Assurance Management Framework for Access Control Systems

by

Hongxin Hu

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved July 2012 by the
Graduate Supervisory Committee:

Gail-Joon Ahn, Chair
Stephen S. Yau
Partha Dasgupta
Nong Ye

ARIZONA STATE UNIVERSITY

August 2012

ABSTRACT

Access control is one of the most fundamental security mechanisms used in the design and management of modern information systems. However, there still exists an open question on how formal access control models can be automatically analyzed and fully realized in secure system development. Furthermore, specifying and managing access control policies are often error-prone due to the lack of effective analysis mechanisms and tools.

In this dissertation, I present an Assurance Management Framework (AMF) that is designed to cope with various assurance management requirements from both access control system development and policy-based computing. On one hand, the AMF framework facilitates comprehensive analysis and thorough realization of formal access control models in secure system development. I demonstrate how this method can be applied to build role-based access control systems by adopting the NIST/ANSI RBAC standard as an underlying security model. On the other hand, the AMF framework ensures the correctness of access control policies in policy-based computing through automated reasoning techniques and anomaly management mechanisms. A systematic method is presented to formulate XACML in Answer Set Programming (ASP) that allows users to leverage off-the-shelf ASP solvers for a variety of analysis services. In addition, I introduce a novel anomaly management mechanism, along with a grid-based visualization approach, which enables systematic and effective detection and resolution of policy anomalies. I further evaluate the AMF framework through modeling and analyzing multiparty access control in Online Social Networks (OSNs). A MultiParty Access Control (MPAC) model is formulated to capture the essence of multiparty authorization requirements in OSNs. In particular, I show how AMF can be applied to OSNs for identifying and resolving privacy conflicts, and representing and reasoning about

MPAC model and policy. To demonstrate the feasibility of the proposed methodology, a suite of proof-of-concept prototype systems is implemented as well.

Dedicated to my family

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my advisor, Prof. Gail-Joon Ahn. He helped me cultivate a taste in research problems, and was a constant source of invaluable advice on navigating through the academic world. I feel truly lucky to have him as my advisor and I sincerely hope that we will remain both collaborators and friends for many years to come. I also wish to express my deep appreciation to my PhD committee members, Prof. Stephen S. Yau, Prof. Partha Dasgupta, and Prof. Nong Ye. The assistance and guidance they provided in the preparation of this dissertation have been invaluable.

Many thanks to the numerous individuals who worked with me on collaborative papers, and on topics related to my research over the last few years, including Prof. Joohyung Lee, Dr. Xinwen Zhang (at Huawei America Research Center), Prof. Yan Zhu (at Peking University), Prof. Wenjuan Xu (at Frostburg State University), Dr. Jing Jin (at Deutsche Bank Global Technologies) and graduate students Yunsong Meng and Dejun Yang. Their feedback and criticism enriched my work and helped me to align it with other research projects.

To the Laboratory of Security Engineering for Future Computing (SEFCOM), it has been a great pleasure working with all lab members during my journey as a doctoral student, particularly Ziming Zhao, Ruoyu Wu, Yiming Jing, Ketan Kulkarni, Jan Jorgensen, Michael Mabey, Pradeep Sekar, Justin Paglierani, Deepinder Mahi, Michael Sanchez, Patrick Trang and Jeong-Jin Seo. This would never be possible without their incessant help and insightful discussions.

Last but not least, I would like to thank my wife Lei Shao. Her support, encouragement, patience and unconditional love enabled me to surpass hardships and complete this work. I also thank my parents Guohui Hu and Yuanzhen Zhang,

and my sisters Qionghua Hu and Xiaoqiong Hu. They were always supporting me and encouraging me with their best wishes. Additionally, I would thank my son Ruijia (Roger) Hu. He gave me the motivation and courage to get through this work.

My dissertation work was supported in part by the grants from National Science Foundation (NSF-IIS-0900970 and NSF-CNS-0831360), and Department of Energy (DE-SC0004308 and DE-FG02-03ER25565).

TABLE OF CONTENTS

Page

LIST OF TABLES x

LIST OF FIGURES xi

CHAPTER 1

1 Introduction 1

1.1 Statement of the Hypothesis 3

1.2 Dissertation Organization 5

2 Related Work 6

2.1 Representation and Analysis of Access Control Model 6

2.2 Representation and Analysis of Access Control Policy 8

Representing and Reasoning about Access Control Policy 8

Anomaly Discovery and Resolution for Access Control Policy . . . 9

Visualization-based Policy Representation 10

2.3 Access Control for Online Social Networks 10

3 Background Information 13

3.1 Role-Based Access Control Standard 13

3.2 Unified Modeling Language and Object Constraint Language 13

3.3 Role-Based Constraints Language 2000 14

3.4 Alloy 15

3.5 Extensible Access Control Markup Language 16

3.6 Answer Set Programming 16

4 Assurance Management Framework (AMF) 19

4.1 Overview 19

4.2 Access Control Model 20

Analysis Approach in AMF for Access Control Model 22

CHAPTER Page
Access Control Model Verification 23
Access Control Model Testing 28
Realization and Analysis of RBAC Model 29
Realization of RBAC Model 30
RBAC Model Representation in UML and OCL . . . 31
RBAC Constraint Specification in OCL 38
Code Generation 43
Analysis of RBAC Model 44
RBAC Model Representation in Alloy 45
RBAC Constraint Specification in Alloy 47
RBAC Function Verification 47
RBAC Constraint Verification 51
Test Case Generation 54
Tool Support. 56
RBAC Authorization Environment (RAE) 57
RBAC Authorization Simulation System (RASS) . . 59
Tool Chain 60
4.3 Access Control Policy 61
Representing and Reasoning about Access Control Policy 63
Example XACML Policy 64
Abstracting XACML Policy Components 66
XACML Policy Analysis using ASP 68
Implementation and Evaluation 72
Anomaly Detection and Resolution for Access Control Policy 74
Anomalies in XACML Policies 76

CHAPTER Page
Underlying Data Structure 79
Conflict Detection and Resolution 82
Conflict Detection Approach 82
Fine-Grained Conflict Resolution 89
Redundancy Discovery and Removal 92
Redundancy Elimination at Policy Level 92
Redundancy Elimination at Policy Set Level 98
Implementation and Evaluation 101
5 Applying AMF to Online Social Networks 108
5.1 Multiparty Access Control for OSNs: Requirements and Patterns . . 111
5.2 Modeling Multiparty Access Control for OSNs 115
MPAC Model 115
MPAC Policy Specification 119
5.3 Identifying and Resolving Privacy Conflicts 121
Privacy Conflict Identification 122
Privacy Conflict Resolution 124
Generating Conflict-Resolved Policy 128
5.4 Logical Representation and Analysis of Multiparty Access Control . 129
Representing Multiparty Access Control in ASP 129
Logical Representation of MPAC Model and Policy 129
Logical Representation of Privacy Conflict Detection and Resolution 131
Reasoning about Multiparty Access Control 131
5.5 Implementation and Evaluation 133
Prototype Implementation 133

CHAPTER Page
Evaluation and Experiment . . . 137
Evaluation of Privacy Conflict Resolution . . . 137
Evaluation of System Usability . . . 140
5.6 Discussion . . . 144
6 Conclusion . . . 146
6.1 Summary . . . 146
Contribution . . . 148
6.2 Future Work . . . 149
Realization and Analysis of Access Control Model . . . 149
Analysis and Management of Access Control Policy . . . 149
Applying AMF to Emerging Domains . . . 150
Social Networks . . . 150
Cloud Computing . . . 151
Mobile Computing . . . 152
Healthcare Systems . . . 152
REFERENCES . . . 154

LIST OF TABLES

Table Page

4.1 Mapping RCL2000 expression to OCL expression for *SSoD-CR*. 42

4.2 Experimental results on real-life XACML policies. 73

4.3 Atomic Boolean expressions and corresponding Boolean variables for P_1. 80

4.4 XACML policies used for evaluation. 105

4.5 Conflict detection and redundancy removal algorithms evaluation. . . . 106

5.1 Usability evaluation for Facebook and *Retinue* privacy controls. 141

5.2 Perceived usability of *Factbook* privacy controls (before using *Retinue*). 142

5.3 Perceived usability of *Retinue* privacy controls (after using *Retinue*). . . 143

LIST OF FIGURES

Figure Page

4.1 Assurance management framework. 19

4.2 Realization and analysis of access control model. 21

4.3 Function verification. 24

4.4 Identifying under-constraint. 26

4.5 Identifying over-constraint. 26

4.6 Constraint verification. 27

4.7 Test case generation for constraints. 28

4.8 Realization and analysis of RBAC model with AMF. 30

4.9 RBAC model representation in UML class diagram. 31

4.10 Translation algorithm from RCL2000 to OCL. 40

4.11 Generated Java code for `CheckStaticConstraints` function. 44

4.12 Translation algorithm from RCL2000 to Alloy. 48

4.13 Structural overview of the RAE. 56

4.14 RAE tool and RASS testbed environment 59

4.15 Toolchain supporting the proposed approach. 61

4.16 Representation and analysis of access control policy. 62

4.17 An example XACML policy. 65

4.18 ASP representation of the example XACML policy. 69

4.19 Anomalies in an example XACML policy. 77

4.20 Representing and operating on rules of XACML policy with BDD. . . . 81

4.21 Authorization space representation for policy P_1 in the example XACML policy. 85

4.22 Aggregation of authorization spaces for policy P_1 in the example XACML policy. 88

Figure Page

4.23 Authorization space representation for policy set PS_1 in the example XACML policy. 89
4.24 Fine-grained conflict resolution framework. 90
4.25 Example of eliminating redundancies at policy level. 93
4.26 Example of rule correlation break. 96
4.27 Example of authorization space segmentation at policy set level for redundancy discovery and removal. 98
4.28 `XAnalyzer` interface. 103
4.29 Evaluation of redundancy removal approach. 106
5.1 Multiparty access control pattern for profile and relationship sharing. . 111
5.2 Multiparty access control pattern for content sharing. 113
5.3 An example of multiparty social network representation. 118
5.4 Example of privacy conflict identification based on accessor space segmentation. 124
5.5 System architecture of *Retinue*. 134
5.6 *Retinue* interfaces. 135
5.7 Example of resolving privacy conflicts. 137
5.8 Conflict resolution evaluation. 138

Chapter 1

Introduction

The advent of emerging technologies such as service-oriented architecture and cloud computing has enabled users to perform business services more efficiently and effectively. However, we still suffer from unintended security leakages by unauthorized actions in business services while providing more convenient services to users through such a cutting-edge technological growth. Access control is one of the most fundamental and pervasive mechanisms in use today to secure these services.

There have been two parallel areas in access control research in recent years. On one hand, there are efforts to develop access control models to fulfil the authorization requirements from real-world application domains. These have turned out several successful and well-established access control models, such as the RBAC96 model [1], the NIST/ANSI standard RBAC model [2, 3], the RT model [4], and the Usage Control model [5]. In parallel, and almost separately, many researchers have devoted to develop policy languages for access control to support policy-based computing, including application-level policies (e.g., XACML [6], SAML [7], Ponder [8] and EPAL [9]), network-level policies (e.g., firewall policy [10] and IPSec policy [11]), and system-level policies (e.g., SELinux policy [12] and AppArmor policy [13]).

Software developers utilize models extensively, particularly in the early software development lifecycle to improve software quality. Since security has become a necessary part of nearly most modern software and information systems, access control models can be leveraged to integrate the security concerns into the software development process [14, 15]. However, several challenging issues should be taken into account for applying access control models in secure system develop-

ments. First, there exists a gap between access control models and building secure systems with such formal models. Access control models are generally described in some forms of formalism, but software developers are often reluctant to fully adopt a formal model for their development tasks. Consequently, it is very desirable to have a mechanism and corresponding tool to aid software developers or system administrators in understanding and articulating a specific access control model in the software analysis and design phases. Second, it is crucial to verify and validate the access control models and associated constraints before actual implementation commences, such that flaws and conflicts in the system design can thus be identified as early as possible, and can be efficiently resolved accordingly. Third, the access control models that are specified with modeling languages should be translated to security enforcement codes to derive appropriate security properties for the system implementation. Besides, the consistency between the design model and its implementation, and the correctness of the translation should be evaluated.

On the other hand, the use of a policy-based approach has recently received considerable attention to accommodate the security requirements covering large, open, distributed and heterogeneous computing environments [6, 8, 10, 12]. Policy-based computing handles complex system properties by separating policies from system implementation and enabling dynamic adaptability of system behaviors by changing policy configurations without reprogramming the systems. Considering that most recent policy language proposals supporting complicated and distributed systems, assuring the correctness of policy specifications becomes a crucial and yet challenging task. Especially, identifying inconsistencies and differences between policy specifications and their expected functions is critical since the correctness of the implementation and enforcement of policies heavily relies on the policy specification. Consequently, the increasing complexity of policy-based comput-

ing strongly demands automated analysis techniques. Without having such analysis techniques in place, most benefits of policy-based techniques and declarative policy languages may be in vain.

Even though some approaches have been proposed from various aspects to address the issues with respect to the representation and verification of access control models [14, 16, 17, 18, 19], as well as the analysis and management of access control policies [20, 21, 22, 23, 24], the preliminary study in this work clearly identifies that there is a need to design a systematic mechanism that is general and flexible enough to reflect and deal with the various *assurance* management requirements rising from both access control system development and policy-based computing. In this research, I would make one step towards this direction. In this dissertation, I define the concept of ***assurance*** as "the process-driven management for access control systems from formal representation to practical enforcement that ensures the correctness and conformance of access control systems".

1.1 Statement of the Hypothesis

Therefore, this research hypothesizes that:

Systematic analysis and practical realization of access control models and policies are necessary to articulate assurance relevant requirements, such as correctness and conformance of access control systems, and to accommodate these requirements for achieving the assurance of access control systems.

In this dissertation, I introduce an Assurance Management Framework (AMF), which enables formal access control models are fully realized in real systems via model representation, constraint specification, and generation of enforcement codes.

Thus, the gap between access control models and the development of access control systems can be minimized. Particularly, model-based verification and model-based testing for access control are articulated in this framework, in which the formal specifications of access control models and constraints are verified, and test cases are derived from the formal specifications automatically. The generated test cases are used to validate whether the secure system design and implementation conform to the formal specifications. Consequently, the analysis and testing of access control models in the AMF framework could provide higher assurance for the design and implementation of access control systems. I then adopt the NIST/ANSI RBAC standard [2] to demonstrate the feasibility of the proposed approach. An RBAC authorization environment `RAE` and a simulation system `RASS` are implemented as well, and they can cooperate with `Alloy Analyzer` [25] to accommodate features addressed in the framework.

Besides, the AMF framework ensures the correctness of access control policies for policy-based computing by adopting automated reasoning techniques and systematic anomaly management mechanisms. I present a logic-based policy management approach for access control policies especially focusing on XACML (eXtensible Access Control Markup Language) policies [6]. Answer Set Programming (ASP) [26, 27] is adopted to formulate XACML policies that allows users to leverage the features of ASP solvers in performing various logical reasoning and analysis tasks such as policy verification, comparison and querying. Moreover, I introduce a policy-based segmentation technique to accurately identify policy anomalies and derive effective anomaly resolutions, along with an intuitive visualization representation of analysis results. In addition, I discuss the implementation of an anomaly analysis tool called `XAnalyzer` and demonstrate how the proposed approach can efficiently discover and resolve policy anomalies.

I further evaluate the applicability of the AMF framework through modeling and analyzing multiparty access control in Online Social Networks (OSNs). A MultiParty Access Control (MPAC) model is formulated to capture the core features of multiparty authorization requirements which have not been accommodated so far by existing access control systems and models for OSNs (e.g., [28, 29, 30, 31]). In the meanwhile, since privacy conflicts are inevitable in multiparty authorization, a systematic mechanism is provided to identify and resolve privacy conflicts for collaborative data sharing in OSNs. In particular, the conflict resolution indicates a tradeoff between privacy protection and data sharing by quantifying privacy risk and sharing loss. Moreover, I introduce an approach for representing and reasoning about MPAC model and policy with ASP. I also discuss a proof-of-concept prototype implementation of the proposed approach called `Retinue` and provide system evaluation and usability study of the methodology.

1.2 Dissertation Organization

The remainder of this dissertation is organized as follows. Chapter 2 discusses related work. Chapter 3 provides background on several technologies. Chapter 4 overviews the AMF framework, elaborates the processes of the analysis and realization of access control model in AMF, and addresses the techniques of the policy reasoning and anomaly management in AMF. Chapter 5 articulates how to apply the AMF framework to OSNs through modeling and analyzing multiparty access control. Finally, Chapter 6 summarizes this dissertation and presents some directions for future work.

Chapter 2

Related Work

In this chapter, I summarize the related work to this dissertation.

2.1 Representation and Analysis of Access Control Model

Many research efforts have been devoted to UML-based modeling of security model. Ahn et al. [16] showed how RBAC model and constraints can be expressed in UML using OCL. Jürjens et al. [32] proposed an extension to UML that defines several new stereotypes towards formal security verification of elements. Alghathbar et al. [33] defined an approach AuthUML that includes a process and a modeling language to express RBAC policies via use cases. Ray et al. [15] specified reusable RBAC policies using UML diagram templates and showed how RBAC policies can be easily integrated with the application. Basin et al. [14] defined a metamodel to generate security definition languages, an instance of which is SecureUML, a platform-independent language for RBAC. Mouheb et al. [34] presented an aspect-oriented modeling approach for specifying and integrating security concerns into UML design models. All of these approaches accommodated security requirements without considering the validation of security model and policy.

One important aspect of access control model analysis is to formally check general properties of access control. In formal verification, a formal specification of a system is proven with a set of higher-level properties that the system should satisfy [35]. Currently, formal verification offers a rich toolbox containing a variety of techniques such as model checking [36], SAT solving [37] and theorem proving [38], for supporting automatic system verification. Schaad and Moffett [17] specified the access control policies under the RBAC96 and ARBAC97 model and a set of separation of duty constraints in Alloy. They attempted to check the con-

straint violations caused by administrative operations. In [39], Shafig et al. explored a Petri-Net based framework to verify the correctness of event-driven RBAC policies in real-time system. Toahchoodee et al. [40] demonstrated how the spatio-temporal aspects in RBAC model can be verified with Alloy. Alloy is also adopted to analyze the formal specifications of an RBAC model and constraints, which are then used for access control system development. In addition, the verified specifications are used to automatically derive the test cases for conformance testing. In [18], Shor et al. demonstrated how the USE tool, a validation tool for OCL constraints, can be utilized to validate authorization constraints against RBAC configurations. The policy designers can employ the USE-based approach to detect certain conflicts between authorization constraints and to identify missing constraints. However, the USE tool mainly focuses on the analysis of OCL constraints and has limitations for specifying models and policies. Similarly, Basin et al. [19] showed security-design models represented with UML and OCL can be validated based on system scenarios, which represent possible run-time instances of systems.

In conformance testing [41], an actual implementation of a system is compared with its specification by means of interactions between the implementation and test cases. The most significant recent development in testing is the application of verification approach which generates test cases from the formal specifications [42, 43]. However, very few studies addressed how access control mechanisms could be tested. Recently, mutation analysis was applied to security policy testing. Masood et al. [44] used formal techniques to conceive a fault model and adopt mutation for RBAC models. Xie et al. [45] proposed a fault model for XACML policies. The mutation operators were introduced to implement the fault model. Pretschner et al. [46] also used mutation analysis and defined security policy mutation operators in order to improve the security tests. However, no existing work

could adopt formal verification technologies for test case generation for the purpose of checking the compliance of access control system design and implementation.

2.2 Representation and Analysis of Access Control Policy

Many research efforts have been devoted to the policy representation and analysis. I only overview some work closely related to this dissertation, mainly focusing on representing and analyzing XACML policies.

Representing and Reasoning about Access Control Policy

In [47], a framework for automated verification of access control policies based on relational first-order logic was proposed. The authors demonstrated how XACML policies can be translated to the Alloy language [48], and checked their security properties using the Alloy Analyzer. However, using the first-order constructs of Alloy to model XACML policies is expensive and still needs to examine its feasibility for larger size of policies. In [49], the authors formalized XACML policies using a process algebra known as Communicating Sequential Processes. This utilizes a model checker to formally verify properties of policies, and to compare access control policies with each other. Fisler et al. [20] introduced an approach to represent XACML policies with Multi-Terminal Binary Decision Diagrams (MTBDDs). A policy analysis tool called Margrave was developed. Margrave can verify XACML policies against the given properties and perform change-impact analysis based on the semantic differences between the MTBDDs representing the policies. Kolovski et al. [21] presented a formalization of XACML using description logic (DL), which is a family of languages that are decidable subsets of first-order logic, and leveraged existing DL reasoners to conduct policy verification. However, existing work could only address part of XACML *combining algorithms*. Also, none of them could handle *conditions* represented in XACML policies.

Several work presented policy analysis tools with the goal of discovering policy anomalies in firewall [22, 23, 24, 50, 51]. However, we cannot directly apply those analysis approaches for XACML due to several reasons. First, the structure of firewall policies is flat but XACML has a hierarchical structure supporting recursive policy specification. Second, a firewall policy only supports one conflict resolution strategy (*first-match*) but XACML has four rule/policy combining algorithms. Last but not the least, a firewall rule is typically specified with fixed fields, while an XACML rule can be multi-valued.

Some XACML policy evaluation engines, such as *Sun* PDP [52] and XEngine [53], have been developed to handle the process of evaluating whether a request satisfies an XACML policy. During the process of policy enforcement, conflicts can be checked if a request matches multiple rules having different effects, and then conflicts are resolved by applying predefined combining algorithms in the policy.

Some work addressed the general conflict resolution mechanisms for access control [54, 55, 56, 57, 58]. Especially, Li et al. [56] proposed a policy combining language `PCL`, which can be utilized to specify a variety of user-defined combining algorithms for XACML. In addition, Bauer et al. [59] adopted a data-mining technique to eliminate *inconsistencies* between access control policies and user's intentions.

Other related work includes XACML policy integration [60, 61] and XACML policy optimization [62]. Since anomaly discovery and resolution are challenging issues in policy integration and redundancy elimination can contribute in policy optimization, all of those related work are orthogonal to this work.

Visualization-based Policy Representation

There are several interfaces that have been developed to assist users in creating and manipulating security policies. Expandable Grid is a tool for viewing and authoring access control policies [63]. The representation in Expandable Grids is a matrix with subjects shown along the rows, resources shown along the columns, and effective accesses for the combinations of subjects and resources in the matrix cells. The SPARCLE Policy Workbench allows policy authors to construct policies in a natural language interface, which are in turn translated into machine-readable policies [64]. Even though these interfaces are useful for authoring access control policies, they cannot effectively represent the results of policy analysis. Moreover, visualization has been widely used in the security arena for better understanding and presenting data related to network attacks [65, 66], intrusion detection [67, 68], and trust negotiations [69]. However, it is rarely adopted for security policy analysis.

2.3 Access Control for Online Social Networks

Access control for OSNs is still a relatively new research area. Several proposals of an access control scheme for OSNs have been introduced (e.g., [28, 29, 30, 31, 70]). Carminati et al. [28] introduced a trust-based access control mechanism, which allows the specification of access rules for online resources where authorized users are denoted in terms of the relationship type, depth, and trust level between users in OSNs. They further presented a semi-decentralized discretionary access control system and a related enforcement mechanism for controlled sharing of information in OSNs [29]. Fong et al. [31] proposed an access control model that formalizes and generalizes the access control mechanism implemented in Facebook. Gates [71] described relationship-based access control as one of the new security paradigms that addresses the requirements of the Web 2.0. Then, Fong [30] recently formulated this

paradigm called a Relationship-Based Access Control (ReBAC) that bases authorization decisions on the relationships between the resource owner and the resource accessor in an OSN. However, none of these work could accommodate privacy control requirements with respect to the *collaborative* data sharing in OSNs.

Recently, semantic web technologies have been used to model and express fine-grained access control policies for OSNs (e.g., [72, 73, 74]). Especially, Carminati et al. [72] proposed a semantic web based framework for social network access control. Three types of policies are defined in this framework, including authorization policy, filtering policy and admin policy, which are modeled with the Web Ontology Language (OWL) and the Semantic Web Rule Language (SWRL). Access control policies regulate how resources can be accessed by the participants; filtering policies specify how resources have to be filtered out when a user fetches an OSN page; and admin policies can determine who is authorized to specify policies. Although they claimed that flexible admin policies are needed to bring the system to a scenario where several access control policies specified by distinct users can be applied to the same resource, however, lack of formal descriptions and concrete implementation of the proposed approach leaves behind the ambiguities of their solution.

Several recent work [75, 76, 77, 78, 79] recognized the need of joint management for data sharing, especially photo sharing, in OSNs. In particular, Squicciarini et al. [78] proposed a solution for collective privacy management for photo sharing in OSNs. This work considered the privacy control of a content that is co-owned by multiple users in an OSN, such that each co-owner may separately specify her/his own privacy preference for the shared content. The Clarke-Tax mechanism was adopted to enable the collective enforcement for shared content. Game theory was applied to evaluate the scheme. However, a general drawback of this solution

is the usability issue, as it could be very hard for ordinary OSN users to comprehend the Clarke-Tax mechanism and specify appropriate bid values for auctions. In addition, the auction process adopted in their approach indicates only the winning bids could determine who was able to access the data, instead of accommodating all stakeholders' privacy preferences.

Measuring privacy risk in OSNs has been addressed recently by several work [80, 81, 82]. Becker et al. [80] presented *PrivAware*, a tool to detect and report unintended information loss through quantifying privacy risk associated with friend relationship in OSNs. In [82], Talukder et al. discussed a privacy protection tool, called *Privometer*, which can measure the risk of potential privacy leakage cased by malicious applications installed in the user's friend profiles and suggest self-sanitization actions to lessen this leakage accordingly. Liu et al. [81] proposed a framework to compute the privacy score of a user, indicating the user's potential risk caused by her/his participation in OSNs. Their solution also focused on the privacy settings of users with respect to their profile items.

Chapter 3

Background Information

This chapter gives a brief introduction to several relevant technologies.

3.1 Role-Based Access Control Standard

RBAC standard was proposed by National Institute of Standards and Technologies (NIST) in 2001 [2] and formally adopted as an ANSI standard in 2004 [3]. The NIST/ANSI RBAC standard is composed of two parts: RBAC *Reference Model* and RBAC *System and Administrative Functional Specification*. The reference model defines sets of basic RBAC elements and relations, such as a set of roles, a set of users, a set of permissions, and relationships between users, roles, and permissions. The system and administrative functional specification identifies all necessary functionalities required by role-based systems. These functionalities are divided into three categories: administrative operations, administrative reviews, and supporting system functions. In addition, the NIST/ANSI RBAC standard has four components: Core RBAC, Hierarchical RBAC, Static Separation of Duty (SSoD) relations, and Dynamic Separation of Duty (DSoD) relations. In this dissertation, I adopt this standard model as a basis for the model representation.

3.2 Unified Modeling Language and Object Constraint Language

Unified Modeling Language (UML) [83] is a general-purpose visual modeling language in which we can specify, visualize, and document the artifacts of software systems. It captures decisions and understanding about systems that must be constructed. UML has become a standard modeling language in the field of software engineering. UML defines notions for building many diagrams–such as use case diagram, class diagram, collaboration diagram and so on–to depict a particular view of a system. In this work, I focus on the class diagram and object diagram of UML.

A class diagram depicts a structural view of information in a system. Classes are defined in terms of their attributes and relationships. The relationships include association, generalization/specialization, and aggregation of classes. An object diagram is an instance of a class diagram. It shows the system states as a collection of objects at a particular point in time. In this work, I concentrate on class and object diagrams for representing RBAC model and system configuration, respectively.

The semi-formal semantics of UML has ambiguity and inconsistency issues [84]. Object Constraint Language (OCL) is a constraint expression language that enables users to describe constraints for UML-based models. OCL constraints typically specify restrictions that state conditions for all instances of the classes. In this work, I adopt OCL to specify RBAC policies. OCL offers a number of advantages over the use of UML diagrams for modeling software systems. First, OCL expressions make the definition of UML graphical models more consistent and precise. Second, OCL has a formal semantic based on mathematical set theory and predicate logic. Thus, it is possible to have OCL expressions ensure whether the model representations are correct and consistent with other elements of the model. Third, UML diagrams and OCL expressions can be integrated to support model-driven system development. As pointed out in [85], the OCL can be regarded as a key ingredient of UML-based model representation.

3.3 Role-Based Constraints Language 2000

Role-based Constraints Language 2000 (RCL2000) [86] is a formal specification language for RBAC policies and helps identify useful role-based authorization constraints such as prohibition, obligation and cardinality constraints. The users of RCL2000 are security policy designers who understand organizational objectives and articulate security policies to support these objectives. RCL2000 also provides *n-ary* expressions and more flexibility in expressing access control constraints [87].

RCL2000 has six entity sets called users (U), roles (R), permissions (P), sessions (S), objects (OBJ), and operations (OP). Additional elements used in RCL2000 are three conflicting sets CR, CP and CU. CR is defined as a collection of conflicting role sets; CP is denoted as a collection of conflicting permission sets; and CU is a collection of conflicting user sets. RCL2000 supports six RBAC system functions user, roles, sessions, permissions, operations and object. Also, RCL2000 defines two nondeterministic functions, OE (one element) and AO (all other). The OE(X) function allows users to get one element from a set X, and AO(X) is used to get a set by taking out one element. In this work, RCL2000 expressions are used to specify formal authorization policies derived from the NIST/ANSI RBAC standard.

3.4 Alloy

Alloy [88] is a structural modeling language based on first-order logic, and designed for the specification of object models through graphical and textual structure. An Alloy model is a structured specification composed with the following components: Signature, Fact, Function, Predicate and Assertion. The Alloy Analyzer [25] is an automated constraint solver for analyzing (verifying and validating) models written in Alloy. Alloy Analyzer provides two kinds of automatic analysis–*simulation* in which the consistency of a fact or predicate is demonstrated by generating a snapshot of the model; and *checking* in which a consequence of the specification is tested by attempting to generate a counterexample for an assertion. The former is useful for demonstrating the feasibility of a specification, where conflicting constraints could be detected, while the latter is for validating the correctness of a certain property in a system, where the assertion could be proved based on the facts defined in the model and within a finite scope of instances.

3.5 Extensible Access Control Markup Language

Extensible Access Control Markup Language (XACML) [6] has become the *de facto* standard for describing access control policies and offers a large set of built-in functions, data types, combining algorithms, and standard profiles for defining application-specific features. At the root of all XACML policies is a *policy* or a *policy set*. A *policy set* is composed of a sequence of *policies* or other *policy sets* along with a *policy combining algorithm* and a *target*. A *policy* represents a single access control policy expressed through a *target*, a set of *rules* and a *rule combining algorithm*. The *target* defines a set of subjects, resources and actions the policy or policy set applies to. For an applicable policy or policy set, the corresponding target should be evaluated to be *true*; otherwise, the policy or policy set is skipped when evaluating an access request. A *rule set* is a sequence of rules. Each *rule* consists of a *target*, a *condition*, and an *effect*. The *target* of a rule decides whether an access request is applicable to the rule and it has a similar structure as the target of a policy or a policy set; the *condition* is a boolean expression to specify restrictions on the attributes in the target and refine the applicability of the rule; and the *effect* is either `permit` or `deny`. If an access request satisfies both the *target* and *condition* of a rule, the response is sent with the decision specified by the *effect* element in the rule. Otherwise, the response yields `NotApplicable` which is typically considered as `deny`.

3.6 Answer Set Programming

Answer set programming (ASP) [26, 27] is a recent form of declarative programming that has emerged from the interaction between two lines of research — nonmonotonic semantics of negation in logic programming and applications of satisfiability solvers to search problems. The idea of ASP is to represent the search problem we are interested in as a logic program whose intended models, called

"stable models (a.k.a. answer sets)," correspond to the solutions of the problem, and then find these models using an answer set solver — a system for computing stable models. Like other declarative computing paradigms, such as SAT (Satisfiability Checking) and CP (Constraint Programming), ASP provides a common basis for formalizing and solving various problems, but is distinct from others such that it focuses on knowledge representation and reasoning: its language is an expressive nonmonotonic language based on logic programs under the stable model semantics [89, 90], which allows elegant representation of several aspects of knowledge such as causality, defaults, and incomplete information, and provides compact encoding of complex problems that cannot be translated into SAT and CP [91].

As the mathematical foundation of answer set programming, the stable model semantics was originated from understanding the meaning of *negation as failure* in Prolog, which has the rules of the form

$$a_1 \leftarrow a_2, \ldots, a_m, not\ a_{m+1}, \ldots, not\ a_n \tag{3.1}$$

where all a_i are atoms and *not* is a symbol for *negation as failure*, also known as *default negation*. Intuitively, under the stable model semantics, rule (3.1) means that if you have generated $a_2, \ldots, a_m$ and it is impossible to generate any of $a_{m+1}, \ldots, a_n$ then you may generate a_1. This explanation seems to contain a vicious cycle, but the semantics are carefully defined in terms of fixpoint.

While it is known that the transitive closure (e.g., reachability) cannot be expressed in first-order logic, it can be handled in the stable model semantics. Given the fixed extent of *edge* relation, the extent of *reachable* is the transitive closure of *edge*.

$$reachable(X,Y) \leftarrow edge(X,Y).$$
$$reachable(X,Y) \leftarrow reachable(X,Z), reachable(Z,Y).$$

Several extensions were made over the last twenty years. The addition of cardinality constraints turns out to be useful in knowledge representation. A cardinality constraint is of the form $lower\{l_1,\ldots,l_n\}upper$ where $l_1,\ldots,l_n$ are literals and $lower$ and $upper$ are numbers. A cardinality constraint is satisfied if the number of satisfied literals in $l_1,\ldots,l_n$ is in between $lower$ and $upper$. It is also allowed to contain variables in cardinality constraints. For instance,

$$more\_than\_one\_edge(X) \leftarrow 2\{edge(X,Y) : vertex(Y)\}.$$

means that $more\_than\_one\_edge(X)$ is true if there are at least two edges connect X with other vertices.

The language also supports choice, counting and aggregates, such as *count*, *sum*, *min*, and *max*. For instance, the following rule uses an aggregate to represent the policy that each referee should be assigned at least 3 proposals but no more than 8 proposals:

$$3 \leq \{assign(R,P) : proposal(P)\} \leq 8 \leftarrow referee(R).$$

The language also has useful constructs, such as strong negations, weak constraints, and preferences. What distinguishes ASP from other nonmonotonic formalisms is the availability of several efficient implementations, answer set solvers, such as Smodels[1], Cmodels[2], Clasp[3], which led to practical nonmonotonic reasoning that can be applied to various level applications.

[1]http://www.tcs.hut.fi/Software/smodels.

[2]http://www.cs.utexas.edu/users/tag/cmodels.html.

[3] http://potassco.sourceforge.net.

Chapter 4

Assurance Management Framework (AMF)

In this chapter, I first give a brief overview of the assurance management framework. I then discuss approaches in AMF for the realization and analysis of access control models, and the analysis and management of access control policies.

4.1 Overview

The AMF framework depicted in Figure 4.1 is designed to manage the assurance of access control systems with respect to both access control model and access control policy. Regarding access control model, the AMF framework enables realization and analysis of formal access control models in secure system development. Model verification and model testing for access control are articulated in this framework, where the formal specifications of access control models are verified, and test cases

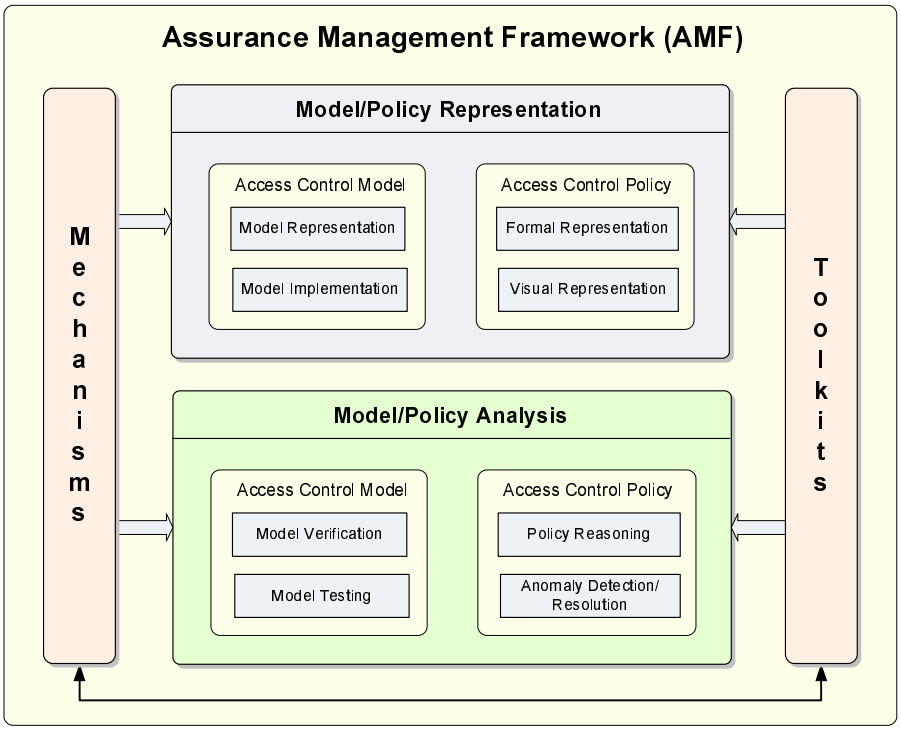


Figure 4.1: Assurance management framework.

are derived from the formal specifications automatically. The generated test cases are in turn used to validate whether the secure system design and implementation conform to the formal specifications. From the perspective of access control policy, the AMF framework ensures the correctness of access control policies for policy-based computing through automated reasoning techniques, and anomaly detection and resolution mechanisms. A logic-based reasoning approach is adopted for access control policies that allows users to leverage the features of logic solvers in performing various logical reasoning and analysis tasks. In addition, this framework contains a comprehensive anomaly detection and resolution mechanism integrated with a visualization-based policy representation that facilitates systematic and effective detection and resolution of access control policy anomalies. Finally, a suite of tools should be developed to support all features addressed in the AMF framework.

4.2 Access Control Model

I first address the processes for realizing and analyzing access control models in the AMF framework, which is shown in Figure 4.2. In the modeling stage, formal specifications of access control models and constraints are verified. Additionally, application-oriented authorization model representation and constraint specification are derived from the formal specifications of access control models and constraints, which can also be utilized to produce test cases. Then, the generated test cases are used to validate the application-oriented models and constraints. In the implementation stage, authorization enforcement codes are generated systematically from the application-oriented specifications. The correctness and conformance of generated codes are also evaluated by using the generated test suites. I divide all tasks into two categories as follows:

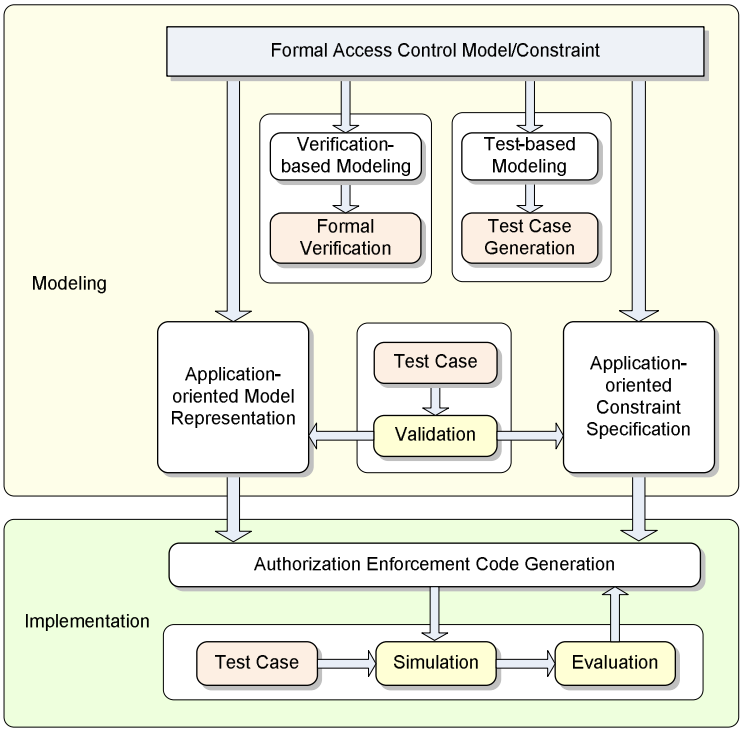


Figure 4.2: Realization and analysis of access control model.

1. Realization of access control model

   - *Application-oriented representation of access control model and constraint.* The representation of an access control model and corresponding constraints should enable software engineers to integrate security aspects into the applications without knowing details of the access control model. In this regard, a well-designed and general-purpose representation should be considered as a means to represent access control models and constraints in an intuitive fashion.

   - *Automatic generation of access control enforcement code.* It is also a crucial aspect to make the transparent transition from system design to secure system implementation. The goal of code generation in AMF is to automatically generate executable modules from the application-

oriented specifications of access control models and constraints by a well-known software engineering mechanism, such as the Model Driven Development (MDD) [92]. The generated authorization modules would be eventually integrated into the real systems to achieve an acceptable degree of assurance in secure system development.

2. Analysis of access control model

   - *Verification of formal access control model.* One of promising advantages in mathematical and logic-based techniques for access control models is that formal reasoning of the authorization properties can be performed. Since the formal access control models serve as a basis for secure system development in AMF, obviously the formal specifications of models should be proved based on the expected authorization properties.
   - *Automatic test case generation from formal specification.* While formal verification can prove violation or satisfaction of properties, it is not sufficient enough to practically guarantee the assurance. The proof only shows that a given formal specification fulfills a set of properties. However, we should consider the actual implementation is influenced by other facts, such as platforms, transformation approaches, compilers, and so on. Consequently, the implemented modules should be further tested.

Analysis Approach in AMF for Access Control Model

I introduce a methodology composing formal analysis and conformance testing for building access control systems. As demonstrated in Figure 4.2, the formal access control model serves as the core of following tasks: (1) formal verification, (2) sys-

tem design, and (3) test generation. In this work, both model verification and model testing are supported by the same formal verification technology for the purposes of automatic verification and test case generation. A notable advantage of using model-based approach is to reduce the complexity of analysis, thus minimizing the state explosion problem.

In order to articulate the methodology clearly, I first define access control model specification as follows:

Definition 1 *(**Access Control Model Specification**). An access control model specification M is defined as $M = (O, F, C)$, where*

- *O is the component representation of an access control model, which defines sets of basic access control entities and relations;*
- *F is a set of access control function specifications, which specify the features required by an access control system; and*
- *C is a set of access control constraint specifications, which define higher-level organizational policies.*

Access Control Model Verification

I take into account the following verification problem for access control models: given an access control model specification M and an access control model property P, does M satisfy P? I consider two kinds of property, access control functional property P_f and access control authorization property P_a. Therefore, the verification of an access control model is decomposed into two steps, access control function verification and access control constraint verification.

Definition 2 *(**Access Control Function Verification**). For an access control model specification $M = (O, F, C)$ and an access control functional property P_f, proving*

whether M satisfies P_f, denoted by $M \models P_f$, is called access control function verification.

That is, if we can determine that M satisfies P_f, it means the access control functional property P_f is held on the access control model specification M. Hence, we can make sure the functional components in a formal model specification M are correct with respect to expected properties.

Figure 4.3 illustrates a reasoning process for the formal verification. The access control model specification M and the functional property P_f are encoded and then fed into a formal verifier. The verifier in turn checks whether the functional property is violated or not. If a functional property violation is encountered, it means the access control model specification does not conform to the functional property, leading the refinement of model specification.

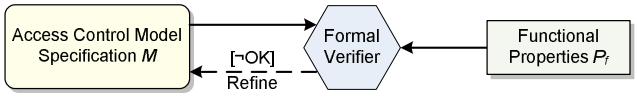


Figure 4.3: Function verification.

A critical task for specifying constraints is to determine whether a set of constraint expressions really reflects the desired authorization requirements properly. Normally, constraints prohibit an action or state occurring in the system. Two issues should be considered carefully while analyzing a given set of constraints against the expected authorization requirements. First, constraints may be too weak, named *under-constraint* to grant undesired system states. A *safety* problem (i.e. the leakage of a right to an unauthorized user) can be resulted from the weak constraints. Second, constraints may be too strong, named *over-constraint* to deny desired system states. Strong constraints can cause *availability* problems. For example, an entitled user cannot own the right to access a resource.

A concept of *authorization state space* is introduced to identify under- and over-constraints in an access control model specification. An authorization state space represents an entire space that an access control system probably covers. In other words, all possible system states of an access control system consist of an authorization state space. Regarding access control requirements, an authorization state space can be divided into two subspaces: (1) the *desired* authorization state subspace S_d, which contains authorization states that should be allowed to occur in an access control system according to the authorization requirements. (2) the *undesired* authorization state subspace S_u, which contains authorization states that should be prohibited to appear in an access control system against the authorization requirements. On one hand, we are able to specify the desired authorization state subspace with the expected authorization properties P_{a^+} and the undesired authorization state subspace with the unexpected authorization properties P_{a^-}, respectively. On the other hand, from the perspective of access control specification, an authorization state space can be divided into permitted authorization state subspace S_p and prohibited/constrained authorization state subspace S_c. The most ideal view of an authorization state space is that the desired authorization state subspace is contained by the permitted authorization state subspace, and the undesired authorization state subspace is included in the prohibited authorization state subspace. It means the specified constraints meet the authorization requirements accordingly. Unfortunately, the ideal view is far from the reality. Two situations may exist in practice.

Figure 4.4 depicts one case, which demonstrates that the permitted authorization state subspace S_p covers *partial* undesired authorization state subspace S_u due to the reason of *under-constraint*.

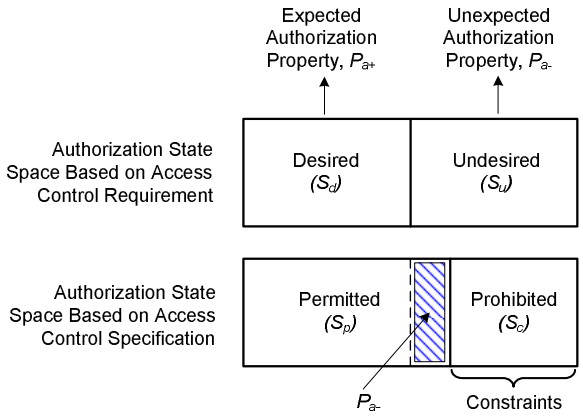


Figure 4.4: Identifying under-constraint.

When the prohibited authorization state subspace S_c is a subset of the undesired authorization state subspace S_u, and there is an overlap between the permitted authorization state subspace S_p and the undesired authorization state subspace S_u, *under-constraint* occurs in the constraint specifications.

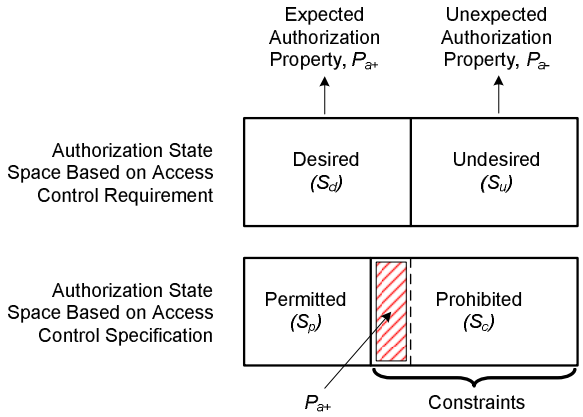


Figure 4.5: Identifying over-constraint.

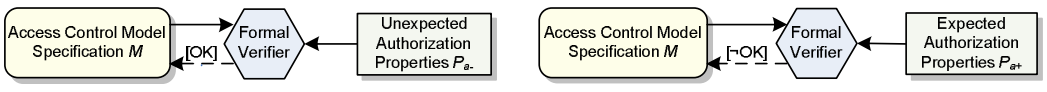
Another case is shown in Figure 4.5. *Over-constraint* is presented in this case, where the permitted authorization state subspace S_p is covered by the desired authorization state subspace S_d, and the prohibited authorization state subspace S_c contains *partial* desired authorization state subspace S_d.

Using formal verification, both *over-* and *under-constraints* for an access control model specification are analyzed automatically with a set of given access control properties. A general definition for access control constraint verification is given as follows:

Definition 3 *(**Access Control Constraint Verification**). For an access control model specification $M = (O, F, C)$ and an access control authorization property P_a, proving whether M satisfies P_a, denoted by $M \models P_a$, is called access control constraint verification.*

In order to identify *under-constraint*, the unexpected authorization property P_{a^-} is used to replace P_a, and an expression that addresses the analysis for *under-constraint* can be defined as follows: $M \models P_{a^-} \Rightarrow C\downarrow$, where C$\downarrow$ denotes *under-constraint*.

As demonstrated on the bottom part of the Figure 4.4, if an unexpected authorization property P_{a^-}, which represents the authorization subspace $S_p \cap S_u$, is satisfied by the access control model specification S_c, *under-constraint* is detected. Figure 4.6 (a) depicts the processes of constraint verification for determining *under-constraint*. If the verifier proves an unexpected authorization property P_{a^-} is held on the access control model specification M, we conclude that the given constraint specifications are too weak, and should be strengthened to exclude undesired authorization properties or contain required authorization properties.



(a) Verifying under-constraint (b) Verifying over-constraint

Figure 4.6: Constraint verification.

The expected authorization property P_{a^+} is utilized to substitute P_a for identifying *over-constraint* as summarized in the following expression: $M \not\models P_{a^+} \Rightarrow C\uparrow$, where C↑ denotes *over-constraint*.

The bottom part of the Figure 4.5 depicts the *over-constraint* situation. Based on the expression, I introduce processes for identifying *over-constraint* as shown in Figure 4.6 (b). If the verifier checks the expected authorization property P_{a^+} is not satisfied by access control model specification M, this points out the defined constraints are too strong. Thus, the constraint definitions should be refined by reducing the restriction of constraints.

Access Control Model Testing

Model-based testing is a software testing method in which the models defined in software construction are used to drive the testing process. Numerous formal verification techniques have been used for model-based testing [93]. The idea of automated test generation from the formal verification is that counterexamples may be generated to illustrate a property violation by the formal verification, and counterexamples are interpreted as test cases. This work intends to use a formal specification of access control model and constraint to automatically derive test cases for testing authorization constraints in access control systems.

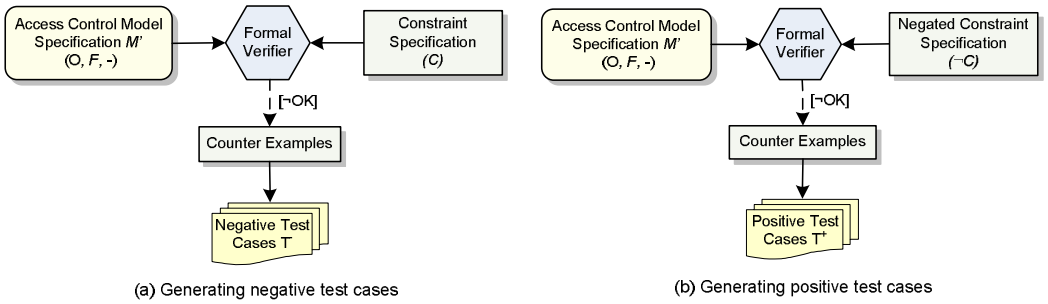


Figure 4.7: Test case generation for constraints.

Two kinds of test cases are generated: one is called *negative test case*, denoted as T^-, which is considered as an undesired access control authorization state that should be denied by the constraints in an access control system. Another test case is named *positive test case*, denoted as T^+. This test case represents a desired access control authorization state, which should be allowed to appear in an access control system.

The following expression specifies the generation of *negative test case* based on the satisfiability verification: $(O,F,-) \not\models C \Rightarrow T^-$. *Negative test case* T^- can be derived from a formal specification, in which an access control model specification $M^{'} = (O,F,-)$ does not satisfy the constraint specification C as demonstrated in Figure 4.7 (a). Since the constraint specification C is taken out from the access control model specification $M^{'}$, the authorization property expressed by constraint specification is not exactly held on the access control model specification. The verifier can generate counterexamples, which are then used to construct negative test cases.

Positive test case T^+ is generated from a formal specification, as we draw the constraint specification C from the access control model specification $M^{'} = (O,F,-)$, and take the negated constraint specification $\neg C$ as the authorization property to verify the access control model specification $M^{'}$. Counterexamples are derived and utilized to build positive test cases. The following expression summarizes this characteristic: $(O,F,-) \not\models \neg C \Rightarrow T^+$. Corresponding process is shown in Figure 4.7 (b).

Realization and Analysis of RBAC Model

I demonstrate the feasibility of the proposed approach for constructing role-based access control systems, leveraging the NIST/ANSI RBAC standard [2, 3] as the underlying authorization model since it includes most of RBAC features [1] and

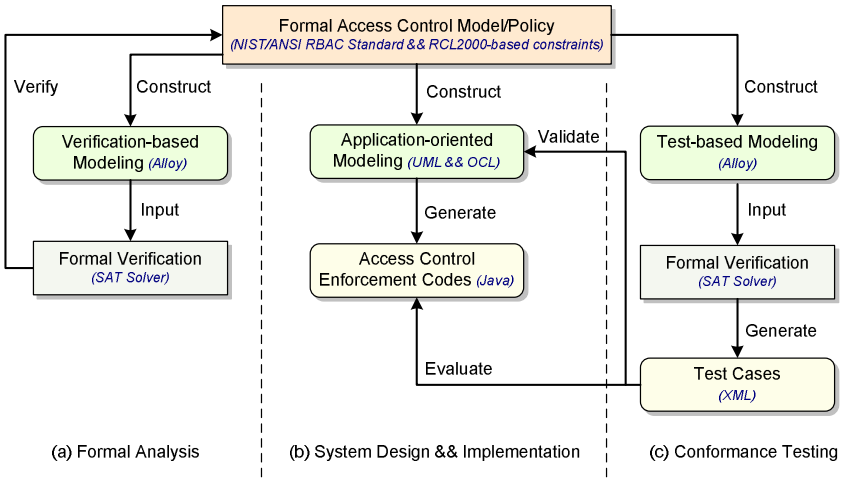


Figure 4.8: Realization and analysis of RBAC model with AMF.

has been widely adopted in information assurance community. Role-based Constraints Language 2000 (RCL2000) [86] is utilized to define authorization constraints formally in RBAC. As demonstrated in Figure 4.8, the formal access control model and constraints are the core of the entire processes for serving the following tasks: (1) formal verification, (2) system design, and (3) test generation. Correspondingly, three high-level access control models–such as verification-based model, application-oriented model, and test-based model–are constructed based on the formal model. In this approach, the formal access control model and associated constraints can be fully translated to application-oriented model representation and constraint specification, which then generate enforcement codes. Also, both model-based verification and model-based testing are supported by the same formal verification technology for the purposes of automatic verification and test case generation.

Realization of RBAC Model

For building an access control system based on a particular access control model, it is very important to have an application-oriented representation of the access control model for software engineers. UML is the standard language in modeling

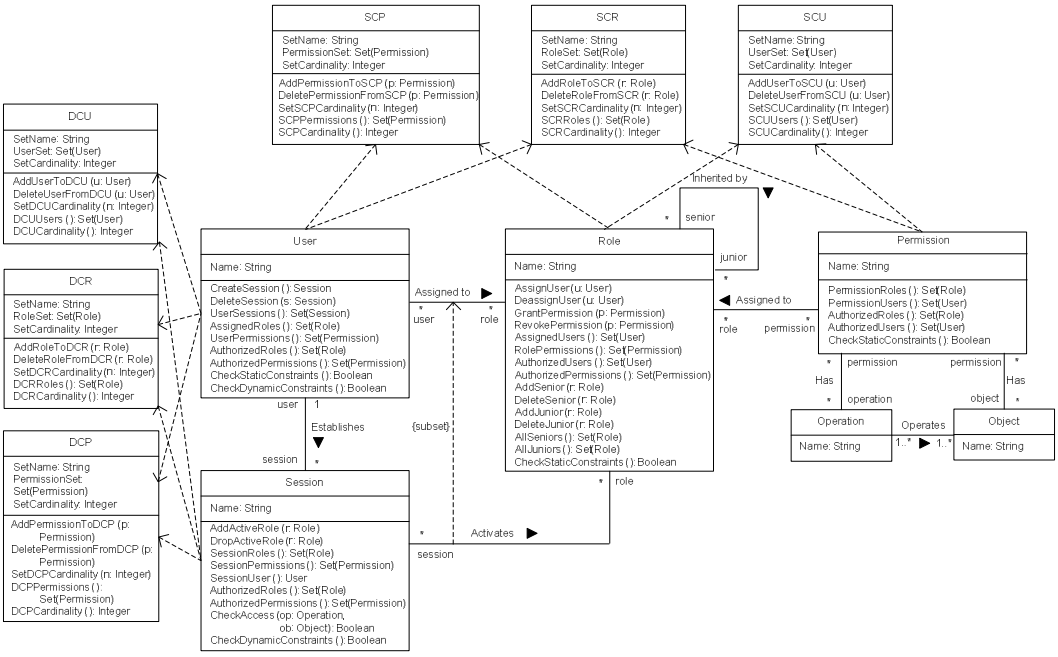


Figure 4.9: RBAC model representation in UML class diagram.

community and the usage of UML for the representation of security models has been recommended [94]. Furthermore, in order to make RCL2000 expressions more meaningful to ordinary system developers, this method translates RCL2000-based constraints into OCL specifications which are closely coupled with UML. Then UML-based modeling and OCL specifications are facilitated to *automatically* generate system modules, called RBAC enforcement codes which can be deployed in an RBAC-centric system implementation.

RBAC Model Representation in UML and OCL The NIST/ANSI RBAC standard defines three models: Core RBAC, Hierarchical RBAC and Constrained RBAC. The Constrained RBAC in the standard adds separation of duty relations. However, there are two limitations in the Constrained RBAC model. First, the *SoD* constraints in the standard are applied only to the activation of roles without considering other components in RBAC model. Second, the standard defines *Static SoD (SSoD)* relations with respect to user-role assignments over pairs of roles and *Dynamic SoD*

(DSoD) relations with the aspect of role activation in a user's session. These two constraints in the standard mainly reflect the simplest separation of duty properties. More fine-grained constraints cannot be defined adequately. Thus special constraint specification languages are desirable to provide much richer expression for RBAC constraints. To reduce these limitations, I extend the Constrained RBAC to consider all aspects of role-based constraints and specify separation of duty constraints with RCL2000 where a variety of separation of duty properties can be expressed.

Figure 4.9 shows a UML class diagram which depicts a complete representation of the NIST/ANSI RBAC model including Core RBAC, Hierarchical RBAC and Constrained RBAC. The representation can be decomposed to partially support Core RBAC or different compositions of three reference models. It contains classes, relationships between classes, and cardinalities in relationships. The basic entities are user, role, permission, and session classes. The permission class is represented as a composition of operation and object classes. The role hierarchy relationship is reflected in role class as a recursive relationship. The standard RBAC model only supports two separation of duty relations: *SSoD* and *DSoD* relations. As discussed above, constraints should be applied to all RBAC entities. Thus, in the model representation, I introduce two components such as SCR (Static Conflicting Roles) and DCR (Dynamic Conflicting Roles) that support *SSoD* and *DSoD* relations in the standard. Four new components SCP, DCP, SCU and DCU are created to support constraints in other RBAC entities such as permissions and users. These six components have dependency relationships with corresponding RBAC components in UML class diagram and are utilized by constraint expressions to identify more fine-grained *SoD* constraints.

The functional specification in the standard defines various functions that role-based systems should provide. Since I use an object-oriented approach to ex-

press these functionalities of the standard, some subtle changes must be conducted in the definition of each function. For example, the RBAC standard defines two functions, `AddInheritance` and `AddAscendant`, to support building a new role inheritance relationship in role hierarchy. The standard explains `AddInheritance` is used to establish a new immediate inheritance relationship between two existing roles and `AddAscendant` is used to create a new role and to add this new roles as an immediate ascendant of an existing role. In object-oriented system design, every function is attached to a class. Therefore, the function `AddInheritance` cannot be used as a single class (`role` class in the model representation). On the other hand, since `CreateRole` function can be implicitly derived from `role` class in the UML class diagram at the implementation stage, `AddAscendant` may not include the operation for creating a new role. In the model representation, I define a function named `AddSenior` which adds an immediate senior role object to current role object instead of adopting two functions proposed in the standard. For role hierarchy, I also add two review functions `AllSeniors` and `AllJuniors` to query all seniors and juniors of a role object, because these two functions can frequently be called by many other functions, as well as by constraints in the presence of role hierarchy. Similarly, several review functions related to role hierarchy are added into the model representation. For example, two review functions, `AuthorizedRoles` and `AuthorizedUsers`, for a permission to find a set of roles that authorize the given permission and to get a set of users that can authorize the given permission through their roles, respectively. For brevity, I elaborate a few typical functional definitions of three components in the standard and corresponding OCL-based definitions.

A. Functional definition of Core RBAC

Administrative commands: These commands are for the creation and maintenance of RBAC element sets and relations by administrators. The func-

tions for adding and deleting an element such as `AddRole` and `DeleteRole` can be created from UML class diagrams in the implementation step. A command specification for `AssignUser` is defined with OCL as follow:

context `Role::AssignUser(u:User)`

pre : `self.user->excludes(u)`

post: `self.user->includes(u)`

Deassigning a user is symmetric to adding a user, the following is the definition of `DeassignUser`. We can also define `GrantPermission` and `RevokePermission` in the same way.

context `Role::DeassignUser(u:User)`

pre : `self.user->includes(u)`

post: `self.user->excludes(u)`

Review functions: These functions are for administrators to query RBAC element sets and relations. Query operations do not change system states and they return a value or a set of values. In OCL, they are defined as a `body` expression. The following OCL definition supports a review function `AssignedUsers`:

context `Role::AssignedUsers():Set(User)`

body: `self.user->asSet()`

Similarly, the definition of `UserPermissions` with OCL as follows:

context `User::UserPermissions():Set(Permission)`

body: `self.role.permission->asSet()`

Supporting system functions: The functions are applied to create and maintain RBAC dynamic properties with regard to users' sessions and access control decisions. `CreateSession` creates a session for a user. The specification is below:

context `User::CreateSession():Session`

post: `result.oclIsNew() and self.session->includes(result)`

`CheckAccess` checks whether an operation on an object is allowed to be performed in a particular session. OCL representation for this function is defined as follows:

context `Session::CheckAccess(op:Operation, ob:Object):Boolean`

pre : `true`

post: `self.SessionRoles()->exists(r|r.permission`

`->exits(p|p.operation->includes(op)`

`and p.object->includes(ob)))`

B. Functional definition of Hierarchical RBAC

As illustrated in Figure 4.9, we have four major functions such as `AddSenior`, `DeleteSenior`, `AddJunior` and `DeleteJunior`, which are used for administrators to maintain inheritance relationships among roles. I define `AddSeniors` in OCL as follows:

context `Role::AddSeniors(r:Role)`

pre : `self.senior->excludes(r)`

post: `self.senior->includes(r)`

I define two new review functions `AllSeniors` and `AllJuniors` for role hierarchy. The following definition is for `AllSeniors`, which is a recursive definition with OCL.

context `Role::AllSeniors():Set(Role)`

body: `self.senior->union(self.senior->`

`collect(r|r.AllSeniors()))->asSet()`

`AllSeniors` and `AllJuniors`, are very useful for other functions and constraints in presence of role hierarchy. The following is an example in applying `AllSeniors` to the definition of `AuthorizedRoles` for user component in RBAC.

context `User::AuthorizedRoles():Set(Role)`

body: `Role.allInstances()->select(r|`

`r.AllSeniors()->including(r)->`

`exists(r1|self.role->includes(r1)))`

Note that the definition of `AuthorizedRoles` should be carefully formulated to reflect the role inheritance with respect to user and session components using `AllSeniors` and with respect to permission component using `AllJuniors`. The definition of
`AuthorizedRoles` for permission is given as follows:

context `Permission::AuthorizedRoles():Set(Role)`

body: `Role.allInstances()->select(r|`

`r.AllJuniors()->including(r)->`

`exists(r1|self.role->includes(r1)))`

C. Functional definition of Constrained RBAC

In this approach, the definitions related to constraint expressions are incorporated with corresponding components in UML-based model representation. I introduce two new system functions `CheckStaticConstraints` and `CheckDynamicConstraints` for RBAC model to enforce constraint expressions and to check conflicts. The functions for constraint checking can be used by other related functions in RBAC model as well[1]. The following two definitions are for `AssignUser` and `AddActiveRole` functions, respectively.

```
context Role::AssignUser(u:User)
pre :  self.user->excludes(u)
post:  self.user->includes(u) and
      if (self.CheckStaticConstraints()->
       isEmpty()) or
       (u.CheckStaticConstraints()->isEmpty())
      then
       self.user->excludes(u)
      endif
```

In `AssignUser`, the assignment operation can affect the status of two objects, user and role objects. Hence, the static constraints for user and role classes need be enforced at the same time to prevent possible conflicts resulted from the assignment operation.

[1]For example, `AssignUser` and `GrantPermission` utilize `CheckStaticConstraints` to check the static assignment relations, and `CreateSession` and `AddActiveRole` employ `CheckDynamicConstraints` to check dynamic attributes related to sessions.

context `Session::AddActiveRole(r:Role)`

pre : `self.user.role->includes(r) and`

`self.role->excludes(r)`

Post: `self.role->includes(r) and`

```
if (self.CheckDynamicConstraints()->
 isEmpty()) or (self.user.
 CheckDynamicConstraints()->isEmpty())
then
 self.role->excludes(r)
endif
```

`AddActiveRole` can also bring the effect to both a session and a user who invokes the session. `CheckDynamicConstraints` for session and user classes should be performed in `AddActiveRole` to avoid possible violations of constraints.

RBAC Constraint Specification in OCL In NIST/ANSI RBAC standard, *SSoD* constraints are defined with two arguments: a role set `rs` that includes two or more roles, and a natural number n, called the cardinality, with the property that $2 \leq n \leq |rs|$ which means a user can be assigned to more than equal to two roles and fewer than the size of role set `rs`. The similar definition is used in *DSoD* constraints with respect to the activation of roles in sessions. The definition of constraints in the standard has limitations. To overcome such obstacles for considering other components in RBAC, I use RCL2000 that defines three sets, CR, CP and CU, as the collections for conflicting role sets, conflicting permission sets and conflicting user sets, respectively. Each conflicting set can include two or more elements. However,

there is no notation for the cardinality attribute in conflicting sets itself. Normally, we can regard the cardinality number *n* in RCL2000 always greater than equal to *two* for each conflicting set. In order to support a more general cardinality property defined in the standard, I extend the definition for CR, CP and CU in RCL2000 to support the cardinality attribute. In addition, two new functions, `GetSet` and `GetCardinality`, are defined in RCL2000 to allow to get the conflicting element set and the cardinality number. I usually write `GetSet` as `GS` and `GetCardinality` as `GC` in constraint expressions. As discussed before, I also extend CR to SCR and DCR, CP to SCP and DCP, and CU to SCP and DCP, to support *SSoD* constraints and *DSoD* constraints separately.

Policy designers can employ RCL2000 to specify complex authorization policies to meet high-level security requirements along with the NIST/ANSI RBAC standard. The next important step is that RCL2000 policy specifications need to be realized in UML-based RBAC representation which, in turn, we need to translate RCL2000 expressions to OCL expressions.

RCL2000 has two nondeterministic functions, `OE` and `AO`. The `OE(X)` function allows users to get one element from a set `X`. Multiple occurrences of `OE(X)` in a single RCL2000 expression all select the same element from a set `X`. With `AO(X)` we can get a set by taking out one element, thus we can express `AO(X)` as `X - {OE(X)}`. The `OE` function can be converted to an OCL expression with `any` operation. For example, `OE(X)` can be translated to `X->any(true)`. Then `AO(X)` can be correspondingly translated to `X-{X->any(true)}`.

RCL2000 supports six RBAC system functions `user`, `roles`, `sessions`, `permissions`, `operations` and `object`. These function expressions can be simply represented in OCL. For example, `roles(u)`, which returns all the roles assigned to the user `u`, can be converted to `u.role`. In RCL2000, `roles`* and `permissions`*

Input: *RCL2000* expression; **Output**: *OCL* expression

Let **Simple-OE** term be either **OE(*set*)** , or **OE(function(*element*))**, where ***set*** is an element of **{U, R, OP, OBJ, P, S, SCR, DCR, SCU, DCU, SCP, DCP, PR, scr, dcr, scu, dcu, scp, dcp, pr}** and ***function*** is an element of **{user, roles, roles*, sessions, permissions, permissions*, operations, object}**

1. AO elimination
  Replace all occurrences of *AO(expr)* with *(expr->{OE(expr)})*;
2. OE elimination
  While There exists **Simple-OE** term in RCL2000 expression
    case (i) **Simple-OE** term is *OE(set)*
      choose *set* term;
      call set_reduction procedure;
    case (ii) **Simple-OE** term is *OE(function(element))*
      choose *function(element)* term;
      call reg_function_reduction procedure;
  End
  Procedure set_reduction
    case (1) *set* is *U*
      put *u:User=User.allInstances->any(true)* to right of existing quantifier(s);
        replace all occurrences of *OE(U)* with *u*;
    case (2) *set* is *R*
      put *r:Role=Role.allInstances->any(true)* to right of existing quantifier(s);
        replace all occurrences of *OE(R)* with *r*;
  ……[a]
  End
  Procedure reg_funtion_reduction
    case (1) *function(element)* term is *user(r)*
      put *u1:User=r.user->any(true)* to right of existing quantifier(s);
      replace all occurrences of *OE(user(r))* with *u1*;
    case (2) *function(element)* term is *roles(u)*
      put *r1:Role=u.role->any(true)* to right of existing quantifier(s);
      replace all occurrences of *OE(roles(u))* with *r1*;
  ……
  End
3. System Functions elimination
  While There exists *function(element)* term in RCL2000 expression
    choose *function(element)* term;
    call sys_function_reduction procedure;
  End
  Procedure sys_function_reduction
    case (1) *function(element)* term is *user(element)*
      replace all occurrences of *user(element)* with *element.user*;
    case (2) *function(element)* term is *roles(element)*
      replace all occurrences of *roles(element)* with *element.role*;
  ……
  End
4. Operators elimination
  replace all occurrences of *set1* $\in$ *set2* with *set2->include(set1)* ;
  replace all occurrences of *set1* $\cap$ *set2* with *set1->intersection(set2)* ;
  replace all occurrences of *set1* $\cup$ *set2* with *set1->union(set2)* ;
  replace all occurrences of |*set*| with *set->size()* ;
  replace all occurrences of $\Rightarrow$ with *implies* ;
  replace all occurrences of $\wedge$ with *and* ;
  replace all occurrences of $\neq$ with <> ;
  replace all occurrences of $\leq$ with <= ;
  replace all occurrences of $\geq$ with >= ;
  replace all occurrences of $\emptyset$ with *{}* ;

[a] Due to the page limit, we omitted the detailed procedures for OE elimination and system function elimination.

Figure 4.10: Translation algorithm from RCL2000 to OCL.

are defined as a variant of `roles` and `permissions` to support role hierarchy. For example, $\texttt{roles}^*(\texttt{u})$ returns a set of roles for which a given user is authorized. In the previous section, I have defined several review functions. Using

such definitions, $\texttt{roles}^*$ and $\texttt{permissions}^*$ functions can be translated. For example, $\texttt{roles}^*(\texttt{u})$ and $\texttt{permission}^*(\texttt{r})$ are converted to `AuthorizedRoles(u)` and `AuthorizedPermissions(r)`.

Each term in RCL2000 is converted to corresponding OCL operator or function. For example, operators like $\neq$, $\leq$, $\geq$, $\Rightarrow$, and $\wedge$ are replaced by $<>$, $<=$, $>=$, `implies`, and `and` operations correspondingly; $\in$, $\cap$, and $\cup$ can be expressed by `include`, `intersection` and `union` functions in OCL respectively. The detailed translation algorithm is described in Figure 4.10.

Next, I illustrate two typical RBAC constraints specified in RCL2000, and give equivalent OCL expressions generated by the translation algorithm.

Constraint 1: (SSoD-CR): The number of conflicting roles, which are from the same conflicting role set, authorized to a user cannot exceed the cardinality number of the conflicting role set.

RCL2000 Expression:

$$|\,\texttt{roles}^*(\texttt{OE}(\texttt{U})) \cap \texttt{GS}(\texttt{OE}(\texttt{SCR}))\,| \leq \texttt{GC}(\texttt{OE}(\texttt{SCR}))$$

Translated OCL Expression:

```
context User
inv: let
       scr:SCR = SCR.allInstances()->any(true)
     in
       self.AuthorizedRoles()->intersection(scr.
         RoleSet)->size() <=scr.SetCardinality
```

Table 4.1: Mapping RCL2000 expression to OCL expression for *SSoD-CR*.

RCL2000	OCL	Meaning
OE(SCR)	scr:SCR = SCR. allInstances()->any(true)	a collection which is a pairs of a conflicting role set and a cardinality for the conflicting role set
OE(U)	self	a single user
roles*(OE(U))	self.AuthorizedRoles()	return all roles that are authorized to a single user considering role hierarchy
GS(OE(SCR))	scr.RoleSet	return a conflicting role set
GC(OE(SCR))	scr.SetCardinality	return the cardinality of a conflicting role set
$\cap$	intersection	return the intersection of two sets
$\mid set \mid$	*set*->size()	return the cardinality number of a set

Table 4.1 explains the mapping from the RCL2000 expression to the OCL expression for this constraint. All components in the RCL2000-based constraint expression can be mapped to corresponding OCL components precisely.

Constraint 2: (User-based DSoD):The number of conflicting roles, which are from the same conflicting role set, activated directly (or indirectly via inheritance) by a user cannot exceed the cardinality number of the conflicting role set.

RCL2000 Expression:

$$\mid \mathtt{roles}^{*}(\mathtt{sessions}(\mathtt{OE}(\mathtt{U}))) \cap \mathtt{GS}(\mathtt{OE}(\mathtt{DCR})) \mid \leq \mathtt{GC}(\mathtt{OE}(\mathtt{DCR}))$$

Translated OCL Expression:

context User

inv: let

```
dcr:DCR = DCR.allInstances()->any(true)
```

in

```
self.session.AuthorizedRoles()->
  intersection(dcr.RoleSet)->size()
  <=dcr.SetCardinality
```

Code Generation The code generation part of the approach enables users to build a real application by creating a platform independent model and then transforming it to platform dependent codes. The objective for code generation is to generate security enforcement codes with some degree of assurance based on model specification represented by UML and OCL. As I addressed in the previous section, all model components and constraints are evaluated so the enforcement codes generated from the model representation should fully reflect features and functionalities of a formal security model, especially the NIST/ANSI RBAC standard in this article. Although I select the Java language as the target language in the framework, I believe this approach can be extended for other languages as well. The process of mapping model specification to enforcement codes could be performed by the adoption of the tools such as Octopus [95] and Dresden OCL toolkit [96]. Due to the page limitation, I omit the discussion of the details of *general* transformation process from UML and OCL to Java codes. Instead, I only discuss several critical issues related to the transformation process.

In the specification of RBAC model, RBAC model elements and relations are defined using the UML class diagrams and the functionalities and constraints of RBAC model are specified with OCL expressions. To implement UML model elements, the classes, attributes, operations and associations need to be translated into corresponding Java classes or operations. Then, each class in the model is mapped to one Java class; an operation for the class is created by one operation in Java class; and an attribute and its association with the class in the model generate a private class member and `get` and `set` operations in the Java class. Also, the basic types of OCL are mapped to corresponding Java types. For example, `Real` in OCL is mapped to `float` in Java. OCL collection type is implemented as a library using `Set` or `List` of Java language. It is a little complicated when implementing this

```
1 public List checkStaticConstraints() {
2     List result = new ArrayList();
3     try {
4           Constriant_SSoD_CR();    //Performing the SSoD_CR constraint
5     } catch (ConstriantException e) {
6           result.add(new ConstriantError(e.getInstance(), e.getMessage()));
7     }
8     try {
9           Constriant_SSoD_CU();    //Performing the SSoD_CU constraint
10    } catch (ConstriantException e) {
11          result.add(new ConstriantError(e.getInstance(), e.getMessage()));
12    }
13    return result;
```

Figure 4.11: Generated Java code for `CheckStaticConstraints` function.

library, because OCL collections have a large amount of predefined operations, such as `select` and `collect`. These operations need be defined as standard operations using Java. Based on the implemented standard OCL library, OCL expressions can directly generate Java codes.

In the implementation, two special system functions, `CheckStaticConstraints` and `CheckDynamicConstraints`, for Constrained RBAC are created automatically to collect and enforce static and dynamic constraint expressions respectively for corresponding components. While we can use a universal function, such as a `CheckConstraints` function, to check all constraints for one component, for the purpose of making checking procedures more efficient, I provide two system functions for constraint checking. Session-related constraint expressions are performed by `CheckDynamicConstraints`, and other constraints are enforced by `CheckStaticConstraints`. Figure 4.11 shows the generated Java codes for `CheckStaticConstraints` function of `user` class. Note that `CheckStaticConstraints` function includes the codes for checking two static *SoD* constraints with CR and CU as well.

Analysis of RBAC Model

I utilize a SAT solver as an underlying formal verification technique to demonstrate automatic analysis and test generation for the formal specifications of an RBAC

model and associated constraints based on the approaches and definitions introduced in Section 4.2. Alloy [88] is used as an intermediate language into which the RBAC model is constructed and the RCL2000-based constraints are translated. Then, using Alloy tool called Alloy Analyzer [25], which uses a SAT solver that supports enumeration, the RBAC model and corresponding constraints are analyzed, and test cases are generated from the RBAC model specifications.

RBAC Model Representation in Alloy The NIST/ANSI standard for RBAC gives an RBAC reference model, which defines sets of basic RBAC elements and relations, including a set of roles, a set of users, a set of permissions, relationships between users, roles, and permissions. I define a primary representation of the NIST/ANSI RBAC model in Alloy as follows:

```
module RBAC

sig User {}

sig Role {}

sig Operation {}

sig Object {}

sig Permission {Operation, Object}

sig Session {}

sig URA {

 ura:  User->Role}

sig PRA {

 pra:  Permission->Role}
```

```
sig US {
 us:  User!->Session}
sig SR {
 sr:  Session->Role}
sig PB {
 pb:  Operation->Object}
```

The above defines the core element sets and relations in an RBAC model. A role hierarchy relation supporting hierarchical RBAC is defined as follows:

```
sig RRA {
 hierarchy:  Role->Role}
```

To specify *SSoD* relations and *DSoD* relations in the context of conflicting roles, which are addressed in the NIST/ANSI RBAC model, I give the following Alloy definitions [2]:

```
sig SCR {
 conflict_role:  set Role,
 cardinality:  Int}
sig DCR {
 conflict_role:  set Role,
 cardinality:  Int}
```

[2]The separation of duty relations in the NIST/ANSI RBAC model can be extended to support conflicting permissions and conflicting users, using several definitions such as {SCP, DCP} and {SCU, DCU}, respectively.

RBAC Constraint Specification in Alloy In order to reason about RCL2000 policy specifications using Alloy tool, we need to translate RCL2000 policy expressions to Alloy statements. Similar to the translation from RCL2000 to OCL, six RBAC system functions `user`, `roles`, `sessions`, `permissions`, `operations` and `object` can be represented in Alloy. For instance, `roles(u)` is converted to `u.(URA.ura)`. Also, `roles`* and `permissions`* are able to converted to Alloy using "*", which denotes a reflexive transitive closure operator, and "$\sim$", which denotes transpose operator to support the role hierarchy. Each term in RCL2000 can be also converted to corresponding Alloy operator. Figure 4.12 gives a detailed translation algorithm.

The following examples give equivalent Alloy expressions for *SSoD-CR* and *User-based DSoD* constraints.

Translated Alloy Expression for SSoD-CR constraint:

```
all u:User | all scr:SCR |
  #((u.(URA.ura).~*(RRA.hierarchy)) &
    scr.conflict_role) <= scr.cardinality
```

Translated Alloy Expression for User-based DSoD constraint:

```
all u:User | all dcr:DCR |
  #(u.(US.us).(SR.sr).~*(RRA.hierarchy) &
    dcr.conflict_role) <= dcr.cardinality
```

RBAC Function Verification The functional specification in the NIST/ANSI standard for RBAC defines various functions that role-based systems should provide. These functionalities are described in the standard using a set-based specification

Input: *RCL2000* expression; **Output**: *Alloy* expression

Let **Simple-OE** term be either **OE(*set*)** , or **OE(function(*element*))**, where ***set*** is an element of {**U, R, OP, OBJ, P, S, SCR, DCR, SCU, DCU, SCP, DCP, PR, scr, dcr, scu, dcu, scp, dcp, pr**} and ***function*** is an element of {**user, roles, roles*, sessions, permissions, permissions*, operations, object**}

1. AO elimination
  Replace all occurrences of *AO(expr)* with *(expr-{**OE**(expr)})*;
2. OE elimination
  While There exists **Simple-OE** term in RCL2000 expression
    case (i) **Simple-OE** term is *OE(set)*
      choose *set* term;
      call set_reduction procedure;
    case (ii) **Simple-OE** term is *OE(function(element))*
      choose *function(element)* term;
      call reg_function_reduction procedure;
  End
  Procedure set_reduction
    case (1) *set* is *U*
      put *all u:User* | to right of existing quantifier(s);
  replace all occurrences of *OE(U)* with *u*;
    case (2) *set* is *R*
      put *all r:Role* | to right of existing quantifier(s);
  replace all occurrences of *OE(R)* with *r*;
  ……[a]
  End
  Procedure reg_funtion_reduction
    case (1) *function(element)* term is *user(r)*
      put *all u1:User= r.~(URA.ura)* | to right of existing quantifier(s);
      replace all occurrences of *OE(user(r))* with *u1*;
    case (2) *function(element)* term is *roles(u)*
      put *all r1:Role= u.(URA.ura)* | to right of existing quantifier(s);
      replace all occurrences of *OE(roles(u))* with *r1*;
  ……
  End
3. System Functions elimination
  While There exists *function(element)* term in RCL2000 expression
    choose *function(element)* term;
    call sys_function_reduction procedure;
  End
  Procedure sys_function_reduction
    case (1) *function(element)* term is *user(r)*
      replace all occurrences of *user(r)* with *r.~(URA.ura)*;
    case (2) *function(element)* term is *roles(u)*
      replace all occurrences of *roles(u)* with *u.(URA.ura)*;
  ……
  End
4. Operators elimination
  replace all occurrences of *set1* ⊆ *set2* with *set2 in set1*;
  replace all occurrences of *set1* ∩ *set2* with *set1 & set2*;
  replace all occurrences of *set1* ∪ *set2* with *set1 + set2*;
  replace all occurrences of | *set* | with # *set*;
  replace all occurrences of ⇒ with =>;
  replace all occurrences of ∧ with &&;
  replace all occurrences of ≠ with !=;
  replace all occurrences of ≤ with <=;
  replace all occurrences of ≥ with >=;
  replace all occurrences of ∅ with *none*;

[a] Due to the page limit, we omitted the detailed procedures for OE elimination and system function elimination.

Figure 4.12: Translation algorithm from RCL2000 to Alloy.

language, *Z*. Prior to applying these functional definitions for role-based system development, the correctness of these definitions needs be checked rigorously. Formal verification is necessary for this objective.

In this subsection, I employ `DeleteRole` function as an example to demonstrate how the formal verification can assist in finding mistakes in the functional specifications. In hierarchical RBAC, the following functional properties must be achieved by the `DeleteRole` function.

1. The existing role is removed from the *Role* date set.
2. Any use-to-role assignment relation established by the role is removed.
3. Any permission-to-role assignment relation established by the role is removed.
4. Any role hierarchy relationship established by the role is removed.

The following is the functional definition of `DeleteRole` for supporting hierarchical RBAC in the NIST/ANSI RBAC standard.

$$
\begin{array}{l}
DeleteRole(role{:}NAME) \lhd \\
\quad role \in ROLES \\
\quad UA' = UA \setminus \{u{:}Users \bullet u \mapsto role\} \\
\qquad assigned\_users' = assigned\_user \setminus \{role \mapsto assigned\_user(role)\} \\
\quad PA' = PA \setminus \{op{:}OPS, obj{:}OBJS \bullet (op,obj) \mapsto role\} \\
\qquad assigned\_permissions' = assigned\_permissions \setminus \{role \mapsto assigned\_permissions(role)\} \\
\quad ROLES' = ROLE \setminus \{role\} \rhd
\end{array}
$$

An Alloy function is constructed based on the above definition as follows:

```
fun DeleteRole(r:Role){

 r in Role =>

  all p:Permission |

  all u:User| (u->r) in URA.ura =>

   URA.ura = (URA.ura - (u->r))) &&

  (all p:Permission | (p->r) in PRA.pra =>
```

```
 PRA.pra = (PRA.pra - (p->r))) &&
(Role = Role - r) }
```

We can also define an Alloy assertion to describe the RBAC functional properties P_f discussed earlier. Corresponding functional properties for `DeleteRole` operation with the notion of hierarchical RBAC are defined as follows:

```
assert Check_DeleteRole {
 all r:Role| all r':Role | all u:User |
 all p:Permission |
  DeleteRole(r) &&
  //The role is removed from the role set
  r !in Role &&
  //Corresponding UA relations are removed
  (u->r) !in URA.ura &&
  //Corresponding PA relations are removed
  (p->r) !in PRA.pra &&
  //Inheritance relations are removed
  (r->r') !in RRA.hierarchy &&
  (r'->r) !in RRA.hierarchy }
check Check_DeleteRole
```

By running Alloy Analyzer, we can validate this assertion against the RBAC model specification, which contains the `DeleteRole` function specification. The

Alloy Analyzer detects counterexamples, which identify violations of the assertion with respect to the function specification. After careful inspection, I found that the functional definition of `DeleteRole` for hierarchical RBAC in the NIST/ANSI RBAC standard misses a step for removing inheritance relations established by the role that is being deleted. In [97], another formal definition of `DeleteRole` function for hierarchical RBAC is given. Using the same approach, I identified that the steps for removing UA relations and PA relations are missed in their specification as well.

RBAC Constraint Verification I now demonstrate how to identify *under-* and *over-constraints* with Alloy using the aforementioned approach in Section 4.2.

Regarding separation of duty principles, the following authorization property considering the role hierarchy is unexpected:

- *Two conflicting roles are authorized to the same user.*

  I specify this unexpected authorization property P_{a^-} in Alloy as follows:

```
pred Check_SSoD[ disj r1,r2:Role, u:User, scr:SCR] {
  //r1 and r2 are mutually exclusive
  r1 in scr.conflict_role &&
  r2 in scr.conflict_role &&
  scr.cardinality = 1 &&
  //r1 and r2 are authorized to the same user
  r1 in u.(URA.ura).~*(RRA.hierarchy) &&
  r2 in u.(URA.ura).~*(RRA.hierarchy) }
run Check_SSoD
```

Suppose the policy designer only defines a simple *SSoD constraint*, which ignores the role hierarchy relation. We can translate the RCL2000 expression for the simple *SSoD constraint* to the Alloy expression, and put it into an Alloy fact as an Alloy constraint as follows:

```
fact SSoD {
 all u:User | all scr:SCR |
  #(u.(URA.ura) & scr.conflict_role)
   <= scr.cardinality }
```

When running the predicate `Check_SSoD` defined above, instances–in which conflicting roles are *indirectly* assigned to a user–are found by Alloy Analyzer. It means the unexpected authorization property is held by the constraint specification. In addition, we can conclude the constraint is too week with respect to the authorization property.

Taking into account the following authorization properties for dynamic separation of duty principle, "*a user cannot activate two conflicting roles in the same session, but can activate them in the different session*," I specify this expected authorization property P_{a^+} in Alloy as follows:

```
assert Check_DSoD {
 all u:User | all disj r1,r2:Role |
 all disj s1,s2:Session | all dcr:  DCR |
  //r1 and r2 are dynamic conflicting roles
  r1 in dcr.conflict_role &&
  r2 in dcr.conflict_role &&
```

```
    dcr.cardinality = 1 &&
    //u creates s1, s2
    (u->s1) in US.us &&
    (u->s2) in US.us &&
    //r1 and r2 cannot be activated in the
    //same session, but can be activated
    //in the different session
    (r1->s1) in ~SR.sr &&
    (r2->s1) !in ~SR.sr &&
    (r2->s2) in ~SR.sr }
check Check_DSoD
```

Assuming the policy designer defines a *User-based DSoD* constraint [3] as I demonstrated before, I define an Alloy fact, which contains this constraint specification.

```
fact DSoD {
  all u:User | all dcr:DCR |
   #(u.(US.us).(SR.sr) & dcr.conflict_role)
    <= dcr.cardinality }
```

Running "`check Check_DSoD`" in Alloy Analyzer, counterexamples are found. It indicates the expected authorization property expressed in assertion `Check_DSoD` is denied by the constraint specification. That is, the constraint is too strong, and

[3] To reduce the complicity, the role hierarchy is omitted in this constraint.

should be weakened to contain the expected authorization properties. If we replace the *User-based DSoD* constraint with the *Session-based DSoD* constraint, the expected authorization property defined in assertion `Check_DSoD` is held.

Test Case Generation As mentioned earlier, *negative test cases* T^- are derived from a formal access control model specification, in which the constraint specification is drawn out and serves as an authorization property for the formal verification, while *positive test cases* T^+ are generated from a formal specification if we take the constraint specification out of the access control model specification, and consider the negated constraint specification as an authorization property.

I take the simple *SSoD_CR constraint* as an example to demonstrate the process of automated test generation. The following assertion is defined to drive the negative test cases for the constraint specification.

```
assert SSoD_CR {
  all u:User | all scr:SCR |
   #(u.(URA.ura) & scr.conflict_role)
    <= scr.cardinality }
check SSoD_CR
```

Checking this assertion against the RBAC model specification, in which *SSoD_CR constraint* has been taken out, counterexamples are generated. These counterexamples are used to construct negative test cases as undesired system states to test the conformance of the *SSoD_CR constraint* in both access control system design and implementation.

To derive positive test cases for the simple *SSoD_CR constraint*, the *negated* constraint specification is used as an authorization property. I define an assertion for this objective as follows:

```
assert Neg_SSoD_CR {

 all u:User | all scr:SCR |

  #(u.(URA.ura) & scr.conflict_role)

   > scr.cardinality }

check Neg_SSoD_CR
```

Note that the above assertion states the number of roles–which are from a conflicting role set–assigned to a user must exceed the cardinality number of the conflicting role set. Supposing the cardinality number is *one*, it means a user must own two or more conflicting roles. Through running this assertion, counterexamples are also generated. Then, positive test cases serving as desired system states are constructed from these counterexamples.

To generate more meaningful test cases for real application domains, Alloy signatures need to reflect all RBAC configuration components of the targeted application domain for producing specialized instances of the defined Alloy module. Then, running the constraint assertion with the scope enables Alloy to generate test cases. Suppose we have a banking system with a user `Bob` and two conflicting roles, `customerServiceRep` and `loanOfficer`. We first need to define the appropriate assignment of user, role and conflicting role set as follows.

```
one sig Bob extends User{}

one sig customerServiceRep,loanOfficer extends
```

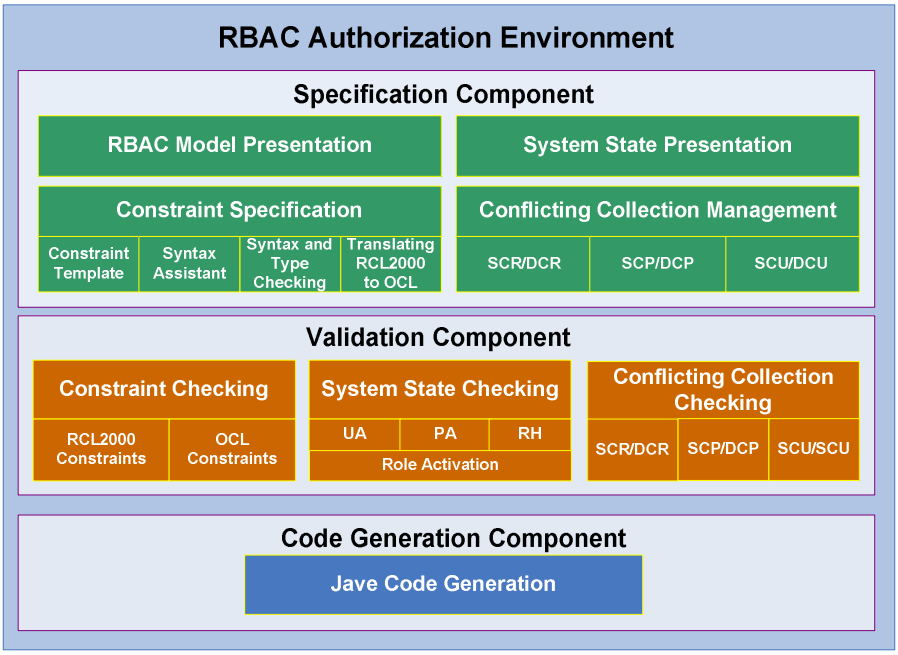


Figure 4.13: Structural overview of the RAE.

```
  Role{}

fact SCR_rules {

  customerServiceRep in SCR.conflict_role &&

  loanOfficer in SCR.conflict_role }
```

Alloy definition is then provided to Alloy Analyzer so that it can run `SSoD` assertion defined earlier with the scope of *one* user and *two* roles. Finally, Alloy Analyzer can generate a negative test case for the conformance testing, such that the user `Bob` is assigned to two conflicting roles, `customerServiceRep` and `loanOfficer`.

Tool Support.

In this section, I introduce the tools developed to support the realization and analysis of RBAC model with AMF.

RBAC Authorization Environment (RAE) To demonstrate the usability of the approach, a tool called RAE was developed which is based on ArgoUML, an open source UML-based modeling tool [98]. RAE tool is composed of three major components, specification component, analysis component and code generation component as shown in Figure 4.13. Specification component is responsible for specifying RBAC model and constraints. Analysis component is in charge of conflict and violation checking so as to validate RBAC model and constraints. Code generation component is used to automatically generate enforcement codes.

Specification component consists of four sub-components: RBAC model representation, system state presentation, constraint specification and collection management. Figure 4.14(a) shows the implementation of the RBAC model presentation and constraint specification components.

- *RBAC model representation* employs UML diagrams to represent RBAC model. This component is implemented based on ArgoUML.
- *System state presentation* is responsible to create snapshots of the system model at particular points using UML object diagrams, which is composed of a set of objects and a set of association links.
- *Constraint specification* provides an environment to easily specify authorization constraints using RCL2000 and OCL. It is further divided into four sub-components: constraints template, syntax and type checking, translating RCL2000 to OCL, and translating RCL2000 to Alloy. To simplify the constraint definition, some reusable constraint templates for typical RBAC constraints are provided in the constraint expression editor. Also it has a syntax assistant for user to construct complicated RCL2000 or OCL constraint expressions conveniently. The syntax checking verifies the constraints expression against the grammar of the specification language. The type check-

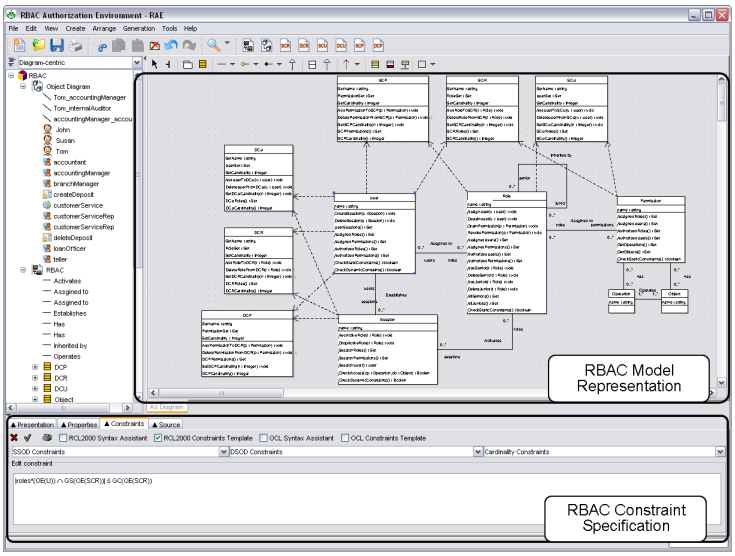
ing ensures that every RCL2000 or OCL constraints expression can be typed correctly.

- *Conflicting collection management*. In the RAE tool, conflicting collections (SCR/DCR, SCP/DCP and SCU/DCU) are maintained separately from the constraint expressions.

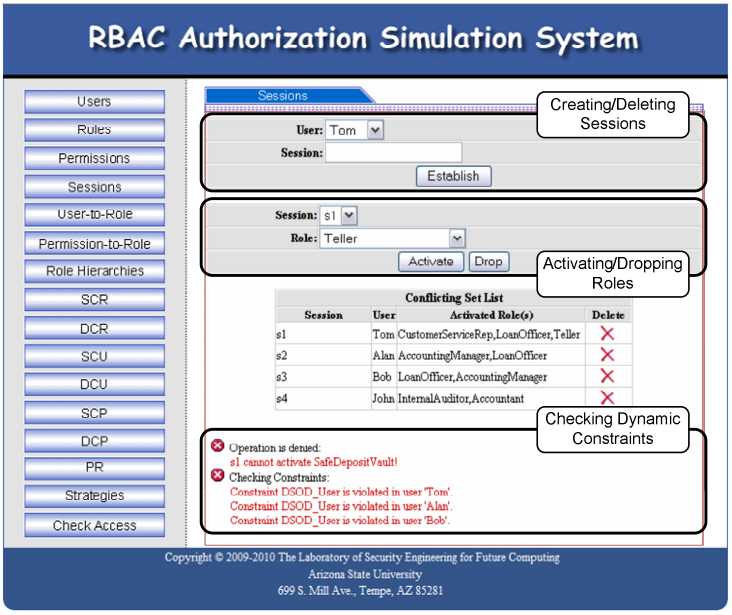
Validation component is composed of three sub-components: constraint checking, system state checking and conflicting collection checking.

- *Constraint checking* tests whether the constraint is violated by the current system state (snapshot), when a new constraint is established or an old constraint is modified. It supports both RCL2000 constraint checking and OCL constraint checking.
- *System state checking* is response for identifying the conflicts whenever a system state is changed. The RAE tool supports four kinds of assignment relation checking: UA (user-to-role assignment), PA (permission-to-role assignment), RH (role-to-role assignment) and role activation. When any assignment occurs, all related authorization constraints are evaluated against the changed system state.
- *Conflicting collection checking* is in charge of detecting the conflicts resulting from any changes of conflicting collections. Changing a conflicting collection affects the semantics of relevant SD constraint expressions.

Code generation component is used to generate java code automatically for RBAC model and constraints. The enforcement codes are used by developers for role-based systems.



(a) RAE: RBAC model and constraint specification.



(b) RASS: simulation of creating session, activating role and checking dynamic constraints.

Figure 4.14: RAE tool and RASS testbed environment

RBAC Authorization Simulation System (RASS) The main purpose of building an RBAC authorization simulation system is to verify the consistency and correctness of the generated RBAC enforcement codes. The following goals can be achieved by the simulation system: (1) executing the RBAC authorization man-

agement operations to check all administrative functions and review functions; (2) applying session-related operations to simulate system actions, such as creating a session and checking dynamic constraints; and (3) using "Check Access" function to simulate the practical processes of authorization checking.

The RAE tool can generate enforcement codes from the model specified with the composition of UML and OCL. The model elements and relations defined with UML are translated to corresponding Java classes and operations. OCL expressions that define the body of a function or a constraint are translated to the body of the corresponding Java methods. The generated codes are plain Java codes, and can be utilized by developers to integrate into a real application system required RBAC features. I built the RASS system through designing a web-based user interface and a storage layer to incorporate the generated RBAC enforcement modules. The web-based user interface provides intuitive interactions between the users and the system functions, which are provided by generated codes, and the storage layer is in charge of storing the RBAC configurations.

In RASS, RBAC function and constraint implementations are verified by running extensive test cases. A snapshot of running test cases in RASS system is shown in Figure 4.14(b). In this snapshot, the simulation of some dynamic system actions, such as creating and deleting a session, activating and dropping a role, and checking dynamic constraints, are presented. Consequently, corresponding system functions, such as `CreateSession` and `DeleteSession`, `AddActivateRole` and `DropActivateRole`, and `CheckDynamicConstraints`, in RBAC model are able to be evaluated under the simulation system.

Tool Chain The toolset in this work constitutes a toolchain with Alloy Analyzer depicted in Figure 4.15 to facilitate the application of the proposed methodology for

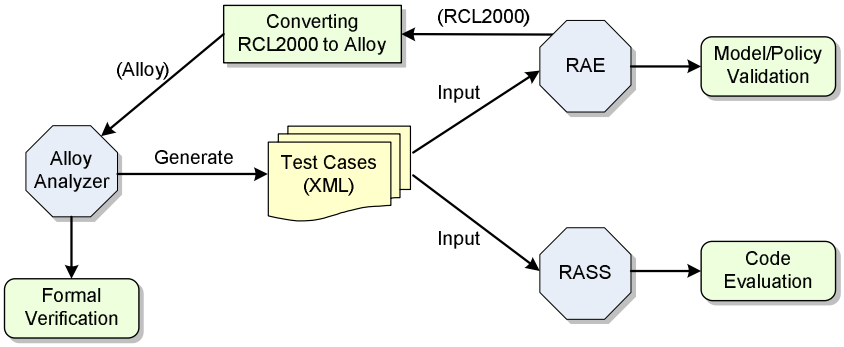


Figure 4.15: Toolchain supporting the proposed approach.

automatic analysis, realization and conformance testing of RBAC model and constraints. The policy designers are able to specify RBAC constraints with RCL2000, and then convert RCL2000-based constraints to Alloy expressions in RAE. The generated Alloy specifications for constraints can be forwarded to Alloy Analyzer. The formal specifications of an RBAC model are also constructed to Alloy, then analyzed by Alloy Analyzer as well. In addition, Alloy Analyzer allows to generate all nonisomorphic instances from an Alloy specification. These instances are then used as test cases, which are fed into RAE to construct system states. Importantly, such cases are checked against constraints to validate the RBAC model and constraint specifications in the stage of system design as well as utilized by RASS to evaluate the generated RBAC codes under the simulation. Meanwhile, constraint anomaly and violation are also analyzed by RAE and RASS.

4.3 Access Control Policy

With the explosive growth of Web applications and Web services deployed on the Internet, the use of a policy-based approach has received considerable attention to accommodate the security requirements covering large, open, distributed and heterogeneous computing environments. In the era of distributed, heterogeneous and Web-oriented computing, the increasing complexity of policy-based computing de-

mands strong support of policy analysis techniques. Without analysis, most benefits of policy-based techniques and declarative policy languages may be in vain.

In order to ensure the correctness of access control policies for policy-based computing, I incorporate automated reasoning techniques and systematic anomaly management mechanisms into the AMF framework as shown in Figure 4.16. For policy reasoning, I introduce a logic-based reasoning approach for access control policies. The approach first converts access control policies to a logic-based representation. Then, by means of off-the-shelf logic solvers, a variety of analysis services can be provided to ensure policy-based system management. The following are several policy reasoning services that can be accomplished by the logic-based reasoning approach.

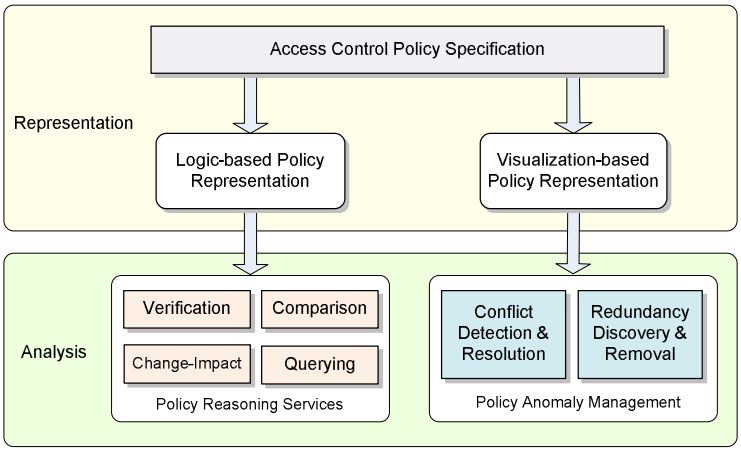


Figure 4.16: Representation and analysis of access control policy.

- *Policy Verification*. Check if the policy satisfies a particular policy property. If not, give a counterexample.
- *Policy Comparison*. For two policies (or policy sets) `P1` and `P2` check if whenever `P1` yields a decision, `P2` will yield the same decision, too. If not, give a counterexample.

- *Change-Impact Analysis*. This analysis consumes two policies that span a set of changes and summarizes the differences between the two policies. The analysis can happen even in the absence of formal properties about the systems.
- *Policy Querying*. Search for policies in the policy set with respect to particular attribute values.

Although the logic-based reasoning approach can also support policy anomaly detection by verifying policies against the given properties [21], it cannot exhaustively identify all potential policy anomalies. Therefore, the AMF framework for access control policy analysis needs additional modules to systematically manage policy anomalies for accurate policy anomaly detection and effective policy anomaly resolution. Since information visualization technique [99] enables users to explore, analyze, reason and explain abstract information by taking advantage of their visual cognition, I further integrate an information visualization technique in the anomaly management mechanism, which contains two major tasks, *conflict detection and resolution* and *redundancy discovery and removal*.

In this section, I address the policy analysis approach using XACML, which has become the *de facto* standard for specifying and enforcing access control policies for various applications and services, especially, in current Web-based computing environment.

Representing and Reasoning about Access Control Policy

XACML has been widely adopted to specify access control policies for various Web applications. With expressive policy languages such as XACML, assuring the correctness of policy specifications becomes a crucial and yet challenging task due to the lack of logical and formal foundation. The logic-based policy reason-

ing approach first turn XACML policies to ASP programs. Compared to a few existing approaches to formalizing XACML policies, such as [20, 21], the formal representation in this work is more straightforward and can cover more XACML features. Furthermore, translating XACML to ASP allows users to leverage off-the-shelf ASP solvers for a variety of analysis services. I also overview a tool `XACML2ASP` and conduct experiments with real-world XACML policies to evaluate the effectiveness and efficiency of the solution in this work.

Example XACML Policy

Consider an example XACML policy for a software development company, which is utilized throughout this section, shown in Figure 4.17. The root policy set ps_1 contains two policies p_1 and p_2 which are combined using $\texttt{first}\text{–}applicable$ combining algorithm. The policy p_1, which is the global policy of the entire company, has two rules r_1 and r_2 indicating that

- all employees can read and change codes during working hours from 8:00 to 17:00 (r_1), and
- nobody can change code during non-working hours (r_2).

On the other hand, each department is responsible for deciding whether employees can read codes during non-working hours. A local policy p_2 for a development department with three rules r_3, r_4 and r_5 is that

- developers can read codes during non-working hours (r_3),
- testers cannot read codes during non-working hours (r_4), and
- testers and developers cannot change codes during non-working hours (r_5).

```
1<PolicySet PolicySetId="ps1" PolicyCombiningAlgId="first-applicable">
2   <Target/>
3   <Policy PolicyId="p1" RuleCombiningAlgId="permit-overrides">
4     <Target/>
5     <Rule RuleId="r1" Effect="permit">
6       <Target>
7         <Subjects><Subject>      employee   </Subject></Subjects>
8         <Resources><Resource>   codes      </Resource></Resources>
9         <Actions><Action>         read       </Action>
10                  <Action>         change     </Action></Actions>
11      </Target>
12      <Condition>              8 ≤ time ≤ 17          </Condition>
13    </Rule>
14    <Rule RuleId="r2" Effect="deny">
15      <Target>
16        <Subjects><Subject>      employee   </Subject></Subjects>
17        <Resources><Resource>  codes      </Resource></Resources>
18        <Actions><Action>         change     </Action></Actions>
19      </Target>
20    </Rule>
21  </Policy>
22  <Policy PolicyId="p2" RuleCombiningAlgId="deny-overrides">
23    <Target/>
24    <Rule RuleId="r3" Effect="permit">
25      <Target>
26        <Subjects><Subject>      developer   </Subject></Subjects>
27        <Resources><Resource>   codes      </Resource></Resources>
28        <Actions><Action>         read       </Action></Actions>
29      </Target>
30    </Rule>
31    <Rule RuleId="r4" Effect="deny">
32      <Target>
33        <Subjects><Subject>       tester     </Subject></Subjects>
34        <Resources><Resource>  codes      </Resource></Resources>
35        <Actions><Action>         read       </Action></Actions>
36      </Target>
37    </Rule>
38    <Rule RuleId="r5" Effect="deny">
39      <Target>
40        <Subjects><Subject>       tester     </Subject>
41                  <Subject>     developer  </Subject></Subjects>
42        <Resources><Resource>  codes      </Resource></Resources>
43        <Actions><Action>         change     </Action></Actions>
44      </Target>
45    </Rule>
46  </Policy>
47</PolicySet>
```

Figure 4.17: An example XACML policy.

Note that the rule combining algorithm for policy p_1 is p*ermit−overrides* and the rule combining algorithm for policy p_2 is d*eny−overrides*.

I consider a subset of XACML that covers more constructs than the ones considered in [100] and [21]. I allow the most general form of *Target*, take into account *Condition*, and cover all four combining algorithms.

XACML components can be abstracted as follows: *Attributes* are the names of elements used by a policy. *Attributes* are divided into three categories: *subject attributes*, *resource attributes* and *action attributes*. A *Target* is a triple $\langle$*Subjects*, *Resources*, *Actions*$\rangle$. A *Condition* is a conjunction of comparisons. An *Effect* is either "p*ermit*," "d*eny*," or "i*ndeterminate*."

- An XACML rule can be abstracted as

  $$\langle \mathit{RuleID}, \mathit{Effect}, \mathit{Target}, \mathit{Condition} \rangle$$

  where *RuleID* is a rule identifier. For example, rule $\mathtt{r}_1$ in Figure 4.17 can be viewed as

  $$\langle \mathtt{r}_1, \mathtt{p}\mathit{ermit}, \langle \mathtt{e}\mathit{mployee}, \mathtt{r}\mathit{ead} \vee \mathtt{c}\mathit{hange}, \mathtt{c}\mathit{odes} \rangle, 8 \leq \mathtt{t}\mathit{ime} \leq 17 \rangle.$$

- An XACML policy can be abstracted as

  $$\langle \mathit{PolicyID}, \mathit{Target}, \mathit{Combining\ Algorithm}, \langle r_1, \ldots, r_n \rangle \rangle$$

  where *PolicyID* is a policy identifier, $r_1, \ldots, r_n$ are rule identifiers and *Combining Algorithm* is either p*ermit–overrides*, d*eny–overrides*, or f*irst–applicable*. For example, policy $\mathtt{p}_1$ in Figure 4.17 is abstracted as:

  $$\langle \mathtt{p}_1, \mathit{Null}, \mathtt{p}\mathit{ermit–overrides}, \langle \mathtt{r}_1, \mathtt{r}_2 \rangle \rangle.$$

- Similarly an XACML policy set can be abstracted as

  $$\langle \mathit{PolicySetID}, \mathit{Target}, \mathit{Combining\ Algorithm}, \langle p_1, \ldots, p_m, ps_{m+1}, \ldots, ps_n \rangle \rangle$$

where *PolicySetID* is a policy set identifier, $p_1, \ldots, p_m$ are policy identifiers, $ps_{m+1}, \ldots, ps_n$ are policy set identifiers, and *Combining Algorithm* is either p*ermit–overrides*, d*eny–overrides*, f*irst–applicable*, or o*nly–one–applicable*. For example, policy set $\mathtt{ps}_1$ can be viewed as

$$\langle \mathtt{ps}_1, \mathit{Null}, \mathtt{f}\mathit{irst–applicable}, \langle \mathtt{p}_1, \mathtt{p}_2 \rangle \rangle.$$

Turning XACML into ASP. I provide a translation module that turns an XACML description into a program in answer set programming (ASP) [26, 27]. This interprets a formal semantics of XACML language in terms of the Answer Set semantics.

The translation module converts an XACML rule

$$\langle \mathit{RuleID}, \mathit{Effect}, \mathit{Target}, \mathit{Condition} \rangle$$

into a set of ASP rules [4]

$$\mathit{decision}(\mathit{RuleID}, \mathit{Effect}) \leftarrow \mathit{Target} \wedge \mathit{Condition}.$$

An XACML policy

$$\langle \mathit{PolicyID}, \mathit{Target}, \mathit{Combining\ Algorithm}, \langle r_1, \ldots, r_n \rangle \rangle$$

can be also translated into a set of ASP rules. In the following we assume that R and R' are variables that range over all rule ids, and V is a variable that ranges over $\{\mathtt{p}\mathit{ermit}, \mathtt{d}\mathit{eny}, \mathtt{i}\mathit{ndeterminate}\}$. In order to represent the effect of each rule r_i ($1 \leq i \leq n$) on policy, I write

$$\mathit{decision\_from}(\mathit{PolicyID}, r_i, V) \leftarrow \mathit{decision}(r_i, V).$$

Each rule combining algorithms is turned into logic programming rules under the stable model semantics as follows:

[4] I identify *Target* with the conjunction of its components. Also, I identify " $\wedge$ " with ",", " $\leftarrow$ " with " :- " and a rule of the form $A \leftarrow B, C \vee D$ as a set of the two rules $A \leftarrow B, C.$ and $A \leftarrow B, D.$

- p*ermit–overrides* of policy p is represented as

$$decision(p,\texttt{permit}) \leftarrow decision\_from(p,R,\texttt{permit}) \wedge Target.$$
$$decision(p,\texttt{deny}) \leftarrow decision\_from(p,R,\texttt{deny}) \wedge\ not\ decision(p,\texttt{permit}) \wedge Target.$$

- d*eny–overrides* of policy p is represented as

$$decision(p,\texttt{deny}) \leftarrow decision\_from(p,R,\texttt{deny}) \wedge Target.$$
$$decision(p,\texttt{permit}) \leftarrow decision\_from(p,R,\texttt{permit}) \wedge\ not\ decision(p,\texttt{deny}) \wedge Target.$$

- f*irst–applicable* of policy p is represented as

$$has\_decision\_from(p,R) \leftarrow decision\_from(p,R,V).$$
$$decision(p,V) \leftarrow decision\_from(p,r_i,V) \wedge \bigwedge_{1\leq k\leq i-1} not\ has\_decision\_from(p,r_k) \wedge Target.$$

The translation of a policy set is similar to the translation of a policy except that the policy combining algorithm o*nly–one–applicable* needs to be taken into account. For instance, o*nly–one–applicable* of policy set *ps* is represented as follows:

$$decision(ps,V) \leftarrow decision\_from(ps,P,V) \wedge 1\{has\_decision\_from(ps,P) : policy(P)\}1.$$
$$decision(ps,\texttt{indeterminate}) \leftarrow 2\{has\_decision\_from(ps,P) : policy(P)\}.$$

Figure 4.18 shows an ASP representation of the example XACML policy in the language of Gringo by applying the translation approach.

XACML Policy Analysis using ASP

Once we represent an XACML into an ASP program Π, we can use off-the-shelf ASP solvers for several automated analysis services. In this section, I mainly illustrate how policy verification can be handled by the policy analysis approach.

```
------------------------------------------------------------
 value(permit;deny;indeterminate).
 rule(r1;r2;r3;r4;r5).
 policy(p1;p2).
 policyset(ps1).
 time(0..23).
 #domain value(V;V1).
 #domain rule(R;R1).
 #domain policy(P).
 #domain time(T).
% domain definition
 subject(employee) :- subject(developer).
 subject(employee) :- subject(tester).
% r1
decision(r1,permit) :- subject(employee),action(read),
              resource(codes),8<=T,T<=17, current_time(T).
decision(r1,permit) :- subject(employee),action(change),
              resource(codes),8<=T,T<=17, current_time(T).
% r2
decision(r2,deny) :- subject(employee),action(change),
                     resource(codes).
% r3
decision(r3,permit) :- subject(developer),action(read),
                       resource(codes).
% r4
decision(r4,deny) :- subject(tester),action(read),
                     resource(codes).
% r5
decision(r5,deny) :- subject(tester),action(change),
                     resource(codes).
decision(r5,deny) :- subject(developer),action(change),
                     resource(codes).
% p1
 decision_from(p1,r1,V) :- decision(r1,V).
 decision_from(p1,r2,V) :- decision(r2,V).
 decision(p1,permit) :- decision_from(p1,R,permit).
 decision(p1,deny) :- decision_from(p1,R,deny),
                      not decision(p1,permit).
% p2
 decision_from(p2,r3,V) :- decision(r3,V).
 decision_from(p2,r4,V) :- decision(r4,V).
 decision_from(p2,r5,V) :- decision(r5,V).
 decision(p2,deny) :- decision_from(p2,R,deny).
 decision(p2,permit) :- decision_from(p2,R,permit),
                        not decision(p2,deny).
% ps1
 decision_from(ps1,p1,V) :- decision(p1,V).
 decision_from(ps1,p2,V) :- decision(p2,V).
 has_decision_from(ps1,p1) :- decision_from(ps1,p1,V).
 decision(ps1,V) :- decision_from(ps1,p1,V).
 decision(ps1,V) :- decision_from(ps1,p2,V),
                    not has_decision_from(ps1,p1).
------------------------------------------------------------
```

Figure 4.18: ASP representation of the example XACML policy.

The problem of verifying a security property F against an XACML description can be cast into the problem of checking whether the program

$$\Pi \cup \Pi_{query} \cup \Pi_{config}$$

has no answer sets, where Π is the program corresponding to the XACML specification, Π_{query} is the program corresponding to the program that encodes the negation of the property to check, and Π_{config} is the following program that generates *arbitrary* configurations.

```
subject_attributes(developer;tester;employee).
action_attributes(read;change).
resource_attributes(codes).

1{subject(X) : subject_attributes(X)}.
1{action(X) : action_attributes(X)}.
1{resource(X) : resource_attributes(X)}.
1{current_time(X) : time(X)}1.
```

If no answer set is found, this implies that the property is verified. Otherwise, an answer set returned by an ASP solver serves as a counterexample that indicates why the description does not entail F. This helps the policy designer find out the design flaws in the policy specification.

For example, consider the example XACML policy shown in Figure 4.17. We need to ensure that a developer cannot change codes during non-working hours. The input query Π_{query} can be represented as follows:

```
working_hours :- 8<=T, T<=17,current_time(T).
check :- decision(ps1,permit),
```

```
  subject(developer),action(change),
  resource(codes),not working_hours.
:- not check.
```

Given the corresponding ASP program of ps_1, the negation of the property, and Π_{config}, Gringo and ClaspD return no answer set from which we conclude that the property is held.

As another example, consider the query if a developer is always allowed to read codes during non-working hours. This query Π_{query} can be represented as

```
working_hours :- 8<=T, T<=17,current_time(T).
check :- decision(ps1,deny),
  subject(developer), action(read),
  resource(codes),not working_hours.
:- not check.
```

A policy designer may intend that this property would follow based on the policy specification. However, the following answer set is found, which indicates a design flaw of the policy.

```
{subject(developer) action(read)
 action(change) resource(codes)
 decision(ps1,deny) decision(p1,deny)
 decision(p2,deny) decision(r2,deny)
 decision(r3,permit) decision(r5,deny)}
```

That is, a developer's request to read the codes is denied if his request also includes changing the codes[5]. From this answer set, the policy designer finds that

[5]XACML supports *multi-valued* requests, which contains multiple id-value pairs in the subject, resource, or action attribute.

p_2, which is supposed to return p*ermit*, returns d*eny*. It is because r_5 returns d*eny*, and the combining algorithm of p_2 is d*eny–overrides*.

In fact, the reason that ps_1 returns d*eny* is because p_1 returns d*eny*. Rule r_1 is not applicable since its *condition* is not satisfied and rule r_2 returns d*eny*. Then, the policy designer realizes the flaw and could disallow the concurrency of two actions within a request. However, even after adding such a constraint, another answer set is found as follows:

```
{subject(developer) subject(tester)
 action(read) resource(codes)
 decision(ps1,deny) decision(p2,deny)
 decision(r3,permit) decision(r4,deny)}
```

That is, when someone is both d*eveloper* and t*ester*, he cannot read codes during non-working hours since rule r_4 disallows it. In this answer set, ps_1 returns d*eny* because p_1 is not applicable and p_2 returns d*eny*. In turn, it is because r_4 returns d*eny*. If we add a constraint disallowing a person to be both d*eveloper* and t*ester* roles simultaneously, the program returns no answer set as intended.

Implementation and Evaluation

A tool called XACML2ASP have been implemented in Java 1.6.3. XACML2ASP can automatically convert core XACML and RBAC constraint expressions into ASP. The generated ASP-based policy representations are then fed into an ASP reasoner to carry out analysis services. I evaluated the efficiency and effectiveness of the proposes approach on several real-world XACML policies. Gringo was employed as the ASP solver for the evaluation. The experiments were performed on Intel Core 2 Duo CPU 3.00 GHz with 3.25 GB RAM running on Windows XP SP2.

In the evaluation, I utilized ten real-world XACML policies collected from three different sources. Six of the policies, *CodeA*, *CodeB*, *CodeC*, *CodeD*, *Continue-a* and *Continue-b* are XACML policies used by [20]; among them, *Continue-a* and *Continue-b* are designed for a real-world Web application supporting a conference management. Three of the policies *Weirdx*, *FreeCS* and *GradeSheet* are utilized by [101]. The *Pluto* policy is employed in ARCHON system,[6] which is a digital library that federates the collections of physics with multiple degrees of meta data richness.

Table 4.2: Experimental results on real-life XACML policies.

Policy	# of Rules	Converting Time(s)	Reasoning Time(s)
CodeA	2	0.000	0.000
CodeB	3	0.000	0.000
CodeC	4	0.000	0.002
CodeD	5	0.000	0.004
Weirdx	6	0.005	0.006
FreeCS	7	0.005	0.006
GradeSheet	14	0.015	0.012
Pluto	21	0.016	0.031
Continue-a	298	0.120	0.405
Continue-b	306	0.125	0.427

Table 4.2 shows the number of rules contained in each policy, the conversion time from XACML to ASP, and the reasoning time using Gringo + claspd for each policy. Note that the reasoning time was measured by enabling Gringo + claspd to generate answer sets representing all permitted requests for each policy. From Table 4.2, we observe that the conversion time from XACML to ASP in `XACML2ASP` is fast enough to handle larger size of policies, such as *Continue-a* and *Continue-b*. It also indicates that the reasoning process for policy analysis in ASP solver is also efficient enough for a variety of policy analysis services.

[6] http://archon.cs.odu.edu/.

In an XACML policy, multiple rules may overlap, which means one access request may match several rules. Moreover, multiple rules within one policy may conflict, implying that those rules not only overlap each other but also yield different decisions. Conflicts in an XACML policy may lead to both safety problem (e.g. allowing unauthorized access) and availability problem (e.g. denying legitimate access). An intuitive means for resolving policy conflicts by a policy designer is to remove all conflicts by modifying the policies. However, resolving conflicts through changing the policies is remarkably difficult, even impossible, in practice from many aspects. First, the number of conflicts in an XACML policy is potentially large, since an XACML policy may consist of hundreds or thousands of rules. Second, conflicts in XACML policies are probably very complicated, because one rule may conflict with multiple other rules, and one conflict may be associated with several rules. Besides, an XACML policy for a distributed application may be aggregated from multiple parties. Also, an XACML policy may be maintained by more than one administrator. Without a priori knowledge on the original intentions of policy specification, changing a policy may affect the policy's semantics and may not resolve conflicts correctly. Furthermore, in some cases, a policy designer may intentionally introduce certain overlaps in XACML policy components by implicitly reflecting that only the first rule is important. In this case, conflicts are not an error, but intended, which would not be necessary to be changed.

Since the *conflicts* in XACML policies always exist and are hard to be eliminated, XACML defines four different combining algorithms to automatically resolve conflicts [6]: *Deny-Overrides*, *Permit-Overrides*, *First-Applicable* and *Only-One-Applicable*. Unfortunately, XACML currently lacks a systematic mechanism

for precisely detecting conflicts. Identifying conflicts in XACML policies is critical for policy designers since the correctness of selecting a combining algorithm for an XACML policy or policy set component heavily relies on the information from conflict diagnosis. Without precise conflict information, the effectiveness of combining algorithms for resolving policy conflicts cannot be guaranteed.

Another critical problem for XACML policy analysis is *redundancy* discovery and removal. A rule in an XACML policy is redundant if every access request that matches the rule also matches other rules with the same effect. As the response time of an access request largely depends on the number of rules to be parsed within a policy, redundancies in a policy may adversely affect the performance of policy evaluation. Therefore, policy redundancy is treated as policy *anomaly* as well. With the significant growth of Web applications deployed on the Internet, XACML policies grow rapidly in size and complexity. Hence, redundancy elimination can be treated as one of effective solutions for optimizing XACML policies and improving the performance of XACML evaluation.

Recently, policy anomaly detection has received a great deal of attention [22, 102, 103, 23], especially, in firewall policy analysis. Corresponding policy analysis tools, such as Firewall Policy Advisor [22] and FIREMAN [23], with the goal of discovering firewall policy anomalies have been developed. However, we cannot directly adopt those prior analysis approaches for XACML due to several reasons. First, most prior approaches mainly have the capability to detect *pairwise* policy anomalies, while a complete anomaly detection should consider all policy components as a whole piece. In other words, prior policy analysis approaches are still needed to be improved [104]. Second, the structure of firewall policies is flat but XACML has a hierarchical structure supporting recursive policy specification. Third, a firewall policy only supports one conflict resolution strategy (*first-match*)

to resolve conflicts but XACML has four rule/policy combining algorithms. Last but not the least, a firewall rule is typically specified with fixed fields, while an XACML rule can be multi-valued. Therefore, a new policy analysis mechanism is desirable to cater those requirements from anomaly analysis in XACML policies.

In this part, I introduce a policy-based segmentation technique, which adopts a binary decision diagram (BDD)-based data structure to perform set operations, for policy anomaly discovery and resolution. Based on this technique, an *authorization space* defined by an XACML policy or policy set component can be divided into a set of disjoint segments. Each segment associated with a unique set of XACML components indicates an overlap relation (either conflicting or redundant) among those components. Accurate anomaly information is crucial to the success of anomaly resolution. For example, conflict diagnosis information provided by a policy analysis tool can be utilized to guide the policy designers in selecting appropriate combining algorithms. Moreover, I observe that current XACML conflict resolution mechanisms are too restrictive by applying only one combining algorithm to resolve all identified conflicts within an XACML policy or policy set component. Also, many other desirable conflict resolution strategies exist [55, 54, 56], but cannot be directly supported by XACML. Thus, I additionally present a flexible and extensible policy conflict resolution mechanism in this dissertation. Besides, I discuss the implementation of a policy analysis tool `XAnalyzer`, which is based on the proposed approach. To evaluate the practicality of the tool, the experiments deal with both real-life and synthetic XACML policies.

Anomalies in XACML Policies

An XACML policy may contain both policy components and policy set components. Often, a rule anomaly occurs in a policy component, which consists of a sequence of rules. On the other hand, a policy set component consists of a set of

```
1<PolicySet PolicySetId="PS1" PolicyCombiningAlgId="First-Applicable">
2   <Target/>
3   <Policy PolicyId="P1" RuleCombiningAlgId="Deny-Overrides">
4     <Target/>
5     <Rule RuleId="r1" Effect="Deny">
6       <Target>
7         <Subjects><Subject>      Designer  </Subject>
8                   <Subject>      Tester    </Subject></Subjects>
9         <Resources><Resource>    Codes     </Resource></Resources>
10        <Actions><Action>        Change    </Action></Actions>
11      </Target>
12    </Rule>
13    <Rule RuleId="r2" Effect="Permit">
14      <Target>
15        <Subjects><Subject>      Designer  </Subject>
16                  <Subject>      Developer </Subject></Subjects>
17        <Resources><Resource>    Reports   </Resource>
18                   <Resource>    Codes     </Resource></Resources>
19        <Actions><Action>        Read      </Action>
20                 <Action>       Change    </Action></Actions>
21      </Target>
22      <Condition>    8:00 ≤ Time ≤ 17:00     </Condition>
23    </Rule>
24    <Rule RuleId="r3" Effect="Deny">
25      <Target>
26        <Subjects><Subject>      Designer  </Subject></Subjects>
27        <Resources><Resource>    Reports   </Resource>
28                   <Resource>    Codes     </Resource></Resources>
29        <Actions><Action>        Change    </Action></Actions>
30      </Target>
31      <Condition>    12:00 ≤ Time ≤ 13:00    </Condition>
32    </Rule>
33  </Policy>
34  <Policy PolicyId="P2" RuleCombiningAlgId="Permit-Overrides">
35    <Target/>
36    <Rule RuleId="r4" Effect="Deny">
37      <Target>
38        <Subjects><Subject>      Developer</Subject></Subjects>
39        <Resources><Resource>    Reports   </Resource></Resources>
40        <Actions><Action>        Change    </Action></Actions>
41      </Target>
42    </Rule>
43    <Rule RuleId="r5" Effect="Permit">
44      <Target>
45        <Subjects><Subject>      Manager   </Subject>
46                  <Subject>      Designer  </Subject></Subjects>
47        <Resources><Resource>    Reports   </Resource>
48                   <Resource>    Codes     </Resource></Resources>
49        <Actions><Action>        Change    </Action></Actions>
50      </Target>
51    </Rule>
52  </Policy>
53</PolicySet>
```

Figure 4.19: Anomalies in an example XACML policy.

policies or other policy sets, thus anomalies may also arise among policies or policy sets. Thus, I address XACML policy anomalies at both policy level and policy set level.

- **Anomalies at Policy Level:** A rule is ***conflicting*** with other rules, if this rule overlaps with others but defines a different effect. For example, the *deny* rule r_1 is in conflict with the *permit* rule r_2 in Figure 4.19 because rule r_2 allows the access requests from a designer to change codes in the time interval [8:00, 17:00], which are supposed to be denied by r_1; and a rule is ***redundant*** if there is other same or more general rules available that have the same effect. For instance, if we change the effect of r_2 to *Deny*, r_3 becomes redundant since r_2 will also deny a designer to change reports or codes in the time interval [12:00, 13:00].

- **Anomalies at Policy Set Level:** Anomalies may also occur across policies or policy sets in an XACML policy. For example, considering two policy components P_1 and P_2 of the policy set PS_1 in Figure 4.19, P_1 is ***conflicting*** with P_2, because P_1 permits the access requests that a developer changes reports in the time interval [8:00, 17:00], but which are denied by P_2. On the other hand, P_1 denies the requests allowing a designer to change reports or codes in the time interval [12:00, 13:00], which are permitted by P_2. Supposing the effect of r_2 is changed to *Deny* and the condition of r_2 is removed, r_4 is turned to be ***redundant*** with respect to r_2, even though r_2 and r_4 are placed in different policies P_1 and P_2, respectively.

A policy anomaly may involve in multiple rules. For example, in Figure 4.19, access requests that a designer changes codes in the time interval [12:00, 13:00] are permitted by r_2, but denied by both r_1 and r_3. Thus, this conflict associates with *three* rules. For another example, suppose the effect of r_3 is changed to *Permit* and the subject of r_3 is replaced by *Manager* and *Developer*. If we only examine *pairwise* redundancies, r_3 is not a redundant rule. However, if we check multiple rules simultaneously, we can identify r_3 is redundant considering r_2 and r_5

together. I observe that precise anomaly diagnosis information is crucial for achieving an effective anomaly resolution. In this work, I attempt to design a systematic approach and corresponding tool not only for accurate anomaly detection but also for effective anomaly resolution.

Underlying Data Structure

The proposed policy-based segmentation technique introduced in subsequent sections requires a well-formed representation of policies for performing a variety of set operations. Binary Decision Diagram (BDD) [105] is a data structure that has been widely used for formal verification and simplification of digital circuits. In this work, I leverage BDD as the underlying data structure to represent XACML policies and facilitate effective policy analysis.

Given an XACML policy, it can be parsed to identify subject, action, resource and condition attributes. Once these attributes are identified, all XACML rules can be transformed into Boolean expressions [106]. Each Boolean expression of a rule is composed of atomic Boolean expressions combined by logical operators $\vee$ and $\wedge$. Atomic Boolean expressions are treated as equality constraints or range constraints on attributes (e.g. $Subject =$ "$student$") or on conditions (e.g. $8:00 \leq Time \leq 17:00$).

Example 1 *Consider the example XACML policy in Figure 4.19 in terms of* atomic Boolean expressions. *The Boolean expression for rule* r_1 *is:*

$(Subject =$ "$Designer$" $\vee Subject =$ "$Tester$"$) \wedge (Resource =$ "$Codes$"$) \wedge (Action =$ "$Change$"$)$

The Boolean expression for rule r_2 is:

$$(Subject = \text{“}Designer\text{”} \vee Subject = \text{“}Developer\text{”}) \wedge (Resource = \text{“}Reports\text{”} \vee Resource = \text{“}Codes\text{”}) \wedge (Action = \text{“}Read\text{”} \vee Action = \text{“}Change\text{”}) \wedge (8:00 \leq Time \leq 17:00)$$

Boolean expressions for XACML rules may consist of atomic Boolean expressions with overlapping value ranges. In such cases, those atomic Boolean expressions are needed to be transformed into a sequence of new atomic Boolean expressions with disjoint value ranges. Agrawal et al. [107] have identified different categories of such atomic Boolean expressions and addressed corresponding solutions for those issues. I adopt similar approach to construct the Boolean expressions for XACML rules.

Table 4.3: Atomic Boolean expressions and corresponding Boolean variables for P_1.

Unique Atomic Boolean Expression	**Boolean Variable**
$Subject = \text{“}Designer\text{”}$	S_1
$Subject = \text{“}Tester\text{”}$	S_2
$Subject = \text{“}Developer\text{”}$	S_3
$Subject = \text{“}Manager\text{”}$	S_4
$Resource = \text{“}Reports\text{”}$	R_1
$Resource = \text{“}Codes\text{”}$	R_2
$Action = \text{“}Read\text{”}$	A_1
$Action = \text{“}Change\text{”}$	A_2
$8:00 \leq Time < 12:00$	C_1
$12:00 \leq Time < 13:00$	C_2
$13:00 \leq Time \leq 17:00$	C_3

Each of the atomic Boolean expression is encoded as a Boolean variable. For example, an atomic Boolean expression *Subject="Designer"* is encoded into a Boolean variable S_1. A complete list of Boolean encoding for the example XACML policy in Figure 4.19 is shown in Table 4.3. I then utilize the Boolean encoding to construct Boolean expressions in terms of Boolean variables for XACML rules.

Example 2 *Consider the example XACML policy in Figure 4.19 in terms of* Boolean variables. *The Boolean expression for rule* r_1 *is:*

$$(S_1 \vee S_2) \wedge (R_2) \wedge (A_2)$$

The Boolean expression for rule r_2 *is:*

$$(S_1 \vee S_3) \wedge (R_1 \vee R_2) \wedge (A_1 \vee A_2) \wedge (C_1 \vee C_2 \vee C_3)$$

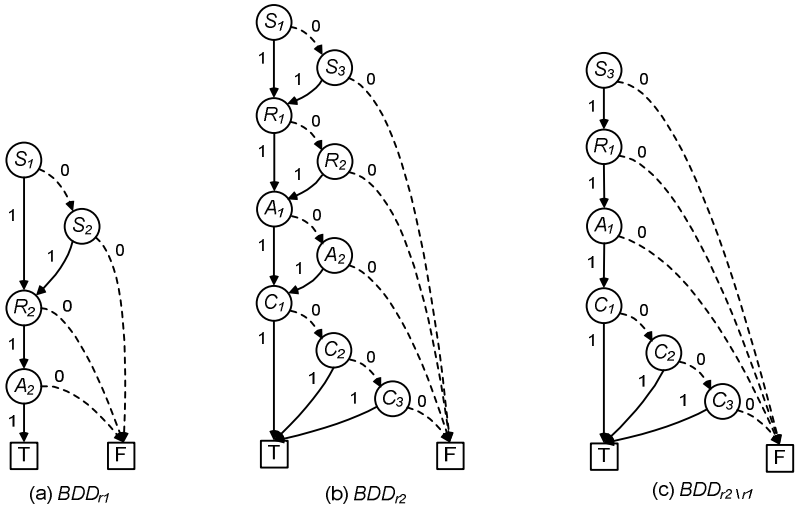


Figure 4.20: Representing and operating on rules of XACML policy with BDD.

BDDs are acyclic directed graphs which represent Boolean expressions compactly. Each nonterminal node in a BDD represents a Boolean variable, and has two edges with binary labels, 0 and 1 for *nonexistent* and *existent*, respectively. Terminal nodes represent Boolean value `T` (True) or `F` (False). Figures 4.20(a) and 4.20(b) give BDD representations of two rules r_1 and r_2, respectively.

Once the BDDs are constructed for XACML rules, performing set operations, such as unions ($\cup$), intersections ($\cap$) and set differences ($\setminus$), required by the policy-based segmentation algorithms (see Algorithm 1 and Algorithm 2) is efficient as well as straightforward. Figure 4.20(c) shows an integrated BDD, which

is the difference of r_2' BDD from r_1' BDD ($r_2 \setminus r_1$). Note that the resulting BDDs from the set operations may have *less* number of nodes due to the canonical representation of BDD.

Conflict Detection and Resolution

I first introduce a concept of *authorization space*, which adopts aforementioned BDD-based policy representation to perform policy anomaly analysis. This concept is defined as follows:

Definition 4 ***(Authorization Space)****. Let R_x, P_x and PS_x be the set of rules, policies and policy sets, respectively, of an XACML policy x. An authorization space for an XACML policy component $c \in R_x \cup P_x \cup PS_x$ represents a collection of all access requests Q_c to which a policy component c is applicable.*

Conflict Detection Approach The proposed conflict detection mechanism examines conflicts at both policy level and policy set level for XACML policies. In order to precisely identify policy conflicts and facilitate an effective conflict resolution, I introduce a policy-based segmentation technique to partition the entire authorization space of a policy into disjoint authorization space segments. Then, conflicting authorization space segments (called *conflicting segment* in the rest of this dissertation), which contain policy components with different effects, are identified. Each conflicting segment indicates a policy conflict.

Conflict Detection at Policy Level. A policy component in an XACML policy includes a set of rules. Each rule defines an authorization space with the effect of either `permit` or `deny`. An authorization space with the effect of `permit` is called as *permitted space* and an authorization space with the effect of `deny` is called as *denied space* in this dissertation.

Algorithm 1 shows the pseudocode of generating conflicting segments for a policy component P. An entire authorization space derived from a policy component is first partitioned into a set of disjoint segments. As shown in lines 17-33 in Algorithm 1, a function called `Partition()` accomplishes this procedure. This function works by adding an authorization space s derived from a rule r to an authorization space set S. A pair of authorization spaces must satisfy one of the following relations: *subset* (line 19), *superset* (line 24), *partial match* (line 27), or *disjoint* (line 32). Therefore, one can utilize set operations to separate the overlapped spaces into disjoint spaces.

Conflicting segments are identified as shown in lines 6-10 in Algorithm 1. A set of conflicting segments $CS : \{cs_1, cs_2, \ldots, cs_n\}$ from conflicting rules has the following three properties:

1. All conflicting segments are pairwise disjoint:
   $cs_i \cap cs_j = \emptyset, 1 \leq i \neq j \leq n$;

2. Any two different requests q and $q^{'}$ within a single conflicting segment (cs_i) are matched by exact same set of rules:
   $GetRule(q) = GetRule(q^{'})$, [7] $\forall q \in cs_i, q^{'} \in cs_i, q \neq q^{'}$; and

3. The effects of matched rules in any conflicting segments contain both "`Permit`" and "`Deny`."

To facilitate the correct interpretation of analysis results, a concise and intuitive representation method is necessary. For the purposes of brevity and understandability, I first employ a two dimensional geometric representation for each authorization space segment. Note that a rule in an XACML policy typically has

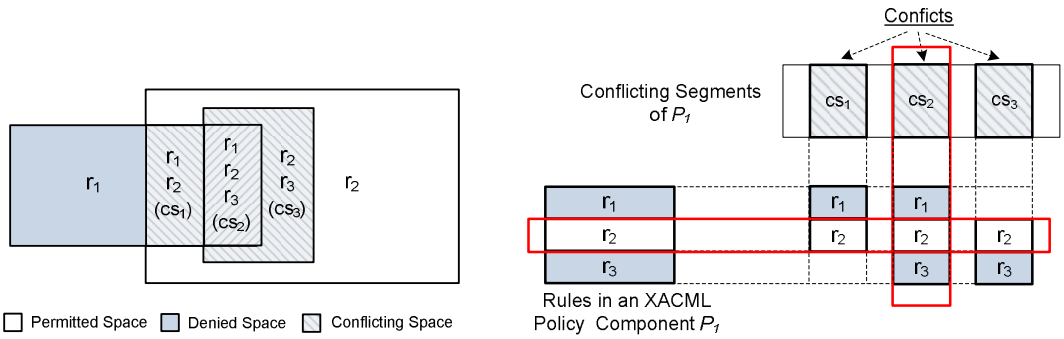
[7] $GetRule()$ is a function that returns all rules matching a request.

Algorithm 1: Identify Disjoint Conflicting Authorization Spaces of Policy P

Input: A policy P with a set of rules.
Output: A set of disjoint conflicting authorization spaces CS for P.

```
1  /* Partition the entire authorization space of P into disjoint spaces*/
2  S.New();
3  S ⟵ Partition_P(P);
4  /* Identify the conflicting segments */
5  CS.New();
6  foreach s ∈ S do
7      /* Get all rules associated with a segment s */
8      R' ⟵ GetRule(s);
9      if ∃r_i ∈ R', r_j ∈ R', r_i ≠ r_j and r_i.Effect ≠ r_j.Effect then
10         CS.Append(s);
11 Partition_P(P)
12 R ⟵ GetRule(P);
13 foreach r ∈ R do
14     s_r ⟵ AuthorizationSpace(r);
15     S ⟵ Partition(S, s_r);
16 return S;
17 Partition(S, s_r)
18 foreach s ∈ S do
19     /* s_r is a subset of s*/
20     if s_r ⊂ s then
21         S.Append(s \ s_r);
22         s ⟵ s_r;
23         Break;
24     /* s_r is a superset of s*/
25     else if s_r ⊃ s then
26         s_r ⟵ s_r \ s;
27     /* s_r partially matches s*/
28     else if s_r ∩ s ≠ ∅ then
29         S.Append(s \ s_r);
30         s ⟵ s_r ∩ s;
31         s_r ⟵ s_r \ s;
32 S.Append(s_r);
33 return S;
```

multiple fields, thus a complete representation of authorization space should be multi-dimensional. Also, I utilize colored rectangles to denote two kinds of authorization spaces: *permitted space* (white color) and *denied space* (grey color), respectively. Figure 4.21(a) gives a representation of the segments of authorization space derived from the policy P_1 in the XACML example policy shown in Figure 4.19. We can notice that five unique disjoint segments are generated. In particular, three conflicting segments cs_1, cs_2 and cs_3 are identified, representing three policy conflicts.



(a) Disjoint segments of authorization space for policy P_1.

(b) Grid representation of policy conflict in policy P_1.

Figure 4.21: Authorization space representation for policy P_1 in the example XACML policy.

When a set of XACML rules interacts, one overlapping relation may be associated with several rules. Meanwhile, one rule may overlap with multiple other rules and can be involved in a couple of overlapping relations (overlapping segments). Different kinds of segments and associated rules can be viewed like Figure 4.21(a). However, it is still difficult for a policy designer or administrator to figure out how many segments one rule is involved in. To address the need of a more precise conflict representation, I additionally introduce a grid representation that is a *matrix-based* visualization of policy conflicts, in which space segments are displayed along the horizontal axis of the matrix, rules are shown along the vertical axis, and the intersection of a segment and a rule is a grid that displays a rule's subspace covered by the segment.

Figure 4.21(b) shows a grid representation of conflicts in the policy P_1 in the example policy. We can easily determine which rules are covered by a segment, and which segments are associated with a rule. For example, as shown in Figure 4.21(b), we can notice that a conflicting segment cs_2, which points out a conflict, is related to a rule set consisting of three rules r_1, r_2 and r_3 (highlighted with a horizontal red rectangle), and a rule r_2 is involved in three conflicting segments cs_1, cs_2 and cs_3 (highlighted with a vertical red rectangle). The grid representation provides a

better understanding of policy conflicts to policy designers and administrators with an overall view of related segments and rules.

Conflict Detection at Policy Set Level. There are two major challenges that need to be taken into consideration when we design an approach for XACML analysis at policy set level.

1. XACML supports four rule/policy combining algorithms: *First-Applicable*, *Only-One-Applicable*, *Deny-Overrides*, and *Permit-Overrides*.

2. An XACML policy is specified recursively and therefore has a hierarchical structure. In XACML, a policy set contains a sequence of policies or policy sets, which may further contain other policies or policy sets.

Each authorization space segment also has an *effect*, which is determined by the XACML components covered by this segment. For nonconflicting segments, the effect of a segment equals to the effect of components covered by this segment. Regarding conflicting segments, the effect of a segment depends on the following four cases of combining algorithm ($\mathscr{CA}$), which is used by the owner (a policy or a policy set) of the segment.

1. $\mathscr{CA}$=*First-Applicable*: In this case, the effect of a conflicting segment equals to the effect of the first component covered by the conflicting segment.

2. $\mathscr{CA}$=*Permit-Overrides*: The effect of a conflicting segment is always assigned with "`Permit`," since there is at least one component with "`Permit`" effect within this conflicting segment.

3. $\mathscr{CA}$=*Deny-Overrides*: The effect of a conflicting segment always equals to "`Deny`."

4. $\mathscr{CA}$=*Only-One-Applicable*: The effect of a conflicting segment equals to the effect of only-applicable component.

To support the recursive specifications of XACML policies, an XACML policy can be parsed and modeled as a tree structure, where each terminal node represents an individual rule, each nonterminal node whose children are all terminal nodes represents a policy, and each nonterminal node whose children are all nonterminal nodes represents a policy set. At each nonterminal node, the target and combining algorithm are stored. At each terminal node, the target and effect of the corresponding rule are stored.

Algorithm 2 shows the pseudocode of identifying disjoint conflicting authorization spaces for a policy set *PS* based on the tree structure. In order to partition authorization spaces of all nodes contained in a policy set tree, this algorithm recursively calls the partition functions, `Partition_P()` and `Partition_PS()`, to deal with the policy nodes (lines 16-17) and the policy set nodes (lines 19-20), respectively. Once all children nodes of a policy set are partitioned, we can then represent the authorization space of each child node (E) with two subspaces *permitted subspace* (E^P) and *denied subspace* (E^D) by aggregating all "`Permit`" segments and "`Deny`" segments, respectively, as follows:

$$\begin{cases} E^P = \bigcup_{s_i \in S_E} s_i & \text{if } Effect(s_i) = Permit \\ E^D = \bigcup_{s_i \in S_E} s_i & \text{if } Effect(s_i) = Deny \end{cases} \tag{4.1}$$

where S_E denotes the set of authorization space segments of the child node E.

For example, since the combining algorithm ($\mathscr{CA}$) of the policy P_1 in the example XACML policy is *Deny-Overrides*, the effects of three conflicting segments shown in Figure 4.21 are "`Deny`". Figure 4.22 shows the result of aggre-

Algorithm 2: Identify Disjoint Conflicting Authorization Spaces of Policy Set *PS*

Input: A policy set *PS* with a set of policies or other policy sets.
Output: A set of disjoint conflicting authorization spaces *CS* for *PS*.

```
1  /* Partition the entire authorization space of PS into disjoint spaces*/
2  S.New();
3  S ⟵ Partition_PS(PS);
4  /* Identify the conflicting segments */
5  CS.New();
6  foreach s ∈ S do
7      E ⟵ GetElement(s);
8      if ∃e_i ∈ E, e_j ∈ E, e_i ≠ e_j and e_i.Effect ≠ e_j.Effect then
9          CS.Append(s);

10 Partition_PS(PS)
11 S''.New();
12 C ⟵ GetChild(PS);
13 foreach c ∈ C do
14     S'.New();
15     /* c is a policy*/
16     if IsPolicy(c) = true then
17         S' ⟵ Partition_P(c);
18     /* c is a policy set*/
19     else if IsPolicySet(c) = true then
20         S' ⟵ Partition_PS(c)
21     E^P.New();
22     E^D.New();
23     foreach s' ∈ S' do
24         if Effect(s') = Permit then
25             E^P ⟵ E^P ∪ s';
26         else if Effect(s') = Deny then
27             E^D ⟵ E^D ∪ s';
28     S'' ⟵ Partition(S'', E^P);
29     S'' ⟵ Partition(S'', E^D);
30 return S'';
```

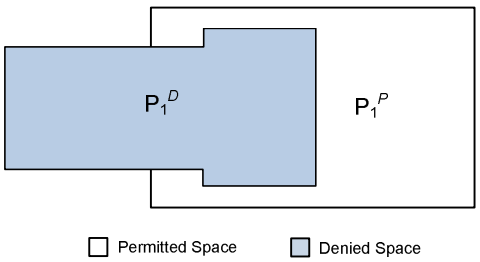
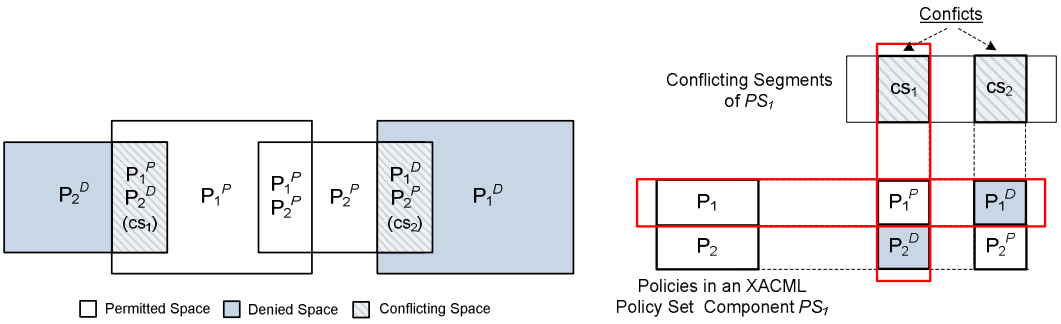


Figure 4.22: Aggregation of authorization spaces for policy P_1 in the example XACML policy.

gating authorization spaces of the policy P_1, where two subspaces P_1^P and P_1^D are constructed.



(a) Disjoint segments of authorization space for policy set PS_1.

(b) Grid representation of policy conflicts in policy set PS_1.

Figure 4.23: Authorization space representation for policy set PS_1 in the example XACML policy.

In order to generate segments for the policy set PS, we can then leverage two subspaces (E^P and E^D) of each child node (E) to partition existing authorization space set belonging to PS (lines 28-29). Figure 4.23(a) represents an example of the segments of authorization space derived from policy set PS_1 in the example policy (Figure 4.19). We can observe that seven unique disjoint segments are generated, and two of them cs_1 and cs_2 are conflicting segments. I additionally give a grid representation of conflicts in the policy set PS_1 shown in Figure 4.23(b). Then, we can easily identify that the conflicting segment cs_1 is related to two subspaces: P_1's *permitted* subspace P_1^P and P_2's *denied* subspace P_2^D, and the policy P_1 is associated with two conflicts, where P_1's *permitted* subspace P_1^P is involved in the conflict represented by cs_1 and P_1's *denied* subspace P_1^D is related to the conflict represented by cs_2.

Fine-Grained Conflict Resolution Once conflicts within a policy component or policy set component are identified, a policy designer can choose appropriate conflict resolution strategies to resolve those identified conflicts. However, current XACML conflict resolution mechanisms have limitations in resolving conflicts effectively. First, existing conflict resolution mechanisms in XACML are too restric-

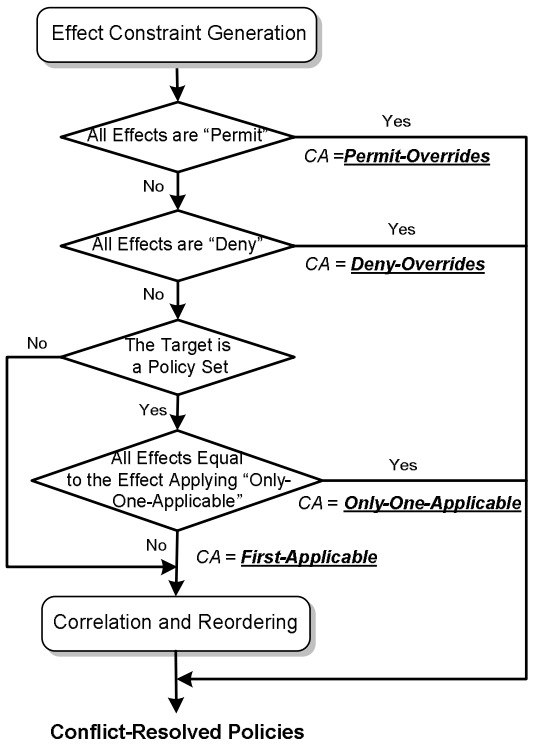


Figure 4.24: Fine-grained conflict resolution framework.

tive and only allow a policy designer to select one combining algorithm to resolve all identified conflicts within a policy or policy set component. A policy designer may want to adopt different combining algorithms to resolve different conflicts. Second, XACML offers four conflict resolution strategies. However, many conflict resolution strategies exist [54, 55, 56], but cannot be specified in XACML. Thus, it is necessary to seek a comprehensive conflict resolution mechanism for more effective conflict resolution. Towards this end, I introduce a flexible and extensible conflict resolution framework to achieve a fine-grained conflict resolution as shown in Figure 4.24.

Effect Constraint Generation from Conflict Resolution Strategy. The conflict resolution framework introduces an *effect constraint* that is assigned to each conflicting segment. An effect constraint for a conflicting segment defines a desired response (either `permit` or `deny`) that an XACML policy should take

when any access request matches the conflicting segment. The effect constraint is derived from the conflict resolution strategy applied to the conflicting segment. A policy designer chooses an appropriate conflict resolution strategy for each identified conflict by examining the features of conflicting segment and associated conflicting components. In the conflict resolution framework, a policy designer is able to adopt different strategies to resolve conflicts indicated by different conflicting segments. In addition to four standard XACML conflict resolution strategies, user-defined strategies [56], such as *Recency-Overrides*, *Specificity-Overrides* and *High-Majority-Overrides*, can be implied in the framework as well. For example, applying a conflict resolution strategy, *High-Majority-Overrides*, to the second conflicting segment cs_2 of policy P_1 depicted in Figure 4.21, an effect constraint *Effect = "Deny"* will be generated for cs_2.

Conflict Resolution Based on Effect Constraints. A key feature of adopting *effect constraints* in the framework is that other conflict resolution strategies assigned to resolve different conflicts by a policy designer can be *automatically* mapped to standard XACML combining algorithms, without changing the way that current XACML implementations perform. As illustrated in Figure 4.24, an XACML combining algorithm can be derived for a target component by examining all effect constraints of the conflicting segments. If all effect constraints are "`Permit`," *Permit-Overrides* is selected for the target component to resolve all conflicts. In case that all effect constraints are "`Deny`," *Deny-Overrides* is assigned to the target component. Then, if the target component is a policy set and all effect constraints can be satisfied by applying *Only-One-Applicable* combining algorithm, *Only-One-Applicable* is selected as the combining algorithm of the target component. Otherwise, *First-Applicable* is selected as the combining algorithm of the target component. In order to resolve all conflicts within the target component by

applying *First-Applicable*, the process of reordering conflicting components is compulsory. Therefore, the first-applicable component in each conflicting segment has the same effect with corresponding effect constraint.

Redundancy Discovery and Removal

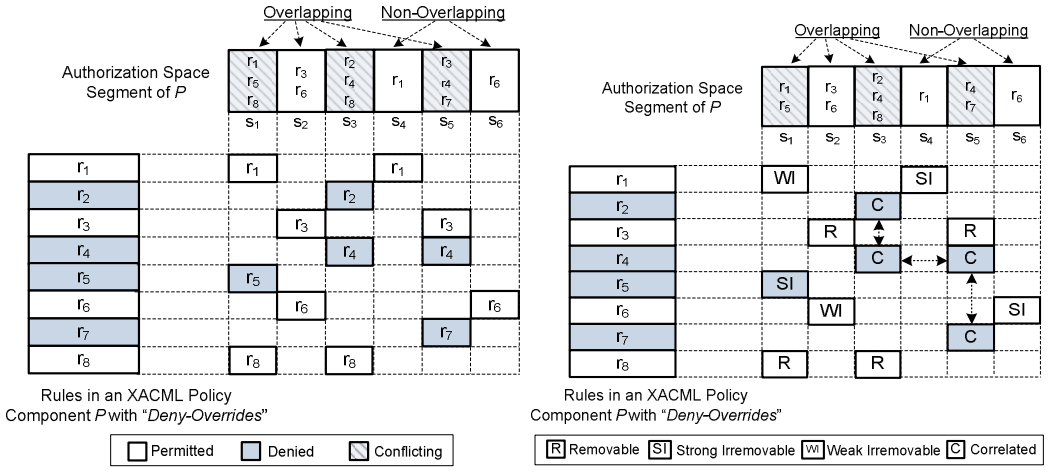
The proposed redundancy discovery and removal mechanism also leverage the policy-based segmentation technique to explore redundancies at both policy level and policy set level. I give a definition of rule redundancy as follows, which serves as a foundation of the redundancy elimination approach.

Definition 5 *(**Rule Redundancy**). A rule r is redundant in an XACML policy p iff the authorization space derived from the resulting policy p' after removing r is equivalent to the authorization space defined by p.*

Redundancy Elimination at Policy Level I employ following four steps to identify and eliminate rule redundancies at policy level: authorization space segmentation, property assignment for rule subspaces, rule correlation break, and redundant rule removal.

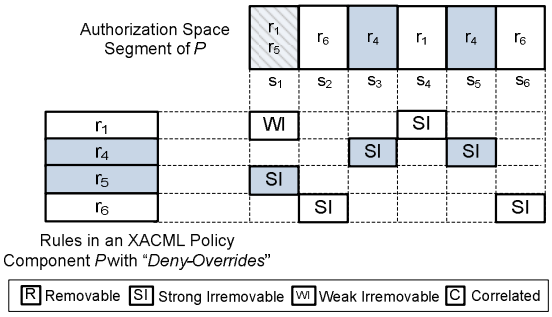
Authorization Space Segmentation. I first perform the policy segmentation function `Partition_P()` defined in Algorithm 1 to divide the entire authorization space of a policy into disjoint segments. I classify the policy segments in following categories: *non-overlapping* segment and *overlapping* segment, which is further divided into *conflicting overlapping* segment and *non-conflicting overlapping* segment. Each *non-overlapping* segment associates with one unique rule and each *overlapping* segment is related to a set of rules, which may conflict with each other (*conflicting overlapping* segment) or have the same effect (*non-conflicting overlapping* segment). Figure 4.25(a) illustrates an authorization space segmentation for a policy with eight rules. In this example, two policy segments s_4 and s_6

are *non-overlapping* segments. Other policy segments are *overlapping* segments, including two *conflicting overlapping* segments s_1 and s_3, and two *non-conflicting overlapping* segments s_2 and s_5.



(a) Authorization space segmentation.

(b) Property assignment.



(c) Redundancy removal.

Figure 4.25: Example of eliminating redundancies at policy level.

Property Assignment for Rule Subspaces. In this step, every rule subspace covered by a policy segment is assigned with a property. Four property values, *removable* (R), *strong irremovable* (SI), *weak irremovable* (WI) and *correlated* (C), are defined to reflect different characteristics of rule subspace. *Removable* property is used to indicate that a rule subspace is removable. In other words, removing such a rule subspace does not make any impact on the original authorization space of an associated policy. *Strong irremovable* property means that a rule subspace cannot

be removed because the effect of corresponding policy segment can be only decided by this rule. *Weak irremovable* property is assigned to a rule subspace when any subspace belonging to the same rule has *strong irremovable* property. That means a rule subspace becomes irremovable due to the reason that other portions of this rule cannot be removed. *Correlated* property is assigned to multiple rule subspaces covered by a policy segment, if the effect of this policy segment can be determined by any of these rules. I next introduce three processes to perform the property assignments to all of rule subspaces within the segments of a policy, considering different categories of policy segments.

Process1: *Property assignment for the rule subspace covered by a non-overlapping segment*. A non-overlapping segment contains only one rule subspace. Thus, this rule subspace is assigned with *strong irremovable* property. Other rule subspaces associated with the same rule are assigned with *weak irremovable* property, excepting the rule subspaces that already have *strong irremovable* property.

Process2: *Property assignment for rule subspaces covered by a conflicting segment*. I present this property assignment process based on the following three cases of rule combining algorithm ($\mathscr{CA}$).

1. $\mathscr{CA}$*=First-Applicable*: In this case, the first rule subspace covered by the conflicting segment is assigned with *strong irremovable* property. Other rule subspaces in the same segment are assigned with *removable* property. Meanwhile, other rule subspaces associated with the same rule are assigned with *weak irremovable* property except the rule subspaces already having *strong irremovable* property.
2. $\mathscr{CA}$*=Permit-Overrides*: All subspaces of "`deny`" rules in this conflicting segment are assigned with *removable* property. If there is only

one "permit" rule subspace, this case is handled which is similar to the *First-Applicable* case. If any "permit" rule subspace has been assigned with *weak irremovable* property, other rule subspaces without *irremovable* property are assigned with *removable* property. Otherwise, all "permit" rule subspaces are assigned with *correlated* property.

3. $\mathscr{CA}$*=Deny-Overrides*: This case is dealt with as the same as *Permit-Overrides* case.

Process3: *Property assignment for rule subspaces covered by a non-conflicting overlapping segment*. If any rule subspace has been assigned with *weak irremovable* property, other rule subspaces without *irremovable* property are assigned with *removable* property. Otherwise, all subspaces within the segment are assigned with *correlated* property.

Figure 4.25(b) shows the result of applying the property assignment mechanism, which performs three property assignment processes in sequence, to the example presented in Figure 4.25(a). We can easily identify that r_3 and r_8 are *removable* rules, where all subspaces are with *removable* property. However, we need to further examine the *correlated* rules r_2, r_4 or r_7, which contain some subspaces with *correlated* property.

Rule Correlation Break and Redundancy Removal. Rule subspaces covered by an overlapping segment are correlated with each other when the effect of overlapping segment can be determined by any of those correlated rules. Thus, keeping one correlated rule and removing others may not change the effect of corresponding segment. Such a correlated relation is called as *vertical rule correlation*, which can be identified in property assignment step. In addition, we can observe that some rule may be involved in several correlated relations. For example, in Figure 4.25(b), r_4 has two subspaces that are involved in the correlated relations with

r_2 and r_7, respectively. This kind of correlated relation is called as *horizontal rule correlation*. Obviously, we cannot resolve a correlation individually and those two dimensions of rule correlation should be take into consideration. Therefore, we can further construct rule correlation groups based on those two kinds of rule correlations so that dependent relationships among multiple correlated rules within one group can be examined together. For example, a correlation group g consisting of three rules r_2, r_4 and r_7 can be identified in Figure 4.25(b) based on two dimensions of rule correlation.

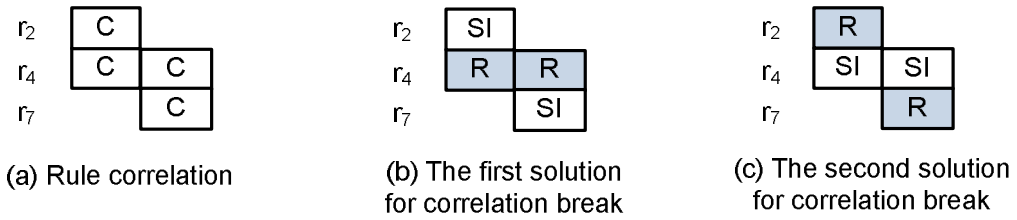


Figure 4.26: Example of rule correlation break.

We can additionally observe that different sequences to break rule correlations in a correlation group may lead to different results for redundancy removal. Figure 4.26(a) shows correlated relations of rules r_2, r_4 and r_7 in the correlation group g. We can break their correlation relations via different sequences. Figure 4.26(b) shows one possible solution. If we first assign two subspaces of r_4 with *removable* property, r_4 becomes a *removable* rule but r_2 and r_7 are turned to an *irremovable* rules. However, regarding another solution represented in Figure 4.26(c), if we first assign the correlated subspace of r_2 with *removable* property, then only r_4 becomes an *irremovable* rule. Both r_2 and r_7 are *removable*. Thus, it is necessary to seek an optimal solution to obtain *maximum* redundancy removal. To achieve this goal, we can compute a correlation degree (*CD*) for each correlated rule r using the following equation:

Algorithm 3: Redundancy Elimination of Policy P: RedundancyEliminate_P(P)

Input: A policy P with a set of rules.
Output: A redundancy-eliminated policy $P^{'}$.

```
1  /* Partition the entire authorization space of P into disjoint spaces*/
2  S.New();
3  S ⟵ Partition_P(P);
4  /* Property assignment for all rule subspaces */
5  PropertyAssgin_P(S);
6  /* Rule correlation break */
7  G ⟵ CorrelatonGroupConstruct(S);
8  foreach g ∈ G do
9      foreach r ∈ g do
10         r.CD ⟵ Σ_{s_i∈CS(r)} 1/(NC(s_i)−1);
11     SP ⟵ GetCorrelatedSubspace(MinCDRule(g))
12     foreach sp ∈ SP do
13         sp.Property ⟵ R ;
14         if |GetCorrelatedSubspace(sp)| = 1 then
15             SP' ⟵ GetCorrelatedSubspace(sp);
16             SP'.Property ⟵ SI ;
17             AssginSI(SP');
18 /*Redundancy removal */
19 P' ⟵ P;
20 foreach r ∈ P' do
21     if AllRemovalProperty(r) = true then
22         P' ⟵ P' \ r;
23 return P';
```

$$CD(r) = \sum_{s_i \in CS(r)} \frac{1}{NC(s_i) - 1} \tag{4.2}$$

Note that $CS(r)$ is a function to return all correlated segments of a rule r, and $NC(s_i)$ is a function to return the number of correlated rules within a segment s_i. Since each policy segment contains multiple correlated rules ($NC(s_i) \geq 2$), $\frac{1}{NC(s_i)-1}$ gives the degree of breakable correlation relations associated with a policy segment s_i if we set a rule r as *removable*. To maximize the number of removable rules for redundancy resolution,the correlation break process selects one rule with the minimal CD as the candidate removable rule each time. For instance, applying this equation to calculate correlation degrees of three rules demonstrated in Figure 4.26(a), $CD(r_2)$ and $CD(r_7)$ equal to 1, and $CD(r_4)$ equals to 2. Thus, we can select either

r_2 or r_7 as the candidate removable rule in the first break step. Finally, two rules r_2 and r_7 become removable rules after breaking all correlations.

The pseudocode of the algorithm for eliminating redundancy at policy level is shown in Algorithm 3. Figure 4.25(c) depicts the result of applying this algorithm to the example given in Figure 4.25(a). Four rules r_2, r_3, r_7 and r_8 were identified as redundant rules and removed from the policy. However, if we leverage *traditional* redundancy detection method [103, 22], which was limited to detect *pairwise* redundancies, to this example, only two redundant rules r_2 and r_7 can be discovered.

Redundancy Elimination at Policy Set Level Similar to the solution of conflict detection at policy set level, we can handle the redundancy removal for a policy set based on an XACML tree structure representation. If the children nodes of the policy set is a policy node in the tree, we perform `RedundancyEliminate_P()` function to eliminate redundancies. Otherwise, `RedundancyEliminate_PS()` function is excused recursively to eliminate redundancy in a policy set component.

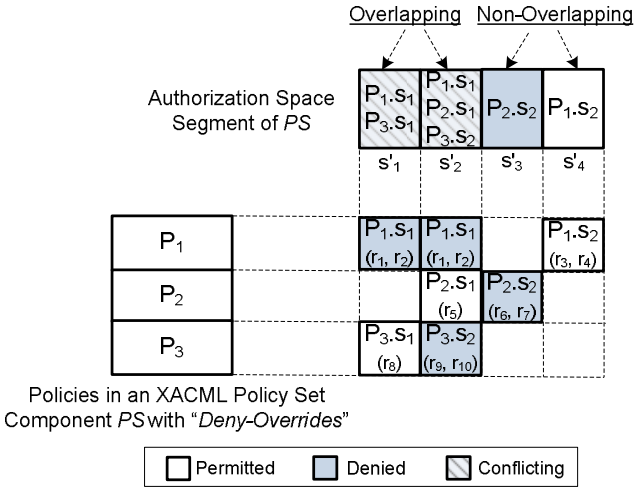


Figure 4.27: Example of authorization space segmentation at policy set level for redundancy discovery and removal.

After each component of a policy set *PS* performs redundancy removal, the authorization space of *PS* can be then partitioned into disjoint segments by performing `Partition()` function. Note that, in the solution for conflict detection at policy set level, authorization subspaces of each child node are aggregated before performing space partition, because we only need to identify conflicts among children nodes to guide the selection of policy combining algorithms for the policy set. However, for redundancy removal at policy set level, both redundancies among children nodes and rule (leaf node) redundancies, which may exist across multiple policies or policy sets, should be discovered. Therefore, we keep the original segments of each child node and leverage those segments to generate the authorization space segments of *PS*. Figure 4.27 demonstrates an example of authorization space segmentation of a policy set *PS* with three children components P_1, P_2 and P_3. The authorization space segments of *PS* are constructed based on the original segments of each child component. For instance, a segment $s_2^{'}$ of *PS* covers three policy segments $P_1.s_1$, $P_2.s_1$ and $P_3.s_2$, where $P_i.s_j$ denotes that a segment s_j belongs to a policy P_i.

The property assignment step at policy set level is similar to the property assignment step at policy level, except that the policy combining algorithm *Only-One-Applicable* needs to be taken into consideration at policy set level. The *Only-One-Applicable* case is handled similar to the *First-Applicable* case. We first check whether the combining algorithm is applicable or not. If the combining algorithm is applicable, the only-applicable subspace is assigned with *strong irremovable* property. Otherwise, all subspaces within the policy set's segment are assigned with *removable* property.

I utilize a similar correlation break mechanism introduced previously to break the correlation relations among the segments of child components of *PS*.

Algorithm 4: Eliminate Redundancies of a Policy Set *PS*: RedundancyEliminate_PS(PS)

Input: A policy set *PS* with a set of policies or other policy sets.
Output: A redundancy-eliminated policy set $PS^{'}$.

```
1  E ⟵ GetChild(PS);
2  foreach e ∈ E do
3      /* e is a policy*/
4      if IsPolicy(e) = true then
5          e' ⟵ RedundancyEliminate_P(e);
6      /* e is a policy set*/
7      else if IsPolicySet(e) = true then
8          e' ⟵ RedundancyEliminate_PS(e)
9      E' ⟵ E' ∪ e';
10 /* Partition the authorization space of PS into disjoint spaces*/
11 foreach e' ∈ E' do
12     foreach s ∈ GetSegment(e') do
13         S ⟵ Partition(S, s);
14 /* Property assignment for all subspaces covered by the segments of PS */
15 PropertyAssgin_PS(S);
16 /* Correlation break */
17 CorrelationBreak_PS(S);
18 /*Redundancy removal for child components of PS */
19 PS' ⟵ PS;
20 foreach e ∈ PS do
21     if AllRemovalProperty(e) = true then
22         PS' ⟵ PS' \ e;
23 /*Redundancy removal for rules of PS */
24 foreach e ∈ PS' do
25     S ⟵ GetSubspace(e);
26     foreach s ∈ S do
27         if s.Property = R then
28             S ⟵ GetSubspaceWithSI(s);
29             foreach s ∈ S do
30                 if OneSISubspace(GetRule(s)) = true then
31                     SP ⟵ GetRuleSubspace(GetRule(s)) \ s;
32                     foreach sp ∈ SP do
33                         sp.Property ⟵ R ;
34             SP' ⟵ GetSubspace(s);
35             foreach sp' ∈ SP' do
36                 sp'.Property ⟵ R ;
37 foreach r ∈ SP' do
38     if AllRemovalProperty(r) = true then
39         SP' ⟵ SP' \ r;
40 return PS';
```

Since there may exist multiple correlated segments of children components with the minimal correlation degree (*CD*) value, I additionally compute a removal value

(*RV*), which indicates the possibility of rule redundancy removal if we set a correlated segment with *removable* property, for all candidate segments using following equation:

$$RV(s) = \sum_{r_i \in R_{HI}(s)} \frac{1}{N_{HI}(r_i)} \tag{4.3}$$

Note that $R_{HI}(s)$ is a function to return all rules having a subspace in the segment s with *strong irremovable* property, and $N_{HI}(r_i)$ is a function to return the number of the rule's subspace with *strong irremovable* property. $\frac{1}{N_{HI}(r_i)}$ measures the possibility of turning a rule to a removable rule if we change a *strong irremovable* rule subspace to be *removable*. The correlated segment with the maximum *RV* value has the highest priority to be chosen for breaking correlations.

After assigning properties to all segments of children components of *PS*, we need to examine whether any child component is redundant. If a child component is redundant, this child component and all rules contained in the child component are removed from *PS*. Then, we need to examine whether there exist any redundant rules. In this process, the properties of all rule subspaces covered by a *removable* segment of a child component of *PS* needs to be changed to *removable*. Note that when we change the property of a *strong irremovable* rule subspace to *removable*, other subspaces in the same rule with dependent *weak irremovable* property need to be changed to *removable* correspondingly. Algorithm 4 shows the pseudocode of eliminating redundancies for a policy set *PS*.

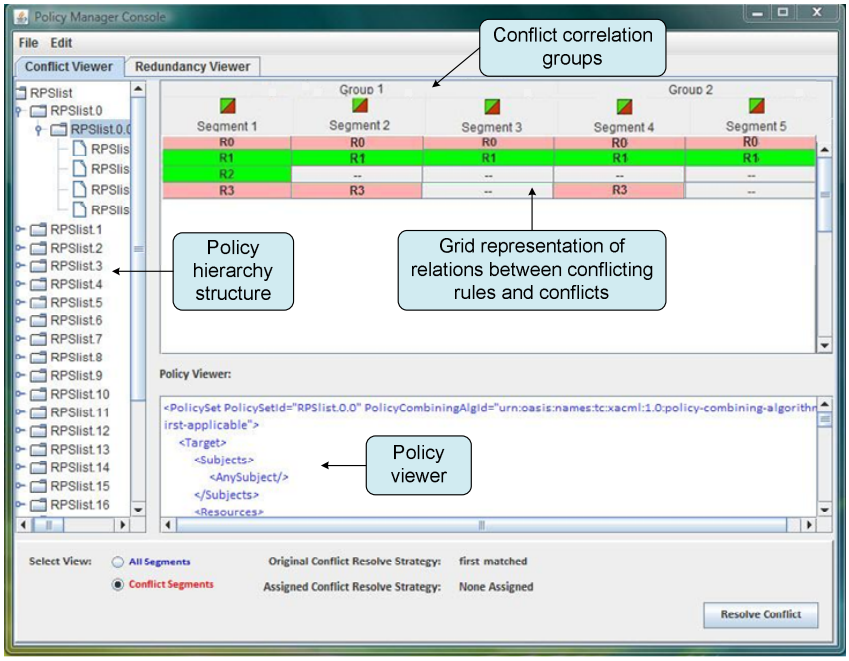
Implementation and Evaluation

A policy analysis tool called `XAnalyzer` have been implemented in Java. Based on the policy anomaly analysis mechanism, it consists of four core components: segmentation module, effect constraint generation module, strategy mapping mod-

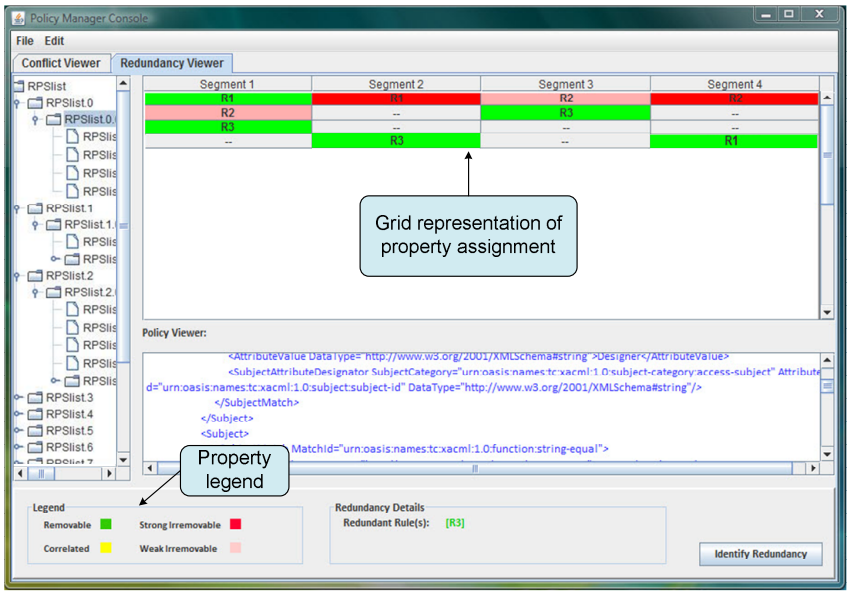
ule, and property assignment module. The segmentation module takes XACML policies as an input and identifies the authorization space segments by partitioning the authorization space into disjoint subspaces. `XAnalyzer` utilizes APIs provided by *Sun* XACML implementation [52] to parse the XACML policies and construct Boolean encoding. JavaBDD [108], which is based on BuDDy package [109], is employed by `XAnalyzer` to support BDD representation and authorization space operations. The effect constraint generation module takes conflicting segments as an input and generates effect constraints for each conflicting segment. Effect constraints are generated based on strategies assigned to each conflicting segment. The strategy mapping module takes conflict correlation groups and effect constraints of conflicting segments as inputs and then maps assigned strategies to standard XACML combining algorithms for examined XACML policy components. The property assignment module automatically assigns corresponding property to each subspace covered by the segments of XACML policy components. The assigned properties are in turn utilized to identify redundancies.

Considering the complexity of tasks involved in the policy analysis, it is desirable to provide intuitive user interfaces for policy designers or administrators for effective policy anomaly detection and resolution. Since the grid representation of policy anomalies offers a succinct view of the interactions of overlapping rules and enables policy designers or administrators to better understand policy anomalies, the grid representation of policy anomalies has been implemented in `XAnalyzer` as well.

`XAnalyzer` provides two policy viewers, *Conflict Viewer* (Figure 4.28(a)) and *Redundancy Viewer* (Figure 4.28(b)), to visualize the outputs of policy conflict analysis and policy redundancy analysis, respectively. In addition, `XAnalyzer` offers flexible ways to handle large-scale policies. There are two kinds of visualiza-



(a) Conflict Viewer.



(b) Redundancy Viewer.

Figure 4.28: `XAnalyzer` interface.

tion interfaces in each viewer: one interface shows an entire snapshot of all anomalies; another interface shows a partial snapshot only containing anomalies within one correlation group. In addition, `XAnalyzer` shows a hierarchical structure of

policies and allow policy designers or administrators to view policy anomalies at different levels independently. The hierarchical structure of policies is presented by a tree of policy components on the left side of the interfaces. A policy designer or administrator can choose a particular policy component such as *Policy* or *Policy Set* node for anomaly analysis. If an administrator chooses a *Policy* node for the analysis, anomalies in that particular *Policy* node are displayed in terms of the rule subspaces involved. Otherwise, if an administrator chooses a *Policy Set* node, all anomalies within that particular node are displayed in terms of *allowed* and *denied* subspaces of policy or policy set components.

I evaluated the efficiency and effectiveness of `XAnalyzer` for policy analysis on both real-life and synthetic XACML policies. The experiments were performed on Intel Core 2 Duo CPU 3.00 GHz with 3.25 GB RAM running on Windows XP SP2. In the evaluation, I utilized five real-life XACML policies, which were collected from different sources. Three of the policies, *CodeA, Continue-a* and *Continue-b* are XACML policies used in [20]; among them, *Continue-a* and *Continue-b* are designed for a real-world Web application supporting a conference management. *GradeSheet* is utilized in [101]. The *Pluto* policy is employed in ARCHON system, [8] which is a digital library that federates the collections of physics with multiple degrees of meta data richness. Since it is hard to get a large volume of real-world policies due to the reason that they are often considered to be highly confidential, I generated four large synthetic policies *SyntheticPolicy-1*, *SyntheticPolicy-2*, *SyntheticPolicy-3* and *SyntheticPolicy-4* for further evaluating the performance and scalability of the tool. These synthetic policies are multi-layered, where each policy component has a randomly selected combining algorithm and each rule has randomly chosen attribute sets from a predefined domain.

[8] http://archon.cs.odu.edu/

Table 4.4: XACML policies used for evaluation.

Policy	Rule (#)	Policy (#)	Policy Set (#)
1 (CodeA)	4	2	5
2 (SamplePolicy)	6	2	1
3 (GradeSheet)	13	1	0
4 (Pluto)	22	1	0
5 (SyntheticPolicy-1)	147	30	11
6 (Continue-a)	312	276	111
7 (Continue-b)	336	305	111
8 (SyntheticPolicy-2)	456	65	40
9 (SyntheticPolicy-3)	572	114	75
10 (SyntheticPolicy-4)	685	188	84

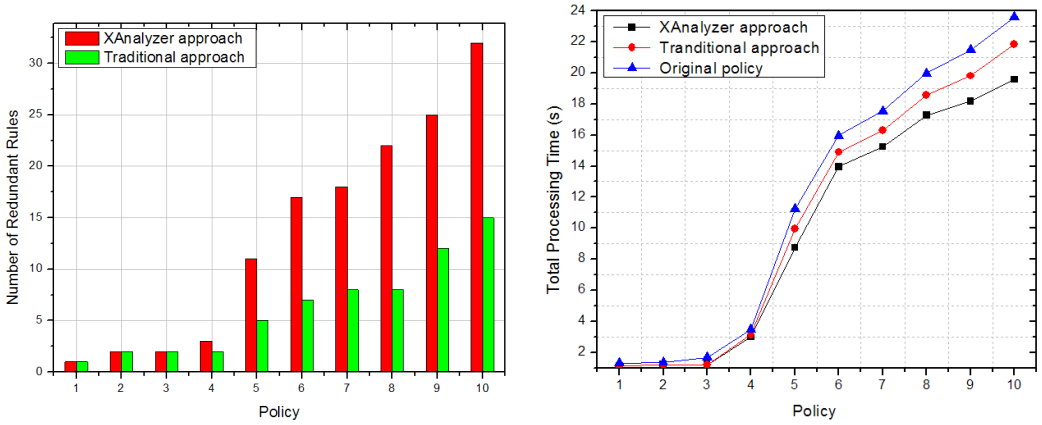
I also use *SamplePolicy*, which is the example XACML policy represented in Figure 4.19, in the experiments. Table 4.4 summarizes the basic information of each policy including the number of rules, the number of policies, and the number of policy sets.

I conducted two separate sets of experiments for the evaluation of conflict detection approach and the evaluation of redundancy removal approach, respectively. Also, I performed evaluations at both policy level and policy set level. Table 4.5 summarizes the evaluation results.

Evaluation of Conflict Detection. Time required by `XAnalyzer` for conflict detection highly depends upon the number of segments generated for each XACML policy. The increase of the number of segments is proportional to the number of components contained in an XACML policy. From Table 4.5, we can observe that `XAnalyzer` performs fast enough to handle larger size XACML policies, even for some complex policies with multiple levels of hierarchies along with hundreds of rules, such as two real-life XACML policies *Continue-a* and *Continue-b* and four synthetic XACML policies. The time trends observed from Table 4.5 are promising, and hence provide the evidence of efficiency of the conflict detection approach.

Table 4.5: Conflict detection and redundancy removal algorithms evaluation.

Policy	Partitions (#)	BDD Nodes (#)	Conflict Detection			Redundant Removal		
			Policy Level(#)	Policy Set Level(#)	Time (s)	Policy Level(#)	Policy Set Level(#)	Time (s)
1 (CodeA)	6	16	1	1	0.082	1	0	0.087
2 (SamplePolicy)	8	34	0	2	0.090	0	2	0.095
3 (GradeSheet)	18	45	0	4	0.098	0	2	0.113
4 (Pluto)	34	78	0	5	0.136	0	3	0.147
5 (SyntheticPolicy-1)	205	112	8	14	0.329	7	4	0.158
6 (Continue-a)	439	135	9	17	0.583	11	7	0.214
7 (Continue-b)	468	146	10	21	0.635	12	7	0.585
8 (SyntheticPolicy-2)	523	209	29	17	0.896	14	8	0.623
9 (SyntheticPolicy-3)	614	227	39	19	0.948	17	10	0.672
10 (SyntheticPolicy-4)	814	265	56	19	1.123	23	12	0.803



(a) Redundancy elimination rate.

(b) Performance improvement.

Figure 4.29: Evaluation of redundancy removal approach.

Evaluation of Redundancy Removal. In the second set of experiments, I evaluated the proposed redundancy analysis approach based on those experimental XACML policies. The evaluation results shown in Table 4.5 also indicate the efficiency of the redundancy analysis algorithm. Moreover, I conducted the evaluation of effectiveness by comparing the redundancy analysis approach with *traditional* redundancy analysis approach [103, 22], which can only identify redundancy relations between *two* rules. Figure 4.29(a) depicts the results of the comparison experiments. From Figure 4.29(a), we can observe that `XAnalyzer` could identify that an average of 6.3% of total rules are redundant. However, *traditional* redundancy analysis approach could only detect an average 3.7% of total rules as redundant rules. Therefore, the enhancement for redundancy elimination was clearly observed

through comparing the proposed redundancy analysis approach with *traditional* redundancy analysis approach in the experiments.

Furthermore, when redundancies in a policy are removed, the performance of policy enforcement is improved generally. For each of XACML policies in the experiments, Figure 4.29(b) depicts the total processing time in *Sun* XACML PDP [52] for responding 10,000 randomly generated XACML requests. The evaluation results clearly show that the processing times are reduced after eliminating redundancies in XACML policies applying either *traditional* approach or this approach, and this approach can obtain better performance improvement than *traditional* approach.

Chapter 5

Applying AMF to Online Social Networks

Online social networks (OSNs) such as Facebook, Google+, and Twitter are inherently designed to enable people to share personal and public information and make social connections with friends, coworkers, colleagues, family and even with strangers. In recent years, we have seen unprecedented growth in the application of OSNs. For example, Facebook, one of representative social network sites, claims that it has more than 800 million active users and over 30 billion pieces of content (web links, news stories, blog posts, notes, photo albums, etc.) shared each month [110]. To protect user data, access control has become a central feature of OSNs [111, 112].

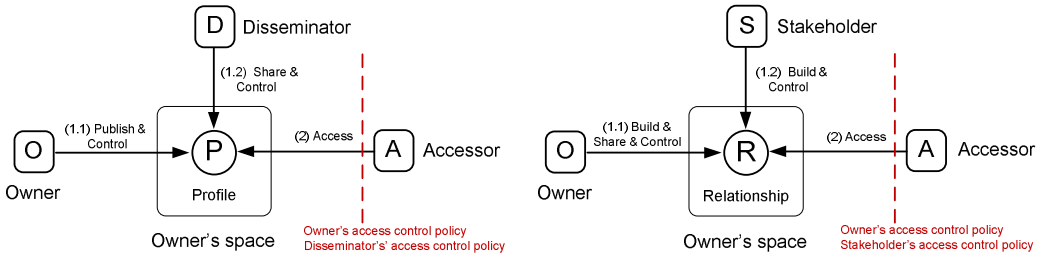
A typical OSN provides each user with a virtual space containing profile information, a list of the user's friends, and web pages, such as *wall* in Facebook, where users and friends can post content and leave messages. A user profile usually includes information with respect to the user's birthday, gender, interests, education and work history, and contact information. In addition, users can not only upload a content into their own or others' spaces but also *tag* other users who appear in the content. Each tag is an explicit reference that links to a user's space. For the protection of user data, current OSNs indirectly require users to be system and policy administrators for regulating their data, where users can restrict data sharing to a specific set of trusted users. OSNs often use *user relationship* and *group membership* to distinguish between trusted and untrusted users. For example, in Facebook, users can allow *friends*, *friends of friends*, *groups* or *public* to access their data, depending on their personal authorization and privacy requirements.

Although OSNs currently provide simple access control mechanisms allowing users to govern access to information contained in their own spaces, users, unfortunately, have no control over data residing *outside* their spaces. For instance, if a user posts a comment in a friend's space, s/he cannot specify which users can view the comment. In another case, when a user uploads a photo and tags friends who appear in the photo, the tagged friends cannot restrict who can see this photo, even though the tagged friends may have different privacy concerns about the photo. To address such a critical issue, preliminary protection mechanisms have been offered by existing OSNs. For example, Facebook allows tagged users to remove the tags linked to their profiles or report violations asking Facebook managers to remove the contents that they do not want to share with the public. However, these simple protection mechanisms suffer from several limitations. On one hand, removing a tag from a photo can only prevent other members from seeing a user's profile by means of the association link, but the user's image is still contained in the photo. Since original access control policies cannot be changed, the user's image continues to be revealed to all authorized users. On the other hand, reporting to OSNs only allows users to either keep or delete the content. Such a binary decision from OSN managers is either too loose or too restrictive, relying on the OSN's administration and requiring several people to report their request on the same content. Hence, it is essential to develop an effective and flexible access control mechanism for OSNs, accommodating the special authorization requirements coming from multiple associated users for managing the shared data collaboratively.

In Chapter 4, I demonstrated the proposed AMF framework with the realization and analysis of access control models, and the representation and analysis of access control policies, separately. However, there is no obvious connection among the adopted formal access control models, such as NIST/ANSI RBAC standard,

and access control policies, such as XACML policies. In this part, I will evaluate the AMF framework using an access control model and *associated* access control policies through modeling and analyzing multiparty access control in OSNs. I begin by examining how the lack of multiparty access control for data sharing in OSNs can undermine the protection of user data. Some typical data sharing patterns with respect to multiparty authorization in OSNs are also identified. Based on these sharing patterns, a multiparty access control (MPAC) model is formulated to capture the core features of multiparty authorization requirements which have not been accommodated so far by existing access control systems and models for OSNs (e.g., [28, 29, 30, 31, 70]). The model also contains a multiparty policy specification scheme. In the meanwhile, a systematic conflict detection and resolution mechanism is addressed to cope with privacy conflicts occurring in collaborative management of data sharing in OSNs. The conflict resolution approach in this work balances the need for privacy protection and the users' desire for information sharing by quantitative analysis of privacy risk and sharing loss.

Another compelling feature of the proposed solution is the support of analysis on multiparty access control model and systems. The correctness of implementation of an access control model is based on the premise that the access control model is valid. Moreover, while the use of multiparty access control mechanism can greatly enhance the flexibility for regulating data sharing in OSNs, it may potentially reduce the certainty of system authorization consequences due to the reason that authorization and privacy conflicts need to be resolved elegantly. Assessing the implications of access control mechanisms traditionally relies on the security analysis technique, which has been applied in several domains (e.g., operating systems [113], trust management [114], and role-based access control [115, 116]). I additionally introduce a method to represent and reason about the model in a logic



(a) A disseminator shares other's profile. (b) A user shares her/his relationships.

Figure 5.1: Multiparty access control pattern for profile and relationship sharing.

program. Besides, a proof-of-concept prototype of the approach is implemented in the context of Facebook. The experimental results based on comprehensive system evaluation and usability study demonstrate the feasibility and practicality of the proposed solution.

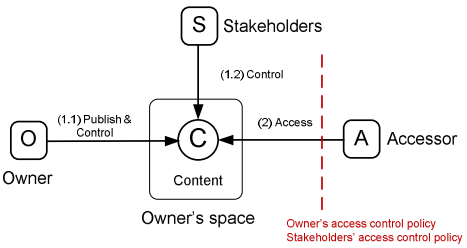
5.1 Multiparty Access Control for OSNs: Requirements and Patterns

In this section, I proceed with a comprehensive requirement analysis of multiparty access control in OSNs. Meanwhile, I discuss several typical sharing patterns occurring in OSNs where multiple users may have different authorization requirements to a single resource. I specifically analyze three scenarios—profile sharing, relationship sharing and content sharing—to understand the risks posted by the lack of collaborative control in OSNs. I leverage Facebook as the running example in this work since it is currently the most popular and representative social network provider. In the meantime, I reiterate that the proposed solution could be easily extended to other existing social network platforms, such as Google+ [117].

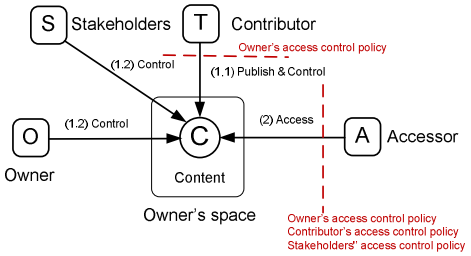
Profile sharing: An appealing feature of some OSNs is to support *social applications* written by third-party developers to create additional functionalities built on the top of users' profile for OSNs [118]. To provide meaningful and attractive services, these social applications consume user profile attributes, such as name, birthday, activities, interests, and so on. To make matters more complicated, social

applications on current OSN platforms can also consume the profile attributes of a user's friends. In this case, users can select particular pieces of profile attributes they are willing to share with the applications when their friends use the applications. At the same time, the users who are using the applications may also want to control what information of their friends is available to the applications since it is possible for the applications to infer their private profile attributes through their friends' profile attributes [119]. This means that when an application accesses the profile attributes of a user's friend, both the user and her friend want to gain control over the profile attributes. If we consider the application is an *accessor*, the user is a *disseminator* and the user's friend is the *owner* of shared profile attributes in this scenario, Figure 5.1(a) demonstrates a profile sharing pattern where a disseminator can share others' profile attributes to an accessor. Both the owner and the disseminator can specify access control policies to restrict the sharing of profile attributes.

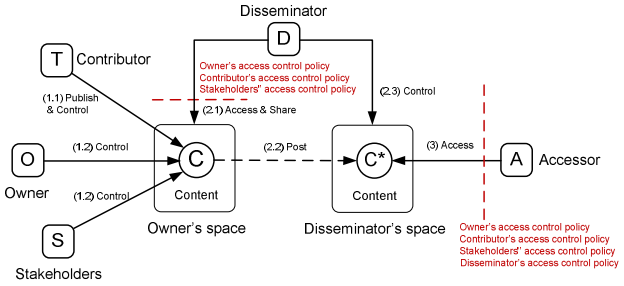
Relationship sharing: Another feature of OSNs is that users can share their relationships with other members. Relationships are inherently bidirectional and carry potentially sensitive information that associated users may not want to disclose. Most OSNs provide mechanisms that users can regulate the display of their friend lists. A user, however, can only control one direction of a relationship. Consider, for example, a scenario where a user Alice specifies a policy to hide her friend list from the public. However, Bob, one of Alice's friends, specifies a weaker policy that permits his friend list visible to anyone. In this case, if OSNs can solely enforce one party's policy, the relationship between Alice and Bob can still be learned through Bob's friend list. Figure 5.1(b) shows a relationship sharing pattern where a user called *owner*, who has a relationship with another user called *stakeholder*, shares the relationship with an *accessor*. In this scenario, authorization require-



(a) A shared content has multiple stakeholders.



(b) A shared content is published by a contributor.



(c) A disseminator shares other's content published by a contributor.

Figure 5.2: Multiparty access control pattern for content sharing.

ments from both the owner and the stakeholder should be considered. Otherwise, the stakeholder's privacy concern may be violated.

Content sharing: OSNs provide built-in mechanisms enabling users to communicate and share contents with other members. OSN users can post statuses and notes, upload photos and videos in their own spaces, tag others to their contents, and share the contents with their friends. On the other hand, users can also post contents in their friends' spaces. The shared contents may be connected with multiple users. Consider an example where a photograph contains three users, Alice, Bob and Carol. If Alice uploads it to her own space and tags both Bob and Carol in the photo, we call Alice the *owner* of the photo, and Bob and Carol *stakeholders* of the photo. All of them may specify access control policies to control over who can see this photo. Figure 5.2(a) depicts a content sharing pattern where the owner of a content shares the content with other OSN members, and the content has multiple stateholders who may also want to involve in the control of content

sharing. In another case, when Alice posts a note stating *"I will attend a party on Friday night with @Carol"* to Bob's space, we call Alice the *contributor* of the note and she may want to make the control over her notes. In addition, since Carol is explicitly identified by *@-mention* (at-mention) in this note, she is considered as a *stakeholder* of the note and may also want to control the exposure of this note. Figure 5.2(b) shows a content sharing pattern reflecting this scenario where a contributor publishes a content to other's space and the content may also have multiple stakeholders (e.g., tagged users). All associated users should be allowed to define access control policies for the shared content.

OSNs also enable users to share others' contents. For example, when Alice views a photo in Bob's space and decides to share this photo with her friends, the photo will be in turn posted in her space and she can specify access control policy to authorize her friends to see this photo. In this case, Alice is a *disseminator* of the photo. Since Alice may adopt a weaker control saying the photo is visible to everyone, the initial access control requirements of this photo should be compliant with, preventing from the possible leakage of sensitive information via the procedure of data dissemination. Figure 5.2(c) shows a content sharing pattern where the sharing starts with an *originator* (*owner* or *contributor* who uploads the content) publishing the content, and then a disseminator views and shares the content. All access control policies defined by associated users should be enforced to regulate access of the content in disseminator's space. For a more complicated case, the disseminated content may be further *re-disseminated* by disseminator's friends, where effective access control mechanisms should be applied in each procedure to regulate *sharing* behaviors. Especially, regardless of how many steps the content has been re-disseminated, the original access control policies should be always enforced to protect further dissemination of the content.

5.2 Modeling Multiparty Access Control for OSNs

In this section, I discuss the formalization of a MultiParty Access Control (MPAC) model for OSNs and a policy scheme for the specification of MPAC policies in OSNs.

MPAC Model

An OSN can be represented by a relationship network, a set of user groups and a collection of user data. The relationship network of an OSN is a directed labeled graph, where each node denotes a user and each edge represents a relationship between two users. The label associated with each edge indicates the type of the relationship. Edge direction denotes that the initial node of an edge establishes the relationship and the terminal node of the edge accepts the relationship. The number and type of supported relationships rely on the specific OSNs and its purposes. Besides, OSNs include an important feature that allows users to be organized in groups [120, 121] (or called circles in Google+ [122]), where each group has a unique name. This feature enables users of an OSN to easily find other users with whom they might share specific interests (e.g., same hobbies), demographic groups (e.g., studying at the same schools), political orientation, and so on. Users can join in groups without any approval from other group members. Furthermore, OSNs provide each member a Web space where users can store and manage their personal data including profile information, friend list and content.

Recently, several access control schemes (e.g., [28, 29, 30, 31]) have been proposed to support fine-grained authorization specifications for OSNs. Unfortunately, these schemes can only allow a single controller, the resource owner, to specify access control policies. Indeed, a flexible access control mechanism in a multi-user environment like OSNs should allow multiple controllers, who are as-

sociated with the shared data, to specify access control policies. As I identified previously in the sharing patterns (Section 5.1), in addition to the *owner* of data, other controllers, including the *contributor*, *stakeholder* and *disseminator* of data, need to regulate the access of the shared data as well. I define these controllers as follows:

Definition 6 *(**Owner**). Let d be a data item in the space of a user u in the social network. The user u is called the owner of d.*

Definition 7 *(**Contributor**). Let d be a data item published by a user u in someone else's space in the social network. The user u is called the contributor of d.*

Definition 8 *(**Stakeholder**). Let d be a data item in the space of a user in the social network. Let T be the set of tagged users associated with d. A user u is called a stakeholder of d, if $u \in T$.*

Definition 9 *(**Disseminator**). Let d be a data item shared by a user u from someone else's space to his/her space in the social network. The user u is called a disseminator of d.*

I now formally define the MPAC model as follows:

- $U = \{u_1, \ldots, u_n\}$ is a set of users of the OSN. Each user has a unique identifier;
- $G = \{g_1, \ldots, g_n\}$ is a set of groups to which the users can belong. Each group also has a unique identifier;

- $P = \{p_1, \ldots, p_n\}$ is a collection of user profile sets, where $p_i = \{q_{i1}, \ldots, q_{im}\}$ is the profile of a user $i \in U$. Each profile entry is a *<attribute: profile-value>* pair, $q_{ij} =< attr_j : pvalue_j >$, where $attr_j$ is an attribute identifier and $pvalue_j$ is the attribute value;

- RT is a set of relationship types supported by the OSN. Each user in an OSN may be connected with others by relationships of different types;

- $R = \{r_1, \ldots, r_n\}$ is a collection of user relationship sets, where $r_i = \{s_{i1}, \ldots, s_{im}\}$ is the relationship list of a user $i \in U$. Each relationship entry is a *<user: relationship-type>* pair, $s_{ij} =< u_j : rt_j >$, where $u_j \in U$, $rt_j \in RT$;

- $C = \{c_1, \ldots, c_n\}$ is a collection of user content sets, where $c_i = \{e_{i1}, \ldots, e_{im}\}$ is a set of contents of a user $i \in U$, where e_{ij} is a content identifier;

- $D = \{d_1, \ldots, d_n\}$ is a collection of data sets, where $d_i = p_i \cup r_i \cup c_i$ is a set of data of a user $i \in U$;

- $CT = \{OW, CB, ST, DS\}$ is a set of controller types, indicating *ownerOf*, *contributorOf*, *stakeholderOf*, and *disseminatorOf*, respectively;

- $UU = \{UU_{rt_1}, \ldots, UU_{rt_n}\}$ is a collection of uni-directional binary user-to-user relations, where $UU_{rt_i} \subseteq U \times U$ specifies the pairs of users having relationship type $rt_i \in RT$;

- $UG \subseteq U \times G$ is a set of binary user-to-group membership relations;

- $UD = \{UD_{ct_1}, \ldots, UD_{ct_n}\}$ is a collection of binary user-to-data relations, where $UD_{ct_i} \subseteq U \times D$ specifies a set of $< user, data >$ pairs having controller type $ct_i \in CT$;

- $relation\_members : U \xrightarrow{RT} 2^U$, a function mapping each user $u \in U$ to a set of users with whom s/he has a relationship $rt \in RT$:

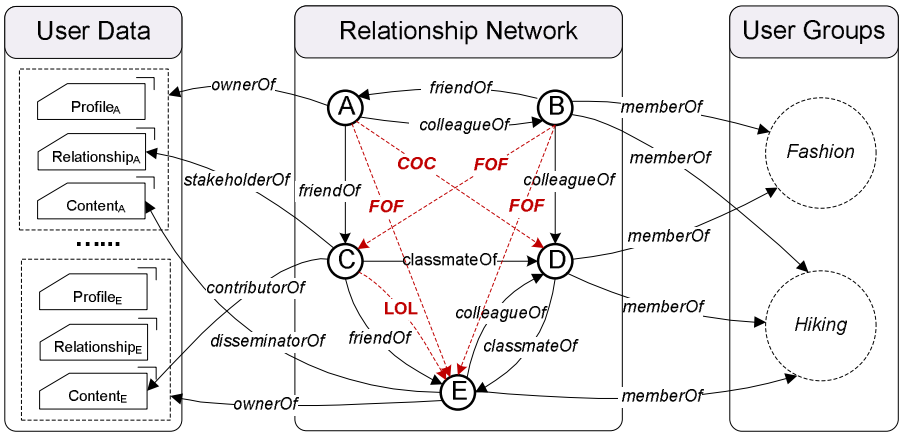


Figure 5.3: An example of multiparty social network representation.

$$relation\_members(u:U, rt:RT) = \{u' \in U \mid (u,u') \in UU_{rt}\};$$

- $ROR\_members : U \xrightarrow{RT} 2^U$, a function mapping each user $u \in U$ to a set of users with whom s/he has a *transitive* relation of a relationship $rt \in RT$, denoted as *relationships-of-relationships* (ROR). For example, if a relationship is *friend*, then its *transitive* relation is *friends-of-friends* (FOF):

$$ROR\_members(u:U, rt:RT) = \{u' \in U \mid u' \in relation\_members(u,rt) \vee (\exists u'' \in U[u'' \in ROR\_members(u,rt)) \wedge u' \in ROR\_members(u'',rt)]\};$$

- $controllers : D \xrightarrow{CT} 2^U$, a function mapping each date item $d \in D$ to a set of users who are the controller with the controller type $ct \in CT$:

$$controllers(d:D, ct:CT) = \{u \in U \mid (u,d) \in UD_{ct}\};$$ and

- $group\_members : G \rightarrow 2^U$, a function mapping each group $g \in G$ to a set of users who belong to the group:

$$group\_members(g:G) = \{u \in U \mid (u,g) \in UG\};$$

$$groups(u:U) = \{g \in G \mid (u,g) \in UG\};$$

Figure 5.3 depicts an example of multiparty social network representation. It describes relationships of five individuals, Alice (A), Bob (B), Carol (C), Dave (D) and Edward (E), along with their relations with data and their groups of interest. Note that two users may be directly connected by more than one edge labeled with different relationship types in the relationship network. For example, in Figure 5.3, Alice (A) has a direct relationship of type *colleagueOf* with Bob (B), whereas Bob (B) has a relationship of *friendOf* with Alice (A). In addition, two users may have transitive relationship, such as *friends-of-friends* (FOF), *colleagues-of-colleagues* (COC) and *classmates-of-classmates* (LOL) in this example. Moreover, this example shows that some data items have multiple controllers. For instance, $Relationship_A$ has two controllers: the owner, Alice (A) and a stakeholder, Carol (C). Also, some users may be the controllers of multiple data items. For example, Carol (C) is a stakeholder of $Relationship_A$ as well as the contributor of $Content_E$. Furthermore, we can notice there are two groups in this example that users can participate in: the "*Fashion*" group and the "*Hiking*" group, and some users, such as Bob (B) and Dave (D), may join in multiple groups.

MPAC Policy Specification

To enable a collaborative authorization management of data sharing in OSNs, it is essential for multiparty access control policies to be in place to regulate access over shared data, representing authorization requirements from multiple associated users. The policy specification scheme is built upon the proposed MPAC model.

Accessor Specification: Accessors are a set of users who are granted to access the shared data. Accessors can be represented with a set of user names, a set of relationship names or a set of group names in OSNs. The accessor specification is formally defined as follows:

Definition 10 *(**Accessor Specification**). Let $ac \in U \cup RT \cup G$ be a user $u \in U$, a relationship type $rt \in RT$, or a group $g \in G$. Let $at \in \{UN, RN, GN\}$ be the type of the accessor specification (*user name*,* relationship type*, and* group name*, respectively). The accessor specification is defined as a set, $accessors = \{a_1, \ldots, a_n\}$, where each element is a tuple $< ac, at >$.*

Data Specification: In OSNs, user data is composed of three types of information, *user profile*, *user relationship* and *user content*.

To facilitate effective privacy conflict resolution for multiparty access control, I introduce *sensitivity levels* for data specification, which are assigned by the controllers to the shared data items. A user's judgment of the sensitivity level of the data is not binary (private/public), but multi-dimensional with varying degrees of sensitivity. Formally, the data specification is defined as follows:

Definition 11 *(**Data Specification**). Let $dt \in D$ be a data item. Let sl be a sensitivity level, which is a rational number in the range [1,5], assigned to dt. The data specification is defined as a tuple $< dt, sl >$.*

Access Control Policy: To summarize the above-mentioned policy elements, I introduce the definition of a multiparty access control policy as follows:

Definition 12 *(**MPAC Policy**). A multiparty access control policy is a 4-tuple $P =< controller, accessor, data, effect >$, where*

- *$controller$ is defined as a 2-tuple $< cn, ct >$, where $cn \in U$ is a user who can regulate the access of data, and $ct \in CT$ is the type of the cn;*

- *accessor is a set of users to whom the authorization is granted, representing with an access specification defined in Definition 10.*
- *data is represented with a data specification defined in Definition 11; and*
- $effect \in \{permit, deny\}$ *is the authorization effect of the policy.*

Suppose a controller can leverage five sensitivity levels: 1 (*lowest*), 2 (*low*), 3 (*medium*), 4 (*high*), and 5 (*highest*) for the shared data. I give a motivating example to demonstrate how multiple controllers are able to specify their privacy concerns over a shared content as follows:

Example 3 *There is a shared photo, funny.jpg, in the social network. Alice is the owner of this photo, and Bob and Carol are two stakeholders of this photo. Alice authorizes her friends to view this photo and she considers the photo has a high sensitivity level; Bob allows the users in hiking group to access this photo and he considers the photo has a medium sensitivity level; and Carol permits her friends of friends to see this photo and she considers the photo has a high sensitivity level. These policies are expressed as:*

$p1 = (<Alice, OW>, \{<friendOf, RN>\}, <funny.jpg, 4>, permit)$.

$p2 = (<Bob, ST>, \{<hiking, GN>\}, <funny.jpg, 3>, permit)$.

$p3 = (<Carol, ST>, \{<friendOfFriend, RN>\}, <funny.jpg, 4>, permit)$.

5.3 Identifying and Resolving Privacy Conflicts

When two users disagree on whom the shared data item should be exposed to, we say a *privacy conflict* occurs. The essential reason leading to the privacy conflicts is that multiple associated users of the shared data item often have different privacy concerns over the data item. For example, assume that Alice and Bob are two

controllers of a photo. Each of them defines a privacy policy stating only her/his friends can view this photo. Since it is almost impossible that Alice and Bob have the same set of friends, privacy conflicts may *always* exist considering collaborative control over the shared data item.

A *naive* solution for resolving multiparty privacy conflicts is to only allow the *common* users of accessor sets defined by the multiple controllers to access the data [79]. Unfortunately, this solution is too restrictive in many cases and may not produce desirable results for resolving multiparty privacy conflicts. Let's consider an example that four users, Alice, Bob, Carol and Dave, are the controllers of a photo, and each of them allows her/his friends to see the photo. Suppose that Alice, Bob and Carol are close friends and have many common friends, but Dave has no common friends with them and has a pretty weak privacy concern on the photo. In this case, adopting the *naive* solution for conflict resolution may turn out that no one can access this photo. Nevertheless, it is reasonable to give the view permission to the common friends of Alice, Bob and Carol. A *strong* conflict resolution strategy may provide a better privacy protection. Meanwhile, it may reduce the social value of data sharing in OSNs. Therefore, it is important to consider the tradeoff between *privacy protection* and *data sharing* when resolving privacy conflicts. To address this issue, I introduce a mechanism for identifying multiparty privacy conflicts, as well as a systematic solution for resolving multiparty privacy conflicts.

Privacy Conflict Identification

Through specifying the access control policies to reflect the privacy concern, each controller of the shared data item defines a set of trusted users who can access the data item. The set of trusted users represents an *accessor space* for the controller. I adopt a space segmentation approach [123] introduced in Section 4.3 to partition accessor spaces of all controllers of a shared data item into disjoint segments. Then,

Algorithm 5: Identification of Conflicting Accessor Space

Input: A set of accessor space, A.
Output: A set of disjoined conflicting accessor spaces, CS.

```
1  /* Partition the entire accessor space */
2  S ⟵ Partition(A);
3  /* Identify the conflicting segments */
4  CS.New();
5  foreach s ∈ S do
6      /* Get all controllers associated with a segment s */
7      C ⟵ GetControllers(s);
8      if |C| < |A| then
9          CS.Append(s);

10 Partition(A)
11 foreach a ∈ A do
12     s_a ⟵ FriendSet(a);
13     foreach s ∈ S do
14         /* s_a is a subset of s*/
15         if s_a ⊂ s then
16             S.Append(s \ s_a);
17             s ⟵ s_a;
18             Break;
19         /* s_a is a superset of s*/
20         else if s_a ⊃ s then
21             s_a ⟵ s_a \ s;
22         /* s_a partially matches s*/
23         else if s_a ∩ s ≠ ∅ then
24             S.Append(s \ s_a);
25             s ⟵ s_a ∩ s;
26             s_a ⟵ s_a \ s;
27     S.Append(s_a);
28 return S;
```

conflicting accessor space segments called *conflicting segments*, which contain accessors that some controllers of the shared data item do not trust, are identified. Each conflicting segment contains at least one privacy conflict. Algorithm 5 shows the pseudocode of generating conflicting accessor space segments for all controllers of a shared data item.

Figure 5.4 gives an example of identifying privacy conflicts based on accessor space segmentation. I use circles to represent accessor spaces of three controllers, c_1, c_2 and c_3, of a shared data item. We can notice that three of accessor spaces overlap with each other, indicating that some accessors within the overlapping spaces are trusted by multiple controllers. After performing the space seg-

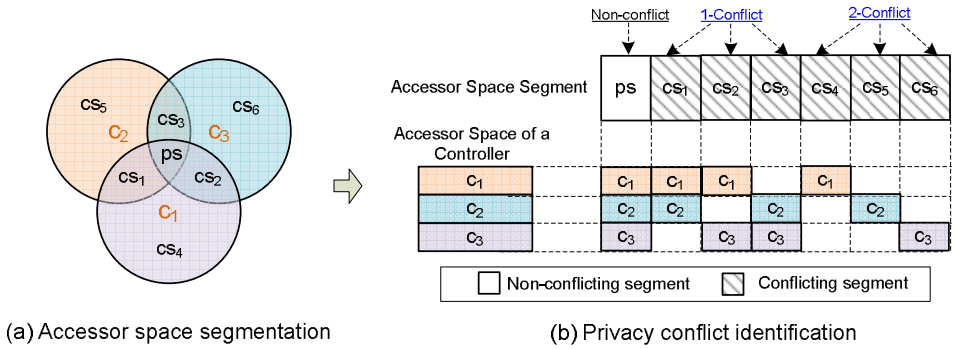


Figure 5.4: Example of privacy conflict identification based on accessor space segmentation.

mentation, seven disjoint accessor space segments are generated as shown in Figure 5.4 (a). To represent privacy conflicts in an intuitive way, I additionally introduce a grid representation of privacy conflicts, in which space segments are displayed along the horizontal axis of a matrix, controllers are shown along the vertical axis of the matrix, and the intersection of a segment and a controller is a grid that displays the accessor subspace covered by the segment. I classify the accessor space segments as two categories: *non-conflicting* segment and *conflicting* segment. *Non-conflicting* segment covers all controllers' access spaces, which means any accessor within the segment is trusted by all controllers of the shared data item, indicating no privacy conflict occurs. A *conflicting* segment does not contain all controllers' access spaces that means accessors in the segment are untrusted by some controllers. Each *untrusting* controller points out a privacy conflict. Figure 5.4 (b) shows a grid representation of privacy conflicts for the example. We can easily identify that the segment *ps* is a *non-conflicting* segment, and cs_1 through cs_6 are *conflicting* segments, where cs_1, cs_2 and cs_3 indicate *one* privacy conflict, respectively, and cs_4, cs_5 and cs_6 are associated with *two* privacy conflicts, respectively.

Privacy Conflict Resolution

The process of privacy conflict resolution makes a decision to allow or deny the accessors within the conflicting segments to access the shared data item. In general,

allowing the assessors contained in conflicting segments to access the data item may cause *privacy risk*, but denying a set of accessors in conflicting segments to access the data item may result in *sharing loss*. The proposed privacy conflict resolution approach attempts to find an optimal tradeoff between privacy protection and data sharing.

Measuring Privacy Risk: The privacy risk of a conflicting segment is an indicator of potential threat to the privacy of controllers in terms of the shared data item: the higher the privacy risk of a conflicting segment, the higher the threat to controllers' privacy. The basic premises for the measurement of privacy risk for a conflicting segment are the following: (a) the lower the number of controllers who trust the accessors within the conflicting segment, the higher the privacy risk; (b) the stronger the general privacy concerns of controllers, the higher the privacy risk; (c) the more sensitive the shared data item, the higher the privacy risk; and (d) the wider the data item spreads, the higher the privacy risk.

Therefore, the privacy risk of a conflicting segment is calculated by a monotonically increasing function with the following parameters:

- *Number of privacy conflicts*: The number of privacy conflicts in a conflicting segment is indicated by the number of the untrusting controllers. The untrusting controllers of a conflict segment i are returned by a function $controllers_{ut}(i)$;
- *General privacy concern of an untrusting controller*: The general privacy concern of an untrusting controller j is denoted as pc_j in the range [1, 5]. The general privacy concern of a controller can be derived from her/his *default* privacy setting for data sharing. Different controllers may have different general privacy concern with respect to the same kinds of data. For example,

public figures may have higher privacy concern on their shared photos than ordinary people;

- *Sensitivity of the data item*: Data sensitivity in a way defines controllers' perceptions of the confidentiality of the data being transmitted. The sensitivity level of the shared data item explicitly chosen by an untrusting controller j is denoted as sl_j in the range [1, 5]. The factor depends on the untrusting controllers themselves. Some untrusting controllers may consider the shared data item with the higher sensitivity; and

- *Visibility of the data item*: The visibility of the data item with respect to a conflicting segment captures how many accessors are contained in the segment i, denoted as n_i. The more the accessors in the segment, the higher the visibility.

The privacy risk of a conflict segment i due to an untrusting controller j, denoted as $PR(i, j)$, is defined as

$$PR(i, j) = pc_j \otimes sl_j \tag{5.1}$$

where, operator $\otimes$ is used to represent any arbitrary combination functions. For simplicity, I utilize the product operator.

In order to measure the *overall* privacy risk of a conflicting segment i, denoted as PR(i), we can use following equation to aggregate the privacy risks of i due to different untrusting controllers. Note that we can also use any combination function to combine the per-controller privacy risk. For simplicity, I employ the summation operator here.

$$PR(i) = n_i \times \sum_{j \in controllers_{ut}(i)} (pc_j \times sl_j) \tag{5.2}$$

Measuring Sharing Loss: When the decision of privacy conflict resolution for a conflicting segment is "deny", it may cause losses in potential data sharing, since there are controllers expecting to allow the accessors in the conflicting segment to access the data item. Similar to the measurement of the privacy risk, five factors are adopted to measure the sharing loss for a conflicting segment. Compared with the factors used for quantifying the privacy risk, the only difference is that I will utilize a factor, *number of trusting controllers*, to replace the factor, *number of privacy conflicts (untrusting controllers)*, for evaluating the sharing loss of a conflicting segment. The overall sharing loss $SL(i)$ of a conflicting segment i is computed as follows:

$$SL(i) = n_i \times \sum_{j \in controllers_t(i)} (5 - pc_j) \times (5 - sl_j) \tag{5.3}$$

where, function $controllers_t(i)$ returns all trusting controllers of a segment i.

Conflict Resolution on the Tradeoff between Privacy Protection and Data Sharing: The tradeoff between privacy and utility in data publishing has been recently studied [124, 125]. Inspired by those work, I introduce a mechanism to balance privacy protection and data sharing for an effective privacy conflict resolution in OSNs.

An optimal solution for privacy conflict resolution should cause a little more privacy risk when allowing the accessors in some conflicting segments to access the data item, and gets lesser loss in data sharing when denying the accessors to access the shared data item. Thus, for each conflict resolution solution s, a resolving score $RS(s)$ can be calculated using the following equation:

$$RS(s) = \frac{1}{\alpha \sum_{i_1 \in CS_p^s} PR(i_1) + \beta \sum_{i_2 \in CS_d^s} SL(i_2)} \tag{5.4}$$

where, CS_p^s and CS_d^s denote *permitted* conflicting segments and *denied* conflicting segments respectively in the conflict resolution solution s. And α and β are preference weights for the privacy risk and the sharing loss, $0 \leq \alpha, \beta \leq 1$ and $\alpha + \beta = 1$.

Then, the optimal conflict resolution CR_{opt} on the tradeoff between privacy risk and sharing loss can be identified by finding the maximum resolving score:

$$CR_{opt} = \max_{s} RS(s) \tag{5.5}$$

To find the maximum resolving score, we can first calculate the privacy risk ($PR(i)$) and the sharing loss ($SL(i)$) for each conflict segment (i), individually. Finally, following equation can be utilized to make the decisions (permitting or denying conflicting segments) for privacy conflict resolution, guaranteeing to always find an optimal solution.

$$Decision = \begin{cases} \text{Permit} & \text{if } \alpha SL(i) \geq \beta PR(i) \\ \text{Deny} & \text{if } \alpha SL(i) < \beta PR(i) \end{cases} \tag{5.6}$$

where, α and β are preference weights for the privacy risk and the sharing loss, $0 \leq \alpha, \beta \leq 1$ and $\alpha + \beta = 1$.

Generating Conflict-Resolved Policy

Once the privacy conflicts are resolved, we can aggregate accessors in *permitted* conflicting segments CS_p and accessors in the *non-conflicting* segment *ps* (in which accessors should be always allowed to access the shared data item) together to generate a new accessor list (AL) as follows:

$$AL = (\bigcup_{i \in CS_p} Accessors(i)) \cup Accessors(ps) \tag{5.7}$$

Using the example shown in Figure 5.4, assume that cs_1 and cs_3 become *permitted* conflicting segments after resolving the privacy conflicts. Therefore, the aggregated accessor list can be derived as $AL = Accessors(cs_1) \cup Accessors(cs_3) \cup Accessors(ps)$. Finally, the aggregated accessor list is used to construct a conflict-resolved privacy policy for the shared data item. The generated policy will be leveraged to evaluate all access requests toward the data item.

5.4 Logical Representation and Analysis of Multiparty Access Control

In this section, I adopt answer set programming (ASP), a recent form of declarative programming [26, 27], to formally represent the model, which essentially provide a formal foundation of the model in terms of ASP-based representation. Then, I demonstrate how the *correctness* analysis and *authorization* analysis [126] of multiparty access control can be carried out based on such a logical representation.

Representing Multiparty Access Control in ASP

I introduce the logical representation of MPAC model/policy and MPAC privacy conflict detection/resolution.

Logical Representation of MPAC Model and Policy

The basic components and relations in the OSN representation can be directly defined with corresponding predicates in ASP. The OSN representation supports *transitive* relationships. For example, David is a friend of Allice, and Edward is a friend of David in a social network. Then, we call Edward is a *friends of friends* of Allice. The *friend* relation between two users Allice and David is represented in ASP as follows:

$$friend(Allice, David).$$

It is known that the transitive closure (e.g., reachability) cannot be expressed in first-order logic [26], however, it can be easily handled in the stable model semantics. Then, *friends-of-friends* can be represented with ASP as follows:

$$friendOfFriend(X,Y) \leftarrow friend(X,Y).$$
$$friendOfFriend(X,Z) \leftarrow friend(X,Y) \wedge friend(Y,Z) \wedge X \neq Z.$$

The translation module converts a multiparty authorization policy

$$(\langle cn, ct\rangle, \{\langle ac_1, at_1\rangle, \ldots, \langle ac_n, at_n\rangle\}, \langle d, sl\rangle, effect)$$

into three ASP rule

$$sensitivityLevel(cn,d,sl).$$
$$decision(cn,U,d,effect) \leftarrow \bigvee_{1\leq k\leq n} ac_k(cn,X) \wedge ct(cn,d).$$
$$decision(cn,U,d,\overline{effect}) \leftarrow \neg decision(cn,U,d,effect).$$

where if $effect$ is *permit* then $\overline{effect}$ is *deny* and vise versa. Then, three privacy policies in Example 3 can be represented in ASP as follows:

```
%P1
sensitivityLevel(alice,funny.jpg,4).
decision(alice,U,funny.jpg,permit) <- owner(alice,funny.jpg)
  & friend(alice,U).
decision(alice,U,funny.jpg,deny) <- not decision(alice,U,funny.jpg,
  permit).
%P2
sensitivityLevel(bob,funny.jpg,3).
decision(bob,U,funny.jpg,permit) <- stakeholder(bob,funny.jpg) &
  memberOf(U, hikingGroup).
decision(bob,U,funny.jpg,deny) <- not decision(bob,U,funny.jpg,
  permit).
```

```
%P3
sensitivityLevel(carol,funny.jpg,4).
decision(carol,U,funny.jpg,permit) <- stakeholder(carol,funny.jpg) &
  friendOfFriend(carol,U).
decision(carol,U,funny.jpg,deny) <- not decision(carol,U,funny.jpg,
  permit).
```

Logical Representation of Privacy Conflict Detection and Resolution

Privacy conflicts can be also identified by logic-based reasoning. Suppose Figure 5.4 represents the accessor space segmentation of the tree policies defined by three controllers, Alice (c_1), Bob (c_2) and Carol (c_3), in Example 3. Accessors in the *conflicting* segment cs_1 are permitted by Alice and Bob, and denied by Carol. Then, we can identify this conflicting segment with the following ASP program:

```
cs(1,U) <- decision(alice,U,funny.jpg,permit) & decision(bob,U,funny.jpg,
  permit) & decision(carol,U,funny.jpg,deny).
```

The aggregation of privacy risks can be computed naturally using the aggregate functions in ASP as follows:

```
pr(I,Val) <- Sigma = #sum [sensitivityLevel(U1,d,SL) : decision(U1,U,d,deny)
 = (pc*SL)]& N = #count {cs(I,U1): user(U1)} & Val=Sigma*N & cs(I,U).
```

Similarly, sharing lost can be computed using ASP rules as follows:

```
sl(I,Val) <- Sigma = #sum [sensitivityLevel(U1,d,SL) : decision(U1,U,d,deny)
  = (5-pc)*(5-SL)] & N = #count {cs(I,U1): user(U1)} & Val=Sigma*N & cs(I,U).
```

Reasoning about Multiparty Access Control

One of the promising advantages in logic-based representation of access control is that formal reasoning of the authorization properties can be achieved. Once we

represent the multiparty access control model into an ASP program, we can use off-the-shelf ASP solvers to carry out automated authorization analysis tasks.

Authorization analysis is employed to examine *over-sharing* (does current authorization state disclose the data to some users undesirable?) and *under-sharing* (does current authorization state disallow some users to access the data that they are supposed to be allowed?). This analysis service should be incorporated into OSN systems to enable users checking potential authorization impacts derived from collaborative control of shared data.

Example 4 *(Checking over-sharing) Alice has defined a policy to disallow her family members to see a photo, party.jpg. Then, she wants to check if any family members can see this photo after applying conflict resolution mechanism for collaborative authorization management considering different privacy preferences from multiple controllers. The input query can be represented as follows:*

```
denyBy(Alice,U) <- decision(Alice,U,party.jpg,deny) & permit(N1) & cs(N1,U).
```

If any answer set is found, it means that there are family members who can see the photo. Thus, from the perspective of Alice's authorization, this photo is *over* shared to some users.

Example 5 *(Checking under-sharing) Bob has defined a policy to authorize his friends to see a photo, funny.jpg. He wants to check if any friends cannot see this photo in current system. The input query can be specified as follows:*

```
permitBy(Bob,U) <- decision(Alice,U,funny.jpg,permit) & deny(N1) & cs(N1,U).
```

If an answer set contains `check`, this means that there are friends who cannot view the photo. Regarding Bob's authorization requirement, this photo is *under* shared with his friends.

In addition to the authorization analysis, the individual privacy risk and sharing lost for each controller can be also calculated by the logic-based reasoning. For instance, the privacy risk of the controller Alice can be represented as follows:

```
ipr(Alice,Y) <- X = #count {denyBy(Alice,U1) : user(U1)} &
  sensitivityLevel(Alice,funny.jpg,N) & Y = X*N.
```

And the sharing lost of the controller can be computed as:

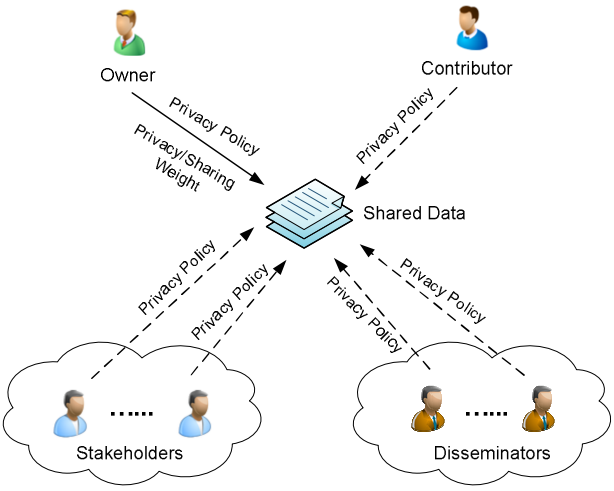
```
isl(Alice,Y) <- X = #count {permitBy(Alice,U1) : user(U1)} &
  sensitivityLevel(Alice,funny.jpg,N) & Y = X*(5-N).
```

5.5 Implementation and Evaluation

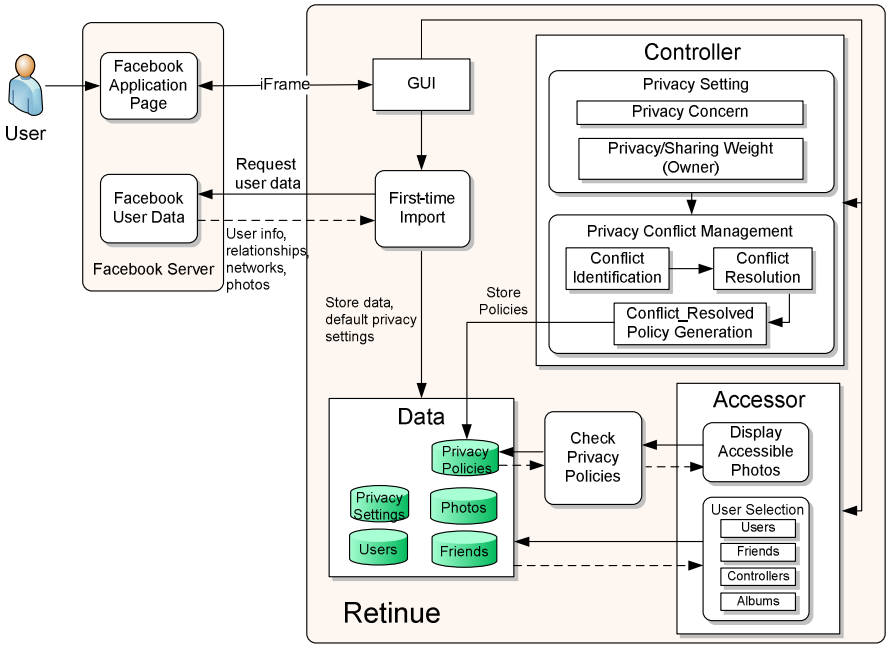
In this section, I address the prototype implementation of the proposed application in the context of Facebook and discuss the experimental results.

Prototype Implementation

A proof-of-concept Facebook application has been implemented for the collaborative management of shared data called *Retinue* (http://apps.facebook.com/retinue_tool). This prototype application enables multiple associated users to specify their privacy concerns to co-control a shared data item. *Retinue* is designed as a third-party Facebook application which is hosted in an Apache Tomcat application server supporting PHP and MySQL databases, with a user interface built using jQuery and jQuery UI and built on an AJAX-based interaction model. *Retinue* application is based on the iFrame external application approach. Using the Javascript and PHP SDK, it accesses users' Facebook data through the Graph API and Facebook Query Language. It is worth noting that the current implementation was restricted to handle photo sharing in OSNs. Obversely, the proposed approach can be generalized to deal with other kinds of data sharing (e.g. videos and comments) in OSNs as long



(a) Collaborative control overview.

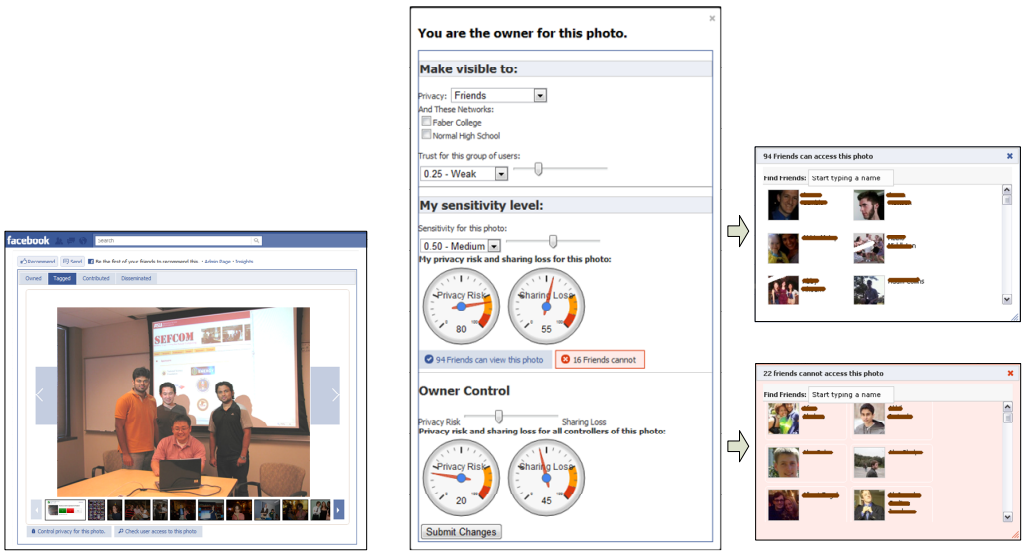


(b) Operational components in *Retinue* application.

Figure 5.5: System architecture of *Retinue*.

as the stakeholder of shared data are identified with effective methods like tagging or searching.

Figure 5.5 shows the system architecture of *Retinue*. The overview of collaborative control process is depicted in Figure 5.5(a), where the *owner* can regulate the access of the shared data. In addition, other controllers, such as *the contribu-*



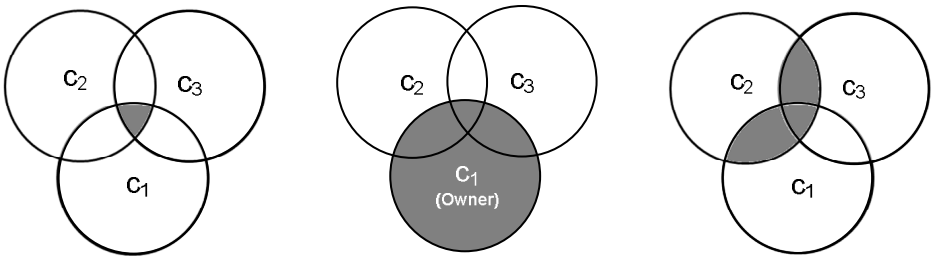
(a) Main interface. (b) Controllers' interfaces.

Figure 5.6: *Retinue* interfaces.

tor, *stakeholders* and *disseminators*, can specify their privacy concerns over the shared data as well. To effectively resolve privacy conflicts caused by different privacy concerns of multiple controllers, the data *owner* can also adjust the preference weights for the privacy risk and the sharing loss to make an appropriate privacy-sharing tradeoff. Figure 5.5(b) shows the core components in *Retinue* application and their interactions. The *Retinue* application is hosted on an external website, but is accessed on a Facebook application frame via an iFrame. The Facebook server handles login and authentication for the application, and other user data is imported on the user's first login. At this point, users are asked to specify their initial privacy settings and concerns for each type of photo. All photos are then imported and saved using these initial privacy settings. Users' networks and friend lists are imported from Facebook server as well. Once information is imported, a user accesses *Retinue* through the application page on Facebook, where s/he can query access information, complete privacy setting for photos in which s/he is a controller, and view photos s/he is allowed to access. The component for privacy conflict man-

agement in *Retinue* application is responsible for the privacy conflict detection and resolution, and the generation of conflict-resolved privacy policy, which is then used to evaluate access requests for the shared data.

A snapshot of the main interface of *Retinue* is shown in Figure 5.6 (a). All photos are loaded into a gallery-style interface. To access photos, a user clicks the "Access" tab and then s/he can view her/his friends' photos that s/he was authorized. To control photo sharing, a user clicks the "Owned", "Tagged", "Contributed", or "Disseminated" tabs, then selects any photo in the gallery to define her/his privacy preferences for that photo. The controllers' interfaces are depicted in Figure 5.6 (b). A controller can select the trusted groups of accessors and assign corresponding trust levels, as well as choose the sensitivity level for the photo. Also, the privacy risk and sharing loss for the controller with respect with the photo are displayed in the interface. In addition, the controller can immediately see how many friends can or cannot access the photo in the interface. If the controller clicks the buttons, which show the numbers of accessible or unaccessible friends, a window appears showing the details of all friends who can or cannot view the photo. The purpose of such feedback information is not only to give a controller the information of how many friends can or cannot access the photo, but as a way to react to results. If the controller is not satisfied with the current situation of privacy control, s/he may adjust her/his privacy settings, contact the owner of the photo to ask her/him to change the weights for the privacy risk and the sharing loss, or even report a privacy violation to request OSN administrators to delete the photo. If the user is the owner of the photo, s/he can also view the overall privacy risk and sharing loss for the shared photo, and has the ability to adjust the weights to balance privacy protection and data sharing of the shared photo.



(a) Naive solution. (b) Facebook solution. (c) *Retinue* solution.

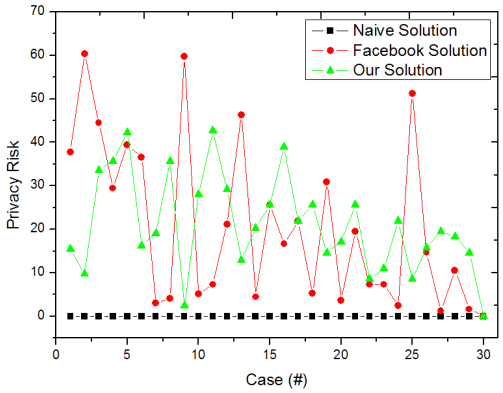
Figure 5.7: Example of resolving privacy conflicts.

Evaluation and Experiment

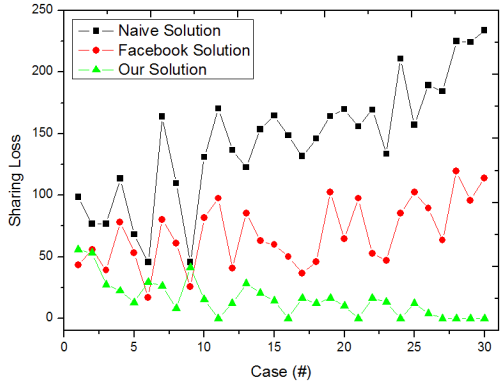
Here I address the evaluation of the proposed conflict resolution approach and system usability study.

Evaluation of Privacy Conflict Resolution

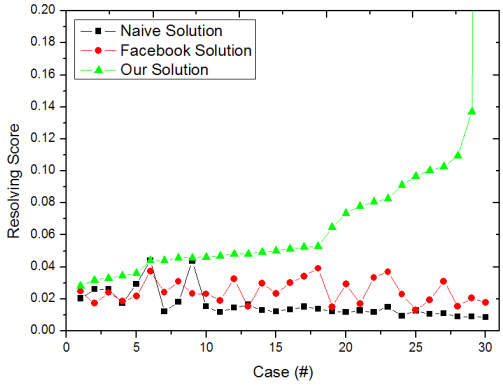
I evaluate the proposed approach for privacy conflict resolution by comparing *Retinue* solution with the *naive solution* and the privacy control solution used by existing OSNs, such as Facebook (simply called *Facebook solution* in the rest of this dissertation) with respect to two metrics, *privacy risk* and *sharing loss*. Consider the example demonstrated in Figure 5.4, where three controllers desire to regulate access of a shared data item. The *naive solution* is that only the accessors in the non-conflicting segment are allowed to access the data item as shown in Figure 5.7(a). Thus, the *privacy risk* is always equal to 0 for this solution. However, the *sharing loss* is the absolute maximum, as all conflicting segments, which may be allowed by at least one controller, are always denied. The *Facebook solution* is that the owner's decision has the highest priority. All accessors within the segments covered by the owner's space are allowed to access the data item, but all other accessors are denied as illustrated in Figure 5.7(b). This is, obviously, ideal for the owner, since her/his *privacy risk* and *sharing loss* are both equal to 0. However, the *privacy risk* and the *sharing loss* are large for every non-owner controller.



(a) Privacy risk.



(b) Sharing loss.



(c) Resolving score.

Figure 5.8: Conflict resolution evaluation.

For *Retinue* solution, each conflicting segment is evaluated individually. Using the same example given in Figure 5.4, suppose cs_1 and cs_3 become *permitted* conflicting segments after resolving the privacy conflicts. Figure 5.7(c) demon-

strates the result of the privacy conflict resolution. *Retinue* solution make a tradeoff between privacy protection and data sharing by maximizing the resolving score, which is a combination of *privacy risk* and *sharing loss*. The worst case of *Retinue* solution is the same as the *naive solution*–only mutually permitted accessors are allowed to access the data item. However, this case only occurs when strong privacy concerns are indicated by each controller. On the other hand, if all accessors have pretty weak privacy concerns, all accessors in conflicting segments may be allowed to access the data, which is not possible with either of other two solutions. Such a case leads to a sharing loss of 0, but does not have an significantly increased privacy risk against other two solutions.

To quantitatively evaluate *Retinue* solution, the experiment used cases where there are three controllers of shared data items and assume that each controller has indicated to allow her/his friends to view the data item. I also utilized the average number of user friends, 130, which is claimed by Facebook statistics [110]. Additionally, assume all controllers share 30 friends with each other, 10 of which are shared among everyone (common users). All settings including privacy concerns, sensitivity levels, and trust levels were randomized for each case, and the privacy risk, sharing loss, and resolving score for each case were calculated. To represent the data sensibly, the samples were sorted from lowest resolving score to highest under the evaluation. Figure 5.8 shows the experimental results with respect to randomly generated 30 user cases.

In Figure 5.8(a), we can observe that the privacy risks for the *naive solution* are always equal to 0, since no untrusted accessors are allowed to view the data item. The privacy risks for *Facebook solution* and *Retinue* solution wavered. Obviously, this depends greatly on the settings of the non-owner controllers. If these controllers are apathetic toward the shared data item, *Facebook solution* will be preferable.

However, it should be noted that *Facebook solution* had very high extrema. This is avoided in *Retinue* solution where high privacy risks will usually result in denying access.

Unsurprisingly, the sharing loss for the *naive solution* was always the highest, and often higher than both other two solutions as shown in Figure 5.8(b). *Retinue* solution usually had the lowest sharing loss, and sometimes is equivalent to the *naive* or *Facebook solution*, but rarely greater than. One may notice that the sharing loss is very low compared to the privacy risks in this experience. This is an inherent effect of *Retinue* solution itself–if sharing loss is very high, users will be granted access to the data item, changing this segment's sharing loss to zero.

As we can notice from Figure 5.8(c), the resolving score for *Retinue* solution is always as good as or better than the *naive* or *Facebook solution*. In the sample data, it was usually significantly better, and rarely was the same as either of other two solutions. It further indicates that *Retinue* solution can always achieve a good tradeoff between privacy protection and data sharing for privacy conflict resolution.

Evaluation of System Usability

Participants and Procedure: *Retinue* is a functional proof-of-concept implementation of collaborative privacy management. To measure the practicality and usability of the proposed mechanism, I conducted a survey study (n=30) to explore the factors surrounding users' desires for privacy controls such as those implemented in *Retinue*. Particularly, I were interested in users' perspectives on the current Facebook privacy system and their desires for more control over photos they do not own. I recruited participants through university mailing lists and through Facebook itself using Facebook's built-in sharing API. Users were given the opportunity to share the application and play with their friends. While this is not a random sampling,

Table 5.1: Usability evaluation for Facebook and *Retinue* privacy controls.

Metric	**Facebook**		**Retinue**	
	Average	Upper bound on 95% confidence interval	Average	Lower bound on 95% confidence interval
Likability	0.20	0.25	0.83	0.80
Simplicity/Usefulness	0.38	0.44	0.72	0.64
Control	0.20	0.25	0.83	0.80

recruiting using the natural dissemination features of Facebook arguably gives an accurate profile of the ecosystem.

In the user study (http://bit.ly/retinue_study), participants were asked to first answer some questions about their usage and perception of Facebook's privacy controls. Users were then instructed to install the application using their Facebook profiles and complete the following actions: set privacy settings for a photo they do not own, set privacy settings for a photo they own, and answer questions about their understanding. As users completed these actions, they were asked questions on the usability of the controls in Retinue.

User Study of Retinue: The criteria for usability evaluation were split into three areas: *likeability*, *simplicity/usefulness*, and *control*. *Likeability* is a measure of a user's satisfaction with a system; *simplicity/usefulness* is a measure how intuitive and useful the system is; and *control* is a measure of the user's perceived control of their personal data. All questions were either True/False or measured on a 5-point likert scale (scaled from 0 to 1 for numerical analysis). For measurement and analysis, a higher number is used to indicate a positive opinion or perception of the system, while a lower number is used to indicate a negative one. I were interested in the average user perception of the system, so I analyzed a 95% confidence interval for the users' answers. This assumes the population to be mostly normal.

Before Using Retinue. Before using *Retinue*, users were asked a few questions about their usage of Facebook to determine the user's perceived usability of

the current Facebook's privacy controls. For the confidence interval, I were interested in determining the average user's maximum positive opinion of Facebook's privacy controls, so I looked at the upper bound of the confidence interval.

An average user asserts at most 25% positively about the *likability* and *control* of Facebook's privacy management mechanism, and at most 44% on Facebook's *simplicity/usefulness* as shown in Table 5.1. This demonstrates an average negative opinion of the Facebook's privacy controls that users currently must use. A detailed table for the perceived usability evaluation of Facebook privacy controls is provided in Table 5.2.

Table 5.2: Perceived usability of *Factbook* privacy controls (before using *Retinue*).

Evaluation Questions	**Average Positive Response**
Likability	***0.20***
If someone uploads a photo of me, Facebook allows me to restrict whether someone else can see it on my Facebook page: True (1) or False (0)	0.33
If someone uploads a photo of me, Facebook allows me to restrict whether someone can see it through the uploader's Facebook page: True(1) or False(0)	0.07
On Facebook, there is a high potential of my data being viewed by parties I wish to hide it from - Strongly agree (1) - Strongly Disagree(5)	0.17
I am concerned with providing personal data (e.g. photos) on Facebook because it could be used or abused in unforeseen ways - Strongly agree (1) - Strongly Disagree(5)	0.22
Simplicity/Usefulness	***0.38***
In terms of everyday use I find the current Facebook privacy controls: Confusing(1) - Intuitive/Understanding (5)	0.42
Facebook gives me adequate control over who can access the content I own: No control(1)-Complete control(5)	0.48
Facebook gives me adequate control over who can access content I have a stake in but do not own: No Control(1) - Complete Control(5)	0.25
Control	***0.20***
I wish I had more control over pictures I am tagged in: Selected (0) or Not (1)	0.13
I wish people I tagged in pictures I own could help control the privacy of these photos: Selected (0) or Not (1)	0.13
I fear that sensitive photos I upload may be seen by people I do not want to view them (e.g. parents, employers) : Selected (0) or Not (1)	0.33
I fear that sensitive photos OTHERS upload may be seen by people I do not want to view them (e.g. parents, employers) : Selected (0) or Not (1)	0.13
I wish I could check if certain users had access to specific photos: Selected (0) or Not (1)	0.17

After Using Retinue. Users were then asked to perform a few tasks in *Retinue*: control settings for a photo they are tagged in, control settings for a photo they own, and check user access to a photo they own or are in. For the confidence

interval, I were interested in determining the average user's minimum positive opinion of *Retinue*'s privacy controls, so I looked at the lower bound of the confidence interval.

An average user asserts at least 80% positively about the *likability* and *control* of *Retinue*'s privacy controls, and at least 64% positively on *Retinue*'s *simplicity/usefulness* as shown in Table 5.1. This demonstrates an average positive opinion of the controls and ideas presented to users in *Retinue*. A detailed table for the perceived usability evaluation of *Retinue* privacy controls is given in Table 5.3.

Table 5.3: Perceived usability of *Retinue* privacy controls (after using *Retinue*).

Evaluation Questions	**Average Positive Response**
Likability	***0.83***
I would use tagged/posted controls on a daily basis if implemented in Facebook	0.8
I would use owner controls on a daily basis if implemented in Facebook	0.73
I would use query controls on a daily basis were they implemented in Facebook	0.8
If Facebook Implemented controls like Retinue's to control photo privacy, I would be happier with privacy controls	0.93
I like the idea of sharing control of my photos with those tagged	0.83
I like the idea of sharing control of my photos with those who posted them on my wall	0.81
I like the idea of being able to control photos in which I am tagged	0.86
Simplicity/Usefulness	***0.72***
Setting my privacy settings for a photo in Retinue is Complicated (1) - Simple (5)	0.48
Setting tagged/posted privacy settings in Facebook is Less complicated (0) - More complicated/Not possible (1)	0.6
Controlling a photo as an owner in Retinue is Complicated (1) - Simple (5)	0.65
Setting privacy for photos I own on Facebook is Less complicated (0) - More complicated/Not possible (1)	0.4
Querying user access to a photo in Retinue is Complicated (1) - Simple (5)	0.82
Querying user access on Facebook is Less complicated (0) - More complicated /Not possible (1)	0.73
If Facebook Implemented controls like Retinue's to control photo privacy, I would be more likely to use privacy controls on a regular basis	0.87
If Facebook Implemented controls like Retinue's to control photo privacy, my photo privacy would be improved	0.88
If Facebook Implemented controls like Retinue's to control photo privacy, my usage of privacy controls would be clearer and more understandable	0.86
If Facebook Implemented controls like Retinue's to control photo privacy, controlling privacy would require less mental effort	0.75
If Facebook Implemented controls like Retinue's to control photo privacy, it would be easier to control photo privacy	0.76
If Facebook Implemented controls like Retinue's to control photo privacy, it would be easier to understand who can access my photos	0.81
Control	***0.83***
I feel more in control of tagged/posted photos on Facebook (1) - Retinue (5)	0.83
I feel more in control of photos I own on Facebook (1) - Retinue (5)	0.75
I feel more aware of others' ability to see my photos on Facebook (1) - Retinue (5)	0.78
If Facebook Implemented controls like Retinue's to control photo privacy, my photos would be better protected	0.88
If Facebook Implemented controls like Retinue's to control photo privacy, I would have more control over my photos	0.9

5.6 Discussion

In the multiparty access control system, a group of users could collude with one another so as to manipulate the final access control decision. Consider an attack scenario, where a set of malicious users may want to make a shared photo available to a wider audience. Suppose they can access the photo, and then they all tag themselves or fake their identities to the photo. In addition, they collude with each other to assign a very low sensitivity level for the photo and specify policies to grant a wider audience to access the photo. With a large number of colluding users, the photo may be disclosed to those users who are not expected to gain the access. To prevent such an attack scenario from occurring, three conditions need to be satisfied: (1) there is no fake identity in OSNs; (2) all tagged users are real users appeared in the photo; and (3) all controllers of the photo are honest to specify their privacy preferences.

Regarding the first condition, two typical attacks, Sybil attacks [127] and Identity Clone attacks [128], have been introduced to OSNs and several effective approaches have been recently proposed to prevent the former [129, 130] and latter attacks [131], respectively. To guarantee the second condition, an effective tag validation mechanism is needed to verify each tagged user against the photo. In the current system, if any users tag themselves or others in a photo, the photo owner will receive a tag notification. Then, the owner can verify the correctness of the tagged users. As effective automated algorithms (e.g., facial recognition [132]) are being developed to recognize people accurately in contents such as photos, automatic tag validation is feasible. Considering the third condition, the current system provides a function to indicate the potential authorization impact with respect to a controller's privacy preference. Using such a function, the photo owner can examine all users

who are granted the access by the collaborative authorization and are not explicitly granted by the owner her/himself. Thus, it enables the owner to discover potential malicious activities in collaborative control. The detection of collusion behaviors in collaborative systems has been addressed by the recent work [133, 134]. The future work would integrate an effective collusion detection technique into MPAC. To prevent collusion activities, the current prototype has implemented a function for owner control, where the photo owner can disable any controller, who is suspected to be malicious, from participating in collaborative control of the photo. In addition, I would further investigate how users' reputations–based on their collaboration activities–can be applied to prevent and detect malicious activities in the future work.

Chapter 6

Conclusion

In this chapter, I summarize the contributions of this dissertation and discuss some directions for future work.

6.1 Summary

Security has become a core ingredient of nearly most modern software and information systems. However, security countermeasures are often integrated into existing systems after significant security problems are discovered during the administration or usage phase. In order to effectively address security aspects in secure system development and management, more convenient and mature mechanisms should be designed.

In this dissertation, I have presented an Assurance Management Framework (AMF) for comprehensive analysis and realization of access control models in secure system development, through access control model representation, constraint specification, generation of enforcement code, and analysis and testing of access control models. I introduced the concept of an authorization state space to assist tasks in identifying unique characteristics of constraints in access control model specification during a course of the model analysis. Corresponding analysis processes for the formal verification and automatic test generation were articulated as well. In addition, I demonstrated how the proposed methodology can be applied to build role-based access control systems by adopting the NIST/ANSI RBAC standard as an underlying security model. For the realization of RBAC model, I showed how the formal RBAC model and constraints can be represented with UML diagrams and OCL expressions. The UML/OCL-based specifications of RBAC model and constraints enable the automatic generation of executable authorization mod-

els. To analyze the RBAC model, I utilized Alloy as an underlying formal verification tool to demonstrate automatic analysis and test case generation for the formal specifications of RBAC model. An RBAC authorization environment `RAE` and a simulation system `RASS` were implemented as well to accommodate core features addressed in the AMF framework.

On the other hand, the AMF framework ensures the correctness of access control policies in policy-based computing through automated reasoning techniques and anomaly management mechanisms. I demonstrated a systematic method to represent XACML policies in ASP that allows users to leverage ASP solvers for a variety of analysis tasks. I also discussed the design of a tool called `XACML2ASP`, which can seamlessly work with existing ASP solvers for XACML policy analysis. In addition, I presented a novel anomaly management mechanism and a grid-based visualization approach, which enable effective detection and resolution of policy anomalies. An anomaly analysis tool for XACML policies called `XAnalyzer` was implemented as well.

Online social networks (OSNs) have experienced tremendous growth in recent years and become a *de facto* portal for hundreds of millions of Internet users. However, OSNs currently do not provide any mechanism to enforce privacy concerns over data associated with multiple users. To this end, I further evaluated the AMF framework through modeling and analyzing multiparty access control in OSNs. I first formulated an access control model to capture the essence of multiparty authorization requirements, followed by a multiparty policy specification scheme. Then, I provided a systematic mechanism to identify and resolve privacy conflicts for collaborative data sharing. The conflict resolution in this work indicates a tradeoff between privacy protection and data sharing by quantifying privacy risk and sharing loss. I also presented a logical representation of the access control

model with ASP for performing various analysis tasks on the model. A proof-of-concept implementation of the proposed solution called `Retinue` was discussed as well, along with the extensive system evaluation and usability study.

Contribution

As a summary, the contributions of this research are as follows:

- Articulation of automated analysis and thorough realization of formal access control models in secure system development via model representation, generation of enforcement code, and verification and testing of access control models.
- A logic-based reasoning approach for access control policies, adopting a logical programming to formulate access control policies that allows users to leverage the features of logic solvers in performing various logical reasoning and analysis tasks.
- A comprehensive anomaly management mechanism incorporated with a visualization-based policy representation to facilitate systematic and effective detection and resolution of access control policy anomalies.
- Evaluation of the applicability of proposed AMF framework through modeling and analyzing multiparty access control, which facilitates a systematic solution for collaborative management of shared data in OSNs.
- Demonstration of the feasibility of the proposed methodology through a suite of proof-of-concept implementations.

6.2 Future Work

This ground-breaking work also has several future research directions:

Realization and Analysis of Access Control Model

The toolset in this work for realizing and analyzing access control models constitutes a set of modules including a formal analysis tool such as Alloy Analyzer to facilitate the features of the proposed methodology in performing tasks related to analysis and conformance testing of role-based authorization systems. As part of future work, I would examine how such a formal analysis can be integrated seamlessly with the toolset. In addition, to address the limitation of using SAT solver for the purpose of formal analysis, I plan to investigate the relationship between the size of represented model and the time required for verification and test case generation in this approach. Furthermore, I would attempt to extend the framework for dealing with more complicated system properties such as temporal and context attributes. Thus, other specification languages and tools for performing formal analysis in the framework would be investigated as well. Moreover, I believe that more practical engineering processes for secure system development should be addressed so that software developers can easily adopt this approach in their development practices [135]. Therefore, I would explore such a practical engineering process for building authorization systems based on the AMF framework.

Analysis and Management of Access Control Policy

I have demonstrated the logic-based policy reasoning approach using ASP and the policy anomaly management mechanism based on BDD. A comparison of various techniques for policy analysis should be conducted, guiding the policy designers to choose appropriate methods for different policy analysis tasks. Also, the coverage of the proposed policy analysis approach needs to be further extended with

respect to more policy features such as handling multi-valued requests, complicated conditions, obligation, and user-defined functions. In addition, it is necessary to enhance the policy analysis tool in this work to provide those features and corresponding analysis services while obscuring the details of the logic formalism. Furthermore, I would evaluate the effectiveness of the proposed policy analysis and management approach and conduct usability studies of the proposed policy visualization approach with subject matter experts. Besides, I plan to extend the proposed analysis approach to handle distributed policy management. Additionally, even though I have applied the anomaly analysis mechanism to XACML [123] and firewall policies [50, 24], I would further explore how the policy analysis and anomaly management mechanism can be applied to more existing access control policy languages [51], such as SAML [7], Ponder [8] and EPAL [9].

Applying AMF to Emerging Domains

I would apply the AMF framework to address the security challenges brought by emerging domains, including social networks [76, 117, 136, 137, 138, 139], cloud computing [140, 141, 142, 143, 144, 145, 146], mobile computing [147, 148, 149], and healthcare systems [57, 58, 150].

Social Networks

As the popularity of OSNs continues to grow, a huge amount of personal and private information uploaded to OSNs. To protect such a large volume of sensitive information, much more research needs to be done in the field of security and privacy for OSNs. In OSNs, collaborative mechanism may allow users to take advantage of the wisdom of crowds when making policy decisions related to user-to-user interaction. I plan to continue this research in the direction of multiparty access control to explore comprehensive decision making methods, such as decision making based on game theory [151], and analysis services for collaborative management of shared

data in OSNs. Also, I would explore more criteria to evaluate the features of the proposed MPAC model. For example, more comprehensive experiments should be conducted to evaluate the effectiveness of MPAC conflict resolution approach based on the tradeoff of privacy risk and sharing loss [136]. In addition, users may be involved in the control of a larger number of shared photos and the configurations of the privacy preferences may become time-consuming and tedious tasks in OSNs. Therefore, I would study inference-based techniques [152] that leverage machine learning and data mining approaches to facilitate both smart user management and automated privacy policy configuration in MPAC. Besides, I plan to systematically integrate the notion of trust and reputation into the MPAC model and investigate a comprehensive solution to cope with collusion attacks for providing a robust MPAC service in OSNs. Also, I would extend this work to address security and privacy challenges for other social network platforms such as Google+ [122].

Cloud Computing

The emerging cloud-computing paradigm is rapidly gaining momentum as an alternative to traditional information technology due to the reason that it provides an extensible and powerful environment for growing amounts of services and data. However, the unique aspects of cloud computing also exacerbate security and privacy challenges. Based on the proposed AMF framework, I am planning to explore various approaches to cope with the access control challenges in clouds. In particular, I noticed that extensive collaborations exist among services provided by different clouds, which might have different security mechanisms and privacy management approaches. Hence, I will explore the approaches to address the heterogeneity among the policies from different cloud domains. And I believe that the use of policy ontology [51] is a promising approach to accommodate such an issue. Additionally, since policy conflicts are inevitable, arising in the interopera-

tion process of multi-domain services in clouds, a more comprehensive mechanism for policy anomaly detection and resolution [123, 24] is essential to prevent the potential security breaches caused by policy integration in cloud computing.

Mobile Computing

Nowadays, OSN providers like Facebook or Google offer their users with web-based Location-based Services (LBSs) like Facebook Places [153] or Google Latitude [154]. To facilitate proactive LBSs, most OSN providers also provide Location-based Social Network Services (LB-SNSs) for mobile platforms like the Android, iOS, or Symbian. However, at present almost none of these implementations guarantees a sufficient degree of information security and privacy as well as location integrity for their users. To address these limitations, I intend to apply the AMF framework to explore solutions that support a secure, privacy-preserving, and location-restricted LB-SNSs for mobile platforms. Furthermore, as smartphones have become an indispensable part of daily life, mobile users are increasingly relying on them to process personal information with feature-rich applications. However, recent studies show that smartphone platforms are vulnerable to a variety of attacks that could bypass these existing security mechanisms [147, 148]. This requires robust security systems for protecting sensitive applications and data on mobile devices. Thus, I would apply the AMF framework to investigate effective mechanisms for enhancing existing smartphone protection systems.

Healthcare Systems

Security and privacy in healthcare systems grow in importance. The adoption of electronic health records, increased regulation, provider consolidation and the increasing need for information exchange between patients, providers and payers, all point towards the need for developing effective security and privacy mechanisms to protect healthcare systems. Moreover, the advent of social networking applica-

tions for healthcare systems, such as Google Health [155] and Microsoft HealthVault [156], has the potential to significantly alter the manner in which patients interact with healthcare providers. Therefore, I would apply the AMF framework to address a variety of security and privacy issues in healthcare systems including data integrity, regulatory compliance, selective sharing, consent delegation, trust management, and so on. In particular, based on a unified logical EHR model [57, 58] and a compliance analysis framework for healthcare systems [150], I would investigate how the AMF framework can systematically manage complex policies to reduce risks in such a dynamic and collaborative environment.

REFERENCES

[1] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman, "Role-based access control models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[2] D.F. Ferraiolo, R. Sandhu, S. Gavrila, D.R. Kuhn, and R. Chandramouli, "Proposed NIST standard for role-based access control," *ACM Trans. Inf. Syst. Secur. (TISSEC)*, vol. 4, no. 3, pp. 224–274, 2001.

[3] ANSI, *American National Standards Institute Inc.*, Role Based Access Control, ANSI-INCITS 359–2004, 2004.

[4] N. Li, J.C. Mitchell, and W.H. Winsborough, "Design of a role-based trust-management framework," in *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*. IEEE, 2005, pp. 114–130.

[5] J. Park and R. Sandhu, "The UCON ABC usage control model," *ACM Transactions on Information and System Security (TISSEC)*, vol. 7, no. 1, pp. 128–174, 2004.

[6] XACML, "OASIS eXtensible Access Control Markup Language (XACML) V2.0 Specification Set," http://www.oasis-open.org/committees/xacml/, 2007.

[7] OASIS, "Security Assertion Markup Language," http://www.oasis-open.org/committees/security/.

[8] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The ponder policy specification language," *Policies for Distributed Systems and Networks*, pp. 18–38, 2001.

[9] P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter, "Enterprise privacy authorization language (EPAL)," http://www.w3.org/Submission/2003/SUBM-EPAL-20031110/.

[10] D.B. Chapman, E.D. Zwicky, and D. Russell, *Building internet firewalls*, O'Reilly & Associates, Inc. Sebastopol, CA, USA, 1995.

[11] M. Condell, C. Lynn, and J. Zao, "Security policy specification language," *Internet Engineering Task Force (IETF) Internet Draft*, 2000.

[12] P. Loscocco and S. Smalley, "Integrating flexible support for security policies into the Linux operating system," in *Proc. 2001 USENIX Annual Technical Conference REENIX Track*, 2001, pp. 29–40.

[13] "AppArmor," http://de.opensuse.org/AppArmor.

[14] D. Basin, J. Doser, and T. Lodderstedt, "Model driven security: From UML models to access control infrastructures," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 15, no. 1, pp. 39–91, 2006.

[15] I. Ray, N. Li, R. France, and D.K. Kim, "Using UML to visualize role-based access control constraints," in *Proceedings of the ninth ACM symposium on Access control models and technologies(SACMAT)*, 2004, pp. 115–124.

[16] G.J. Ahn, S.P. Hong, and M.E. Shin, "Reconstructing a formal security model," *Information and Software Technology*, vol. 44, no. 11, pp. 649–657, 2002.

[17] A. Schaad and J.D. Moffett, "A lightweight approach to specification and analysis of role-based access control extensions," in *Proceedings of the seventh ACM symposium on Access control models and technologies*. ACM, 2002, pp. 13–22.

[18] K. Sohr, G.J. Ahn, M. Gogolla, and L. Migge, "Specification and validation of authorisation constraints using UML and OCL," *Lecture notes in computer science*, vol. 3679, pp. 64, 2005.

[19] D. Basin, M. Clavel, J. Doser, and M. Egea, "Automated analysis of security-design models," *Information and Software Technology*, vol. 51, no. 5, pp. 815–831, 2009.

[20] K. Fisler, S. Krishnamurthi, L.A. Meyerovich, and M.C. Tschantz, "Verification and change-impact analysis of access-control policies," in *Proceedings of the 27th international conference on Software engineering*. ACM New York, NY, USA, 2005, pp. 196–205.

[21] V. Kolovski, J. Hendler, and B. Parsia, "Analyzing web access control policies," in *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 677–686.

[22] E.S. Al-Shaer and H.H. Hamed, "Discovery of policy anomalies in distributed firewalls," in *IEEE INFOCOM*. Citeseer, 2004, vol. 4, pp. 2605–2616.

[23] L. Yuan, J. Mai, Z. Su, H. Chen, C.N. Chuah, and P. Mohapatra, "FIREMAN: A toolkit for firewall modeling and analysis," in *IEEE Symposium on Security and Privacy*. Citeseer, 2006, pp. 199–213.

[24] H. Hu, G.J. Ahn, and K. Kulkarni, "Detecting and resolving firewall policy anomalies," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 3, pp. 318–331, 2012.

[25] D. Jackson, I. Schechter, and I. Shlyakhter, "ALCOA: The Alloy constraint analyzer," in *Software Engineering, 2000. Proceedings of the 2000 International Conference on*. IEEE, 2000, pp. 730–733.

[26] Victor Marek and Mirosław Truszczyński, "Stable models and an alternative logic programming paradigm," in *The Logic Programming Paradigm: a 25-Year Perspective*, pp. 375–398. Springer Verlag, 1999.

[27] Vladimir Lifschitz, "What Is Answer Set Programming?," in *Proceedings of the AAAI Conference on Artificial Intelligence*. 2008, pp. 1594–1597, MIT Press.

[28] B. Carminati, E. Ferrari, and A. Perego, "Rule-based access control for social networks," in *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*. Springer, 2006, pp. 1734–1744.

[29] B. Carminati, E. Ferrari, and A. Perego, "Enforcing access control in web-based social networks," *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 1, pp. 1–38, 2009.

[30] P.W.L. Fong, "Relationship-based access control: Protection model and policy language," in *Proceedings of the first ACM conference on Data and application security and privacy*. ACM, 2011, pp. 191–202.

[31] P.W.L. Fong, M. Anwar, and Z. Zhao, "A privacy preservation model for facebook-style social network systems," in *Proceedings of the 14th European conference on Research in computer security*. Springer-Verlag, 2009, pp. 303–320.

[32] Jan Jürjens, "UMLsec: Extending UML for secure systems development," in *Proceedings of the 5th International Conference on The United Modeling Language*. 2002, pp. 412–425, Springer Verlag.

[33] K. Alghathbar and D. Wijesekera, "authUML: a three-phased framework to analyze access control specifications in use cases," in *Proceedings of the 2003 ACM workshop on Formal methods in security engineering*. ACM, 2003, pp. 77–86.

[34] D. Mouheb, C. Talhi, M. Nouh, V. Lima, M. Debbabi, L. Wang, and M. Pourzandi, "Aspect-priented modeling for representing and integrating security concerns in UML," *Software Engineering Research, Management and Applications 2010*, pp. 197–213, 2010.

[35] E.M. Clarke and J.M. Wing, "Formal methods: State of the art and future directions," *ACM Computing Surveys (CSUR)*, vol. 28, no. 4, pp. 626–643, 1996.

[36] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled, *Model Checking*, MIT Press, 2000.

[37] David G. Mitchell, *A SAT Solver Primer*, EATCS Bulletin (The Logic in Computer Science Column), Volume 85, February, pages 112-133, 2005.

[38] A. Robinson and A.Voronkov, *Handbook of Automated Reasoning*, MIT Press, 2001.

[39] B. Shafiq, A. Masood, J. Joshi, and A. Ghafoor, "A role-based access control policy verification framework for real-time systems," in *Object-Oriented Real-Time Dependable Systems, 2005. WORDS 2005. 10th IEEE International Workshop on*. IEEE, 2005, pp. 13–20.

[40] M. Toahchoodee, I. Ray, K. Anastasakis, G. Georg, and B. Bordbar, "Ensuring spatio-temporal access control for real-world applications," in *Proceedings of the 14th ACM symposium on Access control models and technologies*. ACM New York, NY, USA, 2009, pp. 13–22.

[41] J. Tretmans, "A formal approach to conformance testing," in *Protocol test systems, VI: proceedings of the IFIP TC6/WG6. 1 Sixth International Workshop on Protocol Test Systems, Pau, France, 28-30 September 1993*. North Holland, 1994, p. 257.

[42] C. Artho, H. Barringer, A. Goldberg, K. Havelund, S. Khurshid, M. Lowry, C. Pasareanu, G. Rosu, K. Sen, W. Visser, et al., "Combining test case generation and runtime verification," *Theoretical Computer Science*, vol. 336, no. 2-3, pp. 209–234, 2005.

[43] R.M. Hierons, K. Bogdanov, J.P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, et al., "Using formal specifications to support testing," *ACM Computing Surveys (CSUR)*, vol. 41, no. 2, pp. 1–76, 2009.

[44] A. Masood, A. Ghafoor, and A. Mathur, "Scalable and effective test generation for access control systems that employ RBAC policies," *SERC-TR-285, Purdue University*, 2005.

[45] E. Martin and T. Xie, "A fault model and mutation testing of access control policies," in *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 667–676.

[46] A. Pretschner, T. Mouelhi, and Y.L. Traon, "Model-based tests for access control policies," in *Proceedings of the 2008 International Conference on Software Testing, Verification, and Validation*. IEEE Computer Society, 2008, pp. 338–347.

[47] G. Hughes and T. Bultan, "Automated verification of access control policies," *Computer Science Department, University of California, Santa Barbara, CA*, vol. 93106, pp. 2004–22.

[48] D. Jackson, "Alloy: a lightweight object modelling notation," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 11, no. 2, pp. 256–290, 2002.

[49] J. Bryans, "Reasoning about XACML policies using CSP," in *Proceedings of the 2005 workshop on Secure web services*. ACM, 2005, p. 35.

[50] H. Hu, G.J. Ahn, and K. Kulkarni, "FAME: a firewall anomaly management environment," in *Proceedings of the 3rd ACM workshop on Assurable and usable security configuration*. ACM, 2010, pp. 17–26.

[51] H. Hu, G.J. Ahn, and K. Kulkarni, "Ontology-based policy anomaly management for autonomic computing," in *Proceedings of the 7th Interna-*

tional Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom). IEEE, 2011, pp. 487–494.

[52] "Sun XACML Implementation," http://sunxacml.sourceforge.net.

[53] A.X. Liu, F. Chen, J.H. Hwang, and T. Xie, "Designing fast and scalable policy evaluation engines," *IEEE Transactions on Computers*, 2012, in press.

[54] S. Jajodia, P. Samarati, and VS Subrahmanian, "A logical language for expressing authorizations," in *Proceedings of the 1997 IEEE Symposium onSecurity and Privacy*. IEEE, 1997, pp. 31–42.

[55] I. Fundulaki and M. Marx, "Specifying access control policies for XML documents with XPath," in *Proceedings of the ninth ACM symposium on Access control models and technologies*. ACM, 2004, pp. 61–69.

[56] N. Li, Q. Wang, W. Qardaji, E. Bertino, P. Rao, J. Lobo, and D. Lin, "Access control policy combining: theory meets practice," in *Proceedings of the 14th ACM symposium on Access control models and technologies*. ACM, 2009, pp. 135–144.

[57] J. Jin, G.J. Ahn, H. Hu, M.J. Covington, and X. Zhang, "Patient-centric authorization framework for sharing electronic health records," in *Proceedings of the 14th ACM symposium on Access control models and technologies*. ACM New York, NY, USA, 2009, pp. 125–134.

[58] J. Jin, G.J. Ahn, H. Hu, M.J. Covington, and X. Zhang, "Patient-centric authorization framework for electronic healthcare services," *Computers & Security*, vol. 30, no. 2-3, pp. 116–127, 2011.

[59] L. Bauer, S. Garriss, and M.K. Reiter, "Detecting and resolving policy misconfigurations in access-control systems," *ACM Transactions on Information and System Security (TISSEC)*, vol. 14, no. 1, pp. 2, 2011.

[60] P. Mazzoleni, B. Crispo, S. Sivasubramanian, and E. Bertino, "XACML policy integration algorithms," *ACM Transactions on Information and System Security*, vol. 11, no. 1, 2008.

[61] P. Rao, D. Lin, E. Bertino, N. Li, and J. Lobo, "An algebra for fine-grained integration of xacml policies," in *Proceedings of the 14th ACM symposium on Access control models and technologies*. ACM, 2009, pp. 63–72.

[62] S. Marouf, M. Shehab, A. Squicciarini, and S. Sundareswaran, "Statistics & clustering based framework for efficient XACML policy evaluation," in *IEEE International Symposium on Policies for Distributed Systems and Networks*. IEEE, 2009, pp. 118–125.

[63] R.W. Reeder, L. Bauer, L.F. Cranor, M.K. Reiter, K. Bacon, K. How, and H. Strong, "Expandable grids for visualizing and authoring computer security policies," in *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*. ACM, 2008, pp. 1473–1482.

[64] C.A. Brodie, C.M. Karat, and J. Karat, "An empirical study of natural language parsing of privacy policy rules using the SPARCLE policy workbench," in *Proceedings of the second symposium on Usable privacy and security*. ACM, 2006, p. 19.

[65] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J.M. Wing, "Automated generation and analysis of attack graphs," in *Proceedings of the 2002 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2002, p. 273.

[66] X. Ou, W.F. Boyer, and M.A. McQueen, "A scalable approach to attack graph generation," in *Proceedings of the 13th ACM conference on Computer and communications security*. ACM, 2006, pp. 336–345.

[67] K. Abdullah, C. Lee, G. Conti, and J.A. Copeland, "Visualizing network data for intrusion detection," in *Proceedings of the Sixth Annual IEEE SMC Information Assurance Workshop*. IEEE, 2005, pp. 100–108.

[68] T. Itoh, H. Takakura, A. Sawada, and K. Koyamada, "Hierarchical visualization of network intrusion detection data," *Computer Graphics and Applications, IEEE*, vol. 26, no. 2, pp. 40–47, 2006.

[69] D. Yao, M. Shin, R. Tamassia, and W.H. Winsborough, "Visualization of automated trust negotiation," in *Proceedings of the IEEE Workshop on Visualization for Computer Security (VizSEC 05)*. IEEE, 2005, pp. 65–74.

[70] S. Kruk, S. Grzonkowski, A. Gzella, T. Woroniecki, and H.C. Choi, "D-FOAF: Distributed identity management with access rights delegation," *The Semantic Web–ASWC 2006*, pp. 140–154, 2006.

[71] E.G. Carrie, "Access control requirements for Web 2.0 security and privacy," in *Proceedings of the Workshop on Web 2.0 Security & Privacy (W2SP)*. Citeseer, 2007.

[72] S.A. Brands, *Rethinking public key infrastructures and digital certificates: building in privacy*, The MIT Press, 2000.

[73] N. Elahi, M.M.R. Chowdhury, and J. Noll, "Semantic access control in Web based communities," in *Proceedings of the Third International Multi-Conference on Computing in the Global Information Technology*. IEEE, 2008, pp. 131–136.

[74] A. Masoumzadeh and J. Joshi, "OSNAC: an ontology-based access control model for social networking systems," in *Proceedings of the 2010 IEEE Second International Conference on Social Computing*. IEEE Computer Society, 2010, pp. 751–759.

[75] A. Besmer and H. Richter Lipford, "Moving beyond untagging: Photo privacy in a tagged world," in *Proceedings of the 28th international conference on Human factors in computing systems*. ACM, 2010, pp. 1563–1572.

[76] H. Hu and G.J. Ahn, "Multiparty authorization framework for data sharing in online social networks," in *Proceedings of the 25th annual IFIP WG 11.3 conference on Data and applications security and privacy*. 2011, DBSec'11, pp. 29–43, Springer.

[77] A. Lampinen, V. Lehtinen, A. Lehmuskallio, and S. Tamminen, "We're in it together: interpersonal management of disclosure in social network services," in *Proceedings of the 2011 annual conference on Human factors in computing systems*. ACM, 2011, pp. 3217–3226.

[78] A.C. Squicciarini, M. Shehab, and F. Paci, "Collective privacy management in social networks," in *Proceedings of the 18th international conference on World wide web*. ACM, 2009, pp. 521–530.

[79] K. Thomas, C. Grier, and D. Nicol, "unFriendly: Multi-party privacy risks in social networks," in *Privacy Enhancing Technologies*. Springer, 2010, pp. 236–252.

[80] J. Becker and H. Chen, "Measuring privacy risk in online social networks," in *Proceedings of the 2009 Workshop on Web*. Citeseer.

[81] K. Liu and E. Terzi, "A framework for computing the privacy scores of users in online social networks," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 5, no. 1, pp. 6, 2010.

[82] N. Talukder, M. Ouzzani, AK Elmagarmid, H. Elmeleegy, and M. Yakout, "Privometer: Privacy protection in social networks," in *Proceedings of 26th International Conference on Data Engineering Workshops (ICDEW)*. IEEE, 2010, pp. 266–269.

[83] J.Rumbaugh, I. Jacobson, and G.Booch, *The Unified Modeling Language Reference Manual, Second Edition*, Object Technology Series, Addison Wesley Longman, Reading, Mass, 2004.

[84] R. France, "A problem-oriented analysis of basic UML static requirements modeling concepts," in *Proceedings of the 14th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, New York, NY, USA, 1999, pp. 57–69.

[85] J. Warmer and A. Kleppe, *The Object Constraint Language: Getting your models ready for MDA*, Addison-Wesley, Reading/MA, 2003.

[86] G.J. Ahn and R. Sandhu, "Role-based authorization constraints specification," *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 4, pp. 207–226, 2000.

[87] T. Jaeger and J.E. Tidswell, "Practical safety in flexible access control models," *ACM Transactions on Information and System Security (TISSEC)*, vol. 4, no. 2, pp. 158–190, 2001.

[88] D. Jackson, "Alloy: a lightweight object modelling notation," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 11, no. 2, pp. 256–290, 2002.

[89] M. Gelfond and V. Lifschitz, "The stable model semantics for logic programming," in *Proceedings of International Logic Programming Conference and Symposium*, Robert Kowalski and Kenneth Bowen, Eds. 1988, pp. 1070–1080, MIT Press.

[90] P. Ferraris, J. Lee, and V. Lifschitz, "Stable models and circumscription," *Artificial Intelligence*, 2010.

[91] Vladimir Lifschitz and Alexander Razborov, "Why are there so many loop formulas?," *ACM Transactions on Computational Logic*, vol. 7, pp. 261–268, 2006.

[92] Bran Selic, "The pragmatics of Model-Driven Development," *IEEE Software*, vol. 20, no. 5, pp. 19–25, 2003.

[93] M. Utting and B. Legeard, *Practical Model-Based Testing: A Tools Approach*, Morgan-Kaufmann, 2007.

[94] Michael E. Shin and Gail-Joon Ahn, "UML-based representation of role-based access control," in *Proceedings of the 9th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2000, pp. 195–200.

[95] Octopus, "The Octopus Project," http://www.klasse.nl/octopus, 2004.

[96] Dresden, "Dresden OCL toolkit," http://dresden-ocl.sourceforge.net, 2005.

[97] N. Li, J.W. Byun, and E. Bertino, "A critique of the ANSI standard on role-based access control," *Security & Privacy, IEEE*, vol. 5, no. 6, pp. 41–49, 2007.

[98] ArgoUML, "The ArgoUML Project," http://argouml.tigris.org, 2006.

[99] I. Herman, G. Melançon, and M.S. Marshall, "Graph visualization and navigation in information visualization: A survey," *IEEE Transactions on Visualization and Computer Graphics*, pp. 24–43, 2000.

[100] M.C. Tschantz and S. Krishnamurthi, "Towards reasonability properties for access-control policy languages," in *Proceedings of the eleventh ACM symposium on Access control models and technologies*. ACM, 2006, pp. 160–169.

[101] A. Birgisson, M. Dhawan, U. Erlingsson, V. Ganapathy, and L. Iftode, "Enforcing authorization policies using transactional memory introspection," in *Proceedings of the 15th ACM conference on Computer and communications security*. ACM New York, NY, USA, 2008, pp. 223–234.

[102] F. Baboescu and G. Varghese, "Fast and scalable conflict detection for packet classifiers," *Computer Networks*, vol. 42, no. 6, pp. 717–735, 2003.

[103] E.C. Lupu and M. Sloman, "Conflicts in policy-based distributed systems management," *IEEE Transactions on software engineering*, vol. 25, no. 6, pp. 852–869, 1999.

[104] J.G. Alfaro, N. Boulahia-Cuppens, and F. Cuppens, "Complete analysis of configuration rules to guarantee reliable network security policies," *International Journal of Information Security*, vol. 7, no. 2, pp. 103–122, 2008.

[105] R.E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Transactions on computers*, vol. 100, no. 35, pp. 677–691, 1986.

[106] A. Anderson, "Evaluating XACML as a policy language," in *Technical report*. OASIS, 2003.

[107] D. Agrawal, J. Giles, K.W. Lee, and J. Lobo, "Policy ratification," in *Sixth IEEE International Workshop on Policies for Distributed Systems and Networks, 2005*, 2005, pp. 223–232.

[108] "Java BDD," http://javabdd.sourceforge.net.

[109] "Buddy version 2.4," http://sourceforge.net/projects/buddy.

[110] "Facebook Statistics," http://www.facebook.com/press/info.php?statistics, January 2011.

[111] "Facebook Privacy Policy," http://www.facebook.com/policy.php/.

[112] "Google+ Privacy Policy," http://http://www.google.com/intl/en/+/policy/.

[113] M.A. Harrison, W.L. Ruzzo, and J.D. Ullman, "Protection in operating systems," *Communications of the ACM*, vol. 19, no. 8, pp. 461–471, 1976.

[114] N. Li, J.C. Mitchell, and W.H. Winsborough, "Beyond proof-of-compliance: security analysis in trust management," *Journal of the ACM (JACM)*, vol. 52, no. 3, pp. 474–514, 2005.

[115] G.-J. Ahn and H. Hu, "Towards realizing a formal RBAC model in real systems," in *Proceedings of the 12th ACM symposium on Access control models and technologies*. ACM, 2007, p. 224.

[116] H. Hu and G.-J. Ahn, "Enabling verification and conformance testing for access control model," in *Proceedings of the 13th ACM Symposium on Access control Models and Technologies*. ACM, 2008, pp. 195–204.

[117] H. Hu, G.J. Ahn, and J. Jorgensen, "Enabling collaborative data sharing in Google+," in *Proceedings of 55th IEEE Global Communications Conference*. IEEE, 2012.

[118] "Facebook Developers," http://developers.facebook.com/.

[119] A. Mislove, B. Viswanath, K.P. Gummadi, and P. Druschel, "You are who you know: Inferring user profiles in online social networks," in *Proceedings of the third ACM international conference on Web search and data mining*. ACM, 2010, pp. 251–260.

[120] E. Zheleva and L. Getoor, "To join or not to join: the illusion of privacy in social networks with mixed public and private user profiles," in *Proceedings of the 18th international conference on World wide web*. ACM, 2009, pp. 531–540.

[121] G. Wondracek, T. Holz, E. Kirda, and C. Kruegel, "A practical attack to deanonymize social network users," in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*. IEEE, 2010, pp. 223–238.

[122] "The Google+ Project," https://plus.google.com.

[123] H. Hu, G.J. Ahn, and K. Kulkarni, "Anomaly discovery and resolution in web access control policies," in *Proceedings of the 16th ACM symposium on Access control models and technologies*. ACM, 2011, pp. 165–174.

[124] J. Brickell and V. Shmatikov, "The cost of privacy: destruction of data-mining utility in anonymized data publishing," in *Proceeding of the 14th ACM SIGKDD*. ACM, 2008, pp. 70–78.

[125] T. Li and N. Li, "On the tradeoff between privacy and utility in data publishing," in *Proceedings of the 15th ACM SIGKDD*. ACM, 2009, pp. 517–526.

[126] G.J. Ahn, H. Hu, J. Lee, and Y. Meng, "Representing and reasoning about Web access control policies," in *Proceedings of the 2010 IEEE 34th Annual Computer Software and Applications Conference*. IEEE Computer Society, 2010, pp. 137–146.

[127] J. Douceur, "The sybil attack," *Peer-to-peer Systems*, pp. 251–260, 2002.

[128] L. Bilge, T. Strufe, D. Balzarotti, and E. Kirda, "All your contacts are belong to us: automated identity theft attacks on social networks," in *Proceedings of the 18th international conference on World wide web*. ACM, 2009, pp. 551–560.

[129] P.W.L. Fong, "Preventing Sybil attacks by privilege attenuation: A design principle for social network systems," in *Proceedings of the 2011 IEEE Symposium onSecurity and Privacy*. IEEE, 2011, pp. 263–278.

[130] B. Viswanath, A. Post, K.P. Gummadi, and A. Mislove, "An analysis of social network-based sybil defenses," in *ACM SIGCOMM Computer Communication Review*. ACM, 2010, vol. 40, pp. 363–374.

[131] L. Jin, H. Takabi, and J.B.D. Joshi, "Towards active detection of identity clone attacks on online social networks," in *Proceedings of the first ACM conference on Data and application security and privacy*. ACM, 2011, pp. 27–38.

[132] J.Y. Choi, W. De Neve, K.N. Plataniotis, and Y.M. Ro, "Collaborative face recognition for improved face annotation in personal photo collections shared on online social networks," *IEEE Transactions on Multimedia*, vol. 13, no. 1, pp. 14–28, 2011.

[133] B. Qureshi, G. Min, and D. Kouvatsos, "Collusion detection and prevention with FIRE+ trust and reputation model," in *Proceedings of the 10th International Conference on Computer and Information Technology*. IEEE, 2010, pp. 2548–2555.

[134] E. Staab and T. Engel, "Collusion detection for grid computing," in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE Computer Society, 2009, pp. 412–419.

[135] H. Hu and G.J. Ahn, "Constructing authorization systems using assurance management framework," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 40, no. 4, pp. 396–405, 2010.

[136] H. Hu, G.J. Ahn, and J. Jorgensen, "Detecting and resolving privacy conflicts for collaborative data sharing in online social networks," in *Proceed-*

ings of the 27th Annual Computer Security Applications Conference. 2011, ACSAC'11, pp. 103–112, ACM.

[137] H. Hu, G. Ahn, and J. Jorgensen, "Multiparty access control for online social networks: model and mechanisms," *IEEE Transactions on Knowledge and Data Engineering*, vol. pp, no. 99, pp. 1, 2012.

[138] Z. Zhao, G.J. Ahn, H. Hu, and D. Mahi, "SocialImpact: systematic analysis of underground social dynamics," in *Proceedings of the 17th European conference on Research in computer security*. Springer-Verlag, 2012.

[139] Y. Zhu, Z. Hu, H. Wang, H. Hu, and G.J. Ahn, "A collaborative framework for privacy protection in online social networks," in *Proceedings of the 6th International Conference on Collaborative Computing: Networking, Applications and Worksharing*. IEEE, 2010.

[140] H. Takabi, J.B.D. Joshi, and G.J. Ahn, "Security and privacy challenges in cloud computing environments," *Security & Privacy, IEEE*, vol. 8, no. 6, pp. 24–31, 2010.

[141] Y. Zhu, H. Hu, G. Ahn, and M. Yu, "Cooperative provable data possession for integrity verification in multi-cloud storage," *IEEE Transactions on Parallel and Distributed Systems*, , no. 99, 2012.

[142] Y. Zhu, H. Wang, Z. Hu, G.J. Ahn, H. Hu, and S.S. Yau, "Dynamic audit services for integrity verification of outsourced storages in clouds," in *Proceedings of the 2011 ACM Symposium on Applied Computing*. ACM, 2011, pp. 1550–1557.

[143] Y. Zhu, H. Hu, G.J. Ahn, M. Yu, and H. Zhao, "Comparison-based encryption for fine-grained access control in clouds," in *Proceedings of the second ACM conference on Data and Application Security and Privacy*. ACM, 2012, pp. 105–116.

[144] Y. Zhu, H. Hu, G.J. Ahn, D. Huang, and S. Wang, "Towards temporal access control in cloud computing," in *INFOCOM*. IEEE, 2012, pp. 2576–2580.

[145] R. Wu, G.J. Ahn, H. Hu, and M. Singhal, "Information flow control in cloud computing," in *Proceedings of the 6th International Conference on Collaborative Computing: Networking, Applications and Worksharing*. IEEE, 2010, pp. 1–7.

[146] G.J. Ahn, H. Hu, and J. Jin, "Security-enhanced OSGi service environments," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 39, no. 5, 2009.

[147] W. Enck, M. Ongtang, and P. McDaniel, "Understanding android security," *Security & Privacy, IEEE*, vol. 7, no. 1, pp. 50–57, 2009.

[148] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer, "Google android: A comprehensive security assessment," *Security & Privacy, IEEE*, vol. 8, no. 2, pp. 35–44, 2010.

[149] M. Nauman, S. Khan, and X. Zhang, "Apex: Extending Android permission model and enforcement with user-defined runtime constraints," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*. ACM, 2010, pp. 328–332.

[150] R. Wu, G.J. Ahn, and H. Hu, "Towards HIPAA-compliant healthcare systems," in *Proceedings of the 2nd ACM SIGHIT symposium on International health informatics*. ACM, 2012, pp. 593–602.

[151] J.C. Harsanyi and R. Selten, "A general theory of equilibrium selection in games," *MIT Press Books*, vol. 1, 1988.

[152] A.C. Squicciarini, S. Sundareswaran, D. Lin, and J. Wede, "A3P: adaptive policy prediction for shared images over popular content sharing sites," in *Proceedings of the 22nd ACM conference on Hypertext and hypermedia*. ACM, 2011, pp. 261–270.

[153] "Facebook Places," http://www.facebook.com/places/.

[154] "Google Latitude," http://www.google.com/latitude/.

[155] "Google Health," http://www.google.com/health/.

[156] "Microsoft HealthVault," http://www.microsoft.com/en-us/healthvault/.