Data Mining and Web Intelligence, and Software Internationalisation (AISW'04)*, pages 63–68. Australian Computer Society, 2004.

[56] Audun Jøsang, Claudia Keser, and Theo Dimitrakos. Can we manage trust? In *iTrust 2005*, volume 3477 of *LNCS*, pages 93–107. Springer, 2005.

[57] Audun Jøsang and Simon Pope. Semantic constraints for trust transitivity. In *Proceedings of the 2nd Asia-Pacific conference on Conceptual*

Modelling (APCCM'05), pages 59–68. Australian Computer Society, 2005.

[58] Wolfram Jost. Closing the gap between business and IT. *SOA World Magazine*, 2007.

[59] Jan Jürjens. UMLsec: Extending UML for secure systems development. In *Proceedings of the 5th International Conference on the Unified Modeling Language (UML'02)*, volume 2460 of *LNCS*, pages 412–425. Springer, 2002.

[60] Jan Jürjens. *Secure Systems Development with UML*. Springer, 2005.

[61] Lalana Kagal, Tim Finin, and Anupam Joshi. A policy language for a pervasive computing environment. In *Proceedings of the 4th International Workshop on Policies for Distributed Systems and Networks (POLICY'03)*, pages 63–74. IEEE Computer Society, 2003.

[62] Joost-Pieter Katoen and Lennard Lambert. Pomsets for message sequence charts. In *Formale Beschreibungstechniken für verteilte Systeme*, pages 197–208. Shaker, 1998.

[63] Manuel Koch and Francesco Parisi-Presicce. Visual specifications of policies and their verification. In *Proceedings of the 6th International Conference on Fundamental Approaches to Software Engineering (FASE'03)*, volume 2621 of *LNCS*, pages 278–293. Springer, 2003.

[64] Thomas Koch, Christoph Krell, and Bernd Krämer. Policy definition language for automated management of distributed systems. In *Proceedings of the 2nd IEEE International Workshop on Systems Management (SMW'96)*, pages 55–64. IEEE Computer Society, 1996.

[65] Robert A. Kowalski and Marek J. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.

[66] Ingolf Heiko Krüger. *Distributed System Design with Message Sequence Charts*. PhD thesis, Institut für Informatik, Ludwig-Maximilians-Universität München, July 2000.

[67] Leslie Lamport. How to write a proof. *American Mathematical Monthly*, 102(7):600–608, 1993.

[68] Butler W. Lampson. Protection. In *Proceedings of the 5th Princeton Symposium of Information Science and Systems*, pages 437–443, 1971.

[69] María M. Larrondo-Petrie, Ehud Gudes, Haiyan Song, and Eduardo B. Fernández. Security policies in object-oriented databases. In *Database Security III: Status and Prospects*, pages 257–268, 1989.

[70] Peter Linington. Options for expressing ODP enterprise communities and their policies by using UML. In *Proceedings of the 3rd International Conference on Enterprise Distributed Object Computing (EDOC'99)*, pages 72–82. IEEE CS Press, 1999.

[71] Jorge Lobo, Randeep Bhatia, and Shamim Naqvi. A policy description language. In *Proceedings of the 16th National Conference on Artificial Intelligence and the 11th Innovative Applications of Artificial Intelligence Conference (AAAI'99/IAAI'99)*, pages 291–298. American Association for Artificial Intelligence, 1999.

[72] Niklas Luhmann. Familiarity, confidence, trust: Problems and alternatives. In *Trust: Making and Breaking Cooperative Relations*, chapter 6, pages 94–107. Department of Sociology, University of Oxford, 2000. Electronic edition.

[73] Mass Soldal Lund. *Operational analysis of sequence diagram specifications.* PhD thesis, Faculty of Mathematics and Natural Sciences, University of Oslo, 2008.

[74] Emil Lupu and Morris Sloman. Conflicts in policy-based distributed systems management. *IEEE Transactions on Software Engineering*, 25(6):852–869, 1999.

[75] Tom Lysemose, Tobias Mahler, Bjørnar Solhaug, Jon Bing, Dag Elgesem, and Ketil Stølen. ENFORCE conceptual framework. Technical Report A1209, SINTEF ICT, 2007.

[76] Heiko Mantel. Preserving information flow properties under refinement. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'01)*, pages 78–91. IEEE Computer Society, 2001.

[77] Sjouke Mauw and Michel A. Reniers. High-level message sequence charts. In *Proceedings of the 8th SDL Forum*, pages 291–306. Elsevier, 1997.

[78] Sjouke Mauw and Michel A. Reniers. Operational semantics for MSC'96. *Computer Networks and ISDN Systems*, 31(17):1785–1799, 1999.

[79] Paul McNamara. Deontic logic. In Dov M. Gabbay and John Woods, editors, *Logic and the Modalities in the Twentieth Century*, volume 7 of *Handbook of the History of Logic*, pages 197–288. Elsevier, 2006.

[80] Jonathan D. Moffett and Morris S. Sloman. Policy hierarchies for distributed systems management. *IEEE Journal on Selected Areas in Communications*, 11:1404–1414, 1993.

[81] Bob Moore, Ed Ellesson, John Strassner, and Andrea Westerinen. *RFC 3060 – Policy Core Information Model – Version 1 Specification*, 2001.

[82] OASIS. *eXstensible Access Control Markup Language (XACML) Version 2.0*, 2005.

[83] Object Management Group. *Unified Modeling Language: Superstructure, version 2.1.1*, 2007.

[84] Jaehong Park and Ravi Sandhu. The $UCON_{ABC}$ usage control model. *ACM Transactions on Information and System Security*, 7(1):128–174, 2004.

[85] Dean Povey. Developing electronic trust policies using a risk management model. In *Proceedings of the International Exhibition and Congress on Secure Networking (CQRE (Secure)'99)*, volume 1740 of *LNCS*, pages 1–16. Springer, 1999.

[86] Andreas Prinz. Formal semantics of specification languages. *Telektronikk*, 4:146–155, 2000.

[87] Atle Refsdal. *Specifying Computer Systems with Probabilistic Sequence Diagrams.* PhD thesis, Faculty of Mathematics and Natural Sciences, University of Oslo, 2008.

[88] Atle Refsdal, Bjørnar Solhaug, and Ketil Stølen. A UML-based method for the development of policies to support trust management. In *Trust Management II – Proceedings of the 2nd Joint iTrust and PST Conference on Privacy, Trust Management and Security (IFIPTM'08)*, volume 263 of *IFIP*, pages 33–49. Springer, 2008.

[89] Atle Refsdal and Ketil Stølen. Extending UML sequence diagrams to model trust-dependent behavior with the aim to support risk analysis. *Science of Computer Programming*, 74(1-2):34–42, 2008.

[90] Matti Rossi and Sjaak Brinkkemper. Complexity metrics for systems development methods and techniques. *Information Systems*, 21(2):209–227, 1996.

[91] Javier Rubio-Loyola. *A Methodological Approach to Policy Refinement in Policy-based Management Systems.* PhD thesis, Departament de Teoria del Senyal i Comunicacions, Universitat Politècnica de Catalunya, 2007.

[92] Javier Rubio-Loyola, Joan Serrat, Marinos Charalambides, Paris Flegkas, and George Pavlou. A functional solution for goal-oriented policy refinement. In *Proceedings of the 7th International Workshop*

on Policies for Distributed Systems and Networks (POLICY'06), pages 133–144. IEEE Computer Society, 2006.

[93] Javier Rubio-Loyola, Joan Serrat, Marinos Charalambides, Paris Flegkas, George Pavlou, and Alberto Lluch Lafuente. Using linear temporal model checking for goal-oriented policy refinement frameworks. In *Proceedings of the 6th International Workshop on Policies for Distributed Systems and Networks (POLICY'05)*, pages 181–190. IEEE Computer Society, 2005.

[94] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. Addison Wesley, second edition, 2005.

[95] Ragnhild Kobro Runde, Øystein Haugen, and Ketil Stølen. Refining UML interactions with underspecification and nondeterminism. *Nordic Journal of Computing*, 12(2):157–188, 2005.

[96] Ragnhild Kobro Runde, Atle Refsdal, and Ketil Stølen. Relating computer systems to sequence diagrams with underspecification, inherent nondeterminism and probabilistic choice – Part 1: Underspecification and inherent nondeterminism. Technical Report 346, Department of Informatics, University of Oslo, 2007.

[97] Sini Ruohomaa and Lea Kutvonen. Trust management survey. In *iTrust 2005*, volume 3477 of *LNCS*, pages 77–92. Springer, 2005.

[98] Fred B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security*, 3(1):30–50, 2000.

[99] Fredrik Seehusen, Bjørnar Solhaug, and Ketil Stølen. Adherence preserving refinement of trace-set properties in STAIRS: Exemplified for information flow properties and policies. *Software and Systems Modeling*, 8(1):45–65, 2009.

[100] Fredrik Seehusen and Ketil Stølen. A transformational approach to facilitate monitoring of high-level policies. In *Proceedings of the 9th International Workshop on Policies for Distributed Systems and Networks (POLICY'08)*, pages 70–73. IEEE Computer Society, 2008.

[101] Bikram Sengupta and Rance Cleaveland. Triggered message sequence charts. *IEEE Transactions on Software Engineering*, 32(8):587–607, 2006.

[102] Morris Sloman. Policy driven management for distributed systems. *Journal of Network and Systems Management*, 2(4):333–360, 1994.

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

[104] Bjørnar Solhaug, Dag Elgesem, and Ketil Stølen. Specifying policies using UML sequence diagrams – An evaluation based on a case study. In *Proceedings of the 8th International Workshop on Policies for Distributed Systems and Networks (POLICY'07)*, pages 19–28. IEEE Computer Society, 2007.

[105] Bjørnar Solhaug, Dag Elgesem, and Ketil Stølen. Why trust is not proportional to risk. In *Proceedings of the 2nd International Conference on Availability, Reliability and Security (ARES'07)*, pages 11–18. IEEE Computer Society, 2007.

[106] Bjørnar Solhaug and Tor Hjalmar Johannessen. Specification of policies using UML sequence diagrams. *Telektronikk*, 105(1):90–97, 2009.

[107] Bjørnar Solhaug and Ketil Stølen. Compositional refinement of policies in UML – Exemplified for access control. In *Proceedings of the 13th European Symposium on Research in Computer Security (ESORICS'08)*, volume 5283 of *LNCS*, pages 300–316. Springer, 2008.

[108] Ida Solheim and Ketil Stølen. Technology research explained. Technical Report A313, SINTEF ICT, 2007.

[109] Linying Su, David W. Chadwick, Andrew Basden, and James A. Cunningham. Automated decomposition of access control policies. In *Proceedings of the 6th International Workshop on Policies for Distributed Systems and Networks (POLICY'05)*, pages 3–13. IEEE Computer Society, 2005.

[110] Aashu Virmani, Jorge Lobo, and Madhur Kohli. Netmon: Network management for the SARAS softswitch. In *Proceedings of IEEE/IFIP Network Operations and Management Seminar (NOMS'00)*, pages 803–816, 2000.

[111] Thomas Weigert. UML 2.0 RFI response overview. Object Management Group, document: ad/00-01-07, 1999.

[112] Andrea Westerinen, John Schnizlein, John Strassner, Mark Scherling, Bob Quinn, Shai Herzog, An-Ni Huynh, Mark Carlson, Jay Perry, and Steve Waldbusser. *RFC 3198 – Terminology for Policy-Based Management*, 2001.

[113] Jeannette M. Wing. A specifier's introduction to formal methods. *IEEE Computer*, 23(9):8,10–22,24, 1990.

[114] Marianne Winslett. An introduction to trust negotiation. In *iTrust 2003*, volume 2692 of *LNCS*, pages 275–283. Springer, 2003.