

CERIAS Tech Report 2005-69

**REQUIREMENTS-BASED ACCESS CONTROL ANALYSIS AND
POLICY SPECIFICATION**

by Qingfeng He

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

REQUIREMENTS-BASED ACCESS CONTROL ANALYSIS
AND POLICY SPECIFICATION

by Qingfeng He

A Dissertation Submitted to the Graduate Faculty of
North Carolina State University
In Partial Fulfillment of the
Requirements for the Degree
of Doctor of Philosophy

(Under the direction of Dr. Ana (Annie) I. Anton.)

North Carolina State University, 2005

Abstract

HE, QINGFENG. Requirements-Based Access Control Analysis and Policy Specification. (Under the direction of Dr. Ana (Annie) I. Antón.)

Access control is a mechanism for achieving confidentiality and integrity in software systems. Access control policies (ACPs) define how access is managed and the high-level rules of who can access what information under certain conditions. Traditionally, access control policies have been specified in an ad-hoc manner, leaving systems vulnerable to security breaches. ACP specification is often isolated from requirements analysis, resulting in policies that are not in compliance with system requirements. This dissertation introduces the Requirements-based Access Control Analysis and Policy Specification (ReCAPS) method for deriving access control policies from various sources, including software requirements specifications (SRS), software designs, and high-level security/privacy policies. The ReCAPS method is essentially an analysis method supported by a set of heuristics and a software tool: the Security and Privacy Requirements Analysis Tool (SPRAT). The method was developed in two formative case studies and validated in two summative case studies. All four case studies involved operational systems, and ReCAPS evolved as a result of the lessons learned from applying the method to these case studies. Further validation of the method was performed via an empirical study to evaluate the usefulness and effectiveness of the approach. Results from these evaluations indicate that the process and heuristics provided by the ReCAPS method are useful for specifying database-level and application-level ACPs. Additionally, ReCAPS integrates policy specification into software development, thus providing a basic framework for ensuring compliance between different levels of policies, system requirements and software design. The method also improves the quality of requirements specifications and system designs by clarifying ambiguities and resolving conflicts across these artifacts.

Requirements-Based Access Control Analysis and Policy Specification

by

Qingfeng He

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Department of Computer Science

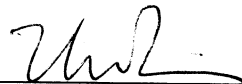
Raleigh, NC

2005

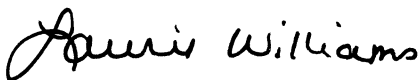
Approved By:



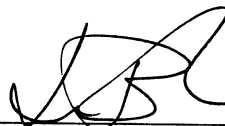
Dr. Ana (Annie) I. Antón
Chair of Advisory Committee



Dr. Ting Yu



Dr. Laurie A. Williams



Dr. Julie B. Earp

*Dedicated to my mom in heaven,
your heart and soul will always be with me;
to my dad;
and to my wife Juanli.*

Biography

Qingfeng He was born in Hubei, China on September 12, 1976. He received the Bachelor of Engineering degree in Electrical Engineering and Automation in 1998 and the Master of Engineering degree in Pattern Recognition and Intelligent Systems in 2001, both from Tsinghua University in China. He entered the doctoral program in the Department of Computer Science at North Carolina State University's College of Engineering in 2001, where he was a member of *The Privacy Place* research group and a member of the Cyber Defense Lab. Qingfeng He was selected as one of the three graduate students in the United States to receive the prestigious CISCO Systems Information Assurance Scholarship in Spring 2005. He was also awarded a U.S. Department of Homeland Security I3P post-doctoral fellowship at Purdue University, which he declined in order to remain in North Carolina with his family. Qingfeng He received his Doctor of Philosophy degree in Computer Science in 2005 from North Carolina State University. He was a member of the Association of Computing Machinery (ACM) and the Institute of Electrical and Electronic Engineers (IEEE). Qingfeng He and his wife, Juanli Guo, were married in 2001. They were expecting their first child in November 2005.

Acknowledgements

This dissertation would not have been possible without the help and support of many people over many years. I must begin by thanking my Ph.D. advisory committee members: in particular, my advisor Dr. Annie Antón. She not only taught me how to be a good researcher, but also taught me how to be a better person. I feel very fortunate and appreciative to have her as my advisor. Dr. Ting Yu has served as a very active committee member and has given me invaluable personal and research guidance. Sincere thanks are also extended to my two other committee members, Dr. Julie Earp and Dr. Laurie Williams, as well as Dr. Purush Iyer, Dr. Peng Ning and Dr. Peter Wurman for their support during my tenure at North Carolina State University (NCSU).

The work presented in this thesis would not have been possible without the support of the National Science Foundation (NSF) via three grants: Information Technology Research (ITR) grants #0113792 and #0325269, and Digital Government (DG) grant #0131886.

Many thanks to Graduate Program Director (GDP), Dr. David Thuent and former GDP, Dr. Edward Davis, both have guided me through my graduate studies at NCSU.

Two professors in the NCSU College of Management deserve a special acknowledgement. Dr. Michael Rappa is a truly inspirational individual who has profoundly influenced me, and Dr. David Baumer has been very supportive of my research and my career.

I've been fortunate to benefit from interactions with professors at other institutions who have provided me with research feedback and career guidance: Dr. Colin Potts, Dr. Eugene H. Spafford, Dr. Steve Fickas, and Prof. Jonathan Moffett. A special thanks to Dr. Colin Potts for his active role in my research projects over the course of the past four years and for his guidance during my empirical research work.

I wish to thank Calvin Powers of IBM Tivoli for his timely feedback on my papers and his support during my job search.

Special thanks are given to Dr. Thomas Honeycutt and David Wright for allowing me to conduct an empirical study in their course, and to the Fall 2004 and Spring 2005 NCSU graduate students who participated.

The Transnational Digital Government (TDG) project team, especially Dr. Jose Fortes, Dr. Stanley Su, Manjiri Patil, Charles McSweeney and Pedro Taveras, contributed to the TDG case study as active collaborators.

Former and current members of *The Privacy Place* research group have been exceptional colleagues: William Stufflebeam, Davide Bolchini, Neha Jain, Jack Frink, Matthew Vail, Carrie Gheen, Hai Yuan, Keith Irwin, Bharathy Sethumadhavan, Paul Otto, Laurie Jones, and Travis Beaux. Paul and Laurie helped me further validate the ReCAPS method and evaluate the SPRAT as participants in the Event Registration System (ERS) case study. I also wish to thank my office mates: Mark Sherriff, Lucas Layman, Sarah Smith, and my former office mate and companion along the same path: Nachiappan Nagappan.

Sincere thanks are given to the NCSU Computer Science Department staff. In particular, Fay Ward was a helpful thesis proofreader, and the following individuals provided administrative and technical support: Margery Page, Vilma Berg, Ginny Adams, Carlos Benavente, Jason Corley, Ron Hartis, Carol Holloman, Missy Seate, and Charlene Lassiter.

My parents, who raised me to be the person I am today, deserve the most acknowledgement. Finally, I thank my beloved wife, Juanli, for understanding, supporting, and loving me.

Table of Contents

List of Figures	x
List of Tables	xii
List of Abbreviations	xiv
Glossary	xvii
Chapter 1 Introduction	1
1.1 Research Context	2
1.2 Data Security and Privacy: A Healthcare Scenario	3
1.3 Access Control Analysis and Policy Specification.....	5
1.4 Motivation of This Work and Problem Statement.....	7
1.5 Overview of This Work.....	10
1.6 Research Methodology and Classification.....	11
1.7 Overview of Remaining Chapters	16
Chapter 2 Background and Related Work.....	17
2.1 Requirements Engineering (RE)	17
2.1.1 Classification of Requirements Engineering Research Efforts	18
2.1.2 Goal-Based Requirements Analysis	20
2.1.3 Scenario-Based Requirements Analysis.....	24
2.1.4 From Requirements Analysis to Software Design	25
2.2 Security Requirements Engineering.....	26
2.2.1 Security and Privacy Requirements Analysis	27
2.2.2 Access Control Analysis in Requirements Engineering	28
2.3 Access Control in Security	31
2.3.1 Access Control	32
2.3.2 Access Control Policies	33
2.3.3 Elements of Access Control Policies.....	33
2.3.4 Current Research Efforts on Access Control Policy Specifications	35
2.3.5 Role Engineering	36
2.4 Security and Privacy Policy Analysis and Specification.....	37
2.4.1 Policy Specification Languages	37

2.4.2	<i>Security Policy Analysis</i>	41
2.4.3	<i>Privacy Policy Analysis</i>	44
2.5	Summary	46
Chapter 3	Formative Case Studies	48
3.1	Security and Privacy Requirements Analysis Tool (SPRAT).....	49
3.1.1	<i>Methodology and Case Study Artifacts</i>	50
3.1.2	<i>Lessons Learned</i>	52
3.1.3	<i>Results</i>	61
3.2	Transnational Digital Government (TDG).....	62
3.2.1	<i>Methodology and Case Study Artifacts</i>	63
3.2.2	<i>Lessons Learned</i>	64
3.2.3	<i>Results</i>	69
3.3	Summary	70
Chapter 4	Requirements-based Access Control Analysis and Policy Specification	72
4.1	Overview of ReCAPS.....	73
4.1.1	<i>An ICOM Model of ReCAPS</i>	73
4.1.2	<i>Assumptions</i>	75
4.1.3	<i>Design Principles</i>	75
4.1.4	<i>Activities</i>	76
4.2	Analysis Process and Heuristics.....	78
4.2.1	<i>Preparation</i>	79
4.2.2	<i>Access Control Rule Identification & Specification</i>	80
4.2.2.1	Identifying Objects	84
4.2.2.2	Identifying Subjects and Actions	86
4.2.2.3	Identifying Conditions	92
4.2.2.4	Identifying Obligations.....	102
4.2.2.5	Summary	104
4.2.3	<i>Access Control Rules Refinement</i>	104
4.2.3.1	Identifying and Removing Redundancies.....	105
4.2.3.2	Identifying and Resolving Conflicts	107
4.2.4	<i>Grouping AC Rules into ACPs</i>	109
4.2.5	<i>Summary</i>	112
4.3	Tool Support	113
4.3.1	<i>Overview</i>	113

4.3.2	<i>ACP Specification Module</i>	116
4.3.3	<i>Design and Implementation</i>	123
4.4	Summary	124
Chapter 5 Validation		125
5.1	Surry Arts Council (SAC) Web Enhancement	127
5.1.1	<i>Methodology and Case Study Artifacts</i>	128
5.1.2	<i>Lessons Learned</i>	129
5.1.3	<i>Summary and Discussion</i>	134
5.2	NCSU College of Management Event Registration System.....	136
5.2.1	<i>Methodology and Case Study Artifacts</i>	137
5.2.2	<i>Lessons Learned</i>	139
5.2.3	<i>Summary and Discussion</i>	147
5.3	An Empirical Evaluation of ReCAPS	149
5.3.1	<i>Experimental Method</i>	150
5.3.2	<i>Results</i>	157
5.3.2.1	Quality of Access Control Policies	160
5.3.2.2	Improvements to Source Documents	165
5.3.2.3	Time Effort.....	166
5.3.3	<i>Summary and Discussion</i>	168
5.4	Summary	171
Chapter 6 Conclusions		173
6.1	Chapter Synopsis	174
6.2	Summary of Contributions	175
6.3	Current Limitations and Plans for Future Work.....	178
6.4	Conclusions.....	181
Appendix A Summary of ReCAPS Analysis Activities.....		182
Appendix B Summary of ReCAPS Heuristics		183
Appendix C Evolution of the ReCAPS Method		188
C.1	ReCAPS Method Summary Version 1 (Pre-SPRAT).....	188
C.2	ReCAPS Method Summary Version 2 (Pre-TDG)	189
C.3	ReCAPS Method Summary Version 3 (Pre-SAC)	194
C.4	ReCAPS Method Summary Version 4 (Pre-ERS).....	202

Appendix D	Experimental Instrumentation for Empirical Study.....	203
D.1	NCSU Informed Consent Form for Research.....	203
D.2	ReCAPS Group Assignment Description and Worksheets	205
D.3	Control Group Assignment Description and Worksheets	216
Bibliography	224

List of Figures

Figure 1.1	Access Control in a Real-World Healthcare Scenario.....	4
Figure 1.2	Policy Hierarchy and Software Development Process	8
Figure 1.3	The ReCAPS Approach for Specifying Access Control Policies	11
Figure 2.1	A Six-Dimentional View of RE Research Efforts	18
Figure 3.1a	Scenario Analysis in the TDG Case Study.....	66
Figure 3.1b	Scenario Analysis in the TDG Case Study.....	67
Figure 4.1	ReCAPS ICOM Model	74
Figure 4.2	ReCAPS Analysis Activities	77
Figure 4.3	A Misuse Case for Project Manager.....	98
Figure 4.4	Purpose Hierarchy for Marketing.....	100
Figure 4.5	Elements Maintained in the SPRAT.....	114
Figure 4.6	SPRAT Screen Shot for ACR Identification.....	117
Figure 4.7	SPRAT Screen Shot for Creating a New Subject	118
Figure 4.8	SPRAT Screen Shot for Viewing Source Detail.....	118
Figure 4.9	SPRAT Screen Shot for Refining ACRs.....	119
Figure 4.10	SPRAT Screen Shot for Viewing ACRs	120
Figure 4.11	SPRAT Screen Shot for Editing ACRs	121
Figure 4.12	SPRAT Screen Shot for Editing an Action	121
Figure 5.1	Evaluation of a Subject's Rule Set.....	159
Figure 5.2	Number of Rules Identified by Each Subject that Were Also Identified by the Experts.....	161
Figure 5.3	Number of False Positive Rules Identified by Each Subject.....	161
Figure 5.4	Number of Rules with Ambiguous Actions/Objects Identified by Each Subject.....	162
Figure 5.5	Number of Rules with Every Element Correctly Specified by Each Subject.....	163
Figure 5.6	Formula to Calculate the Percentage of Rules in Each Category.....	163
Figure 5.7	Percentage of Rules with Conditions Incomplete or Incorrectly Specified	164
Figure 5.8	The ReCAPS Group Provided Better Traceability Support than the Control Group	165

Figure 5.9	Time Spent on the Experiment by Each Subject.....	166
Figure 5.10	Correlation Analysis Shows There is No Correlation between the Effort and the Results in all Subjects	167
Figure 5.11	Correlation Analysis Shows There is No Correlation between the Time Effort and the Results in both ReCAPS and Control Group	167
Figure 6.1	Traceability Support in ReCAPS.....	177
Figure C.1	The Process of Access Control Analysis.....	195

List of Tables

Table 1.1	Shaw's Characterizations of Software Engineering Research Settings [Sha01]	14
Table 1.2	Shaw's Characterizations of Software Engineering Research Approaches and Products [Sha01]	14
Table 1.3	Shaw's Characterizations of Software Engineering Research Validation Techniques [Sha01]	15
Table 2.1	Purposes Defined in P3P1.1 [P3P05]	39
Table 2.2	Primary Purposes Defined in P3P1.1 [P3P05]	40
Table 2.3	Recipients Defined in P3P1.1 [P3P05]	41
Table 3.1	An Example Access Control Matrix in the SPRAT Case Study	51
Table 3.2	An Example Traditional Access Control Matrix.....	52
Table 3.3	Scenario Analysis for SPRAT Requirement FR-PM-3.....	58
Table 3.4	Summary of the SPRAT Case Study Results.....	62
Table 3.5	Summary of the TDG Case Study Results	70
Table 3.6	Inconsistencies Identified in the TDG Case Study.....	71
Table 4.1	Mapping between ACP Elements and Requirements Specification Elements.....	80
Table 4.2	Scenario Analysis for SPRAT Requirement FR-PM-3.....	89
Table 4.3	Subjects and Actions Identified from SPRAT Requirement FR-PM-3.....	89
Table 4.4	Scenario Analysis for SPRAT Requirement FR-GSM-3.....	91
Table 4.5	Actors and Actions Identified from SPRAT Requirement FR-GSM-3.....	92
Table 4.6	Access Control Rules Derived from the Misuse Case	98
Table 4.7	Grouping Access Control Rules into a Policy to Resolve Conflicts	108
Table 4.8	ACPs for Role (System Admin) in the SPRAT Case Study	111
Table 5.1	Scenario Analysis for SAC Requirement FR1.4.....	132
Table 5.2	Rule #1 and Rule #2 in the SAC Case Study	134
Table 5.3	Summary of the SAC Case Study Results.....	135
Table 5.4	Heuristic Usage Frequency in the SAC Case Study	136
Table 5.5	Rule #50 and Rule #70 in the ERS Case Study.....	140
Table 5.6	Source Detail for Rule #50 and Rule #70.....	141

Table 5.7	Rule #60 and Rule #62 in the ERS Case Study.....	144
Table 5.8	Rule #49 and Rule #75 in the ERS Case Study.....	145
Table 5.9	Rule #65 and Rule #78 in the ERS Case Study.....	146
Table 5.10	Source Detail for Rule #65 and Rule #78.....	146
Table 5.11	Rules #76, #77, #78 in the ERS Case Study.....	147
Table 5.12	Summary of the ERS Case Study Results	148
Table 5.13	Heuristics Usage Frequency in the ERS Case Study	149
Table 5.14	Class Distribution of the Subjects Who Participated in the Experiment.....	152
Table 5.15	Knowledge Background of the Subjects Who Participated in the Experiment.....	153
Table 5.16	Summary of the Empirical Evaluation Results	160
Table B.1	ReCAPS Heuristics for Identifying AC Scope (Step 2).....	183
Table B.2	ReCAPS Heuristics for Identifying Objects (Step 2.1).....	183
Table B.3	ReCAPS Heuristics for Identifying Subjects and Actions (Step 2.2).....	184
Table B.4	ReCAPS Heuristics for Identifying Conditions (Step 2.3).....	185
Table B.5	ReCAPS Heuristics for Identifying Obligations (Step 2.4)	186
Table B.6	ReCAPS Heuristics for Specifying AC Rules (Step 2).....	186
Table B.7	ReCAPS Heuristics for Identifying and Removing Redundancies (Step 3.2).....	187
Table B.8	ReCAPS Heuristics for Identifying and Resolving Conflicts (Step 3.3).....	187
Table B.9	ReCAPS Heuristics for Grouping AC Rules in ACPs (Step 3.4)	187
Table C.1	Example ACP from the SPART Case Study	202

List of Abbreviations

AC	Access Control
ACP	Access Control Policy
ACR	Access Control Rule
ALP	Application-Level Policy
ASL	Authorization Specification Language
AWRE	Australian Workshop on Requirements Engineering
BFM	Boundary Flow Modeling
BIA	Border Immigration Agent
CASE	Computer-Aided Software Engineering
CoM	College of Management
COPPA	Children's Online Privacy Protection Act
DAC	Discretionary Access Control
DLP	Database-Level Policy
DR	Dominican Republic
EPAL	Enterprise Privacy Authorization Language
E/R	Entity/Relationship
ERS	Event Registration System
FIP	Fair Information Practice
FR	Functional Requirement
FTC	Federal Trade Commission
GBRAM	Goal-Based Requirements Analysis Method
GLBA	Gramm-Leach-Bliley Act
GQM	Goal-Question-Metric

HIPAA	Health Insurance Portability and Accountability Act
HRU	Harrison-Ruzzo-Ullman
ICOM	Inputs-Controls-Outputs-Mechanisms
ICSE	International Conference on Software Engineering
IFIP	International Federation for Information Processing
IRB	Institutional Review Board
JDBC	Java DataBase Connectivity
KAOS	Knowledge Acquisition in AutOmated Specification
MAC	Mandatory Access Control
MEM	Multilateral Evaluation Mechanism
NCSC	National Computer Security Center
NCSU	North Carolina State University
NFR	Non-Functional Requirement
NSF	National Science Foundation
OAS	Organization of American States
OCL	Object Constraint Language
OECD	Organization for Economic Cooperation and Development
P3P	The Platform for Privacy Preferences Project
PGMT	Privacy Goal Management Tool
PHI	Protected Health Information
POE	Point-Of-Entry
RBAC	Role-Based Access Control
RE	Requirements Engineering
ReCAPS	Requirements-based Access Control Analysis and Policy Specification
REFSQ	International Workshop on Requirements Engineering: Foundation for Software Quality

SAC	Surry Arts Council
SD	Strategic Dependency
SEI	Software Engineering Institute
SMaRT	Scenario Management and Requirements Tool
SPRAT	Security and Privacy Requirements Analysis Tool
SR	Strategic Rationale
SRS	Software Requirements Specifications
TDG	Transnational Digital Government
UML	Unified Modeling Language
W3C	World Wide Web Consortium
XACML	eXtensible Access Control Markup Language
XML	eXtensible Markup Language

Glossary

Access control. *Access control* ensures that every access to a system and its resources is controlled according to a set of predefined policies. It is one of the major security mechanisms used to achieve confidentiality, integrity and privacy in software systems. An access control system is typically described in three ways: access control policies, models, and mechanisms.

Access control mechanism. *Access control mechanisms* define the low-level functions that implement the controls imposed by access control policies. The mechanism must work as a reference monitor, a trusted component intercepting each and every request to the system.

Access control model. *Access control models* formally represent an access control system. They provide ways to reason about the policies they support and prove the security properties of the access control system. Access control models provide a level of abstraction between policies and mechanisms, enabling the design of implementation mechanisms to enforce multiple policies in various computing environments.

Access control policy. *Access control policies* are security requirements that describe how access is managed, what information can be accessed by whom, and under what conditions that information can be accessed. These policies are enforced via a mechanism that mediates access requests and makes grant/deny decisions. In this dissertation, access control policies are comprised of access control rules that are specified in a structured format.

Access control rule. A typical *access control rule* in this dissertation is expressed as a 5-tuple $\langle \textit{subject}, \textit{object}, \textit{action}, \textit{condition}, \textit{obligation} \rangle$, such that a *subject* can perform some *action* on an *object*. Additionally, each access control rule has a *mode* (e.g., allow/deny/refrain/oblige).

Action. An *action* is a simple operation (e.g. read or write) or an abstract operation (e.g. deposit or withdraw) performed by an agent.

Affiliation constraints. *Affiliation constraints* specify a subject's corporate, organizational, or group affiliation.

Attribute constraints. *Attribute constraints* specify a subject must possess some attribute (e.g., digital certificates)) for an access request to be granted.

Authentication constraints. *Authentication constraints* reflect the need for a subject to be authenticated before data access can be granted.

Condition. A *condition* is a provision that must be satisfied before an access request can be granted.

Confidentiality. *Confidentiality* means that information is not disclosed to unauthorized persons, processes or devices.

Consent constraints. *Consent constraints* require a subject to acknowledge consent for their information to be used in some specific way or purpose before data access can be granted for that purpose.

Contextual constraints. *Contextual constraints* reflect the need to consider the context of an access request when making grant/deny decisions, such as the time of the access request or the location from which the access request is made.

Database constraints. *Database constraints* specify restrictions on a database access request, such as when there can be no duplicate entry in a table.

Goal. *Goals* are targets for achievement and high-level objectives of a business, organization or system. They express the rationales for the proposed system and serve as high-level expressions of the system requirements and/or policies.

Integrity. *Integrity* means that unauthorized persons, processes or devices cannot modify information.

Location constraints. *Location constraints* specify a particular location from which the subject can be granted access to a resource.

Misuse case. A *misuse case* is a use case from the point of view of an actor with hostile intent.

Object. An *object* is an entity, such as a data field, table, procedure or application to which access is restricted.

Obligation. An *obligation* is a task that must be fulfilled if a request to access an object is granted.

Privacy. *Privacy* implies that data is protected so that it is used only for authorized business purposes, based on legal requirements, corporate policies and end-user choices.

Privacy constraints. *Privacy constraints* specify restrictions to data access requests in which the data are particularly sensitive (e.g., medical history, financial data).

Purpose constraints. *Purpose constraints* specify that data can be used only for specific purposes.

Recipient constraints. *Recipient constraints* specify which group of people can access the specified data.

Relationship constraints. *Relationship constraints* specify a specific relationship between the subject and the object for an access request.

Requirement. A *requirement* defines capabilities that a system must provide in order to satisfy stakeholders' goals. Requirements can be classified as either *functional requirements*, which describe the functional aspects of a system, or *non-functional requirements*, which describe the properties, attributes, and constraints under which a system must operate.

Scenario. A *scenario* is comprised of a sequence of events that describe possible ways for users to interact with a system.

Security constraints. *Security constraints* specify restrictions that are based on general security principles such as least privileges and separation of duties.

Stakeholder. A *stakeholder* is anyone who claims an interest in a system, such as system developers, sponsors, end-users, and customers.

State constraints. *State constraints* limit data access based upon the reaching of some specific state within the system.

Subject. A *subject* is an entity, such as a user or program agent that may access objects.

Temporal constraints. *Temporal constraints* specify time or date related restrictions that must be enforced before data access can be granted.

Usage constraints. *Usage constraints* specify restrictions on how a subject may access the requested object, such as the number of times the subject can access the object.

Use case. A *use case* is a notation for modeling users' interactions with an existing or envisioned system, the description details the interaction using a sequence of events. In this dissertation, use cases and scenarios are used synonymously.

Chapter 1

Introduction

良好的开端是成功的一半。

Well begun is half done.

—Chinese Proverb

The research in this dissertation is aimed at improving information system security during the early stages of the software development process, namely, requirements analysis and software design. Data security and privacy is important in every software system, but particularly vulnerable in critical infrastructure systems such as medical, immigration, and financial information systems. Specifying correct policies to control users' access to a system and its resources is critical for protecting data security and privacy. It is important to specify access control policies during software development so that security can be built into software products as advocated in this dissertation. However, there has been little reported work [SMJ01] in helping software and security engineers specify access control policies (ACPs) for information systems. ACP specification is often performed in an ad-hoc manner by security engineers, leaving systems vulnerable to security and privacy breaches. Additionally, ACP specification is typically isolated from requirements analysis and software design, a practice that could result in policies that are not in compliance with system requirements. The primary focus of this work is to formulate and validate a method that is useful for specifying ACPs, bringing policies and requirements into better alignment. Specifically, the Requirements-based Access Control Analysis and Policy Specification (ReCAPS) method detailed in this dissertation integrates policy specification into software development and provides procedural support for specifying access control policies for information systems. The objective of this chapter is to establish the context for this research.

This chapter is structured as follows. Section 1.1 introduces the research context of this work. Section 1.2 provides a healthcare scenario to illustrate the role of access control in protecting data security and privacy. Section 1.3 briefly summarizes access control and current policy specification research and practices in the software development process. Section 1.4 introduces the motivation for this work and summarizes the problem statement. Section 1.5 overviews the work presented in this dissertation. Section 1.6 summarizes the research methodology and classifies this work within the context of software engineering research. Section 1.7 provides an overview of the remaining chapters.

1.1 Research Context

This work is situated in the areas of software engineering and information security. Specifically, this work addresses security and privacy problems during the early stages of the software development process, i.e., requirements analysis and software design.

Software engineering refers to the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software [IEE93]. Requirements engineering (RE) is the process of discovering real-world goals for which a software system is intended by identifying stakeholders and their needs, identifying the functions of and constraints on the intended system, and documenting these in a form that is amenable to analysis, communication, and subsequent implementation [NE00]. Example RE activities include requirements elicitation, analysis, modeling, elaboration, specification, negotiation, prototyping, prioritization, reasoning, and validation.

Requirements engineering is a challenging and important phase in software development. Brooks claims that establishing detailed technical requirements is the hardest single part of building a software system [Bro87]. If the work of requirements analysis is not performed correctly, the errors will cripple the resulting software products and are very difficult to rectify

later. The cost of repairing requirements errors increases significantly if these errors are not detected and corrected until a later phase in software development [Boe81]. The later a problem is found during software development, the more expensive it is to fix the problem.

Information security concerns the protection of information against unauthorized disclosure, transfer, modification, or destruction, whether accidental or intentional [ANS01].

Information privacy concerns the ability for individuals, groups and institutions to determine for themselves when, how and to what extent information about them is communicated to others [Wes67].

The work presented in this dissertation involves developing and applying RE methodologies to analyze access-related system requirements (including functional requirements, security and privacy requirements) and high-level security/privacy policies and to specify them as access control policies for enforcement so that a sufficient degree of security and privacy assurance to meet the needs of stakeholders may be achieved.

1.2 Data Security and Privacy: A Healthcare Scenario

Data security and privacy is an important part of information security. As the Internet and e-commerce have prospered, the security and privacy of personal data has become of increasing concern to consumers, developers, and legislators. Security and privacy violations are increasingly disclosed via the Internet, TV, newspaper and other media, such as the ChoicePoint [Cho04] and JetBlue [AHB04] cases. Legislative acts in the U.S. and Europe (e.g., the Health Insurance Portability and Accountability Act (HIPAA) for healthcare [HIP96], the Gramm-Leach-Bliley Act (GLBA) for financial institutions [GLB01], the Children's Online Privacy Protection Act (COPPA) [COP98], and the European Union's Data Protection Directive [DPD97]) require companies and organizations to protect consumers' data security and privacy.

According to a 2002 research report by the Gartner Group, 70% of cases involving unauthorized information access are committed by insiders or people given access to a company's network resources [Bro02]. The average cost of insider attack (\$2.7M) is significantly higher than the average cost of outside attack (\$57K) [CSI02]. More recently, according to the 2004 E-Crime Watch Survey, unauthorized access by an insider is ranked fourth (36%) in the *Types of Electronic Crimes* list—after virus and other malicious code (77%), denial of service attack (44%), and illegal generation of SPAM emails (38%) [CSO04]. All these data show that it is very important to have better security policies to control users' access to a system and its resources.

To obtain a better understanding of the role of access control in protecting data security and privacy, we examine the following real-world scenario in the healthcare domain (see Figure 1.1). Consider a patient who goes to a hospital to see a doctor. For the purpose of this scenario, we assume this is a female patient. The general patient visit process is as follows. She checks in at the front desk. The receptionist at the front desk requires her insurance card or some other ID to check her in. Then she goes through nurses and interns to finally meet the doctor. All these people have direct access to her medical information. After the meeting and diagnosis, the doctor

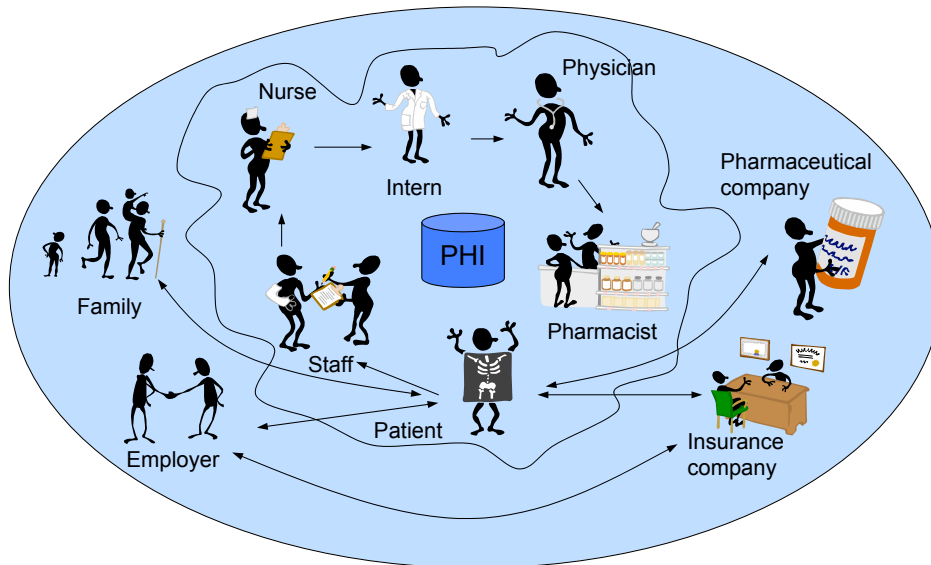


Figure 1.1 Access Control in a Real-World Healthcare Scenario

gives her a prescription, and she goes to the pharmacy to pick up the medicine. During this process, from front desk check-in to medicine pick-up at the pharmacy, there are many people who need to access the patient's personal information, which is often called PHI (protected health information). Additionally, outside the hospital we have other actors, such as the patient's family members or her insurance company who may need to access her personal information.

A major security challenge in this scenario is, given such a complex system, how can we specify correct and complete policies for each actor who needs to access PHI? For example, consider the following two cases:

Case 1: The hospital is a university hospital. The patient's doctor happens to be a faculty member in the university's medical school. Because the patient's case is very special, the doctor decides to use this case in a class he is teaching. What part of this information can he share? If this sharing is allowed under certain conditions, what are the conditions? Are there any obligations for the doctor after the information access?

Case 2: The patient's case is very complicated, and his doctor decides to exchange some ideas with an expert in this field, a doctor at another hospital. What kind of information can the doctor share with the expert?

The above cases concern the security and privacy of medical data. The purpose of answering the above questions is to specify correct and complete access control policies for each user in the system, which is critical for protecting data security and privacy. The work presented in this dissertation provides systematic methodological support for specifying access control policies.

1.3 Access Control Analysis and Policy Specification

The function of access control (AC) is to ensure that every access to a system and its resources is controlled according to a set of predefined policies [SV01]. Access control is one of

the major security mechanisms used to achieve confidentiality, integrity and privacy in software systems [FKC03]. We use these terms as follows:

- *Confidentiality* means that information is not disclosed to unauthorized persons, processes or devices.
- *Integrity* means that unauthorized persons, processes or devices cannot modify information.
- *Privacy* implies that data is protected so that it is used only for authorized business purposes, based on legal requirements, corporate policies and end-user choices.

An access control system is typically described in three ways: access control policies, models, and mechanisms [SV01]. *Access control policies* are security requirements that describe how access is managed, what information can be accessed by whom, and under what conditions that information can be accessed [FKC03]. These policies are enforced via a mechanism that mediates access requests and makes grant/deny decisions. *Access control models* provide a formal representation of an access control system. They provide ways to reason about the policies they support and prove the security properties of the access control system. Access control models provide a level of abstraction between policies and mechanisms, enabling the design of implementation mechanisms to enforce multiple policies in various computing environments. The *access control mechanism* defines the low-level functions that implement the controls imposed by the policies. The mechanism must work as a reference monitor [SV01], a trusted component intercepting each and every request to the system.

Access control analysis entails analyzing system requirements, organizational security and privacy policies, and organizational structures to specify access control policies. Defining access control policies is both a conceptually and practically complex process because software systems can have many users performing various tasks and many resources that need to be protected [Sch00, SMJ01]. Organizational complexity presents another challenge—it is difficult to identify and agree upon a common set of roles and associated permissions within an organization that may

have hundreds of roles to be considered. In practice, ACP specification is often conducted without prescriptive guidance [CIN03, SL02], leaving systems vulnerable to security and privacy breaches. There is a need for systematic procedural support for specifying access control policies.

The specification of access control policies for information systems is the focus of this dissertation. Access control policies are derived from and must comply with security requirements. Thus, the ACP specification process is basically a requirements engineering process. Moffett et al. define security requirements as constraints on functional requirements that are needed to achieve security goals [MHN04]. Ideally, security requirements are analyzed and specified *before* system design rather than as an afterthought [AE01]. These security requirements should drive ACP specification activities, but policy specification efforts often occur after systems are designed and deployed [CIN03]. Because policy specification efforts are often isolated from requirements analysis, the resulting ACPs and requirements may not be compliant with one another—a situation that the method described in this dissertation seeks to avoid.

Researchers recognize the need to bridge the gap between requirements engineering (RE) and complex ACP specification [CIN03, Fon01, HA03]. Existing RE approaches (e.g., KAOS [DLF93], i* [Yu93] and the analytical role modeling framework [CIN03]) provide limited support, as we discuss in Chapter 2. The goal of this dissertation is to develop a method for ACP specification that provides prescriptive guidance and better aligns policies and system requirements.

1.4 Motivation of This Work and Problem Statement

This work is motivated by the observation that policies and requirements are often misaligned [AE01, AEP01, AEC03], resulting in security and privacy violations. The policies referred to here could be high-level policies or access control policies. Requirements and high-

level policies are very similar. They both express desire or worth rather than fact. Their statements are in optative mood, specifying what must or ought to be done. These are the differences between high-level policies and requirements: (1) Policies are more open-ended than requirements in that policies often describe general and high-level goals; whereas requirements are specific, unambiguous and verifiable. (2) The scope of policies is broader than requirements in that requirements cover one system, whereas policies often govern several systems [AEP01].

To better understand the motivation for this work, we examine policy hierarchy and software development, as shown in Figure 1.2. On the left side of this figure, there are different kinds of policies that are specified in different languages. The policy levels shown in Figure 1.2 are different from the policy hierarchies defined by Moffett and Sloman [MS93]. Moffett and Sloman define several relationships that may exist between different levels of policies, such as partitioned targets, goal refinement, delegation of responsibility. We classify policies according to the form of expression. Policies at the top level are stated in natural language; examples include website privacy policies, corporate security policies, and security and privacy laws. These policies are specified by lawmakers or administrative personnel within an organization, such as security and privacy officers. Policies at the middle level are specified in declarative or semi-structured languages, such as Ponder [Dam02] and XACML [OAS05]. These policies instantiate the high-

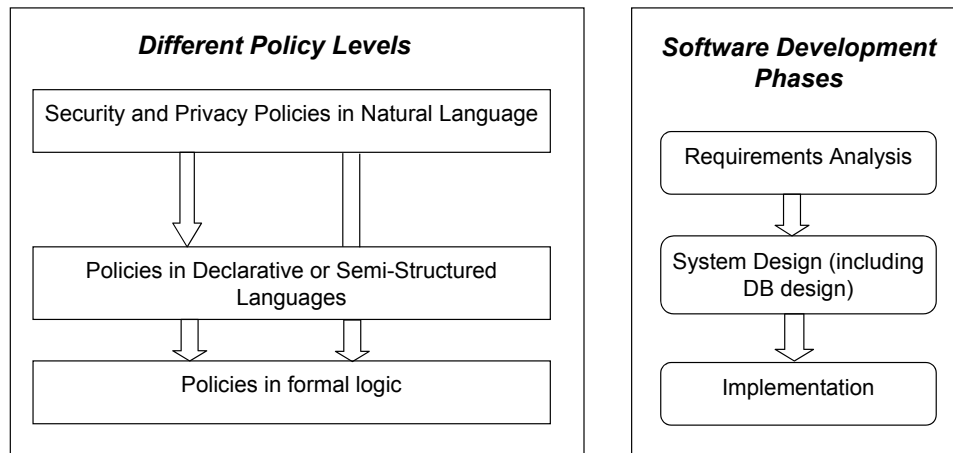


Figure 1.2 Policy Hierarchy and Software Development Process

level policies into rules that describe who has permission to access which object in a specific system. These policies are often specified by system security administrators who manage users and their privileges. The mid-level policies are the focus of this dissertation. Policies at the bottom level are specified in formal logic such as Authorization Specification Language (ASL) [JSS97] and Alloy [Jac02]. This kind of policy is often specified for analysis and verification purpose by security specialists who investigate a system's security properties using formal methods.

On the right side, three software development phases are shown: requirements analysis, system design, and implementation. Requirements analysis focuses on eliciting stakeholders' objectives and specifying them as system requirements. Based on the requirements specification, software designers develop an architectural design for the system, decompose the system into functional modules, define the interfaces between modules, design data flows, generate pseudo codes and test cases for the modules, and design the databases. Based on the requirements and design specifications, software developers implement the system.

Traditionally, policy specification is isolated from requirements analysis and software design, often resulting in either high-level policies that are not in compliance with system requirements, or mid-level access policies that are not in compliance with system requirements.

In conclusion, the problem statement of this work may be summarized as this. *Because ACP specification is typically isolated from requirements analysis, the resulting policies often do not comply with system requirements. This perpetuates the development of systems that neither comply with the software requirements nor adequately protect the information with which they are entrusted. Software and security engineers need methodological support for specifying ACPs and ensuring compliance with software requirements as well as high-level policies.*

1.5 Overview of This Work

The approach presented in this dissertation, Requirements-based Access Control Aalysis and Policy Specification (ReCAPS), integrates policy specification into the software development process, as shown in Figure 1.3.

Moffett treats high-level policies as requirements and low-level policies as an implementation of them [MHN04, Mof99]. We concur with this view. High-level security and privacy policies should be specified as system requirements (either functional or non-functional requirements). Mid-level policies are instances of high-level policies within a specific system’s context. To specify these policies, we must examine system requirements to identify users and their interactions with the system and examine system designs (e.g., database design) to identify the data to be protected.

ACP specification is an iterative process. Although we derive policies from requirements and design, we also improve requirements and design during ACP analysis, by clarifying ambiguities in the requirements and resolving inconsistencies between the requirements and the software design (shown as bidirectional arrows from mid-level policies to requirements analysis and system design in Figure 1.3).

As previously mentioned, this dissertation focuses on the specification of mid-level policies (shown as three bolded boxes in Figure 1.3). There are two major advantages in focusing on mid-level policies. First, they are machine-enforceable, whereas natural languages policies are not. Second, it is relatively easy for software and security engineers (referred to as “analyst(s)” throughout this dissertation) to specify policies that meet organizational security goals using declarative mid-level languages.

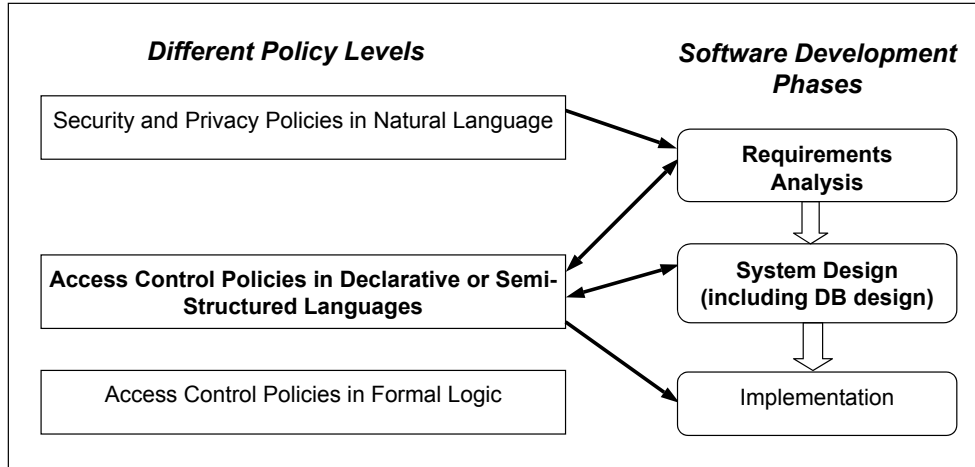


Figure 1.3 The ReCAPS Approach for Specifying Access Control Policies

The major contribution of this approach is the software development scheme introduced in this dissertation that ensures compliance across different levels of policies, system requirements, and software designs by integrating policy specification with requirements analysis and software design. The compliance is achieved in two ways. First, we derive access control policies from system requirements and high-level security and privacy policies. Because security requirements come from these sources, this development scheme helps ensure that a software system is actually enforcing high-level security/privacy laws and policies. Second, we establish traceability links between high-level policies, system requirements, and access control policies. This traceability support helps ensure that any changes in the high-level policies can be easily traced to the corresponding software development artifacts (e.g., requirements specifications, DB designs, ACPs).

1.6 Research Methodology and Classification

The research methodology employed in this dissertation is a conventional software engineering research methodology: the process of conceptualization, empirical exploration, and validation. Before this study, there was no available procedural guidance for specifying access

control policies. The ReCAPS method was developed while analysts were performing qualitative studies of actual development efforts on operational systems. Formative case studies were used to develop the analysis process and heuristics, and summative case studies were used to validate these ideas. The key distinction between formative and summative is that formative case studies involve the development and evolution of the methods simultaneously coupled with validation, whereas summative case studies involve the validation of previously developed methods. It is important to note that the ReCAPS process and heuristics were constantly refined as a result of the lessons learned from these case studies. To summarize the research methodology of this work, we propose a method and try it out on real projects, then refine the method and validate it on real projects. In this research, case studies play a critical role because they are used as the origin and validation of the methodology.

We use Shaw's classification of software engineering research paradigms in terms of research settings, products/approaches and validation techniques [Sha01] to characterize the work in this dissertation. The research settings of this work, according to Shaw's characterizations (Table 1.1), are feasibility, characterization, and methods/means. Some of the corresponding questions are:

- Feasibility: is it possible to ensure compliance between access control policies, system requirements, and high-level security/privacy policies?
- Characterization: what are the quality criteria for a set of access control policies?
- Method/Means: how can we specify access control policies for information systems?
- Method/Means: how is compliance between access control policies, system requirements and high-level security/privacy policies ensured?

The research products of this work, according to Shaw's characterizations (Table 1.2), are descriptive models, techniques, and systems. The research products and approaches of this work are:

- Descriptive models: to organize and report interesting observations from case studies;
- Descriptive models: to structure a problem area (i.e., how to specify ACPs for information systems to ensure compliance), by establishing the relationships between policy specification and software development;
- Descriptive models: to do a careful analysis of the ACP specification in several information systems;
- Techniques: to invent new ways to specify correct access control policies for information systems;
- Systems: to embody the research results in a software tool that supports the ReCAPS method; and
- Systems: to use case studies as a source of insight and for validation of results.

The validation techniques used in this research, according to Shaw's characterizations (Table 1.3), are:

- Persuasion, on the grounds of heuristics and examples presented in this dissertation;
- Implementation, of a software tool that supports the ReCAPS method;
- Evaluation, with respect to the effectiveness and usefulness of the approach in a controlled environment, coupled with statistical analysis; and
- Experience, expressed as lessons learned from case studies.

Table 1.1 Shaw's Characterizations of Software Engineering Research Settings
[Sha01]

Research Settings	Sample Questions
Feasibility	Is there an X, and what is it? Is it possible to accomplish X at all?
Characterization	What are the important characteristics of X? What is X like? What exactly do we mean by X? What are the varieties of X, and how are they related?
Method/Mean	How can we accomplish X? What is a better way to accomplish X? How can we automate doing X?
Generalization	Is X always true of Y? Given Y, what will X be?
Selection	How do I decide between X and Y?

Table 1.2 Shaw's Characterizations of Software Engineering Research Approaches and Products [Sha01]

Research Product	Research Approach or Method
Qualitative or descriptive model	Organize and report interesting observations Create and defend generalizations from real examples Structure a problem area Formulate the right questions Do a careful analysis of a system or its development
Technique	Invent new ways to do some task Develop a technique to choose among alternatives
System	Embody the result in a system Use the system development as both source of insight and carrier of results
Empirical predictive model	Develop predictive models from observed data
Analytic model	Develop structural (quantitative or symbolic) models that permit formal analysis

Table 1.3 Shaw's Characterizations of Software Engineering Research Validation Techniques [Sha01]

Validation Technique	Character of Validation
Persuasion	I have thought hard about this, and I believe that...
Technique	...if you do it the following way, then...
Design	...a system constructed like this would
Example	...walking through this example shows how my idea works
Implementation	Here is a prototype of a system that...
System	...exists in code or other concrete form
Technique	...is represented as a set of procedures
Evaluation	Given these criteria, here is how an object rates...
Descriptive model	...in a comparison of many objects
Qualitative model	...by making subjective judgments against a checklist
Empirical quantitative model	...by counting or measuring something
Analysis	Given the facts, these consequences...
Analytic formal model	...are rigorous, usually symbolic, in the form of derivation and proof
Empirical predictive model	...are predicted by the model in a controlled situation (usually with a statistical analysis)
Experience	I evaluate these results based on my experience and observations about the use of the result in actual practice and report my conclusions in the form of
Quantitative or descriptive model	...prose narrative
Decision criteria	...comparison of systems in actual use
Empirical predictive model	...data on use in practice, usually with statistical analysis

1.7 Overview of Remaining Chapters

The rest of this dissertation is organized as follows.

Chapter 2 provides a survey of the related work in security and software engineering, to position the work presented in this dissertation.

Chapter 3 presents two formative case studies that served as the conceptual origin for the Requirements-based Access Control Analysis and Policy Specification (ReCAPS) method. The discussion in this chapter justifies the heuristics presented in Chapter 4.

Chapter 4 details the ReCAPS method, focusing on the activities software and security engineers (analysts) perform when employing the method. Guidelines and heuristics are also presented to guide software and security engineers through the access control analysis. Examples from two formative case studies are provided to elucidate the heuristics. A software tool—the Security and Privacy Requirements Analysis Tool (SPRAT)—that supports ReCAPS is presented in this chapter.

Chapter 5 discusses the validation efforts for the method presented in this dissertation: (1) two summative case studies involving the specification of access control policies for a web-based e-commerce system and an event registration system for the College of Management at North Carolina State University, and (2) an empirical evaluation in which the method was applied to a small system by individuals who were previously not familiar with the ReCAPS method.

Chapter 6 summarizes the contributions of the dissertation and future work that is needed to further refine the method.

Chapter 2

Background and Related Work

知己知彼者，百战不殆。— 孙武

One who knows the enemy and knows himself will not be in danger in a hundred battles.

—Sun Tzu

The work in this dissertation is interdisciplinary between software engineering (specifically requirements engineering and software design) and information security. Researchers in the security community and software engineering community have investigated access control and policy specification from different angles. To position the work in this dissertation, some of the most relevant previous work in both areas is briefly surveyed.

This chapter is structured as follows. Section 2.1 provides a brief summary of requirements engineering (RE), goal-based, and scenario-based requirements analysis techniques. Section 2.2 surveys security requirements engineering and access control analysis in RE. Section 2.3 provides some background information on access control and access control policies. Section 2.4 surveys the current efforts in security and privacy policy analysis and specification. Section 2.5 concludes this chapter.

2.1 Requirements Engineering (RE)

The process presented in this dissertation is an RE activity that attempts to bridge the gap between requirements and design. Requirements engineering has developed into a relatively mature branch in software engineering over the last 10-15 years. This is evidenced by a large annual international RE conference since 1993, an RE journal founded in 1996, several regional

small workshops (e.g., REFSQ since 1994, AWRE since 1996), and an IFIP Working Group founded in 1993 (IFIP WG 2.9: Software Requirements Engineering). An exhaustive survey of this area is outside the scope of this dissertation. Readers can refer to [NE00, Zav97] for a comprehensive summary of the RE literature. This section briefly explains what requirements engineering is and then surveys previous work on goal-based and scenario-based requirements analysis techniques, which are employed in this dissertation. Finally, this section summarizes previous work on bridging the gap between requirements analysis and software design.

2.1.1 Classification of Requirements Engineering Research Efforts

Requirements analysis and specification is the first phase of the software development process. Requirements engineering (RE) is the process of discovering real-world goals for which a software system is intended by identifying stakeholders and their needs, identifying the functions of and constraints on the intended system, and documenting these in a form that is amenable to analysis, communication, and subsequent implementation [NE00].

Zave classifies RE research efforts according to problems and contributions to solutions [Zav97]. To position the work in this dissertation, this section provides another way to classify RE research efforts from six dimensions, as shown in Figure 2.1.

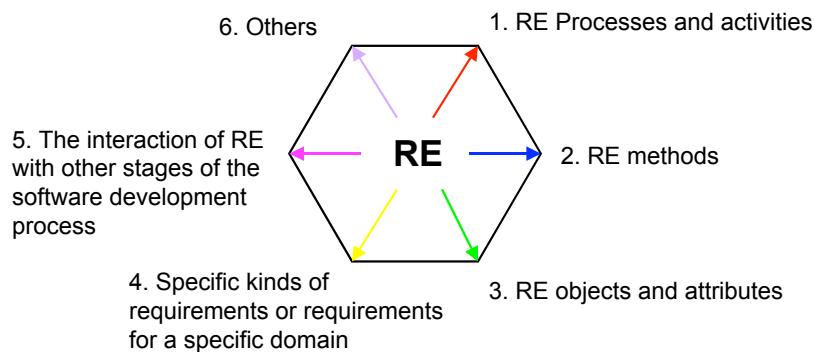


Figure 2.1 A Six-Dimensional View of RE Research Efforts

Research efforts in the **first dimension** focus on the RE process and specific activities. Example RE activities include requirements elicitation, analysis, modeling, elaboration, operationalization, specification, negotiation, prototyping, prioritization, reasoning, and validation. These activities can be roughly classified as either early-, middle-, or late-phase RE activities according to the general sequence in which they occur during the requirements analysis process. Early-phase RE activities focus on understanding the problem domain and eliciting stakeholders' goals. Late-phase RE activities focus on creating good software architecture and design from requirements specifications. Other activities are scattered in the middle of this process, such as modeling, specification, prioritization. As previously mentioned, *the work in this dissertation is a late-phase RE activity, which derives ACPs from requirements specifications and aims to bridge the gap between requirements and design.*

Research efforts in the **second dimension** focus on a specific RE method; for example, goal-based techniques, scenario-based techniques, goal-scenario coupling, use cases, conceptual modeling, formal specification languages, and formal methods. *The work presented in this dissertation employs goal-based, scenario-based and use case-based methods during the analysis process.*

Research efforts in the **third dimension** focus on a specific RE object or attribute. For example, RE objects include functional behavior, intention and viewpoints of stakeholders, and organizations. RE attributes include requirements changes, conflicts and inconsistencies, traceability, evolution, volatility, reusability, social, cultural and cognitive factors of RE. *The work in this dissertation involves traceability.*

Research efforts in the **fourth dimension** focus on a specific kind of requirements, such as non-functional requirements, security and privacy requirements, or requirements for a specific domain (e.g., safety-critical systems, web design, healthcare). *The work in this dissertation involves security and privacy requirements.*

Research efforts in the **fifth dimension** focus on the interaction between RE and other stages of the software development process, such as RE in agile development, requirements and architecture, requirements and software design, requirements and testing, requirements and risk analysis. *The work in this dissertation bridges the gap between requirements and design.*

All other RE research efforts fall in the **sixth dimension**, such as RE and business process engineering, RE and strategic planning, tool support for RE, RE in practice, RE education, and distributed RE. *The work in this dissertation includes a software tool that supports our ACP analysis methodology and techniques.*

2.1.2 Goal-Based Requirements Analysis

Goals are targets for achievements [Ant97]. They are high-level objectives of a business, an organization or a system. In requirements engineering, goals are used as higher-level expressions of a system's requirements. They provide rationales and motives of the proposed system. Goal-driven RE employs goals to elicit, specify, analyze, and validate requirements. Kavakli identifies seven major goal-oriented methods in RE [Kav02]. A complete overview of goal-driven RE techniques is beyond the scope of this dissertation; readers can refer to [Lam01, Kav02] for a more comprehensive summary. In this section, we summarize four goal-based requirements analysis approaches (GBRAM, KAOS, i^* , and NFR) because they are most relevant to security and privacy requirements analysis (see Section 2.2).

The **Goal-Based Requirements Analysis Method (GBRAM)** is an approach for identifying and refining goals into operational requirements [Ant97]. This approach is supported by a rich set of heuristics, guidelines, and recurring question types that help analysts perform the analysis. The approach is essentially a document analysis technique. Example source documents may include enterprise policies, and interview transcripts with stakeholders. Antón and Earp apply this approach to analyze privacy policy documents to derive privacy goal statements. Their findings

are summarized in Section 2.4.3. GBRAM also suggests using scenario analysis to elaborate goals and describe exceptional cases when a goal may fail or be blocked. Thus, analysts can design alternative solutions to solve these problems accordingly.

The **Knowledge Acquisition and Automated Specification (KAOS)** framework is a goal-based requirements acquisition and elaboration method [DLF93, LDM95, DL96, LDL98, LL00, LL02a, LL02b]. KAOS provides a formal and expressive conceptual modeling language, rich requirements elaboration strategies, and tool support to help requirements engineers specify requirements derived from high-level goals.

Three levels are defined in the KAOS framework: the meta level, the domain level and the instance level [LDM95]. The *meta* level defines four types of concepts:

- *meta-concepts* that are supported by the specification language (e.g., goal, entity, object, agent, action);
- *meta-relationships* that link meta-concepts (e.g., a goal is reduced to subgoals, a goal is operationalized into constraints, an agent is assigned constraints);
- *meta-attributes* that characterize meta-concepts and meta-relationships (e.g., the load of agent, the reliability of assignment); and
- *meta-constraints* that constrain meta-concepts and meta-relationships (e.g., a weak constraint must have a restoration action associated with it).

The *domain* level defines domain-specific instances of meta-concepts and meta-relationships (e.g., meeting is an instance of a meta-concept entity, scheduler is an instance of a meta-concept agent). The *instance* level defines specific instances of domain-level concepts (e.g., the ICSE-27 program committee meeting on February 22, 2005 is an instance of meeting defined at the domain level).

The main principal underlying the KAOS framework is that the more knowledge at the meta level, the more knowledge-based guidance can be provided to acquire requirements fragments at

the domain level. Thus, KAOS defines a rich set of meta-concepts and meta-relationships. Some of the most important meta-concepts/relationships are [LDM95, Fon01]:

- *Object*: a thing of interest in the composite system whose instances may evolve from state to state.
- *Action*: an input-output relation over objects. Action applications define state transitions.
- *Agent*: another type of object which acts as processor for some actions.
- *Goal*: is a non-operational objective that the composite system must meet.
- *Constraint*: an operational objective that can be formulated in terms of states controllable by some agent.
- *Scenario*: expresses a typical combination of actions expected to take place in the composite system.

The KAOS framework was employed to analyze security requirements and perform access control analysis, which are discussed in Section 2.2.1 and Section 2.2.2.

The *i framework** is an early-phase RE method used to model and reason about organizational contexts and rationales [Yu93, Yu97, CNY00]. It is comprised of two components: the Strategic Dependency (SD) model and the Strategic Rationale (SR) model. The SD model focuses on describing the external relationships (dependency) among organizational actors, hiding intentional constructs within each actor. The SR model focuses on actors' internal interests and concerns and how they might be addressed by various configurations of systems and environments.

The central concept of the *i** framework is *distributed intentionality*, which means actors are considered to have intentional properties such as goals, abilities, beliefs, and commitments. Actors depend on each other to achieve some goals, perform certain tasks, and furnish some resources. This dependency relationship has two effects. On one hand, by depending on others, an actor may achieve goals that are difficult or impossible to achieve on his own. On the other hand,

an actor becomes vulnerable if the depended-on do not deliver. Four types of dependencies are discussed in *i**: goal dependency, softgoal dependency, task dependency and resource dependency. Actors are also considered *strategic* in the sense that they are concerned about opportunities and vulnerabilities and seek alternatives to better serve their interest.

The *i** framework was employed to analyze security requirements and perform access control analysis, which are discussed in Section 2.2.1 and Section 2.2.2.

The **Non-Functional Requirement (NFR)** framework is a goal-based requirements analysis method that systematically addresses non-functional requirements in the early stages of system development [MCN92, Chu93, CNY00]. The *i** framework discussed previously shares many concepts with the NFR framework, such as softgoals and goal dependencies.

The nature of non-functional requirements can be summarized as subjective, relative and interacting [CNY00]. NFRs can be subjective in the sense that they can be viewed, interpreted and evaluated differently by different people. NFRs are relative since the interpretation and importance of NFRs may vary depending on the particular system being considered. NFRs are often interacting in the sense that attempts to achieve one NFR can hurt or help the achievement of other NFRs.

Due to the above natures of NFRs, the framework represents non-functional requirements as *softgoals*, which are usually *satisficed*, which means they are satisfied within acceptable limits instead of absolutely being accomplished.

The process of using the NFR framework to systematically deal with non-functional requirements starts by acquiring domain knowledge, i.e., the particular domain and the system being developed, system functional requirements and non-functional requirements. After identifying particular NFRs for the domain, the framework provides heuristics to decompose NFRs into more specific softgoals. Then the framework provides heuristics to operationalize softgoals, i.e., providing possible design alternatives or concrete mechanisms for satisficing NFRs in the target system. The next step is to record design rationales (which are also called *arguments*

that are supported by *claims*), select operationalizations and evaluate the impact of decisions. The NFR framework also provides methods to deal with ambiguities, tradeoffs and priorities, and interdependencies among NFRs and operationalizations. They are very important in making design decisions and selecting among alternatives.

Because security requirements are often considered as non-functional requirements, the NFR framework can also be used to analyze security requirements, which is discussed in Section 2.2.1.

2.1.3 Scenario-Based Requirements Analysis

A scenario is comprised of a sequence of events that describe possible ways for users to interact with a system [JBC98]. In requirements engineering, scenarios are widely used to describe the software behavior of a system [WPJ98]. Scenarios are useful for eliciting possible occurrences and the corresponding assumptions (pre-conditions), obstacles, and post-conditions [Als02].

In software engineering, there is a similar notion: use case. A use case includes a use case diagram, which visualizes the interactions of users with a system, and a use case description, which details the interaction using a sequence of events [KG03]. Use cases are widely used in the software development industry because they are included in the Unified Modeling Language (UML) [RJB99, UML05], which is supported by various CASE tools, such as IBM® Rational Rose®¹ and Microsoft® Visio®². According to Kulak and Guiney [KG03], the definition of scenarios and use cases is misaligned. There exist at least three definitions of scenarios: a scenario is either an alternative path, or an instance of a use case, or a synonym for use case. In this dissertation, we treat scenarios and use cases synonymously.

¹ Rational Rose® is a registered trademark of International Business Machines (IBM)® Corporation in the United States and/or other countries.

² Visio® is a registered trademark of Microsoft® Corporation in the United States and/or other countries.

This section focuses on goal-scenario combination approaches. Goals and scenarios have complementary characteristics [Lam01]. Goals are usually abstract and declarative. They are high-level objectives of the business, organization or system. Scenarios are concrete, narrative, and procedural. They describe real situations, using examples and illustrations. Hence combining the benefits of goals and scenarios is an effective way to elicit and validate requirements. Goals are operationalized through scenarios and refined into requirements [AMP94]. Similarly, scenarios can be used to help discover goals [AP98].

The GBRAM uses goal hierarchies to organize requirements as scenarios, goal obstacles, and constraints [Ant96]. Other analysis approaches also organize scenarios hierarchically according to goals and goal obstacles [Coc97]. Rolland et al. propose a bidirectional goal-scenario coupling approach between goal discovery and scenario authoring [RSA98]. Kaindl proposes a systematic design process based on a model combining scenarios with goals and functions [Kai00]. In the combined model, “purpose” serves as a link between functions and goals: a system’s aggregated functions have some purposes, and these purposes match the (sub)goals of the users. Purpose has also been integrated with scenarios to model tasks in one of Kaindl’s early works [Kai95].

2.1.4 From Requirements Analysis to Software Design

During the traditional waterfall software development model [Pre05], requirements analysis and software design are two separate phases. Even though researchers have argued we should relate software requirements with design [Pet78], in practice, requirements, design and code are often misaligned [CHO99]. Although object-oriented analysis and design methods [Boo94] bring requirements and design closer, the gap between requirements analysis and software design still exists.

Specifically, in the requirements engineering community, researchers have never looked at database design. For one reason, when requirements analysis is performed, database design is not there yet. In other words, database design cannot be started until requirements analysis is completed. For another reason, requirements researchers and engineers think the specification of requirements should not be biased by design or implementation. This results in a common understanding that requirements engineers should not look at database design and it is software designers' responsibility to ensure that the design is in compliance with the requirements specifications.

In this dissertation, we propose a different argument from the above view within the context of access control policy specification. ACP specification is a late-phase RE activity. To specify access control policies, software engineers are required to examine not only requirements, but also database designs. Requirements are needed because they provide information about actors and describe their interaction with the system, and the constraints for information accesses. Database designs are needed because they provide information about what objects must be protected via access control. Additionally, to bring requirements and design into better alignment, requirements engineers should examine database design.

In this section, we summarized the general RE literature. The next section will narrow the scope and focus on security and privacy requirements.

2.2 Security Requirements Engineering

Security and privacy requirements are often considered as non-functional requirements [CNY00]. Moffett et al. define security requirements to be constraints on functional requirements that are needed to achieve security goals [MHN04]. They propose a set of core security requirements *artefacts* that form a three-level (goals-requirements-architecture) hierarchical diagram, in which *requirements* are elicited from and operationalize *goals* and system

architecture implements requirements. . This section first summarizes relevant work on security and privacy requirements analysis, then focuses on one type of security requirements: access control requirements.

2.2.1 Security and Privacy Requirements Analysis

Lamsweerde employs KAOS [DLF93] to elaborate security requirements by constructing intentional anti-models [Lam04]. This approach generates malicious obstacles set up by attackers to threaten security goals, and then provides alternative resolutions to counteract these obstacles. The idea is similar to other approaches that capture security requirements through misuse cases [Ale03, SO00, HMA04]. Misuse cases are use cases from the point of view of an actor with hostile intent. Security requirements are specified to protect assets in a system from malicious attacks. Thinking from an attacker's standpoint helps elicit security requirements and provide countermeasure resolutions. In this dissertation, misuse cases are also employed to specify implicit conditions for access control policies.

The *i** framework was initially developed to provide support in modeling, analyzing and redesigning organizations and business processes [CNY00], but it has recently been used to model trust [YL01] as well as security and privacy requirements [LYM02, LYM03, YC02, YC03], such as attack analysis, dependency vulnerability analysis, countermeasure analysis, and viewpoint analysis of actors on privacy. Additionally, the *i** framework is used to reconcile security and privacy requirements with other non-functional requirements by reasoning possible alternatives.

Security requirements are non-functional requirements that can be analyzed using the NFR framework [CNY00]. Basically, security requirements address confidentiality, integrity and availability. These requirements are operationalized into alternative security mechanisms (e.g., password authentication, encryption) and functional requirements to achieve the specific

softgoals (e.g., confidentiality, accountability). Alternatives are evaluated according to design rationales and goal dependencies with functional requirements. The objective of NFR is to provide a systematic method to analyze security requirements and make a variety of alternative security methods and their tradeoffs available to system stakeholders. By evaluating the design decisions, the framework may help provide a system design that can best achieve security requirements (and other non-functional requirements).

He and Antón proposed modeling privacy requirements in role engineering [HA03]. This approach is comprised of a context-based data model and a structured role engineering process. Privacy requirements are modeled as contexts, conditions and obligations in access control policies for enforcement.

2.2.2 Access Control Analysis in Requirements Engineering

Access control requirements as a special type of security requirements have received insufficient attention until recently. RE researchers are investigating methods and tools for analyzing and specifying access-related security requirements. Fontaine [Fon01] employs KAOS, a goal-based requirements acquisition and elaboration method [DLF93], to refine security requirements into specific authorization rules and ACPs expressed in Ponder—a language for specifying management and security policies for distributed systems [Dam02]. Fontaine’s work is an important step towards requirements-level access control analysis for security policy specification. However, the method for mapping KAOS specifications to Ponder policies is not comprehensive in that not all types of Ponder policies can be generated from KAOS specifications. Fontaine has thus far only shown how to specify authorization and obligation policies [Fon01], not refrain and delegation policies. Access control policies not only come from requirements, but also from security and privacy policies. Fontaine’s approach cannot specify ACPs from security and privacy policies, whereas our approach (ReCAPS) can. Additionally,

Fontaine’s approach does not focus on ensuring compliance between ACPs, requirements and design, whereas our approach considers compliance among these artifacts one of the most important design principles, as discussed in Chapter 3.

Liu et al. apply the i^* framework [Yu93], a goal-based requirements analysis method, to support access control analysis by modeling the dependencies among actors, tasks and a system’s resources [LYM03]. However this approach is limited in that it assumes the roles and privileges have been previously derived. It provides no guidance as to how roles and privileges are identified, from where they originate, or how privileges are assigned to these roles. Moreover, it is difficult to model context and constraint information in the i^* framework. These topics remain major challenges in access control analysis during RE.

Generally speaking, the i^* framework is an early-phase RE approach, and its ability lies in modeling dependencies among actors and rationales within actors. Access control analysis is a late-phase RE activity that involves many activities such as organizational structure analysis, organizational security/privacy policy analysis, and detailed business process analysis (e.g. scenario analysis). ACP specification is intertwined with software design (e.g., DB design), which is not the strength of i^* .

Crook et al. propose an analytical role modeling framework to model access control policies [CIN03]. The framework is specifically designed for role-based access control (RBAC) [SCF96] systems and derives roles from organizational structures. Although other researchers have employed RE methods, such as scenarios [NS02] and use cases [FH97], to define needed permissions for roles, this framework was the first to explicitly clarify the importance of providing requirements-level support for modeling access control policies.

Three types of roles may be defined according to organizational structures: roles based on seniority, roles based on function, and roles based on market. In any organization, supervision is a key coordination mechanism. Roles based on seniority reflect the hierarchical lines of supervision. Roles based on function model the capability of an individual or tasks that logically

belong together. Roles based on market models the context of assets being accessed (i.e., on which targets a user may carry out the tasks).

Based on the above organizational structures, the framework defines three types of roles: seniority, functional and contextual roles. The framework is comprised of two-level representations. The meta level defines the role types, asset categories, context types and operations. The instance level defines users, role instances, asset instances, context instances and operation request.

This approach offers two contributions. First, the framework clarifies the need to model ACPs during requirements analysis. Second, the rationale for deriving roles based on organizational structures is very useful. Job positions in an organization can be mapped to *roles* in RBAC. Organizational and seniority hierarchies can be mapped to RBAC *role hierarchies*. This is common in identity management products in which individuals in a particular department and/or division are classified into a specific role, and that role grants them the access rights to specific (e.g., bank) accounts. Deriving roles from organizational structures facilitates the user assignment and authorization management processes in access control. However, this approach is specific for defining roles in RBAC systems, whereas our approach is a general ACP specification approach.

Moffett et al. discuss the relationship between access control policies and requirements engineering [MHN04]. They broadly classify ACPs into three categories: global ACPs that are built into the a system, discretionary ACPs that are basically security requirements, and mandatory ACPs that are essentially mechanisms and should not concern requirements engineering. Our approach is consistent with this view. We specify access control policies from requirements and high-level security and privacy policies and improve these source documents as a result of the analysis.

Brose et al. propose to integrate access control design into the software development process by extending UML to specify access control policies for distributed object systems [BKL02]. We

concur with them that access control policy specification should be an integral part of the software development process. However, this approach does not emphasize the compliance between different levels of policies, requirements and system design; whereas in our approach, ensuring compliance is a prominent design principle.

After surveying access control analysis and policy specification from the software engineering—specifically the RE perspective—the next two sections discuss access control from the security perspective.

2.3 Access Control in Security

It is important to understand what security is before we discuss access control. A computer or a system is *secure* if you can depend on it and its software to behave as you expect. Note the differences between *security* and *trust*. According to Pfleeger [PP02], the word “secure” is binary: something either is or is not secure. However, there can be degrees of trust. A person may totally trust that a system can protect his/her personal data from being disclosed or damaged; whereas he/she may only trust that another system can prevent personal information from being damaged, even though the system may disclose personal information to other parties. *Secure* is a property of the provider. It is a goal. *Trusted* is a property of receiver. It is a characteristic. A *secure* system is asserted as such based on product characteristics; whereas a *trusted* system is judged as such based on evidence and analysis by the receiver. A *secure* system is an absolute. It is not qualified as how, where, when, or by whom the system is used. A *trusted* system is relative; that is, it is viewed in context of use.

There are three common security goals: confidentiality, integrity and availability [PP02]. *Confidentiality* means that information is not disclosed to unauthorized persons, processes or devices. *Integrity* means that unauthorized persons, processes or devices cannot modify information. *Availability* means that an authorized person, process or device should not be

prevented from accessing the information to which he, she, or it has legitimate access. In addition to the above three security goals, privacy is often considered as a separate goal that overlaps with confidentiality but has additional meaning (see Section 2.4.3). *Privacy* means that data is protected so that it is used only for authorized business purposes, based on legal requirements, corporate policies and end-user choices.

2.3.1 Access Control

The function of access control (AC) is to ensure that every access to a system and its resources is controlled according to a set of predefined policies [SV01]. It is one of the major security mechanisms used to achieve confidentiality, integrity and privacy in software systems [FKC03]. An access control system is typically described in three ways: access control policies, access control models and access control mechanisms [SV01]. *Access control policies* are security requirements that describe how access is managed, what information can be accessed by whom, and under what conditions that information can be accessed [FKC03]. These policies are enforced via a mechanism that mediates access requests and makes grant/deny decisions. *Access control models* provide a formal representation of an access control system. They provide ways to reason about the policies they support and prove the security properties of the access control system. Access control models provide a level of abstraction between policies and mechanisms, enabling the design of implementation mechanisms to enforce multiple policies in various computing environments. The *access control mechanism* defines the low-level functions that implement the controls imposed by the policies. It must work as a reference monitor [SV01], a trusted component intercepting each and every request to the system.

2.3.2 Access Control Policies

Access control policies can be broadly grouped into three main policy categories: Discretionary Access Control (DAC), Mandatory Access Control (MAC), and Role-Based Access Control (RBAC). DAC policies enforce access control based on the identity of the requestor and the explicit rules specifying who can or cannot perform specific actions on specific objects. Early discretionary access control models, such as the access control matrix model [Lam74, GF94] and the HRU model [HRU76], provide a basic framework for describing DAC policies. It is the users' discretion to pass their privileges on to other users. Thus, DAC policies are vulnerable to Trojan Horse attacks [SV01].

MAC policies enforce access control based on the security classifications of subjects and objects. For example, the lattice-based multilevel security policy [Den76], policies represented by the Bell-LaPadula model [BL73, BL76] and the Biba model [Bib77] are MAC policies. MAC policies protect indirect information leakages (e.g., Trojan Horse attacks), but are still vulnerable to covert channel attacks [SV01, PP02].

RBAC policies employ roles to simplify authorization management for enforcing enterprise-specific security policies [FKC03, SCF96]. The RBAC model is an alternative to traditional DAC and MAC models and has received increased attention in commercial applications, such as the Oracle 9i DBMS [CS98]. RBAC is now an American National Standard³.

2.3.3 Elements of Access Control Policies

An access control *policy* is comprised of a set of access control rules. A *rule* can have various modes (e.g., allow/deny/oblige/refrain). The work in this dissertation focuses on allow and deny rules. *Allow* rules authorize a subject to access a particular object. *Deny* rules explicitly

³ American National Standard: ANSI INCITS 359-2004.

prohibit a subject from accessing a particular object. When a subject requests to perform an action on an object, the corresponding rules are evaluated by the enforcement engine for that request. A typical access control *rule* is expressed as a 5-tuple $\langle \textit{subject}, \textit{object}, \textit{action}, \textit{condition}, \textit{obligation} \rangle$, such that a *subject* can perform some *action* on an *object* [DD82]. A *subject* is an entity, such as a user or program agent that may access objects. An *object* is an entity, such as a data field, table, procedure or application to which access is restricted. An *action* is a simple operation (e.g. read or write) or an abstract operation (e.g. deposit or withdraw).

An ACP may express additional *conditions* that must be satisfied before an access request can be granted. For example, in healthcare applications, the location from which the access request originates might affect the grant/deny decision [Bez98]. If an access request is from the emergency room, then the request may be granted. In this case, we can specify *the location of the request is emergency room* as a condition for the AC rule. Additionally, in the context of privacy protection, we may specify additional conditions to restrict access to personal data. For example, *purpose* is a standard entity in most privacy policies, as recognized in P3P (The Platform for Privacy Preferences Project) [P3P05]. When a subject (e.g., a nurse) requests to perform an action on an object (in the context of privacy, the object is often personal data such as medical records), the purpose of the operation should be bound to the purposes consented to by the data subjects. This is the purpose binding principle [Fis01], which can be enforced by specifying conditions for ACPs. Karjoth et al. treat purpose the same as subjects, objects, and actions. for a privacy authorization rule because purpose tends to be the primary focus of privacy protection [KS02]. This dissertation does not treat purpose as an independent element because purpose is mainly useful for protecting data privacy.

Obligations [BJW02] are actions that must be fulfilled if a request to access an object is granted. For example, consider *require affiliates to destroy customer data after service is completed*. In this case, “destroy customer data” is an obligation that must be satisfied by affiliates. Obligation-based security policies can be enforced if they can be completely resolved

within an atomic execution [RZF01]. If the obligation is not an immediate action (e.g., it is a task that will be executed in the future), monitoring and auditing its execution might be sufficient for enforcement [BJW02].

In requirements specification, we are concerned with the *actions* for which each actor (*subject*) is responsible, the conditions under which each action can occur (constraints and pre-conditions) and the post-conditions (*obligations*) that must be satisfied. Each of the five access control elements can be mapped to a requirements specification element. This mapping suggests it is possible to derive ACPs from requirements to ensure that ACPs comply with the requirements.

2.3.4 Current Research Efforts on Access Control Policy Specifications

Instead of considering ACP specification from a holistic, real-systems perspective as advocated in this dissertation, current ACP specification research has a much narrower focus, such as policy specification languages, uniform or flexible ways to specify ACPs [JSS01], specifying ACPs for XML documents [FM04], and specifying a specific aspect of access control (e.g., delegation, temporal constraints). There are few reported methods and experiences relating ACP specification in real software systems.

Researchers have been trying to develop more comprehensive and expressive languages, or languages for specific purposes (e.g., for specifying privacy policies) for specifying access control policies. A more detailed survey of policy specification languages is given in Section 2.4.1.

2.3.5 Role Engineering

In the RBAC literature⁴, researchers are investigating role engineering, the process of defining roles, permissions, role hierarchies, constraints and assigning the permissions to the roles [Coy96]. Role engineering is the first step to implementing an RBAC system and essentially a requirements engineering process. Before a software system can realize all the benefits of RBAC, the role engineering activities must occur, yielding a complete specification.

There exist several role engineering approaches, the first of which applies scenarios. Neumann and Strembeck propose a scenario-driven approach for engineering functional roles in RBAC [NS02]. In this approach, each task is depicted using a collection of scenarios, and each scenario is decomposed into a set of steps. Because each step is associated with a particular access operation, each scenario is linked to a set of permissions. The work is limited in that it is only effective to derive functional roles, not seniority or contextual roles [CIN03].

Fernandez and Hawkins suggest determining the needed rights for roles from use cases [FH97]. Epstein proposes a layered model for engineering role-permission assignment by introducing three intermediaries between roles and permissions: jobs, workpatterns, and tasks [Eps02, ES01]. Epstein's approach provides an effective way to assign permissions to roles and aggregate permissions into roles. Roeckle et al. propose a process-oriented approach for role finding to implement role-based security administration [RSW00]. Their approach provides a method to find roles but does not address how to find permissions and how to assign permissions to roles. Unfortunately, neither of these approaches [Eps02, ES01, FH97, RSW00] considers constraints and role hierarchies.

Epstein and Sandhu's UML-based approach documents components of an RBAC model in UML syntax [ES99]. This approach can assist the role engineering process, but it does not

⁴ See ACM RBAC and SACMAT workshop series, <http://www.sacmat.org/>.

provide a method for deriving roles. Kern et al. propose an iterative-incremental life-cycle model of a role in the context of enterprise security management [KKS02]. The role life-cycle concept is very important for security administration; however, this approach fails to support the derivation of roles and permissions. Schimpf argues that role engineering is a critical success factor for enterprise security administration [Sch00]. He proposes organizing a role engineering project and following a clearly defined life-cycle model for roles.

Role engineering is specific to RBAC, whereas the ACP specification approach in this dissertation is a more general approach. Additionally, none of the surveyed approaches specifies privacy constraints in access control policies. In contrast, the ACP specification approach in this dissertation also analyzes privacy requirements and specifies them as conditions in the resulting ACPs.

2.4 Security and Privacy Policy Analysis and Specification

In the operating systems and security communities, researchers have investigated security policy analysis and specification. Recently, as privacy has become of increasing concern to consumers, software developers and legislators, interest in privacy policy analysis has grown. This section first summarizes several policy specification languages, and then briefly surveys previous work on security and privacy policy analysis and specification.

2.4.1 Policy Specification Languages

Although policy specification languages are not the focus of this work, the policies do need to be expressed using some language. We have examined three types of languages: access control policy specification languages, privacy policy specification languages, and formal specification languages. *Access control policy specification languages* include Ponder [Dam02], XACML

[OAS05], Authorization Specification Language (ASL) [JSS97, JSS01], and a variety of role modeling languages [AS99, BBF00, HBM98]. *Privacy policy specification languages* include P3P [P3P05] and EPAL [AHK03]. *Formal specification languages* include Alloy [Jac02], Formal Tropos [FKP03], KAOS [DLF93], Larch [GH93], Object Constraint Language (OCL) [WK99] of UML [RJB99, UML05], and Z [Spi92]. Our analysis concludes that Ponder is the most suitable language for specifying access control policies, whereas Alloy is the most suitable formal notation for automatically verifying security properties as we now discuss.

Ponder is a declarative, object-oriented language for specifying management and security policies for distributed systems [Dam02]. Four types of policies are defined in Ponder: authorization policies, obligation policies, refrain policies and delegation policies. Ponder also provides mechanisms for specifying groups, roles, relationships and management structures. The Ponder language is sufficiently expressive to specify most access control policies. Additionally, its declarative feature is easy to use and allows Ponder policies to be easily translated into implementation. In this dissertation, we express the policies derived from various sources in a form similar to Ponder except that policies in this dissertation are traceable to the sources from which the policies are derived.

Alloy is a lightweight structural object modeling language based on first-order logic. It supports not only object models, but also operation and behavior models. Alloy is based on Z [Spi92]. A comparison of Alloy with Z shows that Alloy has the benefit of simpler semantics, automatic semantic analysis and tool support, and it is compatible with object modeling idioms. Although Alloy is less expressive than OCL, it is based on a language that was tested and used in large-scale industrial developments. OCL has its own limitations: its semantics are complicated; it does not have transitive closure; its syntax has been criticized [DW98]; it is not currently in wide use and there are few publicly available OCL model examples; and it is very difficult to formalize OCL.

Alloy provides tool support, which can generate instances of invariants, simulate the execution of operations, and check user-specified properties of a model. Alloy has been used in a variety of case studies, one of which is using Alloy to check logical consistency among the specifications of RBAC entities, relations and constraints [ZWC02]. Lin et al. also employ Alloy to verify the security properties of a requirements model [LYM03].

In Chapter 4, we will use P3P elements (i.e., purpose, recipient) to specify conditions and obligations for access control rules. As a special section, we present P3P in more detail herein for future references.

P3P is an XML-based policy specification language that can be used to specify an organization's privacy practices in a way that can be parsed and used by policy-checking agents on the user's behalf. P3P predefines a set of values for its elements; for example, Table 2.1 shows the 12 purposes that are defined in P3P1.1, which specify the purpose for which the data is collected or used.

Table 2.1 Purposes Defined in P3P1.1 [P3P05]

Purpose Name	Description
current	Completion and Support of Activity For Which Data Was Provided
admin	Web Site and System Administration
develop	Research and Development
tailoring	One-time Tailoring
pseudo-analysis	Pseudonymous Analysis
pseudo-decision	Pseudonymous Decision
individual-analysis	Individual Analysis
individual-decision	Individual Decision
contact	Contacting Visitors for Marketing of Services or Products
historical	Historical Preservation
telemarketing	Telephone Marketing
other-purpose	Other Uses

Additionally, 23 primary purposes are defined in P3P1.1 to provide a more detailed description of data usage under purpose <current> in Table 2.1 and reason the recipient is collecting data, as shown in Table 2.2.

Table 2.2 Primary Purposes Defined in P3P1.1 [P3P05]

Purpose Name	Description
account	Account and/or Subscription Management
arts	Arts and Entertainment
browsing	Web Browsing
charity	Charitable Donations
communicate	Communications Services
custom	Customization
delivery	Delivery
downloads	Software Downloads
education	Education
feedback	Responding to User
finmgt	Banking and Financial Management
gambling	Online Gambling
gaming	Online Gaming
government	Government Services
health	Healthcare Services
login	Authentication and Authorization
marketing	Advertising, Marketing, and/or Promotion
news	News and Information
payment	Payment and Transaction Facilitation
sales	Sales of Products or Services
search	Search Engines
state	State and Session Management
surveys	Surveys and Questionnaires

P3P defines the following recipients (see Table 2.3), which specify who will receive the collected data.

Table 2.3 Recipients Defined in P3P1.1 [P3P05]

Recipient Name	Description
ours	Ourselves and/or entities acting as our agents or entities for whom we are acting as an agent
delivery	Delivery services possibly following different practices
same	Legal entities following our practices
other-recipient	Legal entities following different practices
unrelated	Unrelated third parties
public	Public fora

2.4.2 Security Policy Analysis

Security policy can be explained in two ways, depending on the context environment. For a system, a security policy is a statement of the security we expect the system to enforce. Some well-known security policies are military security policy [BL73, BL76], Clark-Wilson commercial security policy [CW87], and Chinese Wall security policy [BN89]. These policies may be found in various environments.

For an organization, a security policy is a document that sets out an organization's security-related procedures, rules and principles [WC98]. A security policy defines procedures of how internal and external users interact with an organization's network and how the overall computer architecture topology will be implemented; lists an organization's assets and where these assets will be located; details the security goals, as well as any potential threats and vulnerabilities; assesses security risks; describes how to address security breaches; and clarifies who is the designated security coordinator and the responsibilities for each team member and each

employee. A security policy must be comprehensive, well-thought-out, testable and adaptable. A security policy should be constantly tested and updated as the organization's needs change, as technology changes, and as security issues (e.g. security threats and vulnerabilities) change [WC98].

There are various reasons to establish a security policy for an organization. The basic motivation is the need to protect an organization's assets. Thoroughly planned security policies help minimize break-ins. Also, legal requirements force industry to define certain aspects of an organization's activities and take actions to protect customer data security and privacy. For example, in the healthcare industry, confidentiality of patient records must be maintained. Other reasons to establish a security policy include contractual requirements and regulatory requirements. In short, good security starts with a security policy.

Organizational security policies are source documents for deriving access control policies for the method presented in this dissertation. In the later chapters, when we refer to "high-level security policies", we are referring to the organizational security policies discussed above. How to develop a security policy for an organization is beyond the scope of this dissertation. Readers who are interested in this topic may refer to [SSH97, WC98]. This section focuses on security policy modeling, specification, and consistency checking within the context of a particular system's security policy.

An important element in the development of a secure system is the production of a formal security policy model. A formal security policy model is a "mathematically precise statement of a security policy" according to the National Computer Security Center (NCSC) glossary of computer security terms [NCS88]. The NCSC also defines a general process and some techniques for security policy modeling in its *Guide to Understanding Security Modeling in Trusted Systems* [NCS92].

Dobson and McDermid [DM89] present a modeling language for expressing security policies, in which roles, agents, actions, and data are basic components of a security policy. This

is an early approach for investigating security policy specification languages in which no formal syntax or semantics were defined for a security policy. Dobson and Martin [DM92] distinguish organizational security policies from automated security policies: organizational security policies are the set of laws, rules, regulations, and practices that regulate how an organization protects its resources, whereas automated security policies are the set of restrictions and properties that specify how a computing system prevents resources from being used to violate the organizational security policies. This view is consistent with the view presented in this dissertation as discussed in Chapter 1 (see Figure 1.2). Organizational security policies are the source documents for ReCAPS, and automated security policies (in this dissertation, they are access control policies) are the products of ReCAPS. Dobson and Martin show how to represent organizational security policies using enterprise modeling, and how restrictions and properties of the automated security policy play a key role in the enforcement of the organizational policy [DM92].

Freeman and Neely [FN93] stress the importance of security policy modeling during the requirements analysis and architectural design phases of system development for high assurance software systems. They observe that it is important to clearly define which part of security policies should be formally specified and that the policy modeling team should focus its efforts on what it really needs to model. This dissertation supports the perspective advocated by Freeman and Neely—that it is important to specify access control policies during the requirements analysis and system design phases of software development.

Freeman, Neely and Heckard [FNH94] propose a security policy modeling approach named Boundary Flow Modeling (BFM). Similar to ReCAPS, which derives ACPs from requirements, BFM is driven by security requirements and system architecture, rather than forcing an existing generic security model on the intended system. The BFM approach consists of six steps: (1) identify the policy statements to be modeled, (2) interpret the policy statements, (3) produce model structure with constraints, (4) refine constraints for the system boundary, (5) refine constraints for the system interior, and (6) perform an internal verification of the model.

However, the guidance provided by this approach is not prescriptive; for example, it is unclear how analysts can identify and interpret policy statements when using BFM. Additionally, it is unclear how the method relates the policies to system design.

Researchers have explored logic-based approaches for specifying security policies. Cuppens and Saurel [CS96] propose a logic-based approach, which combines deontic logic to model the concept of permission, obligation and prohibition with a modal logic of action. Additional concepts such as role, responsibility and delegation are also considered. An advantage of a logic-based policy specification approach is that it is relatively easier to verify the security properties of a set of policies, compared with the declarative, semi-structured policy specification approach used in this dissertation. For example, using Standard Deontic Logic (SDL) to provide a precise and non-ambiguous specification of a security policy, Cholvy and Cuppens [CC97] propose a methodology for checking the security policy consistency and querying a policy to discover the actual norms (i.e. permissions, obligations and prohibitions) that apply to a given situation. These logic-based approaches are complementary to the approach presented in this dissertation for specifying access control policies. They are more useful for verification.

The above surveyed approaches focus on security policy modeling, specification, and verification. The work presented in this dissertation focuses on ACP specification. Although access control policy is a kind of security policy, none of the approaches surveyed above provides *procedural* support for specifying access control policies for a software system.

2.4.3 Privacy Policy Analysis

Two major privacy protection principles are the OECD guidelines for data protection [OEC80] and the FTC Fair Information Practice (FIP) Principles [FIP98]. The OECD guidelines define eight privacy principles: collection limitation, data quality, purpose specification, use limitation, security safeguards, openness, individual participation, and accountability. The OECD

principles intend to protect personal data privacy while pursuing free information flow between different organizations and different countries. The five FIP principles (notice/awareness, choice/consent, security/integrity, access/participation, and enforcement/redress) are recommended by the U.S. Federal Trade Commission (FTC) to protect consumer privacy. Both the OECD and FIP principles provide the general privacy requirements with which organizations should comply. In the U.S., several industries have additional legislative acts (e.g. HIPAA and GLBA) regulating their data practices.

Based on these general privacy principles and acts, each organization defines its own privacy policies. These policies are the major privacy requirements that an organization should enforce in their data processing systems. For example, when websites collect information from customers, they need to inform customers for what purpose the data is being collected, who the data recipient is, how long the data will be kept, and how the data will be used. (This is the notice/awareness principle in the FIP principles). The website should also provide opt-in/opt-out choices for customers or obtain customer consent on how to use the collected data (choice/consent principle). The actual data operations of companies and organizations should be consistent with user consented privacy policies (enforcement/redress principle). In this dissertation, organizational privacy policies are source documents for the ReCAPS method to specify access control policies for an organization's information systems.

In addition to protecting the confidentiality and integrity of personal data, Fischer-Hübner summarizes two other privacy protection principles: the purpose binding principle (i.e., data collected for one purpose should not be used for another purpose) and the principle of necessity (i.e., the collection and processing of data shall only be allowed if it is necessary for completing appropriate tasks) [Fis01]. In this dissertation, purposes are specified in the condition part of an access control rule to enforce the purpose binding principle.

Antón et al. have presented a body of work on privacy policy analysis from the requirements engineering perspective [AE01, AE04, AEB04, AEC03, AEP01, AEV05, AHB04]. For example,

Antón and Earp propose strategies to employ scenario management and goal-driven requirements analysis methods for specifying security and privacy policy for secure electronic commerce systems [AE01]. Antón et al. found software requirements and security and privacy policies are often misaligned [AEC03]. This observation further motivates the work in this dissertation as discussed in Chapter 1. Antón et al. apply goal-based requirements analysis to better align software requirements with security and privacy policies [AEC03]. A privacy requirements taxonomy for websites has been presented in [AE04]; in this approach goal-mining techniques are employed to analyze online privacy policies. In this taxonomy, privacy requirements are classified as either privacy protection goals or privacy vulnerabilities. This taxonomy was further used to analyze privacy policies for several domains, including finance [AEB04], healthcare [AEV05], and e-commerce [AHB04]. This body of work forms the basis for the work presented in this dissertation. However, Antón’s work primarily focuses on the level of natural language policies and requirements, whereas the work presented in this dissertation focuses on mid-level level policies (see Figure 1.3 in Chapter 1) and policy analysis and specification in ReCAPS is at the design level. Additionally, Antón’s work is limited to privacy policy, whereas the policy investigated in this dissertation includes both security policy and privacy policy.

2.5 Summary

This section summarized relevant work in requirements engineering and access control. We positioned the work presented in this dissertation in the context of literature. In previous work nobody has related high-level policies, system requirements, access control policies, and database designs. In this dissertation, ensuring compliance across these software artifacts is an important design principle. The approach presented in this dissertation helps bridge the gap between requirements analysis and software design with respect to security and privacy. The next chapter

presents the two formative case studies, which serve as the origin of ideas, methodology and techniques presented in this dissertation.

Chapter 3

Formative Case Studies

实践出真知。

Genuine knowledge comes from practice.

—Chinese Proverb

As mentioned in Section 1.6, the research methodology of this work is the process of conceptualization, empirical exploration, refinement and testing. This is an engineering approach to scientific research, which typically involves studying a problem, proposing solutions, and testing the solutions on real problems. This is in contrast to mathematical approaches to scientific research, which derive research from constructing concepts, often in the form of formal proofs, reflexive induction and reasoning. The process and heuristics presented in this dissertation were developed and evaluated while performing access control analyses on real systems and real policies, not constructed illustrations. This contrasts with other software engineering methodology development approaches, in which methods are developed and later tested on conceptualizations formed in isolation from real applications.

This chapter discusses the development of the Requirements-based Access Control Analysis and Policy Specification (ReCAPS) method, which is detailed in Chapter 4, within the context of its application to two formative case studies. Subsequent summative case studies, discussed in Chapter 5, enabled evaluation and refinement of the method. The key distinction between formative and summative is that formative case studies involve the development and evolution of the methods simultaneously coupled with validation, whereas summative case studies involve the validation of previously developed methods. These case studies unfolded over time, and ReCAPS evolved as a result of its application to the case studies discussed in this chapter.

Each of these case studies involves a real system:

- Security and Privacy Requirements Analysis Tool (SPRAT) (Section 3.1)
- Transnational Digital Government (TDG) Remote Border Control System (Section 3.2)

These projects are discussed in the following sections. In each section, the background of each project is briefly summarized. This is followed by discussions of the methodology employed and the lessons learned through the application of the method.

3.1 Security and Privacy Requirements Analysis Tool (SPRAT)

The Security and Privacy Requirements Analysis Tool (SPRAT) [JAS04] is a research tool under development at North Carolina State University with funding from the U.S. National Science Foundation (NSF). The tool supports goal-based and scenario-based requirements analysis and provides support for analyzing and specifying security and privacy requirements as well as ACPs. It builds upon and extends two existing tools: the Privacy Goal Management Tool (PGMT) [AEB04], and the Scenario Management and Requirements Tool (SMaRT) [SAA03].

With the help of SPRAT, analysts can seamlessly integrate goals, scenarios, requirements, policies and documents in a project repository and conveniently trace from one element (e.g., a goal) to another (e.g., a scenario) during requirements analysis. The SPRAT architecture calls for centralized data storage with distributed client access, with an option that allows local data storage. The envisioned users include requirements analysts, policy makers and security engineers. The information stored in the centralized database is proprietary and needs to be protected. Thus, access control is critical. For example, we have a rich privacy goal repository in the PGMT, which is the result of five case studies conducted over the course of five years. We treat these goals as valuable assets that need to be protected via access control. SPRAT analysts will have certain permissions to access these goals. Additionally, we may grant industry collaborators permission to view some attributes of the data (e.g., only the goal descriptions but

not the classifications and statistical data). In short, there are a variety of access control requirements in the system, making it a sufficiently sophisticated system to analyze using our approach.

3.1.1 Methodology and Case Study Artifacts

The SPRAT case study was conducted by two analysts (a software engineering professor and a PhD student) for approximately 16 person-hours.

Two source documents were used in this case study:

- SPRAT Software Requirements Specification (SRS), Version 1.09
- SPRAT database design E/R diagram

Each analyst had a copy of the source documents and a copy of the method summary, which included access control rule (ACR) identification steps but no guidance for ACR refinement (i.e., eliminating redundancies and resolving conflicts). Both analysts participated in the identification of access control rules, whereas only one analyst (the PhD student) conducted the ACR refinement steps. During the case study, the analysts devoted 30 minutes to initial training, in which the analysts reviewed the method summary together and set up procedures on how to document results. The objective of the training session was to ensure that both analysts had a common understanding of the method and its objective.

After the initial training, the analysts began the ACR identification process (this process is described in more detail in Chapter 4). Specifically, the analysts went through each requirement in the SRS document to identify access control elements. Patterns deemed helpful in identifying elements were documented as heuristics to aid analysts in future analyses. This kind of research approach is appropriate and consistent with grounded theory in which existing phenomena is analyzed to develop an understanding of the current state of a particular subject of interest. Grounded Theory is theory derived from data that has been systematically gathered and analyzed

[GS67]. Therefore, the heuristics presented in this dissertation are not based upon a distinct preconceived theory or hypothesis that we hoped to support or refute. Instead, our initial access control analysis and specification efforts were scientific analyses to develop new theory. Because at the time of this study there was no tool that supported access control analysis, the analysts performed the study manually and documented everything on paper. Specifically, during the ACR identification step, elements were marked on the SRS printouts using color coding and documented in an access control matrix in preparation for refinement in the next step. Each row in the access control matrix represents an access control rule, which is comprised of the following elements: *rule ID*, *mode*, *subject*, *action*, *object*, *condition*, *obligation*, and *source* (these terms are defined in the Glossary and described in more detail in Chapter 4). Each access control element is a column in the access control matrix. An example access control matrix is shown in Table 3.1. This is different from a traditional access control matrix shown in Table 3.2, in which each row presents a subject, each column presents an object, and the cell contains the access rights of the subject on the object. We chose the format shown in Table 3.1 because the format of a traditional access control matrix is insufficient to represent all the necessary information for an access control rule—e.g., conditions and obligations.

Table 3.1 An Example Access Control Matrix in the SPRAT Case Study

Rule ID	Mode	Subject	Action	Object	Condition	Obligation	Source
1	Allow	Analysts	View	Goals	Only the goals in the project that the analyst is assigned to	NULL	FR-GSM-16
2	Deny	Guests	Insert	Scenarios	NULL	NULL	FR-UA-4

Table 3.2 An Example Traditional Access Control Matrix

	File A	Directory B
Alice	Read, write	Read
Administrators		Read, write

Once all possible access control rules are identified from source documents, the ACRs must be refined. In this case study, the analyst found it most efficient to first sort all candidate rules in the access control matrix according to subject and object because redundancies often center around these two elements. Any redundant or conflicting rules were then identified and the analysts removed the redundant rules and resolved any conflicts (the process of removing redundant rules and resolving conflicts is discussed in detail in Chapter 4). Because this was our first formative case study, it was critical to document how we actually went about identifying redundancies and resolving conflicts so that we could then evaluate whether this process was generalizable to other systems and domains of interest. We documented this in the form of heuristics that we applied and validated in our subsequent case studies. We also documented any observations or recurring questions in the identification and refinement process that would be helpful to other analysts.

3.1.2 Lessons Learned

This section summarizes the lessons learned from the SPRAT case study and addresses the integration of these lessons into the ReCAPS method.

Access control analysis is data-centric.

The goal of access control is to protect system resources from unauthorized access. Access control analysis involves identifying system resources that must be protected and ways to protect

them. In information systems, data is the most important system resource. Thus, access control analysis is data-centric.

In contrast, requirements analysis is not data-centric. Functional requirements analysis focuses on the functions of the envisioned system, while non-functional requirements analysis focuses on attributes or constraints on functional requirements. Thus, requirements analysis is function-centric. Even though recent object-oriented analysis methods have expanded the scope of classical requirements analysis methods, the heart of requirement analysis is still specifying functions for a system.

Because requirements specifications are the main source of our approach for deriving access control policies, it is important for analysts to understand the scope of access control before they start the ACP specification process. ReCAPS recognizes this difference between access control analysis and requirements analysis and clearly defines the scope of access control at the beginning, as discussed in Chapter 4. Not all requirements involved access to data. If a requirement does not describe users' access to some *data*, then we cannot derive any access control policies from this requirement. For example, most non-functional requirements—except security and privacy requirements, such as reliability and usability—are not concerned with access control.

Consider the following three SPRAT requirements:

SPRAT FR-SSM-1: *The system shall provide the ability to add a scenario to the system.*

SPRAT FR-UAM-2: *The system shall allow secure storage of passwords in the database.*

SPRAT SR-1: *The system shall allow timestamps for all system interactions.*

Requirement FR-SSM-1 involves user access (i.e., add) to some data (i.e., scenario) in the system; thus, access control-related information is expressed in this requirement. In contrast, because FR-UAM-2 and SR-1 do not describe any user access to specific data or system resources, no access control rules can be derived from either of these requirements.

Checking requirements objects against database design helps ensure consistency.

One of the major challenges in software systems development is maintaining consistency between requirements and design. We observed this to be true in this case study. We identified two inconsistencies between the SRS and the database design in the SPRAT case study. As a result of the analysis, the source documents were inevitable improved.

While the requirements specifications in the SPRAT case study were very comprehensive, they were not as detailed as the database design in terms of information about objects. ReCAPS provides a heuristic (see **IH_{Object}3** on page 84) that compels analysts to check every object identified in the SRS against the database design. This heuristic helps analysts identify inconsistencies between these two artifacts and clarify ambiguous objects identified in the SRS, thus enabling analysts to improve the source documents and the quality of software products.

Consider the following SPRAT requirement:

SPRAT FR-UA-1: The system shall support an administrator level. The administrator will have the following privileges:

- a. Ability to create user groups such as NCSU TPP.org, GT TPP.org*
- b. ...*

We identified an object *user group* from this requirement. However, after checking with the database ER diagram, we did not find any corresponding element in the database design. This example showed that the database design and requirements were inconsistent; because the database design was incomplete, it did not satisfy the requirements. We thus were able to correct this error before it crippled later phases of software development for this project.

The database design may also provide more detailed information about objects than the requirements specifications. This information may be useful during the refinement process. Consider the following two requirements in the SPRAT:

SPRAT FR-GSM-12: The system shall be able to display the context of a goal.

SPRAT FR-GSM-16: *The system shall allow analysts to view elements of a goal returned by a query.*

By checking the database, we were able to learn what elements a *goal* contains, and determine that *context* is an element of a *goal*. This information was not available from the requirements specification but helped us refine the two rules derived from these two requirements: one allows analysts to view the context of a goal; the other allows analysts to view elements of a goal. Because *context* is an element of a *goal*, we determined that the first rule was redundant and, thus, removed the first rule.

Design decisions must be made during the ACP specification process.

In the SPRAT SRS, four kinds of users (user levels or roles) are clearly defined: *System Administrator*, *Project Manager*, *Analyst*, and *Guest*. However, the allowable privileges for each type of users are unclear in the SRS. At this time, it was important to make a design decision about whether the privileges of roles could overlap or not. However, this design decision could not be made without the intervention of stakeholders. After consulting stakeholders, we made the following decisions:

- The privileges of roles cannot overlap to conform to the separation of duties security principle [SCF96]; for example, there are no shared privileges for role *System Administrator* and role *Project Manager*. Thus, the three roles (*System Administrator*, *Project Manager*, and *Analyst*) in the system are independent, and no role hierarchy [SCF96] is defined.
- *System Administrators* are responsible for user management, such as creating/updating/deleting user/group accounts, assigning roles to users, and assigning users to groups.
- *Project Managers* are responsible for managing projects, such as creating new projects and assigning users to a project.

- *Analysts* are responsible for the analysis tasks within a project in which they are assigned, such as goal analysis and scenario analysis.

These design decisions disambiguated requirements for specifying ACPs, and the corresponding requirements were modified accordingly.

However, we delayed the design decision on what privileges should be assigned to *Guests* because even the stakeholders were unclear about that at the time of initial identification. We hypothesized that it would be easier to make that decision by considering all the possible privileges a specific user may have after we produced the access control matrix. However, the need to revisit this was documented to ensure that it would eventually be addressed.

Our experience with the SPRAT case study shows that the above hypothesis is indeed true. After we produced the access control matrix, we showed all the possible privileges to the stakeholders and helped them decide what privileges should be given to *Guests*. This time it was much easier to make the decision than at the initial identification step because the stakeholders were able to see all the possible privileges. As discussed in Chapter 4, ReCAPS requires that analysts document all design decisions made during the ACP specification process. These design decisions are an important part of the project documentation that justifies the rationales for the software design.

The ACP specification process is also a requirements disambiguation process.

During the ACP specification process, we identified many ambiguities in the requirements specifications. We believe this is partially because the ACP specification process is a late-phase RE activity that bridges requirements and design. To identify AC elements, we constantly challenge the requirements specifications by asking questions such as “who can do this?”, “how does he/she interact with the system?”, “what objects are accessed in this process?”, “are these objects designed in the database?”, and “is there any condition or obligation for the data access?”. Answering these questions inevitably helps disambiguate the requirements specifications. Thus,

the ACP specification process is also a requirements disambiguation process, and a side benefit of ReCAPS is the improvements to source documents by identifying and clarifying these ambiguities.

Consider the following SPRAT requirement:

SPRAT FR-GSM-6: *The system shall provide the ability to update an existing goal.*

This is an ambiguous requirement. To disambiguate the requirement, we asked the following questions:

- Who can update an existing goal?
- How does he/she perform the update? (or what are the steps to perform the update?)
- Which elements of a goal can he/she update?
- Are all the elements designed in the database?
- Under what conditions can he/she update an existing goal?
- Are there any obligations if he/she updates an existing goal?

By asking these questions, we were able to learn that only *Analysts* can update an existing goal. *Analysts* can update all elements of a goal if the goal exists in the system. This disambiguation process helped us obtain a better understanding of the requirement and revise it accordingly as this:

SPRAT FR-GSM-6 Revised: *The system shall allow analysts to update all elements of an existing goal.*

The new requirement was more clear and precise than the previous one.

Scenario analysis helps clarify ambiguous requirements and identify AC elements.

Scenario analysis is an effective technique for clarifying ambiguous requirements. In the SPRAT case study, there were requirements that describe only system features without further clarification. ReCAPS suggests using scenario analysis to elaborate ambiguous requirements (see

IH_{subject/action}3 on page 88). A scenario is comprised of a sequence of events, pre-conditions, post-conditions, obstacles, and any goals or requirements that are associated with this scenario. The main elements of a scenario are a sequence of events that describe possible interactions between users and the system. The sequence of events could be organized in normal flow, alternate flow, and iterative flow. Scenario analysis is very helpful in the ACR identification process because we want to know how users interact with the system and control their accesses accordingly. AC elements, such as actors, actions and objects can be derived directly from the events of a scenario.

Consider the following SPRAT requirement:

SPRAT FR-PM-3: The system shall support multi-user analyst results comparison.

This requirement is so ambiguous that without context information and further clarification it is impossible to understand what it means. However, this requirement contains access-related information. To derive access control rules from this requirement, we must clarify the ambiguities. In this case study, the analysts employed scenario analysis techniques to elaborate this requirement, as shown in Table 3.3.

Table 3.3 Scenario Analysis for SPRAT Requirement FR-PM-3

Scenario Name	Project Manager compares goal classification results of several analysts
Pre-conditions	Goals exist in the system and the classification method is defined.
Events	(1) Analysts classify goals independently according to predefined categories. (2) Project managers select analysts whose classification results they wish to compare. (3) The system shall display those goals that are classified differently and how they are different (e.g., by showing the different categories)
Post-conditions	NULL
Obstacles	Analysts did not complete their goal classifications when Project Manager compared their classifications.
Requirements	FR-PM-3: The system shall support multi-user analyst result comparison.
Goals	Get the goals that are classified differently by analysts and how they are different.

The above scenario analysis provides the analysts a better understanding about the requirement. Based on the scenario, the analysts were able to derive access control elements and rules from events, pre-conditions and post-conditions.

It is helpful to specify ACPs from both users' perspective and attackers' perspective.

The purpose of access control is to ensure that every access to the system and its resources is controlled. Thus, it is natural to consider users' interactions with the system and control their access accordingly. If a requirement is described from the system's perspective, we want to find out how users are affected. For example, in the SPRAT case study, consider the following requirement:

SPRAT FR-GSM-12: *The system shall be able to display the context of a goal.*

This requirement is described from the system's perspective, and no users are mentioned. To specify access control policies, we need to find out who will access the information if the system performs this function (i.e., display the context of a goal) and control those users' access accordingly. However, specifying ACPs from only users' perspective is insufficient. Sometimes we need to think from attackers' perspective. What if users have hostile intent? Would users exploit their privileges for their own good? This is especially useful for specifying conditions that restrict users from doing things that compromise the security and privacy of system resources (see **IH_{cond}7** on page 97). For example, in this case study, *Analysts* are able to perform all analysis activities, such as creating/updating/deleting/viewing goals/scenarios/requirements/policies. To help us think from an attacker's perspective, we authored the following concrete scenario to consider what an *Analyst* might do if he/she had hostile intent.

Consider the case when the SPRAT is used by students in a software engineering class for a term project. Students are divided into groups. All members in a group are *Analysts*. The students

are required to specify requirements for a software system using goal analysis and scenario analysis techniques. If *Analysts* have hostile intent, they might want to plagiarize other groups' work. To counteract this malicious intent, we should specify access control policies that prevent *Analysts* from accessing the other groups' work. Thus, we define the following two rules:

- *Analysts* can perform analysis activities only within the project to which they are assigned; and
- *Analysts* cannot assign themselves to any project in the system.

This example shows that considering an attacker's perspective can aid in specifying access control rules that protect system resources from unauthorized access.

Access control rules can be specified at different levels.

In the SPRAT case study, we observed that there are different kinds of access control rules. Some rules are related to users' access to databases; other rules are related to users' access to functions. These different kinds of rules imply that access control can be implemented at different levels, e.g., database level and application level. ReCAPS distinguishes between two kinds of actions in an access control rule: database actions and abstract actions. Database actions are direct operations upon a database, such as insert, update, delete, whereas abstract actions are not, i.e., withdraw (money from a banking account), process (a transaction). Database actions are identified to specify database-level access control policies, whereas abstract actions generally map to application-level access control policies. For example, in this case study, the analysts derived the following access control rules:

Example rule 1: Allow analysts to insert an entry into the table of goals.

Example rule 2: Allow analysts to classify goals.

The first rule is a database-level access control rule, in which the action is a database action (insert) and the object (goals) is an object in the database. The action in the second rule is an

abstract action. This rule can be rephrased as a application-level rule if goal classification is implemented in a function:

***Example rule 3:** Allow analysts to access the goal classification function*

Both database-level and application-level access control rules are useful, and ReCAPS supports both, as discussed in Chapter 4.

3.1.3 Results

Table 3.4 shows the results of the SPRAT case study. The data in Table 3.4 shows that the analysts made significant improvement to the source documents by creating 6 new requirements, revising 27 existing requirements, and identifying 2 inconsistencies between the requirements specifications and the DB design.

During the SPRAT case study, the analysts modified the process described in the initial method summary according to the lessons learned discussed in Section 3.1.2. As previously mentioned, the initial method summary included only the ACR identification steps, but not the ACR refinement steps. The analysts hypothesized that it would be easier to specify access control rules if we identified all the objects first and then specified rules that control users' access to each object. In the actual analysis process, they found this to not be true because it requires analysts to go through the entire document $n+1$ times if there are n objects identified. Specifically, analysts need to go through the entire document to identify all the objects. Then, for each object, analysts need to go through the entire document to identify all accesses to this object. This process was very inefficient. We thus revised the process to two steps: ACR identification and ACR refinement. During ACR identification, analysts must identify all possible rules for each requirement before they move on to the next requirement. After they finish the identification, they must refine the rules produced by removing redundant rules and resolving conflicting rules. This revision made the process much more efficient.

In the SPRAT case study, we created 17 heuristics (IH_{scope1} , IH_{scope3} , $IH_{object1-3}$, $IH_{subject/action1-3}$, $IH_{cond1-3}$, $IH_{cond7-8.a}$, $RH_{redundancy1-2}$, $RH_{redundancy4}$, $RH_{conflict1}$). These heuristics were later validated in our subsequent case studies.

Table 3.4 Summary of the SPRAT Case Study Results

	Pre-ReCAPS	Post-ReCAPS
No. of <i>tables / attributes</i> in the DB design	39/57	41/59
No. of <i>requirements (FR / NFR)</i>	59 (56/3)	65 (62/3)
No. of <i>modified requirements</i> as a result of ReCAPS analysis	N/A	27
No. of <i>inconsistencies</i> identified between the SRS and the DB design as a result of ReCAPS analysis	N/A	2
No. of <i>access control rules</i> created during ACR identification (after ReCAPS Step #2)	N/A	84
No. of <i>final rules</i> (after ReCAPS Step #3)	N/A	73
No. of <i>final policies</i> (after ReCAPS Step #3)	N/A	34

3.2 Transnational Digital Government (TDG)

The Transnational Digital Government (TDG) project, funded by the U.S. National Science Foundation (NSF)⁵, is a collaborative research project involving researchers at seven universities, as well as government agencies in three participating countries: U.S., Belize and Dominican Republic. The project's objective is to research advanced information technologies useful for rapid collection, dissemination, and exchange of information related to transnational border control. The system collects and shares immigration information. The prototype system project is the object of this case study.

⁵ The TDG Project URL: <http://www.acis.ufl.edu/transdg/>

The security and privacy of immigration data is critical in this project. Secure remote sharing of sensitive data requires reliable access control mechanisms. RBAC and distributed trust management were chosen to implement this functionality. Many organizations are involved in the transnational border control process, including remote border stations, governments, police departments, immigration departments, and customs. Information must flow among these organizations as well as across national borders. Different countries have different security and privacy policies and information sharing laws regulating the corresponding activities. Policy enforcement was important throughout the system's development to ensure compliance with the corresponding laws and policies. Using our approach, we specified a set of ACPs for the TDG project in collaboration with the TDG database team at the University of Florida.

3.2.1 Methodology and Case Study Artifacts

The TDG case study was conducted by the same two analysts as the SPRAT case study, for approximately 6 person-hours.

Two source documents were used in this case study:

- TDG Software Requirements Specification (SRS), Version 2.0
- TDG database schema design

Note that in this study, the source document was slightly different from the SPRAT case study. In the TDG case study, we used the database schema design as one of the source documents, whereas in the SPRAT case study, we used the database E/R diagram design. This difference makes the results of these studies stronger because it shows our method applies to various forms of database design.

The analysis process was similar to the process used in the SPRAT case study except that the method was improved after the first case study. Specifically, we divided the analysis process into two steps: identification and refinement. Both analysts participated in the identification of access

control rules, whereas only one analyst (the PhD student) conducted the ACR refinement step. We also created a set of heuristics to help analysts perform the analysis.

Each analyst had a copy of the source documents and a copy of the new method summary. During the case study, the analysts spent 30 minutes on initial training on the analysis method. After the initial training, analysts started the ACR identification process. In contrast to the SPRAT case study, the analysts applied and validated the heuristics created in the SPRAT case study when they went through each requirement in the SRS document. When new patterns were identified and no heuristic could be applied, these patterns were documented and included in the heuristics set for future use and further validation. This is the characteristic of formative case studies: they serve as both the origin and early validation of the method. The ReCAPS process and heuristics were shaped during these case studies.

As in the SPRAT case study, access control elements were marked on the SRS printouts, using color coding and documented in an access control matrix in preparation for refinement in the next step. Once all possible rules were identified, they were refined. Again, the analysts applied the heuristics created in the SPRAT case study and created new ones if necessary. As in the SPRAT case study, the analysts documented any observations or recurring questions in the identification and refinement process.

3.2.2 Lessons Learned

This section summarizes the lessons learned from the TDG case study and addresses the integration of these lessons into the ReCAPS method.

Scenario analysis is useful for clarifying ambiguous requirements and specifying ACPs.

While scenario analysis already demonstrated its usefulness in clarifying ambiguous requirements and specifying ACPs in the SPRAT case study, its effect was even more significant

in the TDG case study, thus validating the heuristics created in the SPRAT case study (see page 57). The requirements specifications of the TDG project are more ambiguous than those of the SPRAT project. Scenario analysis helped us elaborate some ambiguous requirements. Figure 3.1 shows the scenario that border immigration agents at point-of-entry stations check in travelers using the Scenario Management and Requirements Tool (SMaRT) [SAA03]. This scenario contains several episodes, which can be treated as reusable scenarios. For example, Episode 2 describes different ways that a border immigration agent enters the traveler's information in the system.

As previously mentioned, access control elements can be derived from scenario elements. To explain how the analysts derived access control rules from this scenario, consider the watch list checking episode (Event 4—6) in this scenario (see Figure 3.1a). From Event 4, we identified an object “watch list”. We mapped this object to the table `SALIDA_WATCH`, which stores the actual watch list—information about suspicious travelers. The action word “access” is ambiguous. At a minimum, “access” implies “read”. Thus, an access control rule was derived as follows:

Example rule 1: Allow border immigration agents to select table `SALIDA_WATCH`.

Event 5 describes that border immigration agents can query the “watch list”. This is essentially the same as the above rule.

Event 6 describes different cases of the query result. In the interest of space, we only discuss the first case, when the traveler was identified on the watch list (Alternation Branch 6.1). In this case, the system will instruct border immigration agents as to what should be done (Event 6.1.2). Another access control rule was identified here:

Example rule 2: Allow border immigration agents to view actions to be taken upon a traveler being identified on the watch list.

Select Project
Preferences
Sign Out

Do a case insensitive search on the Scenarios.

Scenario

Immigration Agent at border entry stations enter information about visitors
Immigration Agent at border entry station check in a traveler
[View](#) [Edit](#) [Remove](#)
[Convert Current](#)

Details

Name: Immigration Agent at border entry station check in a traveler
User Identifier: N/A
Source Identifier: Slides notes of the TDG conference presentation (TDG: the future in the near-term)
Description:

Event List

- 1.Traveler : present immigration documents
- 2.Episode: Immigration Agent at border entry stations enter information about visitors
 - 2.1.Alternation
 - 2.1.1.Alt. Branch
 - 2.1.1.1.Immigration Agent : enter information in the system via keyboard
 - 2.1.1.2.System : save entered information
 - 2.1.2.Alt. Branch
 - 2.1.2.1.Immigration Agent : ask questions
 - 2.1.2.2.Alternation
 - 2.1.2.2.1.Alt. Branch
 - 2.1.2.2.2.Alt. Branch
 - 2.1.2.2.2.1.System : recognize the spoken language of immigration agent
 - 2.1.2.2.2.2.System : translate one language to another language
 - 2.1.2.2.2.3.System : speak the specified text in either English or Spanish
 - 2.1.2.2.3.Traveler : answer questions
 - 2.1.2.3.Traveler : answer questions
 - 2.1.2.4.Alternation
 - 2.1.2.4.1.Alt. Branch
 - 2.1.2.4.1.1.Immigration Agent : enter information in the system via keyboard
 - 2.1.2.4.1.2.System : save entered information
 - 2.1.2.4.2.Alt. Branch
 - 2.1.2.4.2.1.System : recognize the spoken language of visitor
 - 2.1.2.4.2.2.System : translate one language to another language
 - 2.1.2.4.2.3.Alternation
 - 2.1.2.4.2.3.1.Alt. Branch
 - 2.1.2.4.2.3.1.1.System : speak the specified text in either English or Spanish
 - 2.1.2.4.2.3.2.Alt. Branch
 - 2.1.2.4.2.3.2.1.System : save language translation results
 - 2.1.3.Alt. Branch
 - 2.1.3.1.Immigration Agent : scanning documents
 - 2.1.3.2.Immigration Agent : request system to save scanned documents
 - 2.1.3.3.System : save scanned documents
 - 2.1.4.Alt. Branch
 - 2.1.4.1.Immigration Agent : make photocopies of the immigration documents
 - 2.1.4.2.Immigration Agent : save photocopies of immigration documents offline
- 3.Episode: Immigration agent query the system
 - 3.1.Alternation
 - 3.1.1.Alt. Branch
 - 3.1.1.1.Immigration Agent : author some kind of query via keyboard and provided GUI
 - 3.1.2.Alt. Branch
 - 3.1.2.1.Immigration Agent : ask system a question in spoken language
 - 3.1.2.2.System : recognize the spoken language of immigration agent
 - 3.1.2.3.System : formalize the translated result as a query
 - 3.2.Alternation
 - 3.2.1.Alt. Branch
 - 3.2.1.1.System : return query results
 - 3.2.2.Alt. Branch
 - 3.2.2.1.System : translate query results into specified language
 - 3.2.2.2.System : return translated query results

Figure 3.1a Scenario Analysis in the TDG Case Study

4.Immigration Agent : request to access the "watch list" in the national immigration database
5.Immigration Agent : ask the system whether the traveler is on the "watch list"
6.Alternation
6.1.Alt. Branch
6.1.1.System : answer "YES"
6.1.2.System : instruct what should be done
6.1.3.Immigration Agent : take appropriate actions instructed by the system
6.2.Alt. Branch
6.2.1.System : answer "NO"
6.2.2.Alternation
6.2.2.1.Alt. Branch
6.2.2.1.1.Immigration Agent : does not identify any suspicious behavior of the traveler
6.2.2.1.2.Immigration Agent : take appropriate actions to end the check-in procedure and allow the traveler to enter the country
6.2.2.2.Alt. Branch
6.2.2.2.1.Immigration Agent : identify suspicious behavior of the traveler
6.2.2.2.2.Immigration Agent : request previous border access records about the traveler from the national immigration database
6.2.2.2.3.System : retrieve previous border access records from national immigration database
6.2.2.2.4.Immigration Agent : request the national central immigration server to verify traveler's passport information with the government issuing the passport
6.2.2.2.5.System : retrieve the corresponding information from the country that issued the passport
6.2.2.2.6.Alternation
6.2.2.2.6.1.Alt. Branch
6.2.2.2.6.1.1.Immigration Agent : confirm that all information about the traveler is correct
6.2.2.2.6.1.2.Immigration Agent : take appropriate actions to end the check-in procedure and allow the traveler to enter the country
6.2.2.2.6.2.Alt. Branch
6.2.2.2.6.2.1.Immigration Agent : determine the traveler as a suspicious/wanted/illegal traveler
6.2.2.2.6.2.2.Immigration Agent : create a new record in the national immigration database's "watch list"
6.2.2.2.6.2.3.Immigration Agent : notify appropriate departments and people about the traveler's case
6.2.2.2.6.2.4.Immigration Agent : take other appropriate actions to end the check-in procedure

Goals

Check in travelers
Identify suspicious travelers

Obstacles

Visitor appears nervous and tense
Visitor does not understand English or Spanish
Visitor is accompanied by a minor
Visitor is intoxicated
Visitor needs assistance going through the station

Requirements

No Requirements Exist

Conditions

Preconditions:

Immigration agent is authenticated to access the national immigration server/database [\[References\]](#)
Immigration data can only be used for immigration purpose [\[References\]](#)
visitor has appropriate immigration documents [\[References\]](#)

Postconditions:

audit trail of transnational immigration data access must be generated [\[References\]](#)

Figure 3.1b Scenario Analysis in the TDG Case Study

Event 6.1.3 describes border immigration agents taking actions instructed by the system. This is not enforceable by the system. Thus, no access control rules were derived.

Inconsistencies between requirements specifications and database designs are common.

The TDG project is unique in its distributed development environment. The stakeholder countries are Belize and Dominican Republic. The research team includes five universities in the U.S. Such a distributed environment causes inevitable communication difficulties between all stakeholders. In the TDG case study, we identified 17 inconsistencies between the requirements specifications and the database design. Many of these inconsistencies are due to missing items that were mentioned in the requirements specifications but not designed in the database. For example, the SRS was attached with scanned documents from stakeholders in Belize and Dominican Republic, including arrival/departure records, and entry permit. However, in the database design, some fields that were shown on the arrival/departure records were missing in the database design, such as mode of transportation (airline and flight number, vehicle license number, and vessel name). Given that there are only 25 functional requirements in the system, this number seems large. However, this was not really surprising given that the requirements specifications were authored by the North Carolina State University team, whereas the database was designed by the University of Florida team. Insufficient communication between both the teams yielded inconsistencies between the requirements and the database design. This is a common software engineering problem, and our case study shows that ReCAPS helps identify and eliminate these inconsistencies. This is a side benefit of ReCAPS. This lesson learned from the TDG case study validated the heuristics created in the SPRAT case study on checking requirements specification against database design (see page 54).

The level of detail in an SRS affects the efficiency of ReCAPS on ACP specification.

The SPRAT SRS provides a more detailed functional description than the TDG SRS. In contrast, the requirements description in the TDG project is high-level and stable, yet somewhat ambiguous. We have tried to clarify the requirements, but the results are less than satisfactory due to the distributed environment, large research team and communication complexities that accompany any project of this size. As previously mentioned, our stakeholders were in two developing countries: Belize and Dominican Republic. Our only direct contact with immigration officers in these two countries was during our visit to Belize. At all other times, we contacted them via our stakeholder liaisons in two universities in these countries. Additionally, the research team was comprised of five US universities. Each university was investigating a different aspect of the system. The requirements team joined the project nine months after it had been started. These factors all contributed to the formidable challenge of disambiguating the requirements.

3.2.3 Results

Table 3.5 summarizes the results of the TDG case study. In this case study, we create a new heuristic (IH_{scope2}). Of the 17 heuristics created during the SPRAT case study, we applied and validated 13 of them (IH_{scope1} , IH_{scope3} , $IH_{object1-3}$, $IH_{subject/action1-3}$, $IH_{cond1-3}$, IH_{cond7} , $RH_{redundancy1}$). We identified 17 inconsistencies between the requirements specification and the database design, as shown in Table 3.6. All the inconsistencies were items in the requirements specification that were missing from the database design. We did not make changes to the analysis process, which implied that the analysis process was quite stable at this time.

Table 3.5 Summary of the TDG Case Study Results

	Pre-ReCAPS	Post-ReCAPS
No. of <i>tables / attributes</i> in the DB design	5/66	6/82
No. of <i>requirements (FR / NFR)</i>	44 (25/19)	44 (25/19)
No. of <i>modified requirements</i> as a result of ReCAPS analysis	N/A	4
No. of <i>inconsistencies</i> identified between the SRS and the DB design as a result of ReCAPS analysis	N/A	17
No. of <i>access control rules</i> created during ACR identification (after ReCAPS Step #2)	N/A	18
No. of <i>final rules</i> (after ReCAPS Step #3)	N/A	16
No. of <i>final policies</i> (after ReCAPS Step #3)	N/A	6

3.3 Summary

This chapter presented the two formative case studies that served as the conceptual origin for the Requirements-based Access Control Analysis and Policy Specification (ReCAPS) method.

Each case study detailed in this chapter involved a particular system:

- Security and Privacy Requirements Analysis Tool (SPRAT)
- Transnational Digital Government (TDG) Remote Border Control System

These case studies served as a source of early validation, shaping the ReCAPS method through the lessons learned. The next chapter introduces ReCAPS in detail.

Table 3.6 Inconsistencies Identified in the TDG Case Study

Requirements	Database Design
SRS document appendix: Belize arrival record	Missing permit/visa number in table MAIN
SRS document appendix: Belize arrival record	Missing frequency of visits in table MAIN
SRS document appendix: Belize arrival record	Missing arrival address in Belize in table MAIN
SRS document appendix: Belize departure record	Missing last address in Belize in table MAIN
SRS document appendix: Belize arrival record	Missing mode of transportation: airline and flight number in table MAIN
SRS document appendix: Belize arrival record	Missing mode of transportation: vehicle license number in table MAIN
SRS document appendix: Belize arrival record	Missing mode of transportation: vessel name in table MAIN
SRS requirement 2.3.1: The system shall allow BIAs to determine if the traveler is on a “watch list” of suspicious or wanted individuals	Missing Belize watch list
SRS requirement 2.3.1 annotation from Belize stakeholder	Missing Belize watch list item: name
SRS requirement 2.3.1 annotation from Belize stakeholder	Missing Belize watch list item: date of birth
SRS requirement 2.3.1 annotation from Belize stakeholder	Missing Belize watch list item: nationality
SRS requirement 2.3.1 annotation from Belize stakeholder	Missing Belize watch list item: reason for being on the watch list
SRS requirement 2.3.1 annotation from Belize stakeholder	Missing Belize watch list item: action to be taken upon being identified
SRS document appendix: Dominican Republic Exit Control Form	Missing Dominican Republic passport number in table SALIDA
SRS requirement 2.3.1 annotation from Dominican Republic stakeholder	Missing Dominican Republic watch list item: gender
SRS requirement 2.3.1 annotation from Dominican Republic stakeholder	Missing Dominican Republic watch list item: reason for being on the watch list
SRS requirement 2.3.1 annotation from Dominican Republic stakeholder	Missing Dominican Republic watch list item: action to be taken upon being identified

Chapter 4

Requirements-based Access Control Analysis and Policy Specification (ReCAPS)

罗马不是一天建成的。

Rome was not built in a day.

—*French Proverb*

This dissertation addresses the challenges of specifying access control policies (ACPs) for information systems; this kind of specification is a conceptually and practically complex process. Often ACP specification has been isolated from requirements analysis and software design, resulting in misalignment between different levels of policies and software artifacts [AEC03]. Additionally, ACP specification was often conducted in an ad-hoc manner based on tacit knowledge according to the survey in Chapter 2, leaving systems vulnerable to security and privacy breaches. This chapter introduces the Requirements-based Access Control Analysis and Policy Specification (ReCAPS) method, which integrates policy specification into the software development process and provides procedural support for ACP specification.

This chapter is organized as follows. Section 4.1 provides an overview of the ReCAPS method. Section 4.2 details the process of specifying access control policies and introduces the ReCAPS heuristics, using concrete examples from the two formative case studies, discussed in Chapter 3. Section 4.3 discusses the SPRAT (Security and Privacy Requirements Analysis Tool) ReCAPS module that supports ReCAPS activities.

4.1 Overview of ReCAPS

This overview first discusses ReCAPS within the context of an ICOM (Inputs-Constraints-Outputs-Mechanisms) model. Then the underlying assumptions upon which the method was developed are clarified. The design principles are subsequently explained. Finally, the main activities an analyst performs when applying ReCAPS are briefly summarized.

4.1.1 An ICOM Model of ReCAPS

The fundamental basis for ReCAPS is that access control policies are representations and implementations of security requirements at the software design level. Security requirements are constraints on functional requirements [MHN04]. Thus, in order to specify access control policies for a system, we need to examine the system's functional requirements and non-functional security requirements. Additionally, requirements-level knowledge alone is insufficient to specify access control policies. Requirements are often specified as high-level goals or objectives. They are often very general and implementation-independent. In contrast, access control policies are concerned with a specific subject's privileges to access a specific object. They are tightly connected with system design. Thus, design-level knowledge must also be obtained because we need to include detailed information about subjects and objects in access control policies.

Figure 4.1 portrays the ReCAPS method using a traditional ICOM (Inputs-Controls-Outputs-Mechanisms) model [IDE93]. The sources (inputs) for the approach are various software documents, including the software requirements specification (SRS), the design document and database design (e.g., E/R diagram), as well as the high-level security and privacy policies. For ReCAPS to be effectively applied, the SRS and the DB design are required sources. Other documents, such as the design document and organizational security and privacy policies, are optional (yet helpful) sources. The two required source documents are complementary in that the

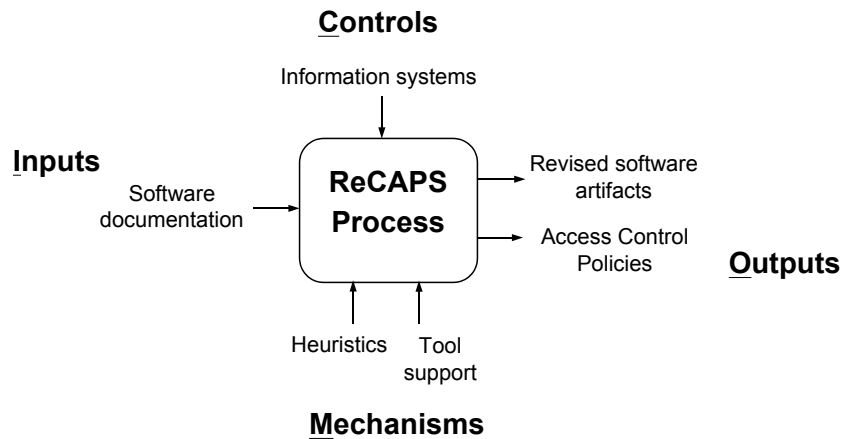


Figure 4.1 ReCAPS ICOM Model

requirements specification justifies the rationale for the ACPs (e.g., why a user is granted a particular privilege to perform a task or access an object), whereas the database design details the objects to which any access must be controlled. Both of these are necessary for ACPs.

The mechanisms of this method include an analysis process, a set of heuristics and software tool support. The ReCAPS method aids software and security engineers (analysts) by providing a detailed process description and associated heuristics that they can apply while examining available source documents to derive and specify ACPs. This process and its associated heuristics are detailed in Section 4.2. During this process, analysts employ requirements engineering (RE) techniques (see Chapter 2), such as goal-based requirements analysis [DLF93], scenario analysis techniques [PTA94], and misuse cases [Ale03, SO00], to help identify AC elements. A software tool (see Section 4.3) was developed to support the ReCAPS activities.

The overall objective of our approach is to produce a comprehensive set of ACP specifications that are aligned and compliant with the software requirements, security policies, and privacy policies. The emphasis on alignment and compliance ensures that analysts are also able to clarify ambiguities in the SRS, as well as identify and resolve any inconsistencies between the SRS and database design. Thus, the source documents inevitably benefit from the process as the SRS and DB design are augmented, which results in more complete, correct and less

ambiguous project documentation. During the ACP specification process, analysts often make design decisions. This documented list of design decisions is another output of the method.

The ReCAPS method applies to only information systems, shown as controls in Figure 4.1.

4.1.2 Assumptions

It is important to clarify the underlying assumptions upon which the ReCAPS method was developed to clearly frame the scope and contributions of this method. ReCAPS is based on the following three assumptions that narrow the scope of our efforts to information systems:

Assumption #1: Analysts need a system's DB design and SRS to specify ACPs for the system. Both are minimal pre-requisite source documents for this approach. Essentially, ACPs bridge the gap between an information system's SRS and its DB design.

Assumption #2: There are various objects in a system to which access may be restricted. For example, an employee may be allowed to print jobs to a local printer, but not to a central printer in the neighboring office. This kind of resource access control is beyond the scope of this dissertation. Instead, we focus on restricting access to data within a DB.

Assumption #3: ACPs are specified for information systems, supported by a database containing sensitive data. We have not investigated ACP specifications for security kernels such as file access in operating systems.

4.1.3 Design Principles

We followed the following design principles when developing the ReCAPS method for ACP specification. These design principles are goals we want to achieve in ReCAPS.

Principle #1: Different levels of policies and system requirements must be brought into better alignment to ensure that unauthorized accesses to sensitive data are prevented. Organizational security and privacy policies, software requirements, and the operational

functioning of policy-enforcing systems are often misaligned. The underlying principle of our approach is thus to ensure compliance between high-level security and privacy policies, access control policies, system requirements and software designs.

Principle #2: Traceability across software artifacts (e.g., between ACPs and requirements) must be maintained. Both policies and requirements may change throughout system development and even after the system is deployed. They are interdependent with respect to change. When one of them changes, it is important to make appropriate changes to the other. Otherwise, inconsistency across these artifacts may cause security and privacy breaches. Traceability helps analysts track changes and maintain consistency between requirements and policies.

Principle #3: Access control analysis and requirements analysis is an iterative process. Both require one to maintain documents containing information with different levels of formality and describing different kinds of information. Therefore, it is helpful to employ an inquiry-driven approach [PTA94] to document important or recurring questions, design decisions, as well as identified inconsistencies that are pending resolution.

4.1.4 Activities

Figure 4.2 portrays the main activities a software or security engineer undertakes to derive ACPs from various available sources. The process is detailed with examples in Section 4.2. This section provides an overview of the main activities.

There are two main kinds of activities in ReCAPS: analysis and refinement. Recall that ACPs are comprised of AC rules and these rules are comprised of elements (e.g., subject, actions, objects). During the *analysis* process, analysts scan available source documents, following the ReCAPS steps while applying the heuristics to identify AC elements and specify AC rules. When analysts examine source documents, they may find ambiguous requirements. In this case, they need to consult stakeholders to clarify the ambiguities. For example, in the TDG case study,

analysts had to consult stakeholders about the check-in process at border-of-entry stations in order to specify access control policies for border immigration agents. The analysis process requires analysts to check requirements against the database design. For example, when we identified an object “watch list”, we checked the database design to make sure this information can be found. Because these artifacts are often produced by different software engineers (i.e., requirements specifications are produced by requirements engineers, whereas database is designed by another group of software engineers, often software designers), analysts may also find inconsistencies between the requirements and the database design. In this case, they need to resolve these inconsistencies as discussed in Section 4.2. Additionally, the checking may reveal items that are clearly described in the requirements specification but are not specified/designed in the database. These missing items must be added to the database design to make it complete. Incomplete designs are more expensive to rectify if they are identified in later phases of software development. Additionally, incomplete designs may cause security and privacy breaches, even system failure if the problems are not identified and corrected.

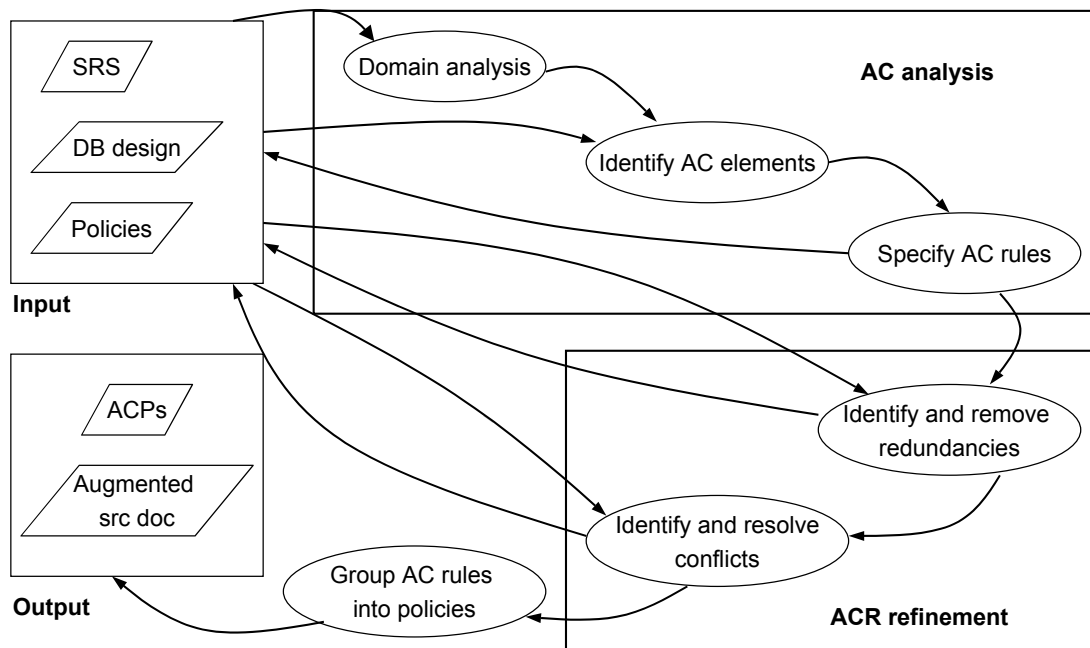


Figure 4.2 ReCAPS Analysis Activities

After the identification step, analysts produce a set of candidate access control rules (see example in Table 3.1 in Chapter 3 on page 51). Because these rules are derived from different sources (e.g., requirements, high-level security and privacy policies), there may exist redundant or conflicting rules. Our ultimate goal is to produce a set of access control policies that can be implemented by software developers. Thus, analysts need to further *refine* these candidate rules by removing any redundancy and resolving all conflicts. To remove redundancies and resolve conflicts among candidate rules, analysts may need to go back to the sources from which the rules were derived to find the source of the conflicts. Analysts may have to consult stakeholders to resolve conflicting rules (see ERS case study in Chapter 5). After the refinement process, all AC rules must be grouped into ACPs.

Analysts often make design decisions during the identification and refinement process. For example, in the SPRAT case study, analysts made a design decision that the privileges of roles cannot overlap (see page 55). Design decisions that are made during the analysis must be documented. These documented decisions are part of project documentation and they justify the rationale for the software design.

4.2 Analysis Process and Heuristics

This section details the steps that analysts perform when applying ReCAPS to specify access control policies for information systems using examples from the two formative case studies.

There are four steps in the ACP specification process:

Step 1: Understand the problem domain;

Step 2: Scan available source documents to identify AC elements and specify AC rules;

Step 3: Refine access control rules; and

Step 4: Group AC rules into ACPs.

The ACP specification process is supported with four kinds of heuristics:

- Identification Heuristics (**IH**): for identifying access control elements;
- Refinement Heuristics (**RH**): for refining access control rules into policies;
- Specification Heuristics (**SH**): for specifying access control rules and policies; and
- Grouping Heuristics (**GH**): for grouping access control rules into policies.

Examples for how to apply these heuristics are explained in detail in this section.

4.2.1 Preparation

The objective of the preparation step is to develop an understanding of the problem domain before analysts start the ACR identification and refinement steps.

Step 1: Understand the problem domain.

To obtain a general understanding of the problem domain, analysts need to quickly scan any available source documents to understand the background of the envisioned system; this includes information about the objective of the system and the main stakeholders. Generally, the software requirements specification (SRS) contains introductory material that summarizes this information and details the stakeholders and the end users. Because the purpose of ACP specification is to control end-users' access to the system, it is important to understand who the end-users are. During the ACP specification process, analysts may need to consult stakeholders to clarify ambiguities. Thus, it is also important to understand who the stakeholders are so that analysts can contact them when needed (for example, to reconcile a conflict or disambiguate a policy or requirement) during the analysis process.

Consider the following information from the introduction section of the TDG SRS document:

TDG SRS Introduction: *The critical stakeholders of this project include the following parties:*

- (1) *The Belize government*
- (2) *The Belize border immigration agents*
- (3) *The Belize immigration agencies*
- (4) *The Dominican Republic (DR) government*
- (5) *The Dominican Republic border immigration agents*
- (6) *The Dominican Republic immigration agencies*
- (7) *The Organization for American States (OAS)*

In this example, we identified seven stakeholders for the TDG system. It is not clear how the stakeholders interact with the system at this time. But in case there is a need to consult stakeholders (or sometimes the person whom served as the source of a requirement) to clarify some ambiguities in the requirements specifications or resolve some conflicts, we know which stakeholder we should contact.

4.2.2 Access Control Rule Identification & Specification

The identification of access control elements from requirements specifications is primarily based on a mapping relationship between access control elements and requirements specification elements, as shown in Table 4.1.

Table 4.1 Mapping between ACP Elements and Requirements Specification Elements

Requirements Specification Elements	Access Control Policy Elements
Actors	Subjects
Actions	Actions
Objects	Objects
Constraints (Pre-conditions)	Conditions
N/A	Obligations

As previously explained in Section 2.3.3, a *subject* in an access control rule is an entity, such as a user or program agent that may access objects. An *object* is an entity, such as a data field, table, procedure or application to which access is restricted. In a sense, subjects and objects are basically the same thing. For example, an application may be specified as an object in one rule, in which a user is restricted to access this application; whereas the same application may be specified as a subject in another rule, in which the application is restricted to access some file in the system. This is what makes the analysis process challenging for analysts. An *action* is a simple operation (e.g. read or write) or an abstract operation (e.g. deposit or withdraw). *Conditions* are restrictions that must be satisfied before an access request is granted. *Obligations* are actions that must be fulfilled if a request to access an object is granted. Each of these elements, except obligations, may be directly mapped from a requirements specification element as we discuss herein.

In requirements engineering, scenarios and goals are employed to model requirements. As discussed in Section 2.1.3, a *scenario* is comprised of a sequence of events that describe possible/envisioned interactions between users and a system. Scenario elements include actors, actions, objects, and pre/post-conditions. Actors, actions and objects in a scenario correspond to subjects, actions and objects in access control rules. A scenario's pre-conditions express what the scenario expects to be true when it begins [Als02]. In ReCAPS, these scenario pre-conditions correspond to conditions in access control rules—restrictions that must be satisfied before an access request may be granted. Thus, pre-conditions may be mapped to conditions in access control rules. Additionally, requirements specifications often specify constraints that place a condition on the achievement of a goal [Ant97]. Constraints can also be mapped to conditions. Goals are targets for achievement and high-level objectives of the business, organization, or system [Ant97]. As discussed in Section 2.3.3, the concept of goals is comparable to purposes in access control policies. Thus, goals in requirements may be mapped to purposes in access control, which are specified as conditions for an access control rule (see page 99).

These mapping relationships facilitate the process of deriving access control policies from requirements specifications. The remainder of this section details the heuristics that provide prescriptive guidance for identifying access control elements from requirements specifications.

Step 2: Scan available source documents to identify AC elements and specify AC rules.

The SRS is the major source from which to derive access control elements. Recall that the five AC elements are: *<subject, object, action, condition, obligation>*. For each requirement in an SRS, analysts need to follow steps 2.1 through 2.4 in the ReCAPS method to identify these elements. It is not necessary to follow these steps in sequential order; an experienced analyst may perform some of these steps in parallel.

Not every requirement contains access control-related information. Thus, as previously mentioned, understanding the scope of access control is important for identifying access control rules.

The following heuristics (**IH_{scope}1-3**) help analysts determine whether or not any access control rules can be derived from a requirement.

IH_{scope}1: Access control analysis is data-centric. Thus, for each requirement, analysts should ask the following question: *does this requirement involve any user access to sensitive information in the system?* If the answer to the above question is yes, then access control rules may be derived from this requirement. Otherwise, no access control rules can be derived from this requirement.

Consider the following TDG requirement:

TDG 2.8.1: The system shall have an Internet portal for MEM-related activities. (MEM: Multilateral Evaluation Mechanism)

Given requirement TDG 2.8.1, it is easy to see that it does not involve any user access to sensitive information in the system. Thus, no access control rules can be derived from this requirement.

In contrast, consider the following SPRAT requirement:

SPRAT FR-GSM-3: The system shall allow analysts to classify goals.

This requirement describes a task that “analysts” can perform, which involves user’s (i.e., analyst) access (i.e., classify) to data (i.e., goals) in the system. Thus, access control rules may be derived from this requirement.

IH_{scope2}: If a requirement describes something that is not enforceable in the system, then no access control rules can be derived from the requirement.

For example, a security requirement may describe some physical security safeguards (e.g., access pass to the building) to control users’ access to the building or system. However, this is not something that can be enforced in the system, thus no access control rule can be derived from it.

IH_{scope3}: No access control rules can be derived from non-functional requirements, except security and privacy requirements.

Non-functional requirements are attributes of a system, such as compatibility, usability, reusability, interoperability, scalability, reliability, and accuracy. Consider the following TDG requirement:

TDG 3.3.2: The system shall provide 90% accuracy of speech recognition functionality.

This requirement defines the expected speech recognition accuracy for the system. Although speech recognition involves users’ access to system resources, accuracy is a system attribute. Thus, no access control rules can be derived from this requirement.

4.2.2.1 Identifying Objects

Step 2.1: Identify objects that need to be protected.

In software based information systems, objects are data to which access needs to be restricted. Three heuristics (**IH_{Object1}**, **IH_{Object2}**, **IH_{Object3}**) are provided to help analysts identify objects:

IH_{Object1}: To identify objects that must be protected in the system, it is helpful to examine the nouns that follow verbs.

Consider the following SPRAT requirement:

SPRAT FR-GSM-3: The system shall allow analysts to classify goals.

Based on heuristic **IH_{Object1}**, the noun “goals” follows the verb “classify”, thus “goal” is tagged as an object. Note that because not every noun is an object, the following heuristic is used to distinguish access-related objects from other objects.

IH_{Object2}: To distinguish access-related objects from other objects, it is helpful to consider whether the candidate object is a system resource that should only be accessed by authorized actors.

In the case of SPRAT FR-GSM-3 above, heuristic **IH_{Object2}** ensures “goals” is the identified object to be protected instead of “analyst”.

IH_{Object3}: To identify the objects that must be included in the database, every object identified in the SRS should be mapped to an object (e.g., a table, a column, a row or a cell) in the database design.

Heuristic **IH_{Object3}** has two benefits. First, it compels analysts to clearly define what the objects are in the database (e.g., a table, a column, a row or a cell in a table). Requirements are often general and ambiguous, but access control policies must be specific. Analysts must check

the requirements against the DB design to make sure the objects are appropriately defined in the DB design. Second, this heuristic helps ensure that the requirements and database design are consistent with one another. Because requirements specification and design specification are often treated separate phases and are often produced by different persons in a traditional software development process, thus inconsistencies inevitably exist between these artifacts.

Consider the following TDG requirement:

TDG 2.3.1: *The system shall allow border immigration agents to determine if the traveler is on the “watch list”.*

Requirements documents have auxiliary notes, such as questions, issues, rationales, and comments. This TDG requirement was annotated with questions and answers from the requirements engineers and stakeholders. These annotations provided detailed information about exactly what is on the watch list as shown in TDG 2.3.1 Annotation below.

TDG 2.3.1 Annotation: *Question C: What data is contained on the “watch list”?*

Belize Answer: Name, Date of Birth, Nationality, Reason for being on the list, Action to be taken.

DR Answer: Basically name, gender, citizenship, watch list inclusion explanation and actions to be taken in case of positive identification.

In the TDG DB design, three items that were mentioned in the annotation for requirement TDG 2.3.1 were missing in the DB design: gender, the reasons for a person’s name being on the watch list, and actions to be taken if person whose name appears on the watch list is encountered at a border station. This example shows how heuristic **IH_{Object}3** helps analysts correct database designs early on, preventing possible costly changes that may not have been identified until well into the development lifecycle.

Access control can be implemented at different levels: database-level and application-level, as discussed in Section 4.2.4. Here we present a heuristic for specifying database-level access control rules with regard to the objects in these kinds of rules.

SH_{DLP}1: To specify database-level access control policies, the object in each rule must be an object (e.g., a table, a column, a row or a cell) in the database.

Note the difference between heuristic **IH_{Object}3** and **SH_{DLP}1**. **IH_{Object}3** compels analysts to check the objects identified from requirements (or other sources) against the database design. It mainly serves as a cross checking mechanism. As long as the checking found there is a mapping (or an inconsistency) between the requirements objects and the database design, **IH_{Object}3** has served its purpose. In an application-level access control rule, the object does not have to be an object in the database. However, to specify database-level access control rules, heuristic **SH_{DLP}1** clearly requires that the object must also be an object in the database.

Consider SPRAT requirement FR-GSM-12:

SPRAT FR-GSM-12: *The system shall be able to display the context of a goal.*

The object identified from this example is *the context of a goal*, which maps to a column *context* in database table *goals*. Thus, to specify database-level access control rules, the object should be specified as: *goals.context*.

4.2.2.2 Identifying Subjects and Actions

Step 2.2: *Identify responsible subjects and actions on the object.*

The following heuristics are used to identify the subjects and actions of an access control rule.

IH_{subject/action}1: Given a requirement expressed in the form “The system shall allow <someone> to <do something>”, the subject is <someone> and the action is <do>.

The pattern in this requirement clearly identifies the subject and the action. Consider TDG requirement 2.2.1:

TDG 2.2.1: *The system shall allow border immigration agents (BIA) at point-of-entry (POE) stations to retrieve a traveler's data from the local database.*

Based on heuristic $IH_{\text{subject/action}1}$, the subject in this example is *border immigration agent (BIA)* and the action is *retrieve*. Because not all requirements are stated in the pattern shown in $IH_{\text{subject/action}1}$, the following heuristics identify several other common patterns that were found to be common in the requirements specifications for the four systems analyzed in this dissertation.

$IH_{\text{subject/action}2}$: Given a requirement expressed in the form “The system shall support/provide the ability (be able) to <do something>”, the subject is unclear. The <do> action has two possibilities as follows.

$IH_{\text{subject/action}2.a}$: The <do> action is performed by an subject. In this case, the analyst would need to consult the stakeholders to disambiguate the requirement by asking precisely who will perform this action. By clarifying the requirement, one can rephrase this requirement as “The system shall allow <someone> to <do something>”, resulting in a less ambiguous requirement specification.

Consider SPRAT requirement FR-GSM-6:

SPRAT FR-GSM-6: *The system shall support the ability to update an existing goal.*

This requirement does not explicitly describe who can (or is allowed to) update an existing goal. This is acceptable for requirements engineers during the early stages of specification, but it is too ambiguous for system designers and software developers. Given such ambiguity in this requirement concerning responsible subjects, the analyst needs to clarify precisely who can update an existing goal. The requirement may then be rephrased as follows:

SPRAT FR-GSM-6 Revised: *The system shall allow analysts to update an existing goal.*

Based on heuristic $IH_{\text{subject/action}1}$, the subject in this example is *analyst* and the action is *update*.

IH_{subject/action}2.b: The <do> action is not performed by any subject. Instead, it is an automated action performed by the system. In this case, the analyst should consult the stakeholder(s) to determine *who* will access *what* information if the system performs this action.

Once again, consider SPRAT requirement FR-GSM-12:

SPRAT FR-GSM-12: *The system shall be able to display the context of a goal.*

This is an ambiguous requirement because it is unclear who can request the system to display this information and who can view the information displayed by the system. Upon asking the stakeholder(s) for clarification this requirement can be rephrased as follows:

SPRAT FR-GSM-12 Revised: *The system shall allow analysts to view the context of a goal.*

Following heuristic **IH_{subject/action}1**, the subject in this example is *analyst* and the action is *view*.

IH_{subject/action}3: Given a requirement expressed in the form, “The system shall provide/support/allow/<do> <something>”, neither the subject nor the action is clear in this case. The <something> in this requirement is often a feature or a function. In this case (the requirement is vague and subject/action cannot be easily identified), analysts should employ scenario analysis to elaborate the requirement.

Consider the following SPRAT requirement:

SPRAT FR-PM-3: *The system shall support multi-user analyst results comparison.*

This requirement is so ambiguous that without contextual information and further clarification it is impossible for most analysts to understand what it means. In cases such as this, scenario analysis is an efficient way to elaborate and clarify the requirement. A scenario is comprised of a sequence of events, pre-conditions, post-conditions, obstacles, and any goals or requirements that are associated with this scenario. Similar to the control structures in a computer program, the sequence of events could be organized as normal flow, alternate flow, and iterative

flow. Scenario analysis enabled us to identify the main events (e.g. the normal flow) in this scenario as shown in Table 4.2.

Table 4.2 Scenario Analysis for SPRAT Requirement FR-PM-3

Scenario Name	Project Manager compares goal classification results of several analysts
Pre-conditions	Goals exist in the system; the classification method is defined; and the analysts who have been selected to compare have completed their classifications.
Events	(1) Analysts classify goals independently according to predefined categories. (2) Project managers select analysts whose classification results they wish to compare. (3) The system shall display those goals that are classified differently and how they are different (e.g., by showing the different categories)
Post-conditions	NULL
Obstacles	Analysts did not complete their goal classifications when Project Manager compared their classifications.
Requirements	FR-PM-3: The system shall support multi-user analyst result comparison.
Goals	Get the goals that are classified differently by analysts and how they are different.

Given an elaborated scenario, the main sources from which to identify subjects and actions are a scenario's sequence of events. Heuristics $IH_{\text{subject/action}1}$ and $IH_{\text{subject/action}2}$ can also be applied while analyzing a scenario's events to identify subjects and actions. From the scenario in Table 4.2 one can derive the subjects and actions shown in Table 4.3 below.

Table 4.3 Subjects and Actions Identified from SPRAT Requirement FR-PM-3

Source	Subject	Action
Event 1	Analyst	Classify
Event 2	Project Manager	Compare
Event 3	Project Manager	View

Two kinds of actions can be derived from requirements specifications and identified during scenario analysis: database actions and abstract actions. *Database actions* are direct operations upon a database, such as insert, update, delete, whereas *abstract actions* are high-level operations in contrast to concrete database actions, such as withdraw (money from a banking account), process (a transaction). Database actions are identified to specify database-level access control policies, whereas abstract actions generally map to application-level access control policies. These different levels of policies are discussed in more details in Section 4.2.4. In ReCAPS, both types of actions are identified from source documents and subsequently specified.

Specifically, the following heuristics are used to specify database actions.

SH_{DLP}2: To specify database-level access control rules, the action in the rules must specify a standard database operation (e.g., create, select, insert, update, and delete).

Consider SPRAT requirement FR-SSM-3:

SPRAT FR-SSM-3: *The system shall provide the ability to delete a scenario from a project without deleting it from the system.*

The action word in this example is *delete*, which is a standard database operation. Thus, *delete* is the action in the derived AC rule.

SH_{DLP}3: When an action can be directly mapped to a database operation (e.g., add, enter, read, view, retrieve, remove, change, and edit), map the action to a standard database operation and specify that database operation as the action in the derived AC rule.

Consider TDG requirement 2.2.1 again:

TDG 2.2.1: *The system shall allow border immigration agents (BIA) at point-of-entry (POE) stations to retrieve a traveler's data from the local database.*

The action word *retrieve* is not a standard database operation, but it can be mapped to *select*, which is a standard database operation. Thus, to specify database-level access control rules, we can use *select* as action instead of *retrieve*.

SH_{DLP}4: When the identified action is an abstract action, use scenario analysis to decompose the action into several database operations.

Consider SPRAT requirement FR-GSM-3 again:

SPRAT FR-GSM-3: *The system shall allow analysts to classify goals.*

Classify is an abstract action that needs to be decomposed into several DB actions in order to specify database-level access control rules. As prescribed by heuristic **IH_{subject/action}3**, this action can be elaborated using scenario analysis, as shown in Table 4.4 (i.e., normal flow).

Table 4.4 Scenario Analysis for SPRAT Requirement FR-GSM-3

Scenario Name	Analysts classify goals.
Pre-conditions	Goals exist in the system and the classification method is defined.
Events	(1) Analysts retrieve goals from the database to a local client. (2) Analysts change some attributes of the goals at the local client. (3) Analysts save the changes in the database.
Post-conditions	Goals are classified.
Obstacles	NULL
Requirements	FR-GSM-3: The system shall allow analyst to classify goals.
Goals	Classify goals according to predefined methods.

In this scenario, Event (1) is an obvious *select* DB operation and Event (3) is actually an *update* DB operation. Event (2) is a local action that does not involve DB operations, so there is no need to specify it in access control policies. Thus, the database actions, shown in Table 4.5, can be derived from this example.

Table 4.5 Subjects and Actions Identified from SPRAT Requirement FR-GSM-3

Source	Subject	Action
Event 1	Analyst	Select
Event 2	Analyst	N/A
Event 3	Analyst	Update

4.2.2.3 Identifying Conditions

Step 2.3: Identify conditions under which a subject is allowed to perform an action on an object.

Defining complete and correct conditions for each access control rule is critical for protecting the security and privacy of system resources. If conditions are not defined correctly, things can go wrong and security and privacy breaches may happen. Defining complete conditions for access control policies is the most challenging part of the specification process based on our experience. It is very difficult to evaluate whether the conditions defined for an access control rule are sufficient to protect the data from unauthorized access, because it is very difficult to evaluate whether a set of access control policies is complete or not. Although we cannot guarantee the following heuristics will identify *all* conditions, our studies to date show that their use does ensure better coverage.

There are *explicit* conditions and *implicit* conditions. Explicit conditions are clearly defined as constraints or pre-conditions in requirements specifications, for example, temporal constraints and location constraints. Heuristic **IH_{cond}1** helps capture this kind of conditions.

IH_{cond}1: When a requirement specifies constraints (e.g., if, unless, when, during, before, and after), specify the constraints as a condition in the resulting AC rule.

Consider TDG requirement 2.5.2:

TDG 2.5.2: The system shall allow border immigration agents (BIAs) at point-of-entry (POE) stations to establish a new record on the watch list for a traveler identified as exhibiting suspicious behavior if none currently exists.

In this example, *if none currently exists* is a constraint for border immigration agents inserting a new record on the watch list. This constraint must be specified in the condition element of the derived AC rule.

IH_{cond2}: When a scenario contains pre-conditions, it is helpful to examine these pre-conditions and specify those that fall within the scope of access control as a condition in the resulting AC rule.

Consider the scenario shown in Table 4.4 (see page 91), the pre-condition for this scenario is “Goals exist in the system and the classification method is defined.” The first part of the pre-conditions—“goals exist in the system”—should be specified as a condition for the resulting access control rules.

Defining implicit conditions is a process of exploration and discovery. Implicit conditions are trickier to identify than explicit conditions. We provide the following heuristics that help analysts discover implicit conditions. There are six kinds of constraints that are useful for identifying conditions: authentication constraints, contextual constraints (temporal, location, relationship, affiliation, attribute, and state), usage constraints, database constraints, security constraints, and privacy constraints (purpose, recipient, and consent). Each of these kinds of constraints is defined below.

IH_{cond3}: (Authentication constraints) If access to the requested data requires authentication, then define *this fact* as a condition in the resulting AC rule.

Authentication constraints reflect the need for a subject to be authenticated before data access can be granted. For example, in both the SPRAT and the TDG, all users must be logged in before they can access any data in the system. Thus, *user is logged in* must be specified as a condition for all access control rules.

IH_{cond4}: (Contextual constraints) If the context of an access request plays a role when making grant/deny decisions, these contextual constraints should be specified as conditions in the resulting rule.

Contextual constraints reflect the need to consider the context of an access request when making grant/deny decisions, such as the time of the access request or the location from which the access request is made. The HIPAA security regulation [HIP96] (Section 142.308(c)(1)(i)(B)) requires the use of either (1) user-based access control, (2) role-based access control, or (3) context-based access control. Context-based constraints are often specified in the condition element of access control rules. Beznosov summarizes several kinds of context information that are important to access control in the U.S. healthcare domain [Bez98]. These contexts not only apply to the healthcare domain, they also generally apply to other domains (e.g. e-commerce, finance). There are six kinds of contextual constraints for an access request that are defined in ReCAPS: temporal constraints, location constraints, relationship constraints, affiliation constraints, attribute constraints, and state constraints. Each of these kinds of contextual constraints is defined as a sub-heuristic below.

IH_{cond}4.a: (Temporal constraints) If a data access is restricted by time or date constraints, specify them as conditions in the resulting AC rule.

Temporal constraints specify time or date related restrictions that must be enforced before data access can be granted. For example, in healthcare applications, time constraints are often defined for shift-related positions such as nurses. In another e-commerce example, users of CARFAX (an online company that provides detailed Vehicle History Reports to paid customers) can only access the system for a period of three months.

IH_{cond}4.b: (Location constraints) If a data access is restricted by any location constraints, specify them as conditions in the resulting AC rule.

Location constraints specify a particular location from which the subject can be granted access to a resource. Given the requirement “A doctor cannot access medical records of patients whom he/she is not responsible for *unless* the request comes from the emergency room,” *request*

comes from the emergency room is a location constraint that must be specified in the access control rule.

IH_{cond}4.c: (Relationship constraints) If there are any relationship constraints between the subject and the object, specify them as conditions in the resulting AC rule.

Relationship constraints specify a specific relationship between the subject and the object for an access request. An example relationship that is very important in privacy protection is ownership. For example, in e-commerce systems, when a user logs in to update his/her personal information, he/she should be able to update *only* his/her *own* personal information, even though in the database all users' personal information may be saved in the same table. The relationship between the subject and the object (e.g., the subject is the owner of the requested data) must be specified as a condition in the resulting AC rule. (Note: One may argue that after personal data is submitted to an organization, the organization owns the data instead of the individual that the data is about. This is a valid argument but the discussion is beyond the scope of this dissertation.)

In healthcare applications, the relationship between the subject and the data subject (i.e., the person that the data is about) often needs to be considered when making authorization decisions. For example, if a healthcare staff (e.g., a nurse) is explicitly assigned to take care of a patient, then she may be allowed to access that patient's medical records. Thus, "responsible for" is a relation that must be specified as a condition in the resulting AC rule.

IH_{cond}4.d: (Affiliation constraints) If there are affiliation constraints for the subject, specify them as conditions in the resulting AC rule.

Affiliation constraints specify a subject's corporate, organizational, or group affiliation. For example, if the subject must be affiliated with a group, department, or organization, to access some resources before access is granted, this must be specified as a condition in the resulting AC rule. For example, "all users that are affiliated with the university are allowed to access the

university library” specifies the required affiliation of all subjects that can be granted access to the library. Note that the affiliation constraints are similar to the recipient constraints (see IH_{cond}8.b, page 101) as explained in that section.

IH_{cond}4.e: (Attribute constraints) If there are any attribute constraints to the subject, specify them as conditions in the resulting AC rule.

Attribute constraints specify a subject must possess some attribute (e.g., digital certificates) for an access request to be granted. For example, if the subject is a certified professional by some organization, then he/she is allowed to access some system resources.

IH_{cond}4.f: (State constraints) If access request can only be granted when the system is in a specific state, then specify this state constraint as a condition in the resulting AC rule.

State constraints limit data access based upon the reaching of some specific state within the system. For example, if the current state of a web server (e.g., number of incoming requests per second) has exceeded a certain limit, then any new access request will be denied. Such state limits should be expressed as a condition in the resulting AC rule.

For another example, consider SPRAT requirement FR-PM-3 again:

SPRAT FR-PM-3: The system shall support multi-user analyst results comparison.

As previously mentioned (see Table 4.2), before a *Project Manager* can compare analysts’ goal classification results, analysts must have completed their classifications. This is a state that the system must be in when a *Project Manager* compares goal classification results. Thus, one condition for the AC rule that allows *Project Manager* to compare analysts’ goal classification result is that analysts have completed their classifications. Otherwise, the data required for the comparison would not be available. Note that this constraint can also be derived from the pre-conditions of the scenario described in Table 4.2 (see page 89) using heuristic IH_{cond}2. This suggests that different analysts can reach the same condition using different heuristics.

IH_{cond5}: (Usage constraints) If there are any usage constraints that restrict the subject's access to the object, specify the constraints as conditions in the resulting AC rule.

Usage constraints specify restrictions on how a subject may access the requested object, such as the number of times the subject can access the object. In e-commerce systems, usage-based information often plays a role in access control [PS02, PS04]. For example, a paid user may be allowed to download music ten times. This usage information must be specified in the condition of the corresponding access control rules.

IH_{cond6}: (Database constraints) If the resulting rule is a database-level AC rule, it is helpful to consider whether there are any database constraints for the action. If yes, specify the constraints as conditions in the rule.

Database constraints specify restrictions on a database access request, such as when there can be no duplicate entry in a table. For example, when an analyst tries to insert a new goal in the repository, a condition for this access rule is that *the goal does not exist in the repository*.

IH_{cond7}: (Security constraints) If is helpful to use general security principles such as least privileges and separation of duties to construct misuse cases that a user may use to exploit the capabilities for hostile intent. Corresponding security constraints should thus be specified in the resulting AC rule.

Security constraints specify restrictions that are based on general security principles such as least privileges and separation of duties. These security principles are important to prevent fraud and errors. Again, consider SPRAT requirement FR-PM-3 (see page 88). According to FR-PM-3, the *Project Manager* is allowed to view goal classification results. *Project Manager* and *Analysts* are not mutually exclusive roles, which means a user can assume both roles at the same time. We thus ask “can a *Project Manager* exploit the capability for his/her own good?” and construct the misuse case shown in Figure 4.3. Note that the misuse case diagram does not employ the standard use case diagram. In this misuse case, user A assumes both roles: *Project Manager* and *Analyst*, whereas user B assumes only the *Analyst* role. The attack pattern is shown as Step (1) – (3) in Figure 4.3.

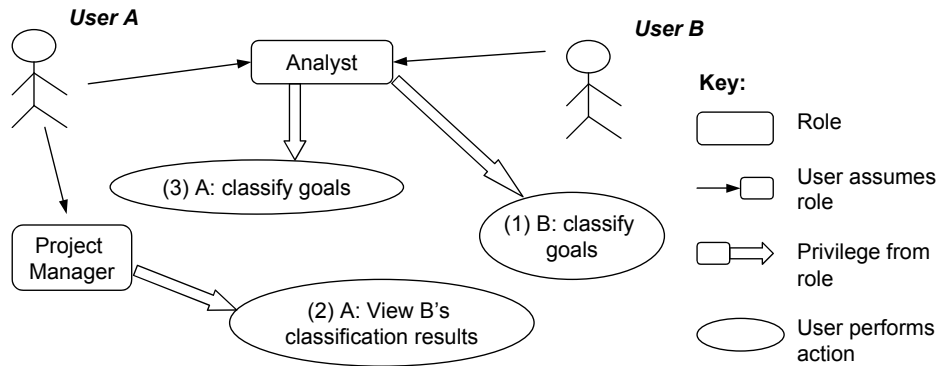


Figure 4.3 A Misuse Case for *Project Manager*

Step 1: *B classifies goals,*

Step 2: *A views B's classification results*

Step 3: *A classifies goals.*

This is not desirable because A would be biased if he/she saw B's classification results before he/she has classified the goals. Misuse case analysis allowed us to specify a deny rule for *Project Manager* to view goal classification results. We document this as Rule 2 in Table 4.6.

Table 4.6 Access Control Rules Derived from the Misuse Case

Policy No	Rule No	Mode	Subject	Action	Object	Condition	Sources
1	1	Allow	Role (Project Manager)	Read	goals.taxonomy	user.loggedIn=TRUE	FR-PM-3
	2	Deny	Role (Project Manager)	Read	goals.taxonomy	Role (user, Analyst) = TRUE AND user.scheduledToClassify = TRUE AND user.classifyingFinished = FALSE	FR-PM-3

IH_{cond}8: (Privacy constraints) If data can only be accessed for a specific purpose, by a certain group of people, or after consent has been obtained, these privacy constraints should be specified as conditions in the resulting AC rule.

Privacy constraints specify restrictions to data access requests in which the data are particularly sensitive (e.g., medical history, financial data). There are three kinds of privacy constraints: purpose constraints, recipient constraints, and retention constraints that are defined in ReCAPS. These constraints correspond to the three privacy elements defined in P3P: purpose, recipient, and consent [P3P05]. Another privacy element *retention* is often specified as obligations in access control rules as discussed in Step 2.4.

IH_{cond}8.a: (Purpose constraints) If specific data can be used for only certain purposes, then specify these purpose constraints as conditions in the resulting AC rule.

Purpose constraints specify that data can be used only for specific purposes. An organization's privacy policy often states that specific types of data will only be used for specific purposes (e.g., telemarketing, payment, research and development). The actual data practices in the organization must always adhere to these purposes. This is the purpose binding principle for privacy [Fis01]. To enforce the purpose binding privacy principle in access control, two kinds of purpose are identified: consumer data purpose and business purpose. Consumer data purpose is consented to by a consumer and recorded by a data collector and expresses how the corresponding collected data can be used. Business purpose is the actual reason for a business task that involves certain consumer data accesses or operations.

Customer consented data purposes are usually high-level and the number of such purposes is limited. According to the Working Draft P3P1.1 Specification [P3P05] released by the World Wide Web Consortium (W3C) on 4 January 2005, there are only 12 purposes defined (see Table 2.1 on page 39). Of course, each organization can define its own vocabulary for data purposes, instead of using the purposes defined in P3P1.1.

Business purposes are defined in each organization according to its business process. They may be defined more specifically than data purposes. For example, the contact purpose may be divided into three categories: phone/fax contact, postal contact, and email contact. However, no matter how business purposes are defined, they must be connected with data purposes. We introduce the purpose hierarchy to support this.

A purpose hierarchy is a structure that organizes purposes and models purpose relations. Purpose relations are partial ordered relations. A partial order is a reflexive, transitive, and anti-symmetric relation. Partial ordered relations support complex purpose hierarchies, such as tree, inverted tree, and lattice structures. We employ the use of a purpose hierarchy to map high-level data purposes to low-level business purposes. If an operation is allowed for a given purpose, it is also allowed for all sub-purposes. Figure 4.4 illustrates a sample hierarchy for the marketing purpose. In this example, email marketing, postal marketing, and phone/fax marketing are sub-purposes of both direct marketing and third-party marketing.

Purpose hierarchy allows unambiguous purpose lookup from business purposes to data purposes. The following is an example of an ambiguous purpose lookup. If a customer consents to have his personal information used only for email marketing purpose, the access decision of an operation (i.e. whether the data access request is granted or denied) with the purpose of direct marketing cannot be determined. This is because email marketing belongs to both the direct marketing and third-party marketing purposes. The system cannot determine its exact parent purpose.

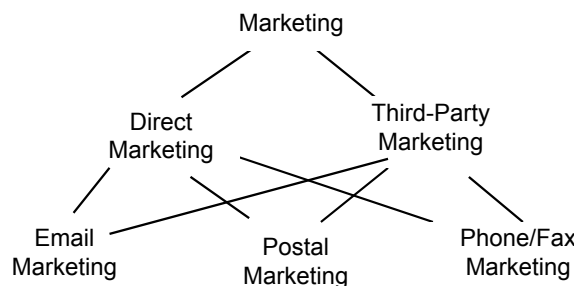


Figure 4.4 Purpose Hierarchy for Marketing

The above problem can be solved by placing restrictions on the purpose hierarchy. We only allow business purposes to be mapped to the lowest level of the purpose hierarchy. The purpose for an operation must be defined as specifically as possible. In this way, data purposes are either in the same level as business purposes or in a higher level. This ensures there are no ambiguous purpose lookups from business purposes to data purposes.

Next, we define the following operator \ll to evaluate the relation between two purposes:

Definition: Given two purposes $p1$ and $p2$, we claim purpose $p2$ contains $p1$ (or purpose $p1$ belongs to $p2$) if, and only if, $p2$ is on the path from the root of the purpose hierarchy down to $p1$ or $p2$ is the same as $p1$, which is represented as $p1 \ll p2$.

Based on the above definition of purpose hierarchy and purpose comparison operator, we can specify purpose constraint in the condition element of an access control rule to enforce purpose binding as follows:

action.BusinessPurpose \ll *object.DataPurpose*

Consider the following example: *A particular piece of data (e.g., credit card information) can be used only for payment (data purpose).* Given the condition in the ACP rule above, a data access request will be evaluated by an enforcement engine; the business purpose will be checked against the requested object's data purpose—payment.

IH_{cond}8.b: (Recipient constraints) If only some recipients can access certain data, then specify recipient constraints as conditions in the resulting AC rule.

Recipient constraints specify which group of people can access the specified data. This heuristic is very similar to the affiliation constraints defined in **IH_{cond}4.d**, which specify the affiliation of a subject (e.g., a group, department, or organization). The difference is that in the context of privacy protection, there is a set of predefined recipients (see Table 2.3 on page 41), whereas the affiliation of a subject may be defined in a specific system.

Similar to the affiliation constraints, recipient constraints may be defined as this: the subject of the AC rule belongs to one of the predefined recipients categories.

IH_{cond}8.c: (Consent constraints) If certain data can be accessed only after consent is obtained, then specify consent constraints as conditions in the resulting AC rule.

Consent constraints require a subject to acknowledge consent for their information to be used in some specific way or purpose before data access can be granted for that purpose. For example, certain medical data can be used for research and development purpose only if written customer consent is obtained. Thus, obtaining customer written consent should be specified as a condition in the resulting AC rule.

The heuristics discussed above provide a general taxonomy of common types of constraints. These heuristics can serve a checklist that helps analysts define conditions for access control rules.

4.2.2.4 Identifying Obligations

Step 2.4: Identify obligations that the subject or system must fulfill if an access request is granted.

As previously stated, obligations are actions that must be fulfilled if an access request to system resources or data is to be granted. Obligations are often triggered by some event. There are two kinds of obligations: security obligations and privacy obligations.

IH_{oblig}1: (Security obligations) If an auditing trail must be recorded upon access to certain data, recording auditing trails must be specified as an obligation in the resulting AC rules.

Security obligations define actions that must be taken after an access request is granted for the sake of additional security, such as accountability. For some classified information, the accountability of information access is very important. Thus, every access to this information

must be logged in the auditing trails. In other cases, only certain kinds of accesses (e.g., update and delete) are important and should be recorded. Thus, logging data accesses in these cases should be specified as obligations for the resulting AC rules.

IH_{oblig}2: (Privacy obligations) It is helpful to evaluate whether there are any privacy obligations in the resulting AC rules.

Privacy obligations specify actions that must be fulfilled if an access request is granted when the data is sensitive and therefore requiring special protections. There are several sub-heuristics to aid analysts in identifying privacy obligations:

IH_{oblig}2.a: If the subject is required to update/delete the data after access, specify updating/deleting data as obligations in the resulting AC rule.

Consider the following requirement: *require affiliates to destroy customer data after 30 days*. In this example, *deleting customer data* is a privacy obligation that must be specified in any corresponding AC rules.

IH_{oblig}2.b: If the subject is required to manipulate the data after access, such as encrypting/decrypting and anonymizing, specify the manipulation action as obligations in the resulting AC rule.

In some applications, data must be encrypted from any access after being used for a certain number of times in a specific timeframe. Thus, encryption is a privacy obligation for the corresponding AC rules.

IH_{oblig}2.c: If the subject is required to notify data subjects after access, specify notification as obligations in the resulting AC rule.

For example, *send a report of customer data access logs to customers every month* may be a privacy obligation for some AC rules.

IH_{oblig}2.d: If the subject is required to obtain consent or authorization after access, specify obtaining consent or authorization as obligations in the resulting AC rule.

In some cases, data may be used for some purpose if consent can be obtained within a certain period of time. This is different from the consent constraints discussed in heuristic **IH_{cond}8.b**. In a consent constraint, consent must be obtained *before* a data access request can be granted, whereas in a consent obligation, a data access request can be granted and consent can be obtained *later*.

4.2.2.5 Summary

This section provided a set of heuristics to help analysts identify AC elements from source documents and specify AC rules. It is important to document the source requirement from which an AC rule was derived because traceability is a primary design principle for this approach. With tool support, the traceability between requirements and access control policies are maintained automatically.

During Step 2, analysts produce a set of candidate AC rules that need to be refined in Step 3.

4.2.3 Access Control Rule Refinement

The access control rules derived from different sources in Step 2 may be redundant or conflict with one another. Thus, these AC rules must be reconciled. Any redundancy must be removed and all conflicts must be resolved.

Step 3: Refine access control rules.

Analysts conduct the following analysis for each subject to refine AC rules.

Step 3.1: Sort the AC rules according to subjects and objects.

Grouping rules according to objects allows analysts to identify redundant and conflicting rules more easily, as described in Steps 3.2 and 3.3.

4.2.3.1 Identifying and Removing Redundancies

Step 3.2: Merge redundant AC rules and create new rules, if necessary.

RH_{redundancy1}: If one rule subsumes another rule, then the second rule is redundant and should be removed.

Consider the following SPRAT requirements:

SPRAT FR-GSM-12: *The system shall be able to display the context of a goal.*

SPRAT FR-GSM-16: *The system shall allow analysts to view elements of a goal returned by a query.*

From both requirements, the following rules can be derived:

Rule 1: *Allow an Analyst to view the context of a goal.*

Rule 2: *Allow an Analyst to view the elements of a goal.*

Because context is a goal element, Rule 2 actually subsumes Rule 1. Thus, Rule 1 is redundant and should be removed.

RH_{redundancy2}: Two logically equivalent rules that are specified in different modes (i.e., one *allow* rule and one *deny* rule) are redundant. One of them must be removed.

This kind of redundancy stems from the mode of an access control rule because an *allow* rule can be specified as a *deny* rule if the conditions are reversed and vice versa. Consider the following two rules:

Rule 1: *allow A to do B if condition C is satisfied*

Rule 2: *deny A to do B if condition C is **not** satisfied.*

These two rules are logically equivalent and thus redundant. Only one is necessary. As a general rule, analysts should keep the rule that is specified from the positive perspective (i.e., mode allow) and remove the deny rule. This is because by default an enforcement engine will deny an access request if no rule is matched during the checking. Additionally, the allow rule contains important access information, If deleted, valid access request may be denied (see example on page 147). Sometimes combining both modes makes it easier to specify an access control policy (see **RH_{conflict}2**, page 108). Note Rule 1 and Rule 2 are not always equivalent. In the security community, researchers have proposed provisional authorizations [JKS00], in which not satisfying a condition does not necessarily lead to the deny of an access request. In this dissertation, we do not consider this special case.

RH_{redundancy}3: If two rules have different (not conflicting) conditions but the same other elements, then combine the two conditions in one rule and remove the other rule.

In this case, combining the conditions means the conditions in both rules must be satisfied.

For example, in the SPRAT, there are two rules:

Rule 1: *Allow an Analyst to view goals in a project if the Analyst is logged in.*

Rule 2: *Allow an Analyst to view goals in a project if the Analyst is assigned to this project.*

Analysts can remove the redundancy by combining the two conditions:

Rule 3: *Allow an Analyst to view goals in a project if the Analyst is logged in and the Analyst is assigned to this project.*

RH_{redundancy}4: Design decisions often introduce new rules and cause some candidate rules to be redundant. Redundant rules must be removed.

For example, in the SPRAT, from the SRS the following three rules for role *System Administrator* can be derived:

Rule 1: Allow System Administrator to create Project Managers.

Rule 2: Allow System Administrator to create Analysts.

Rule 3: Allow System Administrator to create Guests.

Because *Project Managers*, *Analysts* and *Guests* are roles in RBAC, we decided to employ RBAC to control users' access to data. Under this decision, two new AC rules for role *System Administrator* can be created:

Rule 4: Allow System Administrator to create user accounts.

Rule 5: Allow System Administrator to assign roles to users.

In this case, rules 1–3 become redundant and need to be removed. In this way, the specified rules are consistent with the design decision and more flexible than the candidate rules.

4.2.3.2 Identifying and Resolving Conflicts

Step 3.3: Reconcile any conflicting AC rules.

If the requirements or high-level policies from which AC rules were derived are in conflict with one another, the resulting AC rules may conflict as well. Thus, any conflicts must be resolved at this stage of the analysis. There are three kinds of conflicts that arise from the mode, object and condition of AC rules.

RH_{conflict}1: If two rules have the same elements but have conflicting modes, then this is a modality conflict.

Resolution of this kind of conflicts needs to go back to the sources from which these rules were derived.

In one rule the subject is allowed to perform an action on an object, while in another rule, the subject is denied from performing the same action on the same object. This type of conflicts results from sources and can be resolved by removing one of the rules that is not true. Before resolving the conflict in the rules, analysts need to resolve the conflicts in the sources. Sometimes they need to consult stakeholders to decide which is correct and which is not.

RH_{conflict}2: If two rules define conflicting privileges for the same subject on two objects, in which one object subsumes the other object, then this is a partial conflict. There are two ways to resolve this kind of conflicts: (1) specify separate rules for each part of the object; or (2) group rules into an access control policy, in which every rule must be evaluated and satisfied for an access request to be granted.

For example, in the SPRAT project, classification is an attribute of goals. However, classification and goals have different access permissions for Guests as shown below:

Rule 1: *Allow Guest to view goals.*

Rule 2: *Deny Guest to view classification of goals.*

Analysts can resolve this conflict in two ways.

(1) Divide Rule 1 into several separate policies as follows:

Rule 1: *Allow Guest to view elements of goals other than classification.*

Rule 2: *Deny Guest to view classification of goals.*

(2) Group both rules into an access control policy. Both rules in the policy must be evaluated during enforcement. Table 4.7 shows two rules that are grouped together to resolve such kind of conflicts.

Table 4.7 Grouping Access Control Rules into a Policy to Resolve Conflicts

Policy No	Rule No	Mode	Subject	Action	Object	Condition	Sources
1	1	Allow	Role (Guest)	Read	Goals	user.loggedIn=TRUE	FR-UA-4
	2	Deny	Role (Guest)	Read	goals.taxonomy	user.loggedIn=TRUE	FR-UA-4

RH_{conflict}3: If two rules have conflicting conditions but the same other elements, then this is a conditional conflict. Resolution of this kind of conflicts needs to go back to the sources from which these rules were derived.

In this case, analysts cannot simply combine the conditions because the conditions conflict with each other. They have to go back to the sources from which the rules were derived. Sometimes they need to consult stakeholders to decide which is correct, which is not. Only one rule can exist.

Conflict identification is simplified by analyzing the grouped policies; however, the ReCAPS method only detects explicit conflicts among the rules. More rigorous conflicts analysis is beyond the scope of this dissertation.

4.2.4 Grouping AC Rules into ACPs

After the previous ACR identification, specification, and refinement steps, analysts have produced a set of access control rules. Recall that an access control *policy* is comprised of a set of access control *rules*. Upon completion of Step 3, all AC rules must be grouped into policies. The purpose of grouping rules into policies is to facilitate the implementation of rule enforcement. Thus, grouping is based on the decision on how to implement these AC rules in software design.

Access control can be implemented at different levels, e.g., database-level or application-level. The rules/policies listed in Table 4.6 and Table 4.7 are database-level rules/policies; each action in an AC rule is a DB operation and each object is an object in the DB. ReCAPS allows software engineers to specify database-level and application-level access control policies, which are called DLP and ALP respectively. This section presents ways to group AC rules into policies so that these policies can be implemented at application level.

Step 4: Group logically connected AC rules into ACPs.

A set of rules is said to be logically connected if the rules have the same subject, action, object, or any two of these. Thus, there are six possible ways in which AC rules may be grouped:

- Group all AC rules that have the same *subject* into a policy;
- Group all AC rules that have the same *action* into a policy;
- Group all AC rules that have the same *object* into a policy;
- Group all AC rules that have the same *subject* and *object* into a policy;
- Group all AC rules that have the same *subject* and *action* into a policy; and
- Group all AC rules that have the same *action* and *object* into a policy.

It is important to note that it is possible that a rule may be grouped into several policies according to different grouping methods. For example, analysts can group all AC rules for a subject's privileges to access an object into a policy according to the fourth grouping method. Alternatively, analysts could group all AC rules for this subject into an ACP according to the first grouping method. In this case, some rules may belong to two policies. It is more often the case that AC rules are grouped according to the fourth method as prescribed by **GH_{grouping}1**.

GH_{grouping}1: To specify application-level policies, it is often useful to group all the rules that concern a subject's privileges to an object into one policy.

The rationale behind this heuristic is that all privileges for a subject to access an object often are often implemented in the same function. For example, in the SPRAT, there are several AC rules that are defined for role *System Administrator*, concerned with user account management, as shown in Table 4.8.

Table 4.8 ACPs for Role (System Admin) in the SPRAT Case Study

Policy No	Rule No	Mode	Subject	Action	Object	Condition	Sources
1	1	Allow	Role (System Admin)	Insert	Users	user.LoggedIn=true and the username does not exist in the database	FR-UA 1
	2	Allow	Role (System Admin)	Select	Users	User.LoggedIn=true and the username exists in the database	FR-UA 1
	3	Deny	Role (System Admin)	Select	users.pass word	NULL	FR-UA 1
	4	Allow	Role (System Admin)	Update	Users	user.LoggedIn=true and the username exists in the database	FR-UA 1
	5	Allow	Role (System Admin)	Delete	Users	user.LoggedIn=true and the username exists in the database	FR-UA 1
	6	Deny	Role (System Admin)	Delete	Users	Current login user's username == users.username	FR-UA 1
					

In the actual system, user account management is implemented in a single Java class (i.e., *UserMgt* class). Thus, the rules shown in Table 4.12 can be implemented at application-level by controlling users' access to the *UserMgt* class. Thus, these rules can be refined into the following access control policy:

ALP 1: *Allow a user to access Java class UserMgt.java if Role(user) = System Administrator AND user.loggedIn = TRUE*

This policy results in fewer conditions to be checked during run time, making them more efficient and easier to implement. Basically, only user's role and login status need to be checked during enforcement. The other aspects of access control (e.g., System Administrators cannot view

users' passwords (rule #3), System Administrators cannot remove themselves (rule #6)) are encapsulated within the *UserMgt* class.

GH_{grouping}2: If a set of rules are grouped together to resolve conflicts (as described in Step 3.3), these rules must always be grouped together.

As shown in Table 4.7, rules #1 and #2 are grouped together to resolve the partial conflict. Both rules in the policy must be evaluated during enforcement. No matter which policy rules #1 is grouped into, rule #2 must always be grouped in that same policy.

After Step 4, analysts produce a set of ACPs, which are the final outputs of the ReCAPS method.

4.2.5 Summary

In this section, we detailed the ReCAPS analysis steps and heuristics. Analysts start from requirements specifications and other available source documents. They go through the identification, specification, refinement, and grouping steps and produce a set of ACPs at the end of the analysis. The identification step involves requirements analysis, such as scenario analysis; and the grouping step involves design analysis. This demonstrates how the ReCAPS method bridges the gap between requirements analysis and software design.

Analysts often make design decisions during the entire ACP specification process. For example, the SPRAT SRS clearly states that the system shall support four system access levels: administrator, project manager, analyst and guest. Given that access to the system is restricted to these four levels (or roles), RBAC seems suitable for implementing access control. In our analyses, we made additional design decisions. For example, instead of building a role hierarchy, we ensured that the privileges assigned to each role never overlap. At which level to implement access control is another design decision made during the analysis. Obviously, design decisions

have an impact on how ACPs are specified. It is important to document these design decisions as well when performing the analysis.

The main artifacts produced by analysts during the ACP specification process are a set of ACPs, documented design decisions, as well as augmented software documentation.

4.3 Tool Support

Part of the research involved the development of a software tool, the Security and Privacy Requirements Analysis Tool (SPRAT), to support the ReCAPS method. The tool is presented in this section as an enabling technology rather than a major contribution of the thesis. The author of this dissertation was the chief architect for the tool and the developer for the ACP specification module that supports ReCAPS. He also co-authored the software requirements specification (SRS) document and co-designed the database.

4.3.1 Overview

As mentioned in Section 3.1, SPRAT supports goal-based and scenario-based requirements analysis and provides support for analyzing and specifying security and privacy requirements, policies, as well as ACPs. It builds upon and extends two existing tools, the Privacy Goal Management Tool (PGMT) [AEB04], which was previously developed by the author of this dissertation independently, and the Scenario Management and Requirements Tool (SMaRT) [SAA03].

Framework

The underlying idea behind SPRAT is to provide an integrated framework to support goal analysis, scenario analysis, requirements analysis and specification, policy analysis and

specification in a single project and support the traceability from one artifact to another (e.g., from an access control policy to a requirement, from a scenario to a goal). Figure 4.5 illustrates the elements that are maintained in the SPRAT.

In a project, analysts start from various available source documents, such as customer interview transcripts, existing diagrams, textual statements of needs, and high-level security / privacy policies, to derive goals. Then from these goals, they can use scenario analysis to elaborate goals and specify requirements based on goal and scenario analysis. Analysts can specify access control policies from requirements and other source documents as discussed in this dissertation. During this specification process, they can still employ goal and scenario analysis to help elaborate requirements and identify access control elements. The arrows in Figure 4.5 show the traceability links across software artifacts.

Essentially, SPRAT is not limited to security and privacy requirements analysis. It can also be used as a generic requirements and policy analysis and specification tool.

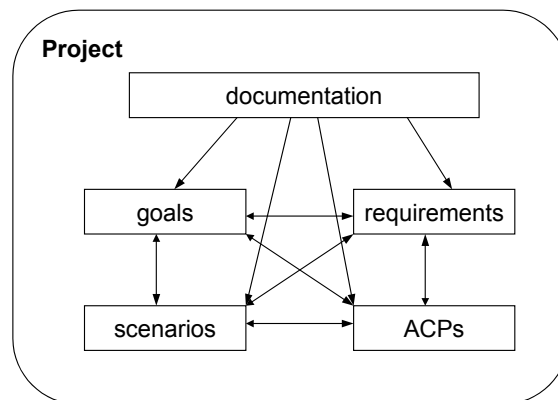


Figure 4.5 Elements Maintained in the SPRAT

Users

It is assumed that the typical SPRAT user will be an experienced requirements engineer with a considerable understanding of goal-based analysis techniques, scenario-based analysis

techniques, and the ReCAPS method. It is also assumed that the typical SPRAT user will have a considerable understanding of basic security concepts, especially in access control.

Potential users of the tool include, but not limited to, university lecturers and students, researchers in software engineering and security, software engineers, requirements engineers, security engineers, security administrators, security and privacy officers, and government law enforcement officers.

Modules and Features

The main modules in the SPRAT are as follows:

- User management
- Document management
- Goal analysis and management
- Scenario analysis and management
- Requirements analysis, specification and management
- Access control analysis and policy specification

The tool comes with a default administrator account, which can be used to create other accounts. In the user management module, administrators can create new user accounts, reset user passwords, remove existing users, and assign roles to users. Users can update their personal information, such as passwords. Project managers can assign/revoke analysts to/from a project that they are responsible for.

In the document management module, analysts can import new documents or input new documents to the repository, and specify the attributes of the document, such as version, date, description, and domain. They can also remove documents from the repository.

In the goal analysis and management module, analysts can create new goals from other sources, update and delete existing goals. Analysts can also view all the goals derived from a certain source document in a project, such as a privacy policy. Analysts can classify goals

according to predefined classification methods. They can also create new classification methods. Additionally, analysts can define the keywords used to specify goals.

In the scenario analysis and management module, analysts can create new scenarios from other sources, update and delete existing scenarios. Analysts can view all the scenarios derived from a certain source document or all the scenarios associated with a goal/requirement. Analysts can specify normal flows, alternate flows, and iterative flows of events in a scenario.

In the requirements analysis, analysis and specification module, analysts can create new requirements, update and delete existing requirements. Analysts can view all the requirements in a project, or all other artifacts that are associated with a specific requirement (e.g., all the goals, scenarios, and ACPs associated with a specific requirement).

In the access control analysis and policy specification module, analysts can derive access control rules from various source documents. They can create new access control elements (i.e., subjects, actions, objects, conditions, obligations), edit and delete existing elements. Analysts can view all the rules derived in a project and refine these rule into policies.

Traceability is a prominent feature of the SPRAT. From a certain artifact (e.g., a requirement), analysts can easily trace all the other artifacts (e.g., goals, scenarios, ACPs) that are associated with this artifact.

4.3.2 ACP Specification Module

The ACP specification module was developed independently by the author of this dissertation to support the two main activities described in ReCAPS: ACR identification and ACR refinement.

ACR Identification

In the ACR identification interface, analysts can choose which source document to work on: requirements specifications, or high-level security and privacy policies. Then the system will import the source statements one by one. Analysts may derive one or more AC rules from each source statement. Figure 4.6 shows the interface for analysts to create new AC rules.

Our design principle for the SPRAT is to reuse all artifacts as much as possible. Thus, when analysts are creating a new AC rule, they will specify each element for this rule. In our current design, analysts will be able to select from a list of existing instances for each element (e.g., subject, action, and object, see drop-down boxes in Figure 4.6). Only when analysts cannot find the appropriate instance from the list, they need to create a new instance for that element. As the reusability increases, the efficiency of the identification process improves. Figure 4.7 shows the interface for creating a new instance of subject. Each subject is either an agent, a role, or a group.

The screenshot shows the 'Create New Access Control Rules' window. At the top, there are navigation buttons '<' and '>', a requirement identifier 'FR17', and a 'View Detail' button. The requirement text is 'The system will allow administrators to modify registrant accounts.' Below this, there are several fields with drop-down menus: 'Mode' (set to 'Authorization: Allow'), 'Subject' (set to 'Role' with a sub-field 'registrant'), 'Action' (set to 'Database Ac...' with a sub-field 'select'), 'Object' (set to 'panelinfo'), 'Condition' (set to 'NULL'), and 'Obligation' (set to 'NULL'). At the bottom, there are four buttons: 'Submit', 'Cancel', 'New...', and 'Edit...'. Below these buttons is a table of existing rules.

Rule ID	Mode	Subject	Action	Object	Condition
43	Authorization: Deny	Role: administrator	Database Action: update	registrants.use...	registrants exi...
46	Authorization: Allow	Role: administrator	Database Action: update	registrants	registrants exi...
85	Authorization: Deny	Role: administrator	Database Action: update	users	

Figure 4.6 SPRAT Screen Shot for ACR Identification



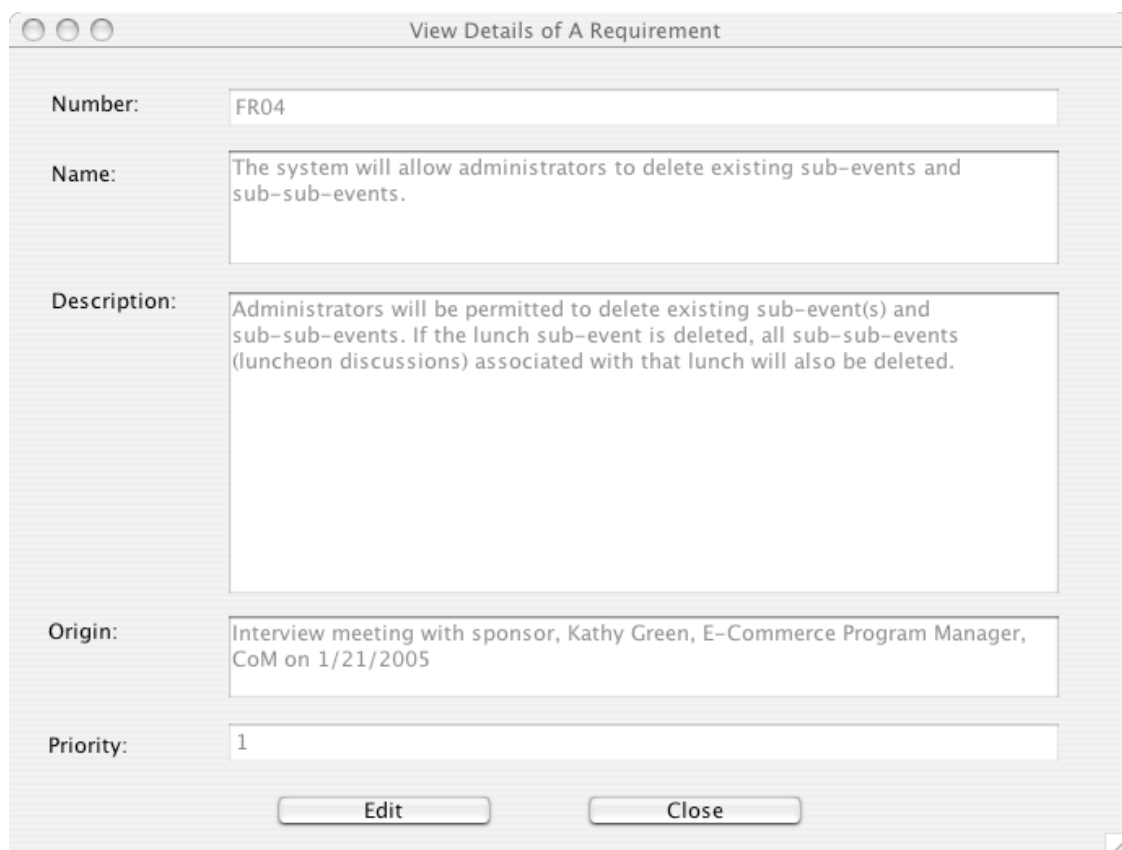
Create A New Subject

Subject: registrant

Type: Role

Submit Clear Cancel

Figure 4.7 SPRAT Screen Shot for Creating a New Subject



View Details of A Requirement

Number: FR04

Name: The system will allow administrators to delete existing sub-events and sub-sub-events.

Description: Administrators will be permitted to delete existing sub-event(s) and sub-sub-events. If the lunch sub-event is deleted, all sub-sub-events (luncheon discussions) associated with that lunch will also be deleted.

Origin: Interview meeting with sponsor, Kathy Green, E-Commerce Program Manager, CoM on 1/21/2005

Priority: 1

Edit Close

Figure 4.8 SPRAT Screen Shot for Viewing Source Detail

Requirements **FR04** [View Detail](#)

The system will allow administrators to delete existing sub-events and sub-sub-events.

Mode: Authorization: Allow

Subject: Role: administrator

Action: Database Action: delete

Object: lunchinfo

Condi... symposium exists AND
 lunchinfo exists

Obligati... delete all lunchinteresttopic entries

[Edit Rule](#) [Edit Element ...](#) [Close](#)

Figure 4.10 SPRAT Screen Shot for Viewing ACRs

Requirements FR10 [View Detail](#)

The system will allow administrators to modify existing sub-events and sub-sub-events.

Mode:

Subject:

Action:

Object:

Condition:

Obligation:

[Submit](#) [Cancel](#) [New...](#) [Edit...](#)

Figure 4.11 SPRAT Screen Shot for Editing ACRs

Type:

[Submit](#) [Clear](#) [Cancel](#)

Figure 4.12 SPRAT Screen Shot for Editing an Action

Each time analysts creates a rule, a traceability link is automatically created between the rule and the source requirement (or statement) from which the rule was derived. This is very helpful because it reduces the effort of documenting sources of AC rules. The left and right arrows at the top-left corner in Figure 4.6 allow analysts to navigate between requirements or source statement. All the rules that have already been derived from this source are listed in the table in Figure 4.6. If an analyst quits the tool in the middle of the analysis process, the tool remembers the current requirement or statement that the analyst is working on. Upon subsequent uses of the tool, the analyst can perform access control analysis on the same document as during the previous session because the tool automatically directs him/her to the requirement or statement at which he/she stopped during the previous session. Additionally, analysts can click the “View Detail” button on the top-right corner to view detailed information about the current requirement or source document. From there analysts are able to trace to the other artifacts that are associated with this requirement. Figure 4.8 shows an example of viewing details about the source. Additionally, if the requirement is ambiguous, analysts may invoke scenario analysis module to author one or more scenarios to elaborate the requirement. The integrative feature of the tool can be best demonstrated in this way.

ACR Refinement

In the ACR refinement interface, analysts are able to view all the rules derived from source documents in a project. They can sort the rules according to subject, action, object, and source. By sorting rules, it is easier to identify redundant rules and conflicting rules. Figure 4.9 shows the ACR refinement interface, which uses a tabular interface to list all the access control rules identified in the current project. Each rule contains a rule ID, mode, subject, action, object, condition, obligation, and source. The rules may be grouped into one or more policies as shown in the first column in Figure 4.9. Analysts can double click on a rule to view the details of this rule, as shown in Figure 4.10. Analysts can delete redundant rules or edit any selected rule. Figure

4.11 shows the interface of editing an existing rule. Analysts can edit also edit every element for a rule as shown in Figure 4.12. Note that changing an element will affect all the rules that contains this element. Additionally, analyst can group several AC rules into a policy or ungroup the rules in a policy. Unfortunately, the tool does not provide automated redundancy identification or conflicts identification. The identification and resolution is done manually by analysts.

4.3.3 Design and Implementation

The SPRAT was developed as a standalone tool using pure Java language to support multiple platforms. The development platform was Eclipse 3.0, an open source software development tool.

SPRAT employs the traditional client/server architecture. All software artifacts, such as goals, scenarios, requirements, ACPs are stored in a project repository using MySQL database. SPRAT client connects to the database via JDBC connection driver. Clients can use a centralized database server hosted at North Carolina State University to save their project artifacts. They also have an option to save project artifacts in a local MySQL database.

There are several reasons that we chose database storage scheme instead of a local files storage scheme.

First, centralized database storage allows easier team collaboration. Every software project is developed by a team, not an individual. Thus, collaboration support is very important for a CASE tool such as SPRAT. A centralized database can ensure the integrity of data. In a local storage scheme, there is a huge problem of data synchronization across different local copies.

Second, project repository has better security and privacy protection in a database. In addition to the authentication mechanism provided by the tool itself, the database management system provides additional mechanisms to protect the project data, which is often proprietary. These mechanisms include access control, encryption, and backup and recovery. If the project is

using local files storage, the only available mechanism to protect data is encryption, which obviously increases the overhead significantly.

Third, the database storage scheme is more scalable and efficient than the local files storage scheme. As mentioned previously, analysts often need to trace across multiple project artifacts. This joint search can be easily achieved in a database system using standard SQL queries. However, it is very inefficient to do a search across several files, especially when the file size is large.

The limitation of the database storage scheme is that the system requires the installation and management of a database during deployment. This causes extra cost in hardware, software, and human resources. However, given the fact that most computers nowadays are connected to the Internet, this is not a problem. Clients can choose to use the centralized database server hosted at North Carolina State University, instead of installing a local database server.

4.4 Summary

This chapter details the Requirements-based Access Control Analysis and Policy Specification (ReCAPS) method in four steps, coupled with heuristics and examples in each step. It is worth noting that some examples are not from the two formative studies due to the limited scope of both studies—for example, the heuristics for defining context constraints and the heuristics for identifying obligations in Section 4.2.2. Next chapter discusses the validation of the ReCAPS process and heuristics via its application to two summative case studies and through utilization of the method in an empirical evaluation.

Chapter 5

Validation

实践是检验真理的唯一标准。

Practice is the criterion of truth.

—Karl Marx

What distinguishes a good software engineering method from other software engineering methods is whether or not it has been validated on real systems. Validation is required before the effectiveness and usefulness of a method may be determined. Software methods, such as the Requirements-based Access Control Analysis and Policy Specification (ReCAPS) method, need early validation while under development. The formative case studies, discussed in Chapter 3, offer early and preliminary validation due to their central role in shaping the ReCAPS method. In contrast, the summative case studies discussed in this chapter seek to validate the method developed during the formative case studies. It is important to note that all four case studies (two formative and two summative) involve operational systems, not constructed illustrations. The two summative validation case studies discussed in this chapter are:

- the Surry Arts Council (SAC) Web enhancement project; and
- the NCSU College of Management (CoM) Event Registration System.

Additional validation in the form of an empirical evaluation, involving software engineering graduate students at NCSU, and development of tool support were completed and are also discussed in this Chapter. The ReCAPS software tool was employed during the NCSU CoM Event Registration System case study, providing an additional level of validation.

It is important to clarify the measurement mechanism for the ReCAPS method in order to appropriately frame the evaluation efforts discussed in this chapter. Basili's Goal-Question-

Metric (GQM) Paradigm [Bas92, BCR94, SB99] is a proven software measurement approach and endorsed by the Department of Defense and the Software Engineering Institute (SEI) in their Practical Software and Systems Measurement workshop. This chapter employs the GQM paradigm to discuss the evaluation efforts.

Our goals for the ReCAPS method are as follows:

- Provide prescriptive guidance on ACP specification for information systems;
- Specify better quality ACPs using ReCAPS than without method guidance;
- Provide traceability support between ACPs and other software artifacts; and
- Improve source documents by clarifying ambiguities and resolve inconsistencies across source documents.

To determine whether or not the ReCAPS method meets the above goals, we ask the following questions:

- Is there any existing method that provides prescriptive guidance on ACP specification for information systems?
- What is the state-of-the-art for ACP specification in practice?
- How can we measure whether a method provides *prescriptive* guidance?
- What are the quality criteria for a set of ACPs?
- Is there any existing ACP specification method that provides traceability support between ACPs and other software artifacts?
- How are the source documents improved as a result of the analysis?

Using the GQM paradigm, the next step is to define the metrics that need to be collected during the evaluation studies in order to answer the above questions. The first two questions and the fifth question were answered in Chapter 2 by surveying the literature. To answer the third question (How can we measure whether a method provides *prescriptive* guidance?), we collect the following qualitative and quantitative evidence from the case studies and the empirical experiment:

- How do analysts feel about the analysis process and heuristics in the summative case studies? This is collected via the lessons learned from each case study.
- How do analysts who are previously unfamiliar with the ReCAPS method feel about the analysis process? This is collected via feedback from the empirical study participants.
- Do analysts who are previously unfamiliar with the ReCAPS method think the heuristics useful for ACP specification? This is collected via feedback from the empirical study participants.
- Are the heuristics reusable? This is measured by the number of times each heuristic was applied in each case study.

The fourth question (What are the quality criteria for a set of ACPs?) is answered and discussed in detail in Section 5.3.2 (see page 157). The sixth question (How are the source documents improved as a result of the analysis?) is measured using the number of inconsistencies identified between the requirements specifications and the database design in each case study.

This chapter discusses the Requirements-based Access Control Analysis and Policy Specification (ReCAPS) method in the context of the two summative case studies and the empirical evaluation, each of which seeks to validate the method. Sections 5.1 and 5.2 present the two summative case studies, respectively. Section 5.3 presents the empirical evaluation. Our discussion is based on the GQM paradigm.

5.1 Surry Arts Council (SAC) Web Enhancement

The objective of this case study was to formally validate the efficacy of the ReCAPS method when used to specify access control policies for a web-based application.

The Surry Arts Council (SAC) is a non-profit organization in Mount Airy, North Carolina that provides opportunities for the local community to experience high-quality performances and instruction in the performing and visual arts. Surryarts.org is the Web portal for disseminating

information about the Council, such as information about arts events in the community. The SAC contracted a group of students at North Carolina State University (NCSU) to enhance its website with e-commerce functionality so that customers can purchase tickets, memberships and other items online.

The SAC Web enhancement project (a.k.a., the SAC project) is a standard web-based application, in which security and privacy are critical features. Customer personal information (e.g., contact information, payment information, and purchase history) is considered sensitive data that must be protected from unauthorized access. Thus, access control is important in this project. Additionally, the SAC project is different from the two formative case studies in its nature. The SPRAT project is a standalone software tool with limited scope in access control; and the TDG project is a research prototype (rather than a fully functional system) that is being deployed for preliminary use; whereas the SAC project is a real, deployed application in which real customers' data security and privacy are vulnerable. This characteristic makes the SAC project an ideal case for validating the ReCAPS method.

5.1.1 Methodology and Case Study Artifacts

The SAC case study was conducted by one analyst, a PhD student, for approximately six person-hours. The task of the case study was to derive database-level access control policies from four available source documents for the SAC web enhancement project. The four source documents used in this case study were:

- SAC Web Enhancement Software Requirements Specification (SRS), Version 1.9
- SAC Web Enhancement design document, Version 1.2 (including database schema design)
- SAC Security Policy, Version 1.9
- SAC Privacy Policy, Version 1.4

In the two formative case studies the available documentation was limited. In contrast, three additional kinds of source documents were available for use in this case study: design document, security policy and privacy policy. This allowed us to validate the generality of the ReCAPS method and the associated heuristics. By generality, we mean the applicability of the method and associated heuristics within the context of broader and different kinds of software documentation.

The analyst in this study participated in the two formative case studies; thus, he was familiar with the methodology—no time was need for initial methodology training. The analyst commenced this case study by first focusing on the identification process. Similar to the previous two formative case studies, the SAC case study was conducted manually (without software support) and everything was documented on paper. For example, during the ACR identification step, the analyst marked the elements on the SRS printouts using color coding and then documented this information in an access control matrix in preparation for refinement in the next step. Additionally, any observations or recurring questions were documented.

5.1.2 Lessons Learned

This section summarizes the lessons learned from the SAC case study. In Chapter 3, the presentation of each case study also includes a discussion of the lessons learned. The difference between the lessons learned outlined in this chapter and the lessons learned presented in Chapter 3 is that the lessons learned during summative case studies are oriented much more toward validation than development. Thus, the following discussion seeks to provide examples that confirm and augment the ReCAPS method presented in Chapter 4.

Understanding the scope of access control is important to ACP specification.

Heuristics **IH_{scope}1-3** have proven very useful in terms of identifying whether any access control rules may be derived from available sources. In the SAC case study, all three heuristics

were applied frequently. For example, we cannot derive any access control rules from the following SAC requirements:

***SAC C 1.1:** Surryarts.org is now hosted by ADVI, a local hosting company in Mount Airy. The new site will operate on a new host that supports the new features.*

Heuristic **IH_{scope}1** applies to this requirement. This requirement defines some external environment settings that are not concerned with user access to sensitive information in the system. No access control rules were derived from this requirement.

***SAC C 1.2:** Surry Arts Council personnel can only do credit card processing manually.*

Heuristic **IH_{scope}2** applies here. Processing credit card transactions manually is not enforceable by the system. Thus, no access control rules were derived from this requirement.

***SAC NFR 1.1:** The Surry Arts Council logo will be clearly visible on surryarts.org.*

Heuristic **IH_{scope}3** applies here. This non-functional requirement defines the visibility feature of the system. No access control rules were derived here, either.

Mapping objects identified in source documents to the database design yields missing items or inconsistencies between the requirements and the database design.

Objects may be identified using heuristics **IH_{object}1-2**. Objects identified from source documents may be ambiguous and must be mapped to an object in the database. Heuristic **IH_{object}3** helps analysts clearly define the objects. This checking process yields several missing items and inconsistencies between the requirements and the database design in the SAC case study. To list a few of them:

- FR 1.3 describes that the system shall allow clients to purchase memberships on the Surry Arts Council website (surryarts.org). However, in the *useracct* table of the database design, there is no field that reflects whether or not a user is a member.

- In the Order/Payment Module, the design document states that if the user is not logged in at a certain point, the system will ask the user to log in, create an account or purchase the item without an account. However, in the database design, the *email* field in the *payments* table is set to NOT NULL and references the *email* field in the *useracct* table as a foreign key, which means every email address that appears in the *payments* table must also appear in the *useracct* table. In other words, users must have an account to make purchases. This is a conflict with the design document.
- FR1.9 and the Admin Module of the design document state that items in the Catalog Module will be added, deleted and modified from the admin interface. Setting the *quantity* to zero can disable an item. However, in the catalog table of the database design, there is no field for the quantity of product items. The missing quantity field was thus added to the *Catalog* table to resolve this conflict.

Scenario analysis may be employed to elaborate ambiguous requirements.

The requirements specification of the SAC project was ambiguous and poorly written.

Consider the following SAC requirement:

SAC FR 1.4: The system shall support online ticket sales.

This requirement is ambiguous. Heuristic **IH_{subject/action}3** was used to elaborate the requirement.

We authored the following scenario shown in Table 5.1 to describe the shopping experience of a client.

The scenario shown in Table 5.1 is not complete. For the sake of space, we intentionally omitted some branches, such as the client quitting the system in the middle of the process and the client not providing all required information in the required format. The scenario is specific enough for analysts to derive AC elements and specify AC rules.

Table 5.1 Scenario Analysis for SAC Requirement FR 1.4

Scenario Name	Clients purchase tickets on surryarts.org website.
Pre-conditions	Surryarts.org website is up and functional.
Events	<ul style="list-style-type: none"> (1) A client browses the surryarts.org website for recent shows and ticketing information. (2) System displays the requested information about the shows and ticketing information (3) The client requests to purchase a certain number of tickets for selected shows. (4) System puts the items in a shopping cart and shows a list of all items in the cart. (5) The client continues shopping. Go back to (1). (6) The client proceeds to check out. (7) System asks a) log in? b) create an account? c) check out without an account? (8) If the client selects a), then go to authentication scenario. (9) If the client selects b), then go to user registration scenario. (10) If the client selects c), then system displays a form that must be filled by the clients about billing and shipping information. (11) The client fills required information in the specified format and submits the order. (12) System displays a message that shows the order has been placed and an automated email confirmation has been sent to the client's email address.
Post-conditions	NULL
Obstacles	Tickets are sold out.
Requirements	FR 1.4: The system shall support online ticket sales.
Goals	Buy tickets online on surryarts.org website.

Explicit constraints should be specified in the condition part of AC rules.

Heuristic **IH_{cond1}** helps analysts to define explicit conditions that are specified in source documents. In the SAC case study, we applied this heuristic and specified explicit conditions. Consider the following statement in the SAC design document:

SAC Order/Payment Module: *Users will be able to view their previous orders (that were not canceled or deleted) at any point to see the state of their orders or to print an invoice.*

The above statement contains a clause that defines that certain orders may be viewable to users. Thus, *orders not cancelled or deleted* must be specified in the condition part of the rule that allows users to view previous orders.

Implicit conditions are more challenging to identify than explicit conditions.

Recall that explicit conditions are constraints that are clearly defined in the requirements specifications, whereas implicit conditions are implied but not explicitly defined. In the SAC case study, we realized that identifying implicit conditions is challenging and analysts need prescriptive guidance for how to identify them. To address this need, we created a set of heuristics for identifying and specifying these conditions as completely as possible. The set of heuristics (**IH_{cond3-8}**) provides a classification of frequently used conditions that help analysts ensure better coverage. In the SAC case study, we applied several heuristics (**IH_{cond3}**, **IH_{cond4.c}**, **IH_{cond6}**, **IH_{cond8.a}**) for identifying implicit conditions. Consider the two rules shown in Table 5.2.

Rule #1 allows users to create a new account by inserting an entry into the table *useracct*. We applied heuristic **IH_{cond6}** to determine if any database constraint exists that applies to this rule and identified a precondition: when a user tries to create a new account, the same account cannot already exist in the system. Because email is the primary key in the table *useracct*, we define a condition for this rule: “useracct.email does not exist.”

Table 5.2 Rule #1 and Rule #2 in the SAC Case Study

	Rule #1	Rule #2
Mode	<i>Authorization: Allow</i>	<i>Authorization: Allow</i>
Subject	<i>Agent: user</i>	<i>Agent: user</i>
Action	<i>Database Action: insert</i>	<i>Database Action: read</i>
Object	useracct	useracct
Condition	Useracct.email does not exist AND user provides all required info in the right format	useracct.email exists AND user is logged in AND current login user.email == useracct.email
Obligation	NULL	NULL
Sources	FR1.5	FR1.5

Rule #2 allows users to view their account information. We applied heuristics **IH_{cond}3** and **IH_{cond}4.c** to identify implicit conditions. Specifically, authentication is required for all users because of privacy concerns, thus account information is protected from unauthorized access via authentication. Applying heuristic **IH_{cond}3** resulted in the following condition being specified: “user is logged in.” Additionally, a user can only access his/her own account information, not other users’ account information. Thus, the relationship constraint between the subject and the object is important and must be specified as a condition in the rule. Otherwise, privacy breaches may happen because any user can access any other user’s account information. Thus, we specified the following condition after applying heuristic **IH_{cond}4.c**: current login user.email == useracct.email. This condition defines the relationship constraint between the subject (i.e., current login user) and the object (i.e., useracct) for an access request to be granted.

5.1.3 Summary and Discussion

Table 5.3 summarizes the results of the SAC case study. We did not make changes to the analysis process. This implies the analysis process is quite stable at this time.

Table 5.3 Summary of the SAC Case Study Results

	Pre-ReCAPS	Post-ReCAPS
No. of <i>tables / attributes</i> in the DB design	5/59	5/61
No. of <i>requirements</i>	15	15
No. of <i>modified requirements</i> as a result of ReCAPS analysis	N/A	15
No. of <i>inconsistencies</i> identified between the SRS and the DB design as a result of ReCAPS analysis	N/A	7
No. of <i>access control rules</i> created during ACR identification (after ReCAPS Step #2)	N/A	38
No. of <i>final rules</i> (after ReCAPS Step #3)	N/A	34
No. of <i>final policies</i> (after ReCAPS Step #3)	N/A	21

During the SAC case study, we created ten heuristics (**SH_{DLP}1-4**, **IH_{cond}4.c**, **IH_{cond}6**, **RH_{redundancy}3**, **RH_{conflict}3**, **GH_{grouping}1-2**) based on the experience and lessons learned. Specifically, heuristics **SH_{DLP}1-4** were created to specify database-level AC rules. Heuristic **IH_{cond}4.c** was created to specify relationship constraints between the subject and the object of an access control rule. This was based on the fact that each user can only access his/her *own* personal information when he/she logs into the SAC.org website, not other users' personal information. Heuristic **IH_{cond}6** was created to specify database constraints for database-level AC rules. Heuristics **RH_{redundancy}3** and **RH_{conflict}3** were created to reflect the fact that conditions may cause redundancy or conflicts between access control rules. Heuristics **GH_{grouping}1-2** were created to group AC rules into ACPs. Additionally, it became evident that heuristics were needed to aid in specifying special kinds of conditions and obligations that had not been encountered in our formative case studies. Four new heuristics were created to aid in this task (**IH_{cond}4-5**, **IH_{oblig}1-2**). Heuristics **IH_{cond}4-5** define a set of constraints that often have an impact for an enforce engine to make grant/deny decisions on an access request. Heuristics **IH_{oblig}1-2** define security and privacy obligations that may exist in access control rules.

We observed that some heuristics were employed more often than others (see Table 5.4). It is worth noting that this does not mean that the heuristics that were not employed in the SAC case study are not useful; it simply depends on the context and characteristics of the target system. Although some heuristics were not applied to the SAC project, they did prove useful during the formative case studies. Thus, we fully expect them to be valuable for analyzing other systems that are more similar in nature to the systems analyzed during our formative analyses. Table 5.4 shows that a large number of the ReCAPS heuristics were applied at least five times during the SAC project's ACP specification effort. In particular, there were ten heuristics that were applied at least ten times.

Table 5.4 Heuristics Usage Frequency in the SAC Case Study

Usage Frequency	Heuristics
Applied > 10 times	IH _{scope} 1, IH _{object} 1-3, IH _{subject/action} 1, SH _{DLP} 1-2, IH _{cond} 3, IH _{cond} 6, GH _{grouping} 1
Applied >= 5 and < 10 times	IH _{scope} 2-3, SH _{DLP} 3-4, IH _{cond} 1-2, IH _{cond} 4.c
Applied < 5 times	IH _{subject/action} 2-3, IH _{cond} 8.a, RH _{redundancy} 1-3, RH _{conflict} 1-2, GH _{grouping} 2
Not used	IH _{cond} 4.a-b, IH _{cond} 4.d-f, IH _{cond} 5, IH _{cond} 7, IH _{cond} 8.b-c, IH _{oblig} 1-2, RH _{redundancy} 4, RH _{conflict} 3

5.2 NCSU College of Management Event Registration System

The objective of the second summative case study was to further validate the ReCAPS method by having analysts with some or no experience with the method apply it within the context of a Web-based application. Additionally, the case study sought to validate the

usefulness of the Security and Privacy Requirements Analysis Tool (SPRAT; see Section 4.3) to support ReCAPS analysis activities.

The Event Registration System (ERS) is sponsored by the College of Management at North Carolina State University (NCSU) to simplify the event registration process for the College of Management annual graduate symposium. The project, which began in January of 2005 and will be deployed in October of 2005, is under development by a group of graduate students from the College of Management and the Department of Computer Science at NCSU.

Similar to the SAC project, the ERS project is also a web-based application, in which users' personal information must be protected from unauthorized access. Access control plays an important role in this project.

5.2.1 Methodology and Case Study Artifacts

The ERS case study was conducted by three analysts for approximately 24 person-hours. The task was to specify database-level access control policies for the Event Registration System using the ReCAPS method with the aid of tool support available in the SPRAT. Among the three analysts, one analyst was a ReCAPS expert; one had some experience with the method; and the other had no experience at all. This team setup allowed us to observe how analysts with different levels of prerequisite knowledge successfully apply or fumble with the method. It is worth mentioning that all analysts were at the same stage in terms of understanding the problem domain. No analyst had any knowledge about the ERS project before the case study.

Four source documents were used in this case study:

- ERS Software Requirements Specification (SRS), Version 2.3
- ERS database schema design
- ERS Security Policy, Version 2.2
- ERS Privacy Policy, Version 1.2

A major difference between this case study and the previous three is that tool support was available to support the analysts during this case study. As mentioned in Section 3.1, the Security and Privacy Requirements Analysis Tool (SPRAT), discussed in Section 4.3, supports ReCAPS analysis activities. At the time of this case study, the SPRAT ACP specification module SPRAT had been implemented, thus the analysis activities were much less manual and much more efficient. Other modules, such as those that support goal analysis and scenario analysis, were still under development; however, the ACP specification module is the only module needed to support this validation effort.

The source requirements, the security policy, and the privacy policy were manually entered into the database via SQL queries because the corresponding modules in SPRAT (i.e., the requirements specification module and the document management module) had not been completed. Upon commencing the ACP specification process, analysts were prompted to specify which source document they wished to work on. As discussed in Chapter 4, the system provides an interface that supports the analysts as they navigate between requirements or statements of the given source document as they specify access control elements and rules. Upon completing the ACR identification process, analysts use the refinement tool to sort all candidate rules, make appropriate changes to rules and elements (e.g., remove redundant rules), and finally group rules into policies. It is worth mentioning that the identification and refinement processes still require human analysis; the tool provides a mechanism to help analysts document access control rules and elements, automatically record the origins of each rule to support traceability, and sort rules so that redundancies and conflicts can be identified more easily.

To ensure all analysts in this case study had sufficient training with the ReCAPS method, about 75 minutes were devoted to initial ReCAPS training. The objective of the training session was to ensure that all analysts have a common understanding on the method and its objective. The detailed analysis steps and all available heuristics were reviewed during the training session. Then the analysts started from the identification process and then the refinement process.

Although source documents, except database schema design, were already available in the SPRAT, each analyst also had a printed hardcopy of the source documents during the case study. Access control rules were documented using the SPRAT, but findings, observations, and questions were documented either on the source documents or on additional paper. During the case study, the three analysts took turns “driving” (the keyboard). All analysts participated in the identification and refinement process.

5.2.2 Lessons Learned

This section summarizes the lessons learned from the ERS case study.

The SPRAT is efficacious in supporting the ReCAPS method, especially traceability.

The SPRAT support was helpful during the ERS case study and its benefits can be summarized in several ways. First, reusing access control elements improves the efficiency and consistency of the ACP specification process. The SPRAT allows analysts to select from a list of existing elements when specifying a new access control rule. Only when no appropriate element exists, do analysts need to create a new one. After the element is created, it joins the repository of existing elements and may be reused in access control rules that are subsequently created. Second, the tool automatically establishes traceability links between access control rules and the source requirements or statements. This helps analysts by ensuring that traceability is maintained. The traceability support was a tremendous aid to analysts in this study, especially in contrast to previous studies in which the need to manually maintain traceability was a tremendous burden. In the previous case studies, analysts had to document the sources of each access control rule manually on the paper, which is tedious and difficult—not to mention prone to human error. The SPRAT allows analysts to easily trace the sources of each rule as demonstrated in the ERS case study. Third, the SPRAT allows analysts to analyze the collective rules in a system more

conveniently. For example, the tool allows analysts to sort all the rules according to different elements (e.g., first sort by subjects, then by objects). In this way, it is easier to analyze the collective privileges of a subject and identify any redundancy that may exist in the rule set. Fourth, the SPRAT will eventually allow (upon completion of additional modules) analysts to employ RE techniques, such as scenario analysis, to elaborate ambiguous requirements and identify access control elements accordingly.

Traceability is essential during the ACR refinement process.

During the ERS case study, maintaining traceability between access control rules and source requirement or statement proved to be extremely important throughout the ACR refinement process. Often two very similar rules that were potentially redundant or in conflict with one another were identified during the refinement process. However, because these rules were derived from different sources they had to be traced back to the sources from which these rules were derived to determine why they were specified differently and whether they could be merged or reconciled. Consider the following two access control rules derived from ERS source documents as shown in Table 5.5:

Table 5.5 Rule #50 and Rule #70 in the ERS Case Study

	Rule #50	Rule #70
Mode	<i>Authorization: Allow</i>	<i>Authorization: Allow</i>
Subject	<i>Agent: anyone</i>	<i>Agent: anyone</i>
Action	<i>Database Action: insert</i>	<i>Database Action: insert</i>
Object	registrants	registrants
Condition	registrants.username does not exist AND registrants.email does not exist	value (registrants.username) == current users.username
Obligation	NULL	NULL
Sources	FR18	Security Policy: II. Security Policy

Rules #50 and #70 are very similar. All elements are identical except for the condition and source. The rules were derived from different sources: Rule #50 was derived from requirement FR18, whereas Rule #70 was derived from Security Policy document, section II: Security Policy.

According to heuristics **RH_{redundancy}3** (see page 106) and **RH_{conflict}3** (see page 109), these rules were either redundant or in conflict. To reconcile this situation in the ERS, we had to trace back to the sources from which these two rules were derived as shown in Table 5.6.

Table 5.6 Source Detail for Rule #50 and Rule #70

FR18	Security Policy: II. Security Policy
<p>The system will allow invitees who were sent the URLs via post-cards earlier as well as non-invitees who access the URL, to create user accounts on the web-based system.</p> <p>Description: As per instructions from the sponsor, there will be no restriction on a user account creation upon registration. Any person may go to the system URL to register and thereby create an account. The default privilege is that of a ‘regular user’ and cannot be altered by the registrant. The administrator only can make a VIP account. (Regular users are not sent VIP dinner invitations). The user has to indicate if he is a student registered with College of Management or has another affiliation.</p> <p>Origin: Interview meeting with sponsor, Kathy Green, E-Commerce Program Manager, CoM on 1/21/2005</p> <p>Priority: 1</p>	<p>The user creates an account by choosing a mandatory username and password (also has to provide other mandatory information mentioned below) at the website to register for the symposium.</p>

From these information sources and the database schema design, it is clear that Rule #70 assumes that users are required to take two steps to register for the symposium. First, users create an account by providing a username as password. Second, users provide registration information

such as address, affiliation, and food preference. This is an assumption according to the database schema design. The *registrants* table stores detailed information about registrants, whereas the *users* table stores only username and password. The two tables are associated via username. Based on this assumption, users may create an account and quit the system. They may come back later to provide registrant information. In this case, the condition to insert an entry in the *registrants* table is the new value for the username field must be the current user's username.

Rule #50 assumes that users must provide all information (including username, password, and registrant information) on the same Web page at one time. Thus, the condition for this rule is that the username and email address do not exist in the *registrants* table because username is the primary key and email is the unique key. Upon careful examination, it became evident that the condition portion of both rules was incomplete. According to the first assumption, the username and provided email address still should not exist when a user inserts an entry in the *registrants* table. According to the second assumption, the value for the username field in the *registrants* table should be the username in the *users* table. Thus, the condition in Rule #70 and Rule #50 were merged and Rule #70 was removed.

The above example exemplifies how analysts need to be able to trace rules to the source documents (including database designs) from which the rules were derived. Traceability support is thus very important in ReCAPS.

Checking the requirements against the DB design aids in identifying inconsistencies between source documents.

The usefulness of heuristic **IH_{Object}3** was demonstrated again in the ERS case study. Many inconsistencies between the requirements specification and the database design were identified. A few example inconsistencies that would have lead to expensive repairs had they not been identified cripple later phases of the software development process are as follows:

- Requirement FR16 states that the administrator should be permitted to delete/query responses submitted by the registrants via the evaluation form. The report would include: (1) date of the response; and (2) responses in the form of evaluation form. However, applying heuristic **IH_{object}3** revealed that in the database schema design, the response date was missing.
- Requirement FR19 states that a VIP registrant can log into the system and accept/reject the VIP dinner invitation. However, applying heuristic **IH_{object}3** revealed that there was no field that stores information about acceptance/rejection decisions in the database schema design. Nor was there a field to store whether a VIP has been sent a dinner invitation.
- Requirement FR25 states that the system will provide an option to the registrant to remind/change his password. The reminder function will send the registrant's password to the registrant's registered email address. However, the privacy policy section regarding the Password Policy, states that a hash function is used to encrypt passwords before passwords are stored in the database. Hash functions are known to be one-way functions, which means it is computationally infeasible to retrieve the plain text from the encrypted hash code. How can the remind process retrieve the password from the database and send the password to the registrant's registered email address? Applying heuristics **IH_{object}3** helped identify this inconsistency.

Global conditions eliminate the need to repeatedly specify the same condition in each AC rule.

A global condition is one that applies to the entire system, does not change within the lifetime of the system, and applies to all access control policies. In the ERS system, all users must log into the system to perform tasks, but any user can create an account. Thus, "user is

logged in” is a condition for all access control rules. To be more efficient in ACP specification and enforcement, the following global condition was specified for all access control rules:

Global condition: user is logged in

Once this global condition is defined, there is no need to define it yet again in each rule; thus ACP specification can be accomplished more efficiently by defining global conditions. Because a global condition encompasses all access control rules, the global condition must be checked by an enforcement engine before other conditions are checked.

Subsumption is a major indicator of redundancy between AC rules.

Heuristic $RH_{\text{redundancy}}1$ (see page 105) describes a specific kind of redundancy: subsumption. Subsumption occurs when the subject (or object) in one rule contains the subject (or object) in another rule. During the ERS case study, subsumption was a major indicator of potentially redundant access control rules. Heuristic $RH_{\text{redundancy}}1$ was frequently applied and proved effective in identifying this kind of redundancy. Eight subsumption redundancies were identified in the ERS case study. There are two kinds of subsumption (object subsumption and subject subsumption) as we now discuss. Consider the two rules shown in Table 5.7:

Table 5.7 Rule #60 and Rule #62 in the ERS Case Study

	Rule #60	Rule #62
Mode	<i>Authorization: Allow</i>	<i>Authorization: Allow</i>
Subject	<i>Role: registrant</i>	<i>Role: registrant</i>
Action	<i>Database Action: update</i>	<i>Database Action: update</i>
Object	registrants.veglunch	registrants
Condition	current users.username == registrants.username	current users.username == registrants.username
Obligation	NULL	NULL
Sources	FR21	FR22

The rules in Table 5.7 exemplify *object subsumption*. The only difference between Rule #60 and Rule #62 is the object. The object in Rule #60 is the *veglunch* field in table *registrants*, whereas the object in Rule #62 is the entire table *registrants*. Obviously Rule #62 subsumes Rule #60. Thus, Rule #60 was removed from the set of access control rules in the ERS during the refinement process.

Table 5.8 shows an example of *subject subsumption*. Administrator is subsumed by anyone. Thus, Rule #75 is subsumed by Rule #49. Thus, Rule #75 was removed from the set of access control rules in the ERS.

Table 5.8 Rule #49 and Rule #75 in the ERS Case Study

	Rule #49	Rule #75
Mode	<i>Authorization: Allow</i>	<i>Authorization: Allow</i>
Subject	<i>Agent: anyone</i>	<i>Role: administrator</i>
Action	<i>Database Action: insert</i>	<i>Database Action: insert</i>
Object	users	users
Condition	users.username does not exist	users.username does not exist
Obligation	NULL	NULL
Sources	FR18	Security Policy: II. Security Policy, 1. Identification and Authentication Policy

Modal redundancy may exist because rules are derived from different sources.

Modal redundancy occurs when two logically equivalent rules are specified with opposite modes (i.e., allow and deny). Modal redundancy may exist between access control rules because access information may be specified in different ways in different sources. Heuristic **RH_{redundancy}2** helps analysts identify this kind of redundancy; in this case, the redundancy can be resolved by simply removing the deny rule. Consider the two ERS access control rules shown in Table 5.9.

Table 5.9 Rule #65 and Rule #78 in the ERS Case Study

	Rule #65	Rule #78
Mode	<i>Authorization: Allow</i>	<i>Authorization: Deny</i>
Subject	<i>Role: registrant</i>	<i>Role: registrant</i>
Action	<i>Database Action: select</i>	<i>Database Action: select</i>
Object	Registrants	Registrants
Condition	current users.username == registrants.username	current users.username != registrants.username
Obligation	NULL	NULL
Sources	FR22	Security Policy: II. Security Policy, 5. Private Information Protection Policy

Although the mode of the two rules shown in Table 5.9 is different, Rules #65 and #78 are logically equivalent and thus redundant. They were created during the identification process because access control was specified in two ways in the available ERS sources (see Table 5.10).

Table 5.10 Source Detail for Rule #65 and Rule #78

FR22	Security Policy: II. Security Policy, 5. Private Information Protection Policy
<p>The registered users can login through a user interface and access their information.</p> <p>Description: The registered users can login to the website and access and modify their information and their choices of sub events and sub-sub-events.</p> <p>Priority: 1</p>	<p>The registrants can query the attendees of a particular panel discussion, but only the names (first and last) would be revealed to other registrants and not the other private information.</p>

Requirement FR22 clearly defines what registered users *can* do. In the security policy source (Security Policy: II. Security Policy, 5. Private Information Protection Policy), access

information is specified in a different manner. From this statement, three AC rules were derived during the identification process (see Table 5.11).

Table 5.11 Rules #76, #77 and #78 in the ERS Case Study

	Rule #76	Rule #77	Rule #78
Mode	<i>Authorization: Allow</i>	<i>Authorization: Allow</i>	<i>Authorization: Deny</i>
Subject	<i>Role: registrant</i>	<i>Role: registrant</i>	<i>Role: registrant</i>
Action	<i>Database Action: select</i>	<i>Database Action: select</i>	<i>Database Action: select</i>
Object	registrants.fname	registrants.fname	Registrants
Condition	registrants exists AND registrants.privacy_flag is set to share AND (panelusermapping.username == registrants.username)	registrants exists AND registrants.privacy_flag is set to share AND (panelusermapping.usernam e == registrants.username)	current users.username != registrants.username
Obligation	NULL	NULL	NULL
Sources	Security Policy: II. Security Policy, 5. Private Information Protection Policy	Security Policy: II. Security Policy, 5. Private Information Protection Policy	Security Policy: II. Security Policy, 5. Private Information Protection Policy

Careful examination of Tables 5.9 and 5.11 reveals the existence of modal redundancy. In this case, Rule #65 (see Table 5.9) was kept in the system, whereas Rule #78 was removed because by default an enforcement engine will deny an access request if no rules are matched when the enforcement engine checks the access request against the rule set. If Rule #65 were removed, then registrants would not be able to view their personal information, which is not desirable because it would conflict with requirement FR22.

5.2.3 Summary and Discussion

Table 5.12 summarizes the results of the ERS case study. Although it was unnecessary to apply all the ReCAPS heuristics during the ERS project, the existing ReCAPS process and

heuristics proved sufficient and efficacious. Thus, this summative case study served to further validate the usefulness and efficiency of the ReCAPS process and heuristics. Moreover, there was no need for additional heuristics that were not already available in the ReCAPS; this suggests the analysis method and associated heuristics are stable and suitable for transaction intensive, web-based information systems.

Table 5.12 Summary of the ERS Case Study Results

	Pre-ReCAPS	Post-ReCAPS
No. of <i>tables / attributes</i> in the DB design	16/87	16/89
No. of <i>requirements</i>	29	29
No. of <i>modified requirements</i> as a result of ReCAPS analysis	N/A	23
No. of <i>inconsistencies</i> identified between the SRS and the DB design as a result of ReCAPS analysis	N/A	47
No. of <i>access control rules</i> created during ACR identification (after ReCAPS Step #2)	N/A	85
No. of <i>final rules</i> (after ReCAPS Step #3)	N/A	68
No. of <i>final policies</i> (after ReCAPS Step #3)	N/A	20

Most of the inconsistencies that were identified were characterized by discrepancies with regard to data format. Specifically, the format for data fields defined in the requirements specifications were often inconsistent with the actual format defined for these fields in the database schema design. For example, the location of an event is defined as “free text up to 255 characters” in requirement FR1; whereas the field *place* in table *symposium* is defined as `varchar(50)`. Clearly, the fact that ReCAPS helped the analysts identify so many inconsistencies is yet another indicator of how applying the ReCAPS method improves software quality.

As in the SAC project, some heuristics were applied more often than others in the ERS case study as well (see Table 5.13). This does not mean the heuristics that were not used in the ERS case study are not useful. As previously mentioned, those heuristics were not applied in the

summative case studies are more appropriate for systems with similar scope and domain as found in the formative case study projects. These heuristics proved effective in these other contexts. Table 5.13 shows that most of the ReCAPS heuristics are very helpful in the ACP specification process. Ten heuristics were applied at least ten times and eight heuristics were applied at least five times.

Table 5.13 Heuristics Usage Frequency in the ERS Case Study

Usage Frequency	Heuristics
Applied ≥ 10 times	$IH_{scope}1$, $IH_{object}1-3$, $IH_{subject/action}1$, $SH_{DLP}1-2$, $IH_{cond}3$, $IH_{cond}6$, $GH_{grouping}1$
Applied ≥ 5 and < 10 times	$SH_{DLP}3-4$, $IH_{cond}1-2$, $IH_{cond}4.c$, $RH_{redundancy}1-3$
Applied < 5 times	$IH_{scope}2-3$, $IH_{subject/action}2-3$, $IH_{oblig}1$, $RH_{conflict}2-3$, $GH_{grouping}2$
Not used	$IH_{cond}4.a-b$, $IH_{cond}4.d-f$, $IH_{cond}5$, $IH_{cond}7-8$, $IH_{oblig}2$, $RH_{redundancy}4$, $RH_{conflict}1$

5.3 An Empirical Evaluation of ReCAPS

All four previous case studies were conducted by analysts who are somewhat familiar with the ReCAPS method. Thus, two important attributes remain to be validated: the effectiveness / usefulness of the method in comparison to other approaches (none of which exists) and whether analysts who are unfamiliar with the method can effectively employ it without substantial training. Specifically, these two factors are concerned with the following two questions:

- Does applying the ReCAPS method produce better results than other methods?
- Can individuals without prerequisite knowledge about the ReCAPS method employ it effectively without significant training?

In order to answer these two questions, we performed an empirical study in a graduate-level software engineering class at North Carolina State University during the Spring 2005 semester. As previously mentioned, there is no prescriptive guidance on ACP specification. The closest related work is Fontain's mapping approach from KAOS specifications to Ponder policies [Fon01], as discussed in Chapter 2. However, it is difficult to compare ReCAPS with Fontain's approach in an educational environment. Firstly, Fontain's approach starts from KAOS specifications, which are expressed in a formal language. It requires significant training for a group of undergraduate and graduate students to understand and use KAOS specifications, which is unsuitable for a class assignment in an educational setting. Secondly, the experimenter would have to manipulate the source documents and produce KAOS specifications from these source documents for the group that applies Fontain's approach before the experiment. This results in unfair comparison because the two groups start from different inputs to derive ACPs. Thirdly, Fontain's approach was published as a project report and has not been validated or peer reviewed. Since the work was published in 2001, there was no follow-up work available to the public. Due to the above concerns, we were forced to compare the ReCAPS with no method support at all. The subjects were thus separated into two groups: a ReCAPS group and a *control* group. The hypothesis for this empirical evaluation is that ReCAPS allows analysts to better specify access control policies than does the lack of a method (which is the current state of the art). This section presents the experimental method employed and discusses the results of the investigation.

5.3.1 Experimental Method

This experiment compares the use of ReCAPS in the specification of access control policies for information systems with a control condition in which no method was stipulated. This section explains the experimental method employed for this validation effort.

Design

The subjects were divided into two groups, each of which was asked to use one of the following methods:

- the Requirements-based Access Control Analysis and Policy Specification (ReCAPS) method;
- no method but the same criteria for quality ACPs was provided to both groups.

The independent measures involve a group using ReCAPS and a control group. Ideally, it would better demonstrate the ReCAPS method's effectiveness by comparing it with an existing method for the same purpose. However, after an extensive review of the literature and upon consulting experts in the research community and industry, we were unable to identify such a method that could be used for comparison. Thus, we had no choice but to compare ReCAPS with a control group.

Subjects

The subject population in the experiment was a group of students (undergraduate and graduate students) enrolled in a graduate level software engineering class (CSC 510: Software Engineering) at North Carolina State University in the Spring 2005 semester. Students in the class were given the opportunity to voluntarily participate in an experiment for which they would be compensated with extra credit to help raise their course grade. All participating subjects signed a consent form that was approved by the North Carolina State University Institutional Review Board.

Students participating in the experiment possessed varied backgrounds: 3 PhD students; 28 master's students, and 17 undergraduate seniors. Of the 48 students enrolled in the course, 28 participated. Of the 28 participants, 26 submitted valid responses. An assignment was deemed valid if the participant completed the entire assignment, including the time/effort form and evaluation questions. Of the 26 valid responses, 2 were completed by PhD Students, 12 were

completed by MS students, and 12 were completed by undergraduate students. Table 5.14 details the distribution of subject participants among the two experiment groups.

Table 5.14 Class Distribution of the Subjects Who Participated in the Experiment

	# of PhD Students	# of MS Students	# of Undergrad Students	Total
ReCAPS Group	1	7	6	14
Control Group	1	5	6	12
Total	2	12	12	26

Their knowledge in software engineering, databases, and security differed significantly. To minimize noise that would have occurred with unbalanced groups, a survey was conducted before the study to collect information from each of the subjects concerning their background in software engineering, security and access control, as well as their class level (e.g. Senior, Master, PhD student). The course instructors blinded any identifying information (e.g. students' names) from the individual conducting the empirical study to ensure that the students were assigned to groups in an unbiased manner. All 48 students were then assigned to the two groups in a balanced fashion. Students were first grouped according to their expertise and class level. Allocation was random within a group with similar expertise and the same class level. For example, when an undergraduate student with limited security knowledge was assigned to one group, another undergraduate student with similar level of security knowledge was assigned to the other group.

Although the above method ensures that all 48 students were divided into two balanced groups, it is important to know whether those 26 students who participated and submitted valid results still have equivalent background. We checked these 26 students' background and Table 5.15 shows the number of students in each category. Each student's background in security, software engineering, and database was ranked as good (G), medium (M), or no (N).

Table 5.15 Knowledge Background of the Subjects Who Participated in the Experiment

	Security			Software Engineering			Database			Total
	G	M	N	G	M	N	G	M	N	
ReCAPS Group	6	4	4	6	2	6	11	0	3	14
Control Group	8	2	2	9	0	3	8	0	4	12
Total	14	6	6	15	2	9	19	0	7	26

The above data suggests that the ReCAPS group was less confident about their security and software engineering skills than the control group; whereas the control group was less confident about their database background than the ReCAPS group.

Materials

The project used in the experiment was a simplified version of the Surry Arts Council (SAC) website e-commerce enhancement project. The SAC project is well-suited for this empirical analysis because the system involves various kinds of access to sensitive information (e.g., customer contact information, financial information, purchase history). The security and privacy of this information is critical to the success of the system, thus it is compelling from a security perspective. Additionally, the project is a typical e-commerce application, which does not require the experiment subjects to possess much additional or new domain knowledge. The original requirements specifications were a little bit too complicated and extensive for a controlled experiment in a class. Thus, we simplified the requirements specifications to ensure the project was self-contained for the experiment and that it did not require a major time commitment on the part of the study participants.

Two types of documents from the SAC project were used as the sources for this study: a software requirements specification (SRS) document and a database schema design. The

simplified SRS contained ten requirements (in contrast to 15 requirements in the full SRS), and the DB schema design contains five tables (as in the original DB design). The source documents for both the ReCAPS and control groups were exactly the same.

What differed in the materials for both groups were the instructions. The ReCAPS group was provided an assignment description, which summarized the method presented in this dissertation, including the main activities and several heuristics. The control group was provided a different assignment description, which only summarized necessary background information for how to complete the study (much of this was identical to that provided to the ReCAPS group, but all ReCAPS context was removed), what the results look like (e.g., what is access control, what an access control policy is comprised of).

Although no specific method for how to specify access control policies was provided to the control group, it is important to note that both groups were given the exact same set of criteria of a good set of access control policies with examples of rules and policies. The instructions made it clear that these criteria would be used to evaluate their results. The ReCAPS instructions did include directions on how to ensure these criteria are met, whereas the control group knew the criteria but the only guidance this group received about how to achieve the criteria were examples of correct access control rules and policies. These quality criteria are discussed in detail in the following *Measurements* section.

The subjects in both groups were provided with blank worksheets upon which they were asked to document: the access control rules they derived from source documents as well as inconsistencies identified between the requirements specifications and the database design. Each subject was required to document the amount of time he/she spent on each step of the study (including the amount of time devoted to reading and developing a solid understanding of the assignment instructions). Upon completing the study, each subject was required to answer several qualitative evaluation questions about their experience with the assignment and the approach(es)/techniques they employed. Although the subjects' answers to these questions are

anecdotal at best, they do provide valuable insights about the ReCAPS method, which we discuss later in this section.

In summary, the materials used in this empirical study are as follows:

- NCSU Consent Form for Research;
- SAC project software requirements specifications document;
- SAC project database schema design;
- questions about their experiences with the assignment and the ReCAPS method;
- time effort form; and
- various assignment worksheets for documenting assignment results.

Procedure

An initial pilot study was run in a graduate level software engineering class in the Fall 2004 semester. There were 29 students enrolled in the course, 23 students participated, but only 12 students submitted valid responses. Pilot studies are necessary prerequisites for the design of a sound empirical study. For example, this pilot student enabled us to revise the materials to ensure that students in the subsequent empirical study fully understood what constituted a “valid response.” The access control rules produced during this initial study were given to an independent security expert to evaluate the quality of the rules specified by the students in the pilot study. These evaluation criteria were included in the final empirical study to ensure that all participants in both groups had a common understanding how their results would be evaluated. These criteria are discussed in detail in Section 5.3.2.

Before conducting this empirical experiment that involved human subjects in research, we obtained permission and approval from the NCSU Institutional Review Board (IRB). Subjects were asked to sign a consent form before the study to comply with university policy for any empirical research that involves human subjects. The experiment was run as an optional

homework assignment and students who participated received up to two points extra credit towards their final grade in the course.

Questionnaires were administered to the students regarding their expertise level and class level. As previously mentioned, students were then assigned to the two groups (the ReCAPS group and the control group) in a balanced fashion.

The experimenter spent 10 minutes introducing the assignment in class to all students and then handed the assignment to each student. Students were aware that there were two groups in the study but none of them knew which group he/she was in. Subjects were given one week to complete the study as an optional homework assignment without supervision. The assignment was self-paced; subjects were free to work anytime, anywhere, and as much as they wished. However, subjects were instructed to complete the assignment independently and limit their references to only the materials provided.

The main task for the subjects in this study was to produce a set of access control policies derived from two source documents. Additionally, subjects were required to improve both source documents during the analysis and specification process; for example, if they identified an inconsistency or missing requirement/data element, they were asked to document this in the respective document(s). All results were documented using the provided worksheets. Additionally, subjects were required to document the amount of time they spent on each step of the study. It is important to note, however, that subjects were assured that the amount of time they spent on the assignment would in no way affect their grades. Upon completing the study, subjects were required to answer several qualitative questions concerning their experience during the assignment.

To emphasize that the main part of the study was the set of access control policies produced, students were informed that the ACPs would comprise 60% of their final grade on the assignment. The entire grading policy was disclosed in the assignment description. Subjects were instructed that they would not receive any partial credit for incomplete assignments to ensure that

they remembered to complete the time/effort form as well as answer the questions upon assignment completion.

5.3.2 Results

This section discusses the statistical analysis results of the empirical evaluation of the Requirements-based Access Control Analysis and Policy Specification (ReCAPS) method. We employed the Mann-Whitney U Test [Rob73, Sie73] for the statistical analysis. The Mann-Whitney U Test is a non-parametric method for testing the significant differences between two independent groups with non-normal data and small sample size.

Measurements

Three main aspects of the experiment may be compared across the two groups of subjects: the quality of resulting ACPs, the improvements to the two source documents, and the time effort. We employed the following eight criteria to evaluate the quality of resulting ACPs. These quality criteria were determined together with a security expert⁶.

- (1) All possible access control rules are specified.
- (2) Each rule is within the scope of access control.
- (3) Each action is a database operation and each object is an object in the DB.
- (4) The conditions for each rule are correctly specified and as completely as possible.
- (5) Each rule is traceable to the sources from which it was derived.
- (6) No two rules are redundant.
- (7) No two rules conflict with one another.
- (8) Logically connected rules are grouped together.

⁶ Based on conversation and discussion with Dr. Ting Yu at North Carolina State University.

These eight criteria were used to evaluate the resulting ACPs. The improvements to both source documents were measured using the number of inconsistencies identified between the requirements specifications and the database design, the number of ambiguities and inconsistencies identified within the requirements specifications, and the number of problems identified within the database design.

It is important to note that all students were aware that their results would be evaluated using the same quality criteria and what these criteria were, as mentioned in section *Material* (see page 153). This ensures that all students have a common understanding about the objective of the study and prevents the experimental results from being skewed.

Evaluation Method

We followed the following process to quantitatively evaluate the quality of access control rules derived by subjects. First, we counted all the rules derived by the subjects and compared these rules with a rule set that was derived from the same sources and specified by experts. For each rule in the experts' rule set, we compared whether there was a semantically equivalent rule in the subject's rule set. If yes, we counted the subject-identified rule, otherwise we counted it as incorrect. Note that the form of rules as specified by the subjects did not have to be identical to that of those specified by the experts. In this way, we were able to create two sets of rules for each subject: a set of rules *A* that are in the experts' rule set and a set of rules *B* that are false positive (either outside the scope of access control or cannot be derived from sources, see Figure 5.1).

For each rule in rule set *A*, we analyzed: whether each action is a database operation and each object is an object in the DB, whether the conditions are correct and as completely specified as possible, and whether the rule is traceable to the sources from which it was derived. . For the entire rule set *A*, we analyzed whether there were any redundant or conflicting rules and whether logically connected rules were grouped together. A set of rules is said to be logically connected if

the rules have the same subject, action or object. For example, a set of rules that specify the SAC Manager’s privileges is logically connected because these rules all have the same subject: SAC Manager.

For each rule in rule set *B*, we documented whether it is “outside the scope of access control” or “cannot be derived from sources”.

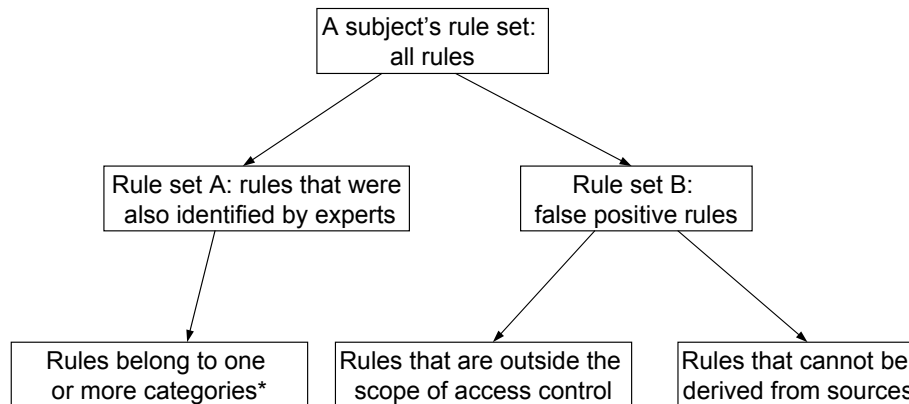


Figure 5.1 Evaluation of a Subject’s Rule Set

*: E.g., ambiguous actions/objects, incomplete conditions, etc.

We used the number of access control rules identified by each subject to measure criteria 1—4 and 6—7. However, traceability (Criterion 5) is evaluated as a binary measure: a subject is either able to trace all rules to the sources or is not. We did not find a case in which a subject was able to trace some rules to their sources but not others. In all cases, all rules were traced or none were. With respect to grouping (Criterion 8), the rules were graded on a scale of 0—10 based on the grouping in the subject’s rule set. Criterion 8 was scored as 10 if all rules were grouped into policies, otherwise only partial credit was given, depending on based on how many rules were correctly grouped.

We have discussed the measurement and evaluation method for the empirical study. Next we discuss the results from three perspectives: the quality of access control policies, the improvements to source documents, and the time effort.

5.3.2.1 Quality of Access Control Policies

Table 5.16 compares the performance of the ReCAPS group against the performance of the Control group with respect to the eight evaluation criteria required of a set of high-quality set of access control rules and policies. In Table 5.16, the evaluation criteria are listed in the first column and the second column specifies whether the ReCAPS group outperformed the Control group, based upon whether the Mann-Whitney U test revealed the results were statistically significant. As shown in Table 5.16 and Figure 5.2, the ReCAPS group identified more access control rules that were also identified by the experts than the control group (Criterion 1). The average number of rules identified by the ReCAPS group that were also in the experts' rule set is 22.29, compared with 15.17 rules identified by the control group. The results of this comparison are statistically significant (Mann Whitney U = 152.5, $p < 0.001$, two-tailed test). This data shows that the ReCAPS group provided better coverage of the access control rules and policies than the control group.

Table 5.16 Summary of Empirical Evaluation Results

Evaluation Criteria	ReCAPS > Control
(1) All possible access control rules are specified.	Significant, $p < 0.001$
(2) Each rule is within the scope of access control.	Not significant
(3) Each action is a database operation and each object is an object in the DB.	Significant, $p < 0.001$
(4) The conditions for each rule are correctly specified and as completely as possible.	Not significant
(5) Each rule is traceable to the sources from which it was derived.	Significant, $p < 0.001$
(6) No two rules are redundant.	Not significant
(7) No two rules conflict with one another.	Not significant
(8) Logically connected rules are grouped together.	Significant, $p < 0.001$

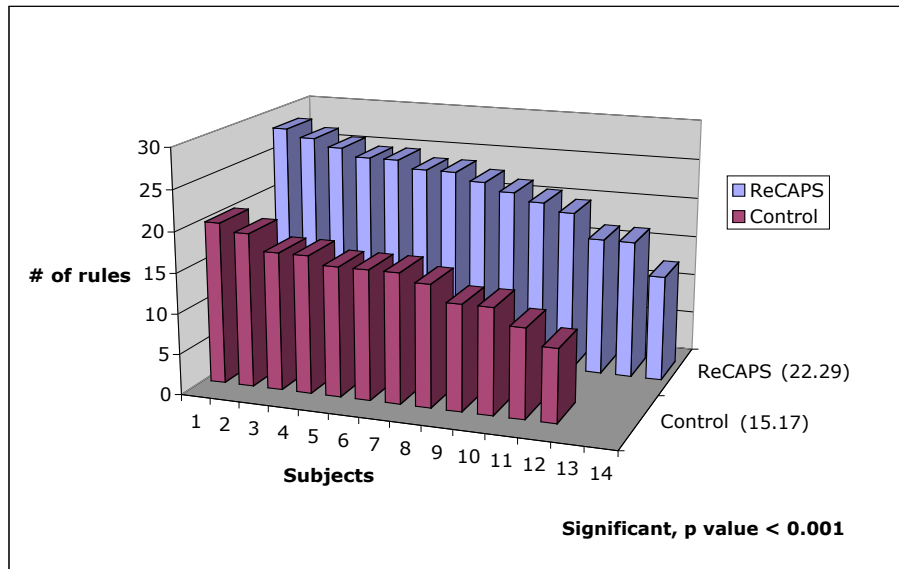


Figure 5.2 Number of Rules Identified by Each Subject That Were Also Identified by the Experts: The ReCAPS Group Outperformed the Control Group by Identifying More Rules That Were also Identified by the Experts.

If we subtract the number of rules that were identified by the experts from the total number of rules identified by each subject, we obtain the number of false positive rules (recall that false positive rules are either outside the scope of access control or cannot be derived from sources). The ReCAPS group identified fewer false positive rules than the control group, as shown in

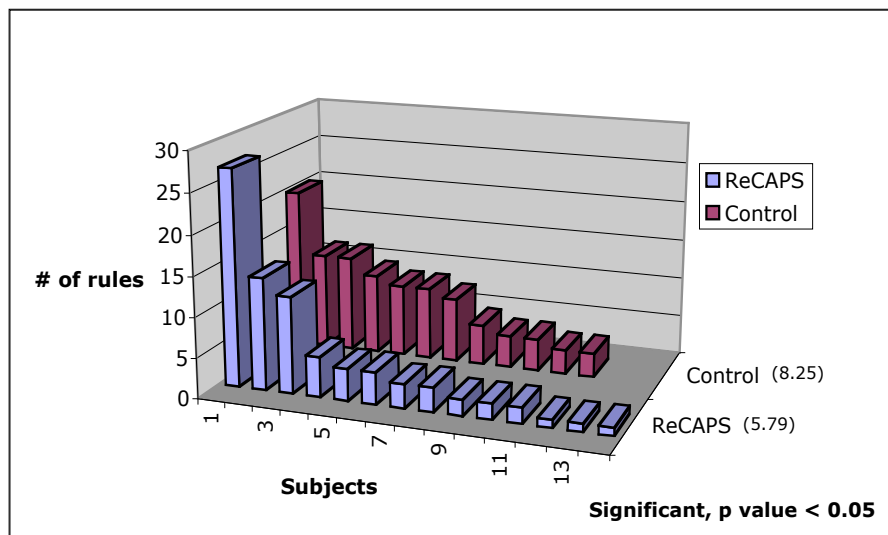


Figure 5.3 Number of False Positive Rules Identified by Each Subject: The ReCAPS Group Outperformed the Control Group by Identifying Fewer Number of False Positive Rules.

Figure 5.3. The average number of false positive rules identified by the ReCAPS group is 5.79, compared with 8.25 false positive rules identified by the control group. The results of the comparison are statistically significant (Mann Whitney $U = 122.5$, $p < 0.05$, two-tailed test). This data further demonstrates that the ReCAPS not only ensures better coverage; it also reduces false positive access control rules.

In the experiment, subjects were asked to specify database-level access control policies, in which every action is a DB operation and every object is an object in the DB. This helps minimize the number of ambiguous actions and objects identified. The ReCAPS group outperformed the control group by identifying fewer rules with ambiguous actions/objects, as shown in Figure 5.4. The average number of rules with ambiguous actions/objects identified by the ReCAPS group is 2.71, compared with 14.58 rules identified by the control group. The results of the comparison are statistically significant (Mann Whitney $U = 160.0$, $p < 0.001$, two-tailed test).

An examination of those rules in which every element was correctly specified reveals that the ReCAPS group yielded more rules with correctly specified elements than the control group,

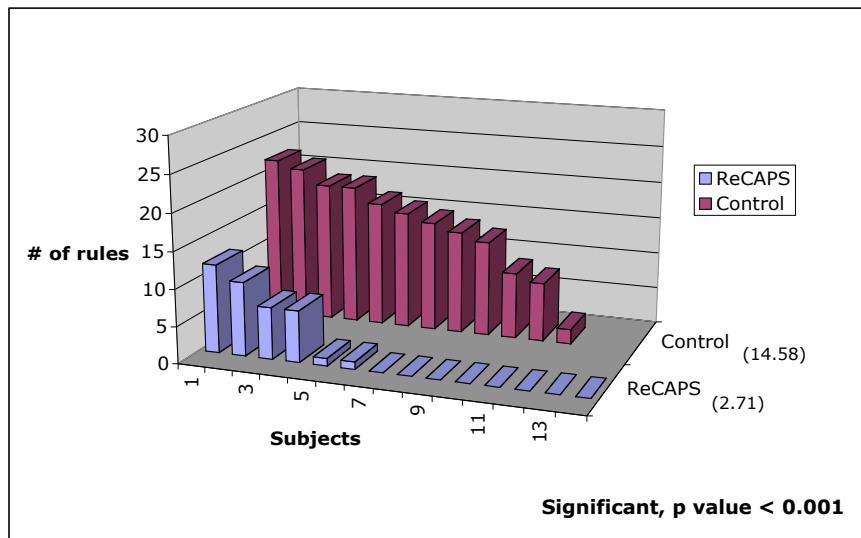


Figure 5.4 Number of Rules with Ambiguous Actions/Objects Identified by Each Subject: The ReCAPS Group Outperformed the Control Group by Identifying Fewer Rules with Ambiguous Actions/Objects.

as shown in Figure 5.5. Here, *correctly* means that each action is a DB action, each object is an object in the DB, and any conditions are fully and correctly specified. The average number of rules with every element correctly specified by individuals in the ReCAPS group is 9.43, compared with 1.25 rules specified by individuals in the control group. The results of this comparison are statistically significant (Mann Whitney U = 153.5, $p < 0.001$, two-tailed test).

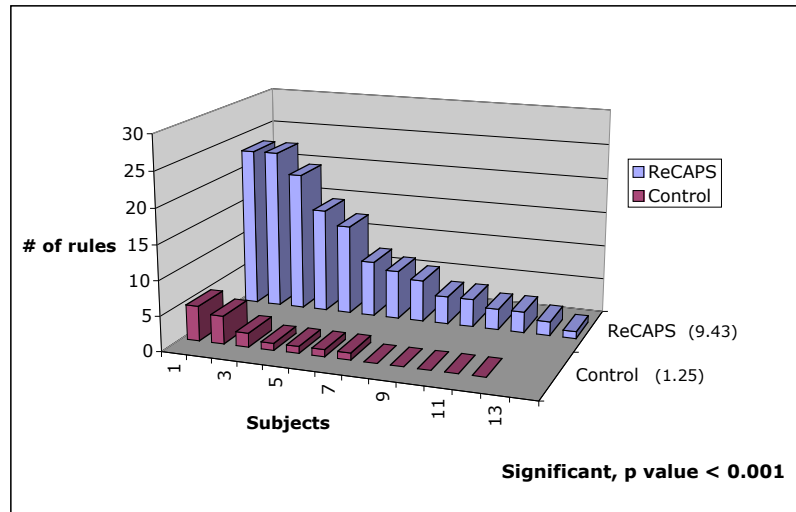


Figure 5.5 Number of Rules with Every Element Correctly Specified: The ReCAPS Group Outperformed the Control Group by Identifying More Rules with Every Element Correctly Specified.

The above data (see Figure 5.1—Figure 5.5) are compared using the number of rules in each category (e.g., # of rules identified by each subject that were also identified by the experts, # of false positive rules identified by each subject) by each subject. Another way to evaluate the data is to calculate the percentage of rules in each category, as shown in Figure 5.6.

$$\text{Percentage} = \frac{\text{Number of rules in a category identified by the subject}}{\text{Total number of rules identified by the subject}}$$

Figure 5.6 Formula to Calculate the Percentage of Rules in Each Category

If the percentage is used to evaluate the data, all of the previously presented statistically significant results concerning the comparison of the ReCAPS group and the Control group are

still statistically significant. Moreover, using this approach, the results of evaluation criteria #4 become statistically significant using Mann-Whitney U test, as shown in Figure 5.7. The average percentage of rules whose conditions are incorrectly or incompletely specified by individuals in the ReCAPS group is 45%, compared with 64% of the rules specified by individuals in the control group. The results of the comparison are statistically significant (Mann Whitney U = 123.5, $p < 0.05$, two-tailed test).

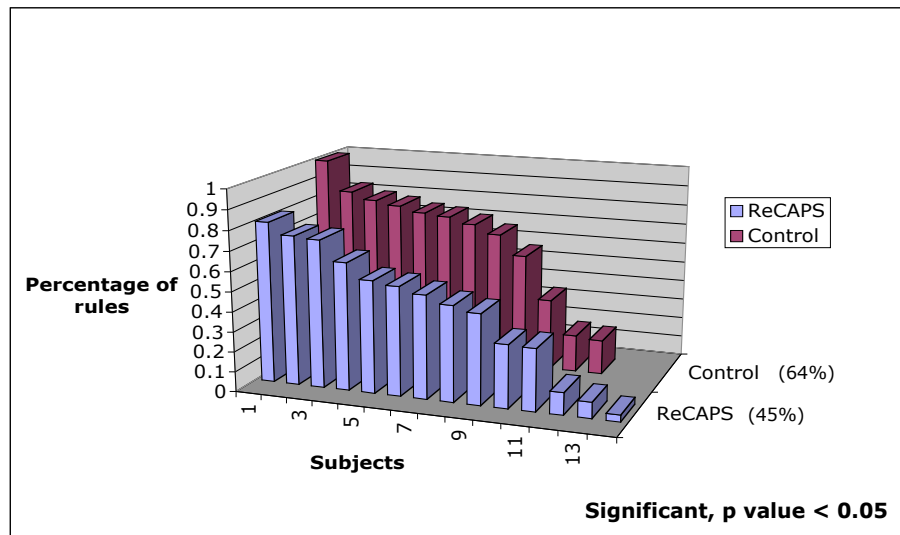


Figure 5.7 Percentage of Rules with Conditions Incomplete or Incorrectly Specified by Each Subject: The ReCAPS Group Outperformed the Control Group by Identifying a Lower Percentage of Rules with Conditions Incomplete or Incorrectly Specified.

As previously mentioned, traceability was evaluated in a binary fashion in this study. The individuals in the ReCAPS group consistently out performed the individuals in the control group with regard to maintaining traceability, as shown in Figure 5.8. The results of the comparison are statistically significant (Mann Whitney U = 154.0, $p < 0.001$, two-tailed test).

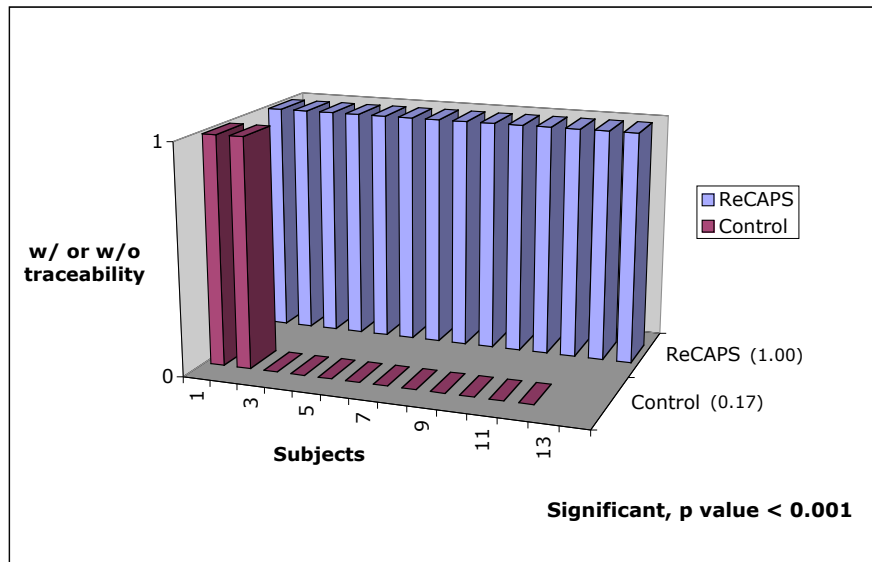


Figure 5.8 The ReCAPS Group Provided Better Traceability Support than the Control Group

In summary, the ReCAPS group outperformed the control group in all eight evaluation criteria. However, the Mann-Whitney U test results are only statistically significant in four of the eight evaluation criteria (1, 3, 5, 8). In the remaining four criteria (2, 4, 6, 7), the results are positive, but anecdotal at best because they are not statistically significant.

5.3.2.2 Improvements to Source Documents

The empirical study provided only anecdotal evidence that the ReCAPS group outperformed the control group by identifying more inconsistencies between the requirements specifications and the database design, more ambiguities/inconsistencies within the requirements specifications, and more problems in the database design. Although these results are not statistically significant, the data suggests that improvement to source documents is a side-benefit of the approach, not a main contribution (see Tables 5.3 and 5.12).

5.3.2.3 Time Effort

The average amount of time spent on the assignment by individuals in the ReCAPS group was 4.53 hours, compared with 3.79 hours by individuals in the control group, as shown in Figure 5.9. The results of this comparison are statistically significant (Mann Whitney U = 124.0, $p < 0.05$, two-tailed test).

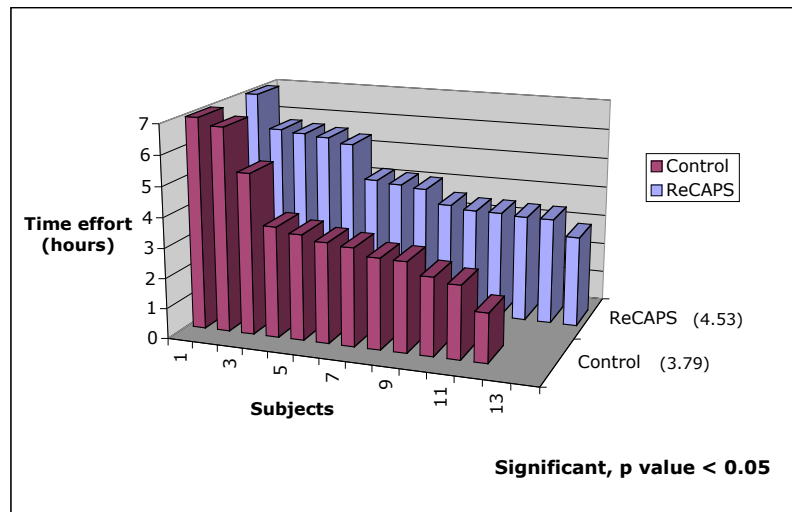


Figure 5.9 Time Spent on the Experiment by Each Subject: The ReCAPS Group Spent More Time on the Assignment than the Control Group.

Because of this fact, one might challenge the validity of the above data by suggesting that the ReCAPS group outperformed the control group simply because they spent more time than the control group. To address this, we performed a correlation analysis between the number of rules that were in the experts' rule set identified by the subjects and the amount of time spent on the assignment by the subjects. The result shows there is no strong correlation between these two factors, as shown in Figure 5.10. The variance R^2 (0.0326) would lead to a correlation coefficient r close to 0.18. The correlation result is not statistically significant either, with p value greater than 0.05.

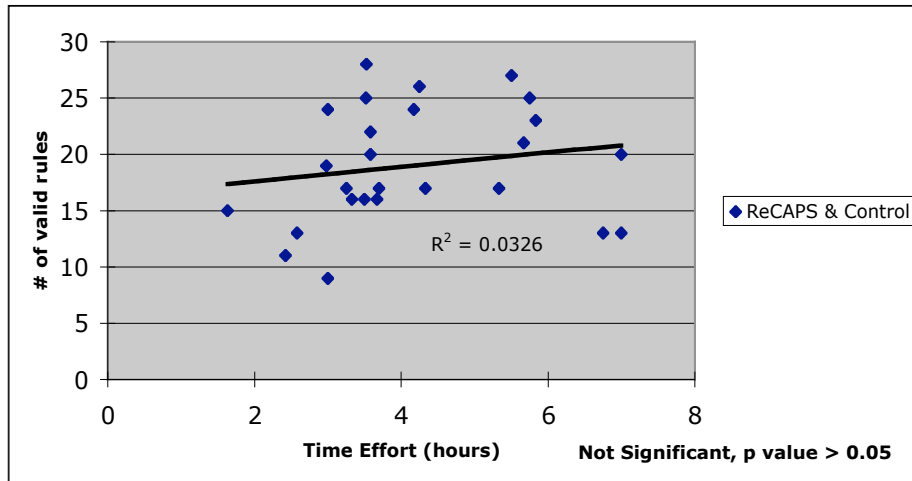


Figure 5.10 Correlation Analysis Shows There is No Correlation between the Time Effort and the Results in all Subjects

We conducted the same correlation analysis within the ReCAPS group and the control group. The results are shown in Figure 5.11. The variance R^2 for the ReCAPS group is 0.0939 (correlation coefficient $r = 0.31$) and the variance R^2 for the control group is 0.1221 (correlation

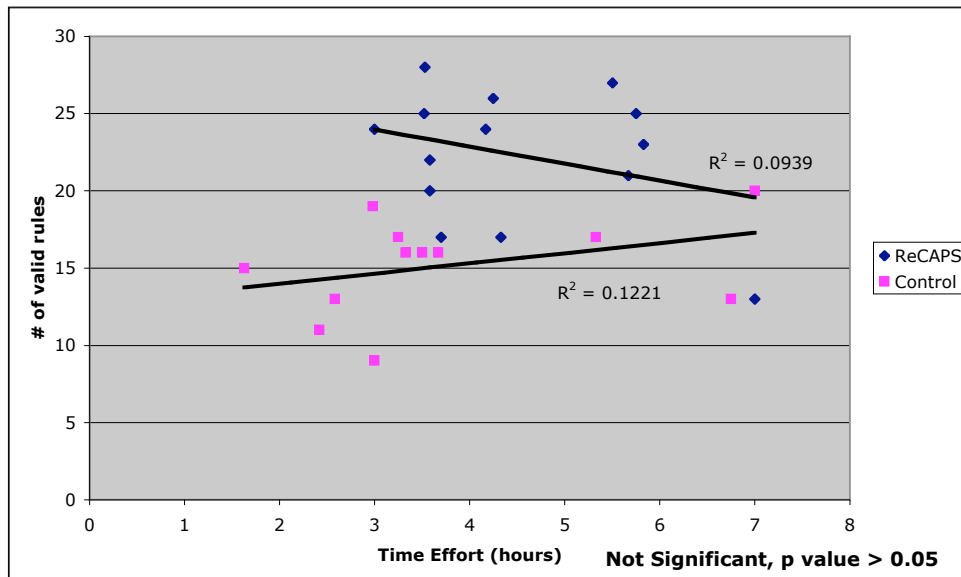


Figure 5.11 Correlation Analysis Shows There is No Correlation between the Time Effort and the Results in both ReCAPS and Control Group

coefficient $r = 0.35$). Neither of the correlation analysis result is statistically significant, either, with p value greater than 0.05. Based on the above analysis, we can conclude that it is unlikely that time/effort alone can account for the results in this empirical study.

5.3.3 Summary and Discussion

As previously stated, this empirical study sought to evaluate the effectiveness / usefulness of the method in comparison to other approaches (none of which exists) and whether analysts who are unfamiliar with the method can effectively employ it with reasonable training. The study demonstrated the feasibility of employing the ReCAPS method as a policy specification method. The results are supported by the similar specification of ACPs for the Surry Arts Council website e-commerce enhancement project across the ReCAPS and control groups. Four of the evaluation criteria for high-quality access control rules and policies were shown to yield statistically significant results in which the ReCAPS group outperformed the Control group. Additional anecdotal evidence supports the remaining four criteria as well.

Because we were limited in the size and complexity of the assignment that we could design for an optional extra credit assignment, only a subset of the ReCAPS heuristics were evaluated in this empirical study. The ReCAPS group was provided with 18 heuristics ($IH_{scope}1-2$, $IH_{object}1-3$, $IH_{subject/action}1$, $IH_{subject/action}3$, $SH_{DLP}1-2$, $SH_{DLP}4$, $IH_{cond}1$, $IH_{cond}3$, $IH_{cond}6$, $IH_{cond}8$, $RH_{redundancy}1-2$, $RH_{conflict}1-2$). These heuristics were selected for this empirical study by design. First, because the study was being conducted within the confines of a homework assignment, the selected heuristics could not require excessive prerequisite knowledge nor could so many heuristics be included without risking lack of participation due to the assignment's length and complexity. Second, the set of heuristics needed to stand on their own and be representative of the ReCAPS' mission. Third, the selected heuristics needed to be broadly applicable in the assignment. Based on these

considerations, we ensured that the set of heuristics used in this empirical study supported the identification, specification and refinement of access control rules.

The empirical study results presented in Section 5.3.2 and the feedback received from the ReCAPS group suggests that this subset of heuristics is very helpful in guiding analysts as they derive access control rules and specify access control policies. One student made the following statement in addressing whether the heuristics helped him/her perform the analysis efficiently:

“I believe it was very efficient. Several assumptions I would have overlooked were analyzed more meticulously because I was forced to state them.”

Another student commented on the heuristics as follows:

“The heuristics, just as the breaking of the access control rule into four parts, helped me to wrap my head around each section. This made the assignment manageable. Rather than trying to do everything at once in a non-logical order, the heuristics force me to do each step in order and sometimes in parallel.”

These anecdotal statements provide further insights into how the ReCAPS helps analysts. The remaining heuristics will need to be evaluated in follow up studies given that we were not able to evaluate all of them in this study (e.g., heuristics for constructing misuse cases to specify implicit conditions, heuristics for deriving ACPs from high-level security and privacy policies) due to limitations on the scale of the study. Thus, it was not possible to evaluate the heuristics as a complete set (e.g., whether they are complete and sufficiently comprehensive).

The above results are very encouraging, but they require careful interpretation. Because there are no existing documented ACP specification approaches that provide prescriptive guidance comparable to ReCAPS, any evaluation of ReCAPS necessarily involves comparing the performance of subjects applying ReCAPS to those using a different type of method or using no method at all. Examples of non-comparable methods include best industry practices. This

evaluation strategy was rejected, because we were unable to obtain a written version of best industry practices from anywhere.

The alternative used in the current empirical study—comparing performance of individuals in the ReCAPS group to that of individuals in a control group that was given no process guidance—is open to the objection that *any* method may help people, at least initially, because of the so-called Hawthorne Effect [FK78]. By virtue of being the focus of an investigation and by manipulating their behavior in an obvious way (here, training in ReCAPS), subjects are more attentive to the task at hand and perform better than they would have otherwise. Such a difference stems not from the treatment (ReCAPS) but from the fact that there is a treatment.

There are two responses to this concern. First, it is still important to confirm that the ReCAPS group significantly outperformed the control group. For the Hawthorne Effect to be a potential threat to validity, there has to be a potentially valid effect to threaten. The statistical analysis reveals that ReCAPS improved performance at the $p < 0.001$ level. Research in process interventions is notoriously vulnerable to individual differences, task variables, experimental demand characteristics and seemingly random properties of the experimental materials, e.g., different worksheets used for each group. Plausibly justified software engineering techniques and methods often fail to show any advantage when subjected to experimental test. In many cases, this may be because the techniques do not help at all; but in view of the factors just mentioned, it is important to replicate such studies before accepting such results at face value. Thus, any significant result, such as the one obtained in the current study, is clear *prima facie* evidence that the treatment effect yields a genuine benefit.

The second response concerns the mechanism through which the Hawthorne Effect is supposed to work. As suggested above, it could be that being subject to a visible treatment caused subjects to pay more attention to the task or take it more seriously, presumably because the attention they were receiving by being subjects in an experiment was reinforcing and they wanted to do well. However, since there was no significant difference between the time spent on

the task by the subjects in the two groups, we can safely conclude that the results are not merely due to one group focusing on the problem for longer. It is of course possible that the ReCAPS subjects paid more careful attention and used the time they had more effectively and would have done so with any instructions at all. However, this residual effect is unlikely for the following reason. The Hawthorne Effect is most likely to arise in a repeated measures design where the subjects are aware of the nature of changing treatments. (The original interventions at the Hawthorne Plant, from which the effect gets its name, are a classic example of subjects being aware that they were being subjected to one treatment after another.) In such a case, a control condition is manifestly less a treatment than an experimental condition. However, in an independent group design, such as the one used in the current study, the subjects in the experimental condition had no experience of the control condition with which to compare it. Subjectively, therefore, they were as likely to improve their performance as the ReCAPS subjects. We conclude that the difference between the ReCAPS and control groups can be attributed to the ReCAPS process and heuristics and not to any methodological artifact. Such a conclusion requires future replication before it can be accepted as definitive, but it is better supported currently than alternative explanations.

5.4 Summary

This chapter has presented three validation efforts (two summative case studies and an empirical evaluation) for the Requirements-based Access Control Analysis and Policy Specification (ReCAPS) method. Each case study detailed in this chapter involves a particular system:

- the Surry Arts Council (SAC) web enhancement project; and
- the NCSU College of Management Event Registration System.

The ReCAPS method provides analysts with the prescriptive procedural guidance and support needed for the identification and refinement of access control policies for software-based information systems. The two summative case studies validated the efficacy of the ReCAPS process and associated heuristics to derive access control policies from various source documents. Combining the results in Table 5.4 and Table 5.13, we applied and validated 28 (**IH_{scope}1-3**, **IH_{object}1-3**, **IH_{subject/action}1-3**, **SH_{DLP}1-4**, **IH_{cond}1-4**, **IH_{cond}6**, **IH_{cond}8**, **IH_{oblig}1**, **RH_{redundancy}1-3**, **RH_{conflict}1-3**, **GH_{grouping}1-2**) of 32 total heuristics in the two summative case studies. Four heuristics (**IH_{cond}5**, **IH_{cond}7**, **IH_{oblig}2**, **RH_{redundancy}4**) were not used in either case study. Among these 28 heuristics, ten of them (**IH_{scope}1**, **IH_{object}1-3**, **IH_{subject/action}1**, **SH_{DLP}1-2**, **IH_{cond}3**, **IH_{cond}6**, **GH_{grouping}1**) were applied at least ten times and five of them (**SH_{DLP}3-4**, **IH_{cond}1-2**, **IH_{cond}4.c**) were applied at least five times in each of the summative case studies. The empirical evaluation involving the Surry Arts Council web enhancement project demonstrated the feasibility of employing ReCAPS as a reasonable analysis method and showed that other people who were not familiar with the method can apply it effectively without significant training. The next chapter presents the conclusions of this dissertation and discusses plans for future work.

Chapter 6

Conclusions

高瞻远瞩。

Stand on a higher mountain and you will see farther.

—Chinese Proverb

This dissertation presents the Requirements-based Access Control Analysis and Policy Specification (ReCAPS) method, which was designed to improve information system security at the early stages of the software development process; especially during requirements analysis and software design. This work was motivated by the fact that ACP specification was without systematic procedural support and was often performed in an ad-hoc manner, resulting in systems that are vulnerable to security and privacy breaches. Additionally, policy specification was traditionally not part of the software development process and was especially isolated from requirements analysis and software design. This could lead to access policies that are not in compliance with system requirements. The ReCAPS method integrates policy specification into the software development process and derives access control policies from various source documents. ReCAPS provides prescriptive procedural guidance and tool support for specifying access control policies for information systems.

The research in this dissertation was developed while performing analysis on operational systems. The Requirements-based Access Control Analysis and Policy Specification (ReCAPS) method and the Security and Privacy Requirements Analysis Tool (SPRAT) were developed by applying them to their own development as well as to other operational systems. Through these various case studies, the ReCAPS method was evaluated and evolved as a result of the lessons learned from applying the method to the experiences detailed throughout this dissertation.

This chapter is organized as follows. Section 6.1 provides a synopsis of each chapter. Section 6.2 summarizes the contributions of this work. Section 6.3 discusses plans for future work. Section 6.4 concludes the dissertation.

6.1 Chapter Synopsis

Chapter 1 introduced and articulated the problem addressed in this work. Specifying complete and correct policies that control users' access to a system and its resources is important for the protection of data security and privacy in information systems. Traditionally, the ACP specification process has lacked systematic support and has been isolated from requirements analysis in software development, resulting in access policies that are not compliant with system requirements. This fact motivated the research presented in this dissertation, a systematic method for specifying access control policies for information systems.

Chapter 2 provided a survey of the related work in security and software engineering, to position the work presented in this dissertation. The ReCAPS builds upon this prior work by integrating ACP specification in the software development process to ensure compliance between policies, requirements, and software design in information systems.

Chapter 3 presented two formative case studies that served as the conceptual origin for the ReCAPS method: the Security and Privacy Requirements Analysis Tool (SPRAT) and the Transnational Digital Government (TDG) remote border control project. The ReCAPS method was simultaneously developed and informally validated while being applied to real projects; thus, the ReCAPS method evolved as a result of its application to the case studies that were discussed in this chapter.

Chapter 4 detailed the ReCAPS method. The ReCAPS method is an analysis process that is supported by a set of heuristics and a software tool. The presentation of the ReCAPS process steps and associated heuristics focused on the activities an analyst performs when employing the

method, progressing from identification, specification, refinement, to grouping. A set of heuristics was presented to guide software engineers through access control analysis, employing examples from the two formative case studies to elucidate the heuristics.

Chapter 5 presented three validation efforts for the method presented in this dissertation: (1) a summative case study involving the specification of access control policies for the Surry Arts Council (SAC) web enhancement project; (2) a summative case study involving the specification of access control policies for the NCSU College of Management Event Registration System (ERS), using the SPRAT; and (3) an empirical evaluation in which the method was applied to a small system by individuals who were previously not familiar with the ReCAPS method.

6.2 Summary of Contributions

The primary contribution of this dissertation is the introduction of the Requirements-based Access Control Analysis and Policy Specification (ReCAPS) method, which is comprised of:

- a process description that details the steps an analyst takes to perform this kind of analysis;
- a set of heuristics validated within the context of four operational systems and an empirical study to assist analysts in identifying and refining access control rules; and
- a software tool that supports the ReCAPS analysis activities.

ACP specification is typically isolated from requirements analysis and software design. It is often conducted without methodological support or systematic guidance. ReCAPS supports access control analysis and ACP specification and offers three main advantages not currently available:

- ReCAPS integrates ACP specification with requirements analysis and software design;
- ReCAPS provides traceability support to help ensure compliance between policies and requirements; and

- ReCAPS offers prescriptive guidance for ACP specification with a rich set of heuristics.

Although ReCAPS is essentially an analysis method with support of a set of heuristics and a tool, its most important contribution goes beyond that of a method. The new software development scheme that builds ACP specification as an explicit part of the software development process created by ReCAPS is significant. By integrating ACP specification with requirements analysis and software design, ReCAPS provides a basic framework for ensuring compliance between different levels of policies, system requirements and software design. The impact of this compliance is significant. One of the problems that plague companies and organizations is the degree of confidence they have in claiming that their information systems are enforcing security/privacy laws and global policies. This problem also plagues law enforcement agencies; currently there is a lack of technology to help them measure an organization's accountability for enforcing laws. The ReCAPS approach is a promising step in the right direction towards solving these problems. First, we derive access control policies from system requirements and high-level security and privacy policies. Because the sources are where security requirements come from, this development scheme helps ensure that a software system is actually enforcing high-level security / privacy laws and policies. Second, we establish traceability links between high-level policies, system requirements, and access control policies. This traceability support helps ensure that any changes in the high-level policies can be easily traced to the corresponding software development artifacts (e.g., requirements specifications, DB designs, ACPs), where appropriate changes can be made.

Figure 6.1 portrays the traceability links that are maintained across various artifacts in the ReCAPS. When an ACP is derived, we establish a link between the policy and its sources (e.g., a particular requirement, a particular section of the security policy or privacy policy, or a particular table/field in the database design). When we employ RE techniques to elaborate requirements, we also build requirements traceability links between each requirement and its origin, goals, scenarios and stakeholders. By establishing the links across these artifacts, we can manage policy

and/or requirements evolution. In the event of a change in a policy or requirement, our approach allows analysts to quickly locate the affected requirements or policies for subsequent modification. By ensuring consistency between ACP, requirements and software designs, our approach improves the quality of ACPs and helps bridge the gap between requirements and design. Note that laws are outside the dotted rectangle in Figure 6.1 because they are outside the scope of this dissertation. Ideally, high-level security/privacy policies and requirements specifications should specify the security and privacy requirements governed by law. The traceability between high-level security / privacy policies, requirements specifications and laws is outside the scope of this dissertation.

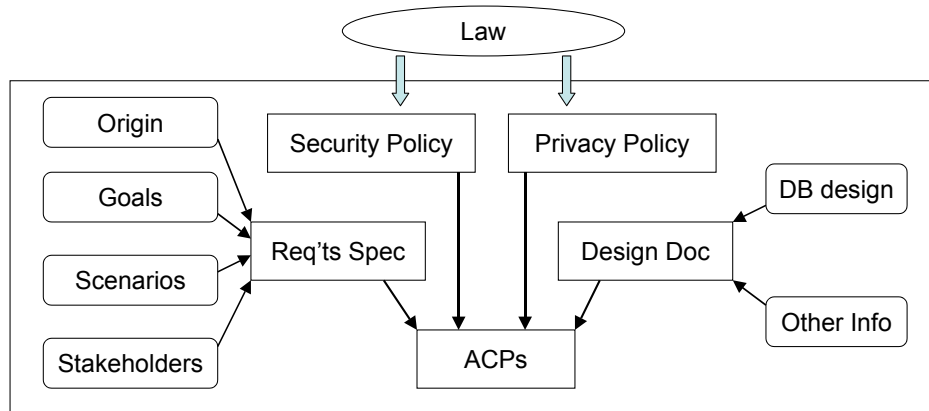


Figure 6.1 Traceability Support in ReCAPS

As previously mentioned, prescriptive guidance on ACP specification is greatly needed when developing secure software systems. ReCAPS provides the first methodological support of this kind to date. We have validated the ReCAPS process and heuristics within the context of two summative case studies and an empirical case study. Our validation efforts to date have shown that the ReCAPS process and heuristics are helpful for analysts in specifying better quality ACPs.

Because ReCAPS is inherently inquiry-driven and iterative, it offers the same benefit as other inquiry-driven approaches. As such, it helps analysts ensure that many of the ambiguities, inconsistencies, and conflicts that often plague requirements specifications, software designs and corresponding policies are identified and eliminated.

6.3 Current Limitations and Plans for Future Work

The work presented in this dissertation addresses some of the fundamental problems with ACP specification; however, our approach does have its limitations, and work remains to be done in these areas. This section discusses some of the main limitations of the ReCAPS method and provides an overview of areas of future interest and work.

The main limitations of the ReCAPS method are:

- the method assumes that a requirements specification document and a database design are available;
- formal analysis of the resulting ACPs for completeness and consistency is not available at this time;
- the method does not provide much support for defining roles for RBAC systems; and
- delegation and refrain policies have not been investigated in ReCAPS.

The first area for future work involves extensions to the ReCAPS method. Plans for extending ReCAPS can be summarized as pursuing the following four directions:

- specify ACP during requirements analysis and use ReCAPS to drive database design;
- provide formal analysis to the resulting ACPs to support automatic reasoning;
- define roles for RBAC systems; and
- investigate other types of access control policies, such as delegation and refrain policies.

The fact that the ReCAPS' assumes that an initial requirements specification document and a database design are available implies that when analysts specify access control policies using the ReCAPS method, requirements analysis and database design have already been completed. We are considering moving the ACP specification process a bit earlier in the software development process and integrating ACP specification with requirements analysis. Our prior experiences

suggest this is feasible. As reported in Chapter 3 and Chapter 5, the ACP specification process is a requirements disambiguating process. For example, scenario analysis is used in heuristics **IH_{subject/action}3** and **SH_{DLP}4**. Although the ReCAPS heuristics for identifying AC elements need to be slightly revised when no available source documents are available, the heuristics are broadly applicable. For example, the heuristics for identifying conditions can be reused during requirements analysis without any change. Additionally, all the heuristics for refining AC rules can still be used.

The challenging part is to identify objects when an initial database design is unavailable. Currently, ReCAPS allows analysts to check the objects identified from source documents against the database design. When the database design is not available, the checking is impossible. We are considering extending ReCAPS to use it to drive database design. Object-oriented analysis methods may be helpful in this process.

The work presented in this dissertation focuses on the identification and specification of access control policies for information systems. The heuristics for refining AC rules are still limited. For example, the conflicts identification heuristics can only detect explicit conflicts. More rigorous conflict identification methods are needed. Formal methods can help in this matter. Additionally, to evaluate whether a set of access control policies satisfies some security properties, we need formal analysis on the resulting ACPs. Our plan in this direction is to develop formal semantics for the policies being specified and develop a tool that can transform the policies into Alloy specifications, which can be used to perform automated analysis using model checking techniques [Hol97].

Role-based access control (RBAC) is widely used in many applications to simplify authorization management. Defining roles for a complex organization and system can be very complex. Although ReCAPS supports three kinds of subjects—agent, role and group—it does not provide much support on defining roles for RBAC systems. In RBAC systems, role definition and

management should be part of the ACP specification process. A structured role engineering process and a life-cycle model to manage roles are needed. Furthermore, organization theory may be investigated to define roles for RBAC systems [CIN03].

In this dissertation, we found authorization and obligation policies were sufficient to represent the policies in all four case studies. However, we have yet to investigate the other two types of access control policies: delegation and refrain policies. Our plans for extending ReCAPS also include investigating these kinds of policies.

In addition to the above extensions to ReCAPS, we plan to integrate the SPRAT with other software tools and further validate the ReCAPS using the tool in an industrial setting. The above discussions are aimed at extending ReCAPS to enrich the method and make it more useful. Broader areas of future work are also under consideration.

ReCAPS is mainly an analysis method that can be used during the development of new software systems. However, there are many legacy systems in place in most organizations and this raises additional questions that must be addressed: Can the ReCAPS method be used as a checking method to check whether a legacy system is enforcing the high-level security and privacy policies? Can the ReCAPS method be used to check whether two legacy systems with different high-level policies are in compliance with one another? Answers to these questions have practical significance. For example, when the HIPAA took effect on April 14, 2003, regulated organizations wanted to know whether their legacy systems were in compliance with the new law. In another example, if two organizations merge, both organizations have legacy systems in place that are enforcing different policies. Can ReCAPS help them ensure that their legacy systems are in compliance with one another? We believe the traceability support that ReCAPS offers is a promising solution on this matter and industry has already expressed an interest in this level of support offered by the ReCAPS.

The ACP specification method supported by ReCAPS is a top-down method. We start from requirements and database design to specify ACPs. If we want to migrate an existing system to a

new system, we might want to employ a bottom-up or hybrid method that contains both top-down and bottom-up techniques. In legacy systems, there are users who are performing tasks. There are data elements that are in the database, being retrieved and updated. How can we extract access control policies from these existing resources? We believe data-mining techniques may be useful in this case.

6.4 Conclusions

ACP specification is a conceptually complex process and involves extensive human analysis. In this dissertation, we presented the Requirements-based Access Control Analysis and Policy Specification (ReCAPS) method that provides procedural guidance for this purpose. The use of our approach in the practice of requirements engineering results in more complete, more accurate access control policies that will be enforced by the software system. Additionally, the use of our approach in software development improves the quality of software systems by identifying and resolving inconsistencies early. Finally, the traceability support across software artifacts provided by our approach helps ensure consistencies across these artifacts.

Appendix A

Summary of ReCAPS Analysis Activities

Step 1: Understand the problem domain.

Step 2: Scan available source documents to identify AC elements and specify AC rules.

Step 2.1: Identify objects that need to be protected.

Step 2.2: Identify responsible subjects and actions on the object.

Step 2.3: Identify conditions under which an actor is allowed to perform an action on an object.

Step 2.4: Identify obligations that the actor or system must fulfill if an access request is granted.

Step 3: Refine access control rules.

Step 3.1: Sort the AC rules according to subjects and objects.

Step 3.2: Merge redundant AC rules and create new rules, if necessary.

Step 3.3: Reconcile any conflicting AC rules.

Step 4: Group logically connected AC rules into ACPs.

Appendix B

Summary of ReCAPS Heuristics

Table B.1 ReCAPS Heuristics for Identifying AC Scope (Step 2)

Code	Heuristic	Page
IH _{scope1}	Access control analysis is data-centric. Thus, for each requirement, analysts should ask the following question: <i>does this requirement involve any user access to sensitive information in the system?</i> If the answer to the above question is yes, then access control rules may be derived from this requirement. Otherwise, no access control rules can be derived from this requirement.	82
IH _{scope2}	If a requirement describes something that is not enforceable in the system, then no access control rules can be derived from the requirement.	83
IH _{scope3}	No access control rules can be derived from non-functional requirements, except security and privacy requirements.	83

Table B.2 ReCAPS Heuristics for Identifying Objects (Step 2.1)

Code	Heuristic	Page
IH _{object1}	To identify objects that must be protected in the system, it is helpful to examine the nouns that follow verbs.	84
IH _{object2}	To distinguish access-related objects from other objects, it is helpful to consider whether the candidate object is a system resource that should only be accessed by authorized actors.	84
IH _{object3}	To identify the objects that must be included in the database, every object identified in the SRS should be mapped to an object (e.g., a table, a column, a row or a cell) in the database design.	84

Table B.3 ReCAPS Heuristics for Identifying Subjects and Actions (Step 2.2)

Code	Heuristic	Page
IH _{subject/action1}	Given a requirement expressed in the form “The system shall allow <someone> to <do something>”, the subject is <someone> and the action is <do>.	86
IH _{subject/action2}	Given a requirement expressed in the form “The system shall support/provide the ability (be able) to <do something>”, the subject is unclear. The <do> action has two possibilities as follows.	87
IH _{subject/action2.a}	The <do> action is performed by an subject. In this case, the analyst would need to consult the stakeholders to disambiguate the requirement by asking precisely who will perform this action. By clarifying the requirement, one can rephrase this requirement as “The system shall allow <someone> to <do something>”, resulting in a less ambiguous requirement specification.	87
IH _{subject/action2.b}	The <do> action is not performed by any subject. Instead, it is an automated action performed by the system. In this case, the analyst should consult the stakeholder(s) to determine <i>who</i> will access <i>what</i> information if the system performs this action.	88
IH _{subject/action3}	Given a requirement expressed in the form, “The system shall provide/support/allow/<do> <something>”, neither the subject nor the action is clear in this case. The <something> in this requirement is often a feature or a function. In this case (the requirement is vague and subject/action cannot be easily identified), analysts should employ scenario analysis to elaborate the requirement.	89

Table B.4 ReCAPS Heuristics for Identifying Conditions (Step 2.3)

Code	Heuristic	Page
IH _{cond1}	When a requirement specifies constraints (e.g., if, unless, when, during, before, after, etc.), specify the constraints as a condition in the resulting AC rule.	92
IH _{cond2}	When a scenario contains pre-conditions, it is helpful to examine these pre-conditions and specify those that fall within the scope of access control as a condition in the resulting AC rule.	93
IH _{cond3}	(Authentication constraints) If access to the requested data requires authentication, then define <i>this fact</i> as a condition in the resulting AC rule.	93
IH _{cond4}	(Contextual constraints) If the context of an access request plays a role when making grant/deny decisions, these contextual constraints should be specified as conditions in the resulting rule.	93
IH _{cond4.a}	(Temporal constraints) If a data access is restricted by time or date constraints, specify them as conditions in the resulting AC rule.	94
IH _{cond4.b}	(Location constraints) If a data access is restricted by any location constraints, specify them as conditions in the resulting AC rule.	94
IH _{cond4.c}	(Relationship constraints) If there are any relationship constraints between the subject and the object, specify them as conditions in the resulting AC rule.	95
IH _{cond4.d}	(Affiliation constraints) If there are affiliation constraints for the subject, specify them as conditions in the resulting AC rule.	95
IH _{cond4.e}	(Attribute constraints) If there are any attribute constraints to the subject, specify them as conditions in the resulting AC rule.	96
IH _{cond4.f}	(State constraints) If access request can only be granted when the system is in a specific state, then specify this state constraint as a condition in the resulting AC rule.	96
IH _{cond5}	(Usage constraints) If there are any usage constraints that restrict the subject's access to the object, specify the constraints as conditions in the resulting AC rule.	97
IH _{cond6}	(Database constraints) If the resulting rule is a database-level AC rule, it is helpful to consider whether there are any database constraints for the action. If yes, specify the constraints as conditions in the rule.	97
IH _{cond7}	(Security constraints) If is helpful to use general security principles such as least privileges and separation of duties to construct misuse cases that a user may use to exploit the capabilities for hostile intent. Corresponding security constraints should thus be specified in the resulting AC rule.	97
IH _{cond8}	(Privacy constraints) If data can only be accessed for a specific purpose, by a certain group of people, or after consent has been obtained, these privacy constraints should be specified as conditions in the resulting AC rule.	99
IH _{cond8.a}	(Purpose constraints) If specific data can be used for only certain purposes, then specify these purpose constraints as conditions in the resulting AC rule.	99
IH _{cond8.b}	(Recipient constraints) If only some recipients can access certain data, then specify recipient constraints as conditions in the resulting AC rule.	101
IH _{cond8.c}	(Consent constraints) If certain data can be accessed only after consent is obtained, then specify consent constraints as conditions in the resulting AC rule.	102

Table B.5 ReCAPS Heuristics for Identifying Obligations (Step 2.4)

Code	Heuristic	Page
IH _{oblig} 1	(Security obligations) If an auditing trail must be recorded upon access to certain data, recording auditing trails must be specified as an obligation in the resulting AC rules.	102
IH _{oblig} 2	(Privacy obligations) It is helpful to evaluate whether there are any privacy obligations in the resulting AC rules.	103
IH _{oblig} 2.a	If the subject is required to update/delete the data after access, specify updating/deleting data as obligations in the resulting AC rule.	103
IH _{oblig} 2.b	If the subject is required to manipulate the data after access, such as encrypting/decrypting, anonymizing, etc., specify the manipulation action as obligations in the resulting AC rule.	103
IH _{oblig} 2.c	If the subject is required to notify data subjects after access, specify notification as obligations in the resulting AC rule.	103
IH _{oblig} 2.d	If the subject is required to obtain consent or authorization after access, specify obtaining consent or authorization as obligations in the resulting AC rule.	104

Table B.6 ReCAPS Heuristics for Specifying AC Rules (Step 2)

Code	Heuristic	Page
SH _{DLP} 1	To specify database-level access control policies, the object in each rule must be an object (e.g., a table, a column, a row or a cell) in the database.	86
SH _{DLP} 2	To specify database-level access control rules, the action in the rules must specify a standard database operation (e.g., create, select, insert, update, delete, etc.).	90
SH _{DLP} 3	When an action can be directly mapped to a database operation (e.g., add, enter, read, view, retrieve, remove, change, edit, etc.), map the action to a standard database operation and specify that database operation as the action in the derived AC rule.	90
SH _{DLP} 4	When the identified action is an abstract action, use scenario analysis to decompose the action into several database operations.	91

Table B.7 ReCAPS Heuristics for Identifying and Removing Redundancies (Step 3.2)

Code	Heuristic	Page
RH _{redundancy1}	If one rule subsumes another rule, then the second rule is redundant and should be removed.	105
RH _{redundancy2}	Two logically equivalent rules that are specified in different modes (i.e., one <i>allow</i> rule and one <i>deny</i> rule) are redundant. One of them must be removed.	105
RH _{redundancy3}	If two rules have different (not conflicting) conditions but the same other elements, then combine the two conditions in one rule and remove the other rule.	106
RH _{redundancy4}	Design decisions often introduce new rules and cause some candidate rules to be redundant. Redundant rules must be removed.	106

Table B.8 ReCAPS Heuristics for Identifying and Resolving Conflicts (Step 3.3)

Code	Heuristic	Page
RH _{conflict1}	If two rules have the same elements but have conflicting modes, then this is a modality conflict. Resolution of this kind of conflicts needs to go back to the sources from which these rules were derived.	107
RH _{conflict2}	If two rules define conflicting privileges for the same subject on two objects, in which one object subsumes the other object, then this is a partial conflict. There are two ways to resolve this kind of conflicts: (1) specify separate rules for each part of the object; or (2) group rules into an access control policy, in which every rule must be evaluated and satisfied for an access request to be granted.	108
RH _{conflict3}	If two rules have conflicting conditions but the same other elements, then this is a conditional conflict. Resolution of this kind of conflicts needs to go back to the sources from which these rules were derived.	109

Table B.9 ReCAPS Heuristics for Grouping AC Rules into ACPs (Step 4)

Code	Heuristic	Page
GH _{grouping1}	It is often useful to group all the rules that concern a subject's privileges to an object into one policy.	110
GH _{grouping2}	If a set of rules are grouped together to resolve conflicts (as described in Step 3.3), these rules must always be grouped together.	112

Appendix C

Evolution of the ReCAPS Method

This appendix contains the method summaries that were used in the four case studies. Version 1 is the version before the SPRAT case study was conducted (Pre-SPRAT). Version 2 includes the changes made in the SPRAT case study, which was used in the TDG case study (Pre-TDG). Version 3 is the Pre-SAC version and Version 4 is the Pre-ERS version.

C.1 ReCAPS Method Summary Version 1 (Pre-SPRAT)

Inputs for the case study:

- (1) SPRAT software requirements specifications (SRS) document, Version 1.09
- (2) SPRAT database design E/R diagram

Process for the case study:

Step 1: Read the SRS document, identify what objects need to be protected and associate them with affected stakeholders.

These objects are data that are considered private and sensitive by the stakeholders. These objects should not be publicly available and only authorized people can access them.

Needs heuristics to guide how to identify objects.

Step 2: Identify candidate attributes for each object.

Identification Heuristic: Refer to the database design if necessary.

Analysis Heuristic: Does any attribute need to be protected differently from the object? What are these attributes? How do they need to be protected? Classify the way of protection according

to Ponder policy types: authorization, obligation, delegation, and refrain policies. Treat each attribute as an object if it requires a different way to protect.

Step 3: Identify privacy related preferences associated with each object.

This step can be skipped in this case study because privacy preferences are not supported in the SPRAT.

Step 4: Identify possible actions on the objects and the responsible actors.

Identification Heuristic: Read SRS document to identify possible actions.

Identification heuristic: Scenario analysis might be helpful to identify more detailed actions.

Step 5: Identify conditions under which an actor is allowed to perform an action on an object.

Step 6: Identify obligations that the actor or system must fulfill if an access request is granted.

Step 7: Identify the reasons why an actor is performing an action or accessing an object and document these reasons as purposes. Example purposes include telemarketing, research and development.

Step 8: Document the results of Step 4-7 and specify access control policies.

C.2 ReCAPS Method Summary Version 2 (Pre-TDG)

Inputs for the case study:

- (1) TDG software requirements specifications (SRS) document, Version 2.0
- (2) TDG database schema design

(Note: The TDG case study was conducted in two separate sessions. The database schema design was not available in the first session. The analysts completed the case study in the second session after they obtained the database design.)

Mini-Tutorial:

Access control policies are data-centric rather than function-specific (e.g., logging into a system is not codified as an ACP). If a requirement does not describe *access* to some *data*, then we cannot derive any access control related information from this requirement.

An experienced analyst can go through the requirements, identifying AC elements at the same time. It is not necessary to follow the steps below in sequential order.

When we need to make a design decision to disambiguate requirements, this time of analysis is the right time to make such design decisions. However, sometimes we need to delay these design decisions until we create the AC matrix when it is hard to make the decision or the privileges given to a user might change. We believe it is easier to consider all the possible privileges a specific user may have when AC matrix is finished.

Observation: AC related information can be mainly derived from functional requirements. For the SPRAT case study, we were unable to derive any AC information from the NFRs. We need to determine if this is true for all systems to modify our methodology & heuristics accordingly.

Process for the case study:

Step 1: Read the Introduction section of the SRS document to develop a general understanding of the envisioned or planned system.

Identification Heuristic: It is often useful to look through the Introduction Section to identify stakeholders and actors.

Step 2: For each requirement in the SRS document:

Step 2.1: Identify those objects that need to be protected and associate them with the affected actors. These objects are data that are considered private or sensitive by the users. These objects should not be publicly available and only authorized users can access them.

Identification Heuristic: Objects can be identified by looking at the words that follow verbs.

Identification Heuristic: Every object identified in the SRS should also appear in the database design. This is very useful because it forces analysts to disambiguate requirements early on and commit to what the objects really are.

Step 2.2: Identify privacy related preferences associated with each object.

Step 2.3: Identify responsible actors and possible actions on the identified objects.

Identification Heuristic: If a requirement is stated as follows “The system shall allow SOMEONE to DO SOMETHING”, then the actor (=SOMEONE) and action (=DO) is very clear.

Identification Heuristic: If a requirement is stated as follows “The system shall provide/support the ability to DO SOMETHING”. Use of the word “ability” implies that a user triggers some action in the system. We need to ask requirements engineers to disambiguate the requirement by asking who will perform this action and who will be affected by this requirement. We can often rephrase the requirement in this way “The system shall allow SOMEONE to DO SOMETHING”.

Identification Heuristic: If a requirement is stated as follows “The system shall provide/support SOMETHING”, then we need to determine who will access what information if the system performs this action.

Identification Heuristic: When requirements are vague or when it is impossible to identify any actor/action directly from requirements, we can use scenario analysis to elaborate the requirements. The actor, action and object in an event can be mapped to the subject, action and object of an access control policy, respectively. The pre-conditions can be mapped to the condition of an access control policy.

Step 2.4: Identify conditions under which an actor is allowed to perform an action on an object. Some of these conditions are already defined in the requirements specifications using words such as when, before, after, unless.

Identification Heuristic: Does a user have to be authenticated before he/she can access the requested data?

Step 2.5: Identify obligations that the actor or system must fulfill if an access request is granted.

Step 2.6: Identify the reason why an actor is performing an action or accessing an object and document these reasons as purposes. Example purposes include telemarketing, research and development.

Step 2.7: Document the results of Step 2.1-2.6 and fill all access rights in an access control matrix.

Note: If tool support is available, the step is not necessary because we document everything throughout the analysis. If no tool support is available, we mark the elements on hard copies and then document this information after the analysis.

Step 3: Specify what permissions should be assigned to each actor (e.g., administrators, project managers, analysts and guests in the SPRAT case study). We need to determine access

rights for those actors that can be easily determined first. This process is the reconciliation and abstraction process.

Step 3.1: We need to go through the access control matrix and sort the actions according to objects (e.g. goals, scenarios, requirements).

When we have the SPRAT tool, it will greatly help analysts during this step of process. For example, the tool will allow analyst to sort permissions according to objects, which helps analysts to group permissions or remove redundant permissions.

Step 3.2: We need to remove redundant permissions.

Example: For analyst, we removed several redundant permissions, which are basically the same but derived from different requirements. For example, “view contexts of goals” is contained in the permission “view elements of goals”. We can remove the permission “view contexts of goals”.

Step 3.3: We need to merge existing permissions based on implementation decisions. If possible, we may need to create new permissions.

Example: We merge “create project managers, create analysts, create guests” into a new permission: “create user account”. At the same time, we create a new permission: “assign roles to users”. This is based on our implementation decision. We decide to employ RBAC to control of the access of users to resources. Thus, it makes sense to create users and assign roles to users. For “reset user password”, we simply convert this candidate permission as final permission.

Step 3.4: If there are conflicting rules, such as two rules with the same element but conflicting mode (i.e., allow vs. deny), we need to resolve the conflict.

Step 3.5: Convert those permissions that are independent and stable as final permissions.

Example: For project manager, we did not perform any merge permissions or create any new permission. We simply convert existing candidate permissions as final permissions. This is because the final permissions are independent.

Note: In step 2, we have delayed the decision of assigning which set of permissions to some users/roles (e.g., guests in the SPRAT case study). Now it is the right time to make the decision because at this time we have the access control matrix available and the next step is design and implementation.

Step 4: Decide how to implement access control.

Observation: To identify potential security vulnerabilities, it is helpful to investigate whether a user assumes privileges that conflict with the separation of duties principle.

C.3 ReCAPS Method Summary Version 3 (Pre-SAC)

Inputs for the case study:

- (1) SAC software requirements specifications (SRS) document, Version 1.9
- (2) SAC design document, Version 1.2
- (3) SAC Security Policy, Version 1.9
- (4) SAC Privacy Policy, Version 1.4

Process and heuristics for the case study:

Step 1: Read the Introduction section of the SRS.

The SRS for both projects contains introductory material, which yielded a general understanding of the envisioned system, the stakeholders and the end users. It is important for

system designers to understand the concerns of both end-users and stakeholders so they can control end-users' access to data accordingly.

Step 2: Scan the SRS to identify access control elements.

Before analysts can derive any access control elements, it is important to understand what is inside or outside the scope of access control. If a requirement involves a user's electronic access to some data in the system, then the user part is inside the scope of access control; otherwise, it is outside the scope. For example, a requirement may state physical security safeguards to control users' access to the building or system. This is outside the scope of access control. Additionally, when a requirement states a user's interaction with the system, only the user part is inside the scope of access control. If a requirement states something that is not enforceable in the system, then it is outside the scope of access control.

Actors, actions, objects, etc. can be identified using traditional inquiry-driven analysis. For each requirement in the SRS, follow steps 2.1 through 2.6 (see Figure C.1) to identify these elements. It is not necessary to follow these steps in sequential order, they merely serve as a checklist; an experienced analyst may perform some of these steps in parallel.

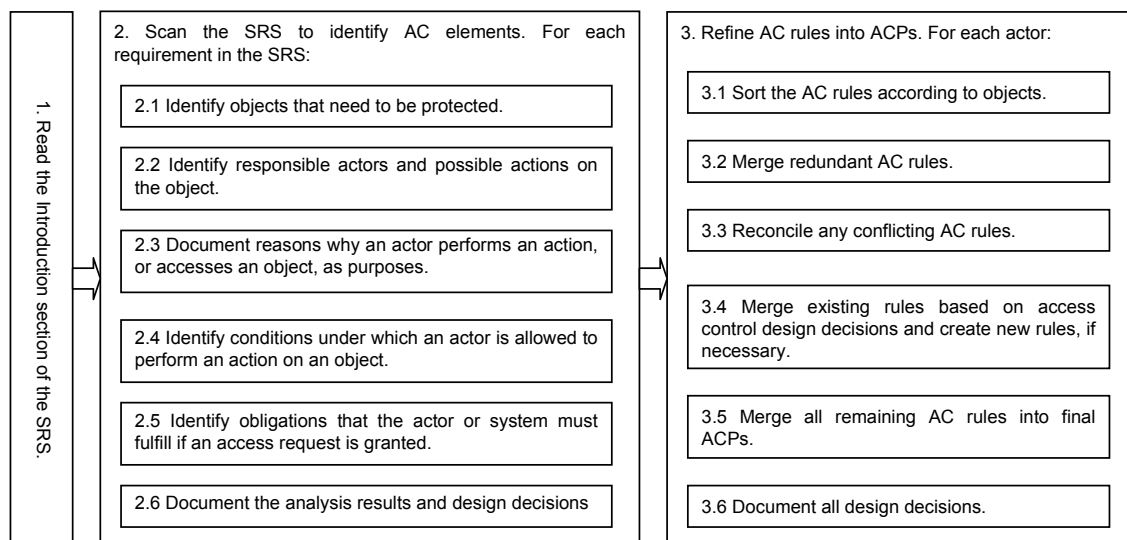


Figure C.1 The Process of Access Control Analysis

Step 2.1: Identify objects that need to be protected.

Objects are data to which access needs to be restricted. Three general heuristics (H1, H2 and H3) can be used for identifying these objects:

H1: Objects are nouns that can typically be identified by looking at the nouns that follow verbs.

Consider the following SPRAT requirement:

FR-GSM-3: The system shall allow analysts to classify goals.

The noun “goals” follows the verb “classify”, thus “goal” is tagged as an object. Note that because not every noun is an object, the following heuristic is used to distinguish access-related objects from other objects.

H2: Objects are system resources that should only be accessed by authorized actors.

In the case of FR-GSM-3 above, heuristic H2 ensures “goals” is the identified object to be protected instead of “analyst”.

H3: Every object identified in the SRS should also appear in the database design.

Heuristic H3 forces analysts to ensure that the requirements and database design are consistent with one another. Consider the following TDG requirement:

2.3.1: The system shall allow border immigration agents to determine if the traveler is on the “watch list”.

This requirement was annotated with stakeholder comments on what data is contained in the “watch list”. Three of these items were missing in the database schema: gender, the reasons for a person’s name being on the watch list, and actions to be taken if person whose name appears on the watch list is encountered at a border station.

Step 2.2: Identify responsible actors and possible actions on the object.

H4: If a requirement is stated as “The system shall allow <someone> to <do something>”, then the actor is <someone> and the action is <do>.

Consider the following TDG requirement:

2.5.2: The system shall allow border immigration agents to create a new record in the “watch list”.

The actor is *border immigration agent* and the action is *create*.

H5: If a requirement is stated as “The system shall provide/support the ability to <do something>”. The word “ability” implies that a user triggers some action in the system. We need to ask requirements engineers to disambiguate the requirement by asking *who will perform this action* and *who will be affected by this requirement*. We can often rephrase the requirement in this way “The system shall allow <someone> to <do something>”.

For example, the following requirement is in the SPRAT case study:

FR-GSM 6: The system shall support the ability to update an existing goal.

This requirement does not explicitly describe who can update existing goals. This is fine for requirements engineers, but it is ambiguous for system designer and software developers. By asking who will perform this action, we clarify that it is analyst’s responsibility to update existing goals. The requirement may be rephrased as follows:

FR-GSM 6 Revised: The system shall allow analysts to update an existing goal.

Then we can apply H4 to this requirement and identify the actor is *analyst* and the action is *update*.

H6: If a requirement is stated as “The system shall provide/support/allow/<do> <something>”, one must determine who will access what information if the system performs this action.

Consider the following requirement from the SPRAT case study:

FR-GSM 12: The system shall display the context of a goal.

This is a very ambiguous requirement. It is unclear in this requirement who can request the system to display this information and who can view the information displayed by the system. By asking requirements engineers to clarify that, we may rephrase the requirement as follows:

FR-GSM 12 Revised: The system shall analysts to view the context of a goal.

In this way, we can identify the actor is *analyst* and the *action* is view.

H7: When requirements are vague or when it is impossible to directly identify any actor/action, scenario analysis can be used to elaborate the requirements.

Consider the following SPRAT requirement:

FR-PM-3: The system shall support multi-user analyst results comparison.

This requirement was so ambiguous that without clarification it is impossible to understand what it means. Scenario analysis enabled us to identify the main events in this scenario as follows:

Event 1: Analysts classify goals independently according to predefined categories.

Event 2: Project managers select analysts whose classification results they wish to compare.

Event 3: The system shall display those goals that are classified differently and how they are different (e.g., by showing the different categories)

This analysis yields the following actors and actions, which are of interest for specifying ACPs:

Actor: Analyst *Action:* classify

Actor: Project Manager *Action:* request

Actor: Project Manager *Action:* view

Step 2.3: Document reasons why an actor performs an action, or accesses an object, as purposes.

This step is necessary only when the purpose of an access request affects grant/deny decisions. Purpose is often used in the context of privacy protection. Example purposes include telemarketing, payment, research and development. If the purpose of an access needs to be considered when making grant/deny decisions, we specify the purpose in the *condition* part of an AC rule as follows:

action.BusinessPurpose << object.DataPurpose

The symbol << means business purpose is contained in data purpose. Consider the following example. A particular piece of data (e.g., credit card information) is supposed to be used only for payment (data purpose). Given the condition in the ACP rule above, a data access request will be evaluated by an enforcement engine (to either grant or deny access); the business purpose of this access will be checked against the requested object's data purpose—payment.

Step 2.4: Identify conditions under which an actor is allowed to perform an action on an object.

Some of these conditions are clearly defined in the requirements specifications using words such as when, before, after, unless. Others are implicit. Consider SPRAT requirement FR-PM-3 mentioned previously. If a user assumes the role of *Project Manager* and *Analyst* at the same time, as an analyst, he can classify goals; as a project manager, he can view other analysts' classification results. However, access to the information (classification results) is withheld until he is finished with his own classification of the goals. This condition must be satisfied before a Project Manager can view the classification results. We document this rule as follows:

IF Role (user, Project Manager) = TRUE AND Role (user, Analyst) = TRUE AND
user.scheduledToClassify = TRUE AND user.classifyingFinished = FALSE

THEN viewClassificationResults = DENY

Step 2.5: Identify obligations that the actor or system must fulfill if an access request is granted.

Analysts often make design decisions during the ACP specification process. For example, the SPRAT SRS clearly states that the system shall support four system access levels: administrator, project manager, analyst and guest. Given that access to the system is restricted to these four levels (or roles), RBAC seems suitable for implementing access control. In our

analyses, we made additional design decisions. For example, instead of building a role hierarchy, we ensured that the privileges assigned to each role never overlap.

During Step 2, analysts produce a set of candidate AC rules that are used in Step 3 to specify ACPs.

Step 3: Refine AC rules into ACPs

Recall that an access control *policy* is comprised of a set of access control *rules*. The access control rules derived from different requirements could be redundant or in conflict with each other. Thus, we have to reconcile the different rules identified in Step 2 and specify the collective privileges that should be assigned to each actor. We conduct the following analysis for each actor:

Step 3.1: Sort the AC rules according to objects (e.g., goals, scenarios, requirements).

Grouping rules according to objects allows analysts to identify redundant and conflicting rules more easily as described in Steps 3.2 and 3.3.

Step 3.2: Merge redundant AC rules.

As common in requirements specification, synonymous words are often interchanged. In the SPRAT SRS, for example, “view elements of goals” encompasses “view contexts of goals” because context is a goal element. These two rules were merged, yielding: “view elements of goals”.

Step 3.3: Reconcile any conflicting AC rules.

If the requirements from which AC rules were derived are in conflict with one another, the resulting AC rules may conflict as well. Thus, any conflicts must be resolved at this stage of the analysis. For example, if two rules have the same elements but conflicting mode (i.e., allow vs. deny), these two rules conflict with one another.

Step 3.4: Merge existing rules based on access control design decisions and create new rules, if necessary.

In the SPRAT, we decided to employ RBAC to control users' access to data. Based on this decision, we merged the three privileges for the System Administrator role (create Project Managers, create Analysts and create Guests) into: "create user account" and created a new privilege: "assign roles to users". In this way, the rules we specified are consistent with the design decision and more flexible than the candidate rules.

Step 3.5: Merge all remaining AC rules into final ACPs.

The objective is to refine all the candidate rules into ACPs. The previous steps help eliminate redundant rules, reconcile conflicting rules, and refine rules according to decision decisions. Any remaining AC rules should be merged as ACPs. For example, in the SPRAT, we did not merge or create any privilege for the Project Manager role. Instead, we simply merged the AC rules into ACPs because these rules did not fit any of the cases described in Steps 3.2 through 3.4.

Step 3.6: Document all design decisions.

Recall that analysts make important design decisions during ACP specification. Flexible access control is often desirable. In the SPRAT, for example, *Project Manager* can specify what information within the system is accessible to *Guests* and how the information can be accessed. But, increasing flexibility may also increase access control implementation complexity as well as system development costs. Analysts must conscientiously make design decisions based on qualitative tradeoff analysis (e.g., between implementation complexity and access control flexibility). These design decisions must be documented.

The main artifacts produced by analysts during the ACP specification process for a given project are a set of ACPs, documented design decisions, as well as augmented requirements and database design specifications. As previously mentioned, we specify ACPs as a group of rules

that contain five elements: *<subject, object, action, condition, obligation>*. Table C.1 portrays a SPRAT ACP, which contains two access control rules: a *Deny* rule and an *Allow* rule. In this example, the ACP subject is a role—Project Manager. The ACP object is GoalClassificationResults, which can be mapped to table *clOptions* in the database design. The ACP action is *View*. In the *Allow* rule, Project Manager is allowed to view GoalClassificationResults under normal circumstances. However, the ACP also specifies a Deny rule that restricts the Project Manager’s ability to view GoalClassificationResults under a particular condition. Both rules will be evaluated by the access control enforcement engine to make grant/deny decisions when a data access request occurs.

Table C.1 Example ACP from the SPRAT Case Study

Policy No	Rule No	Mode	Subject	Action	Object	Condition	Sources
1	1	Allow	Role (Project Manager)	View	clOptions	user.loggedIn=TRUE	FR-PM-3
	2	Allow	Role (Project Manager)	View	clOptions	Role (user, Analyst) = TRUE AND user.scheduledToClassify = TRUE AND user.classifyingFinished = FALSE	FR-PM-3

C.4 ReCAPS Method Summary Version 4 (Pre-ERS)

The ERS case study was conducted after an initial version of the thesis was completed. Chapter 4 of this thesis was used as the method summary by the analysts for training purpose. Thus, no method summary is attached in this section.

Appendix D

Experimental Instrumentation for Empirical Study

This appendix contains the experimental instrumentation for the empirical study that we conducted in a graduate-level software engineering class (CSC 510) in Spring 2005. Section D.1 contains the NCSU Informed Consent Form for Research. Section D.2 contains the assignment description and worksheets that were available only to the ReCAPS group. Section D.3 contains the assignment description and worksheets that were available only to the control group. The two source documents (i.e., requirements specifications and database schema design) are proprietary information and not available to the public.

D.1 NCSU Informed Consent Form for Research

North Carolina State University
INFORMED CONSENT FORM for RESEARCH

Title of Study: Deriving access control policies from requirements specifications and database designs

Principal Investigator: Qingfeng He

Faculty Sponsor: Dr. Thomas L. Honeycutt
Dr. Annie I. Antón

We are asking you to participate in a research study. The purpose of this study is to investigate the effectiveness and usefulness of a research technique for specifying access control policies for information systems.

INFORMATION

If you agree to participate in this study, you will be asked to use the specified method to specify access control policies for a software system by analyzing two source documents. The estimated amount of time required for the total duration of the study is 4-6 hours.

RISKS

N/A

BENEFITS

By performing this study, you will (1) better understand the role of access control analysis in bridging the gap between requirements and design, and (2) be able to specify access control policies for information systems using the specified method.

CONFIDENTIALITY

The information in the study records will be kept strictly confidential. Data will be stored securely. No reference will be made in oral or written reports, which could link you to the study.

COMPENSATION

For participating in this study you will receive up to 2 points extra credit towards final grade for CSC 510. If you withdraw from the study prior to its completion, you will not receive any partial extra credit.

CONTACT

If you have questions at any time about the study or the procedures, you may contact the researcher, Qingfeng He, at 900 Main Campus Dr, Venture III Suite 165C-197, Raleigh, NC 27695-8207, or (Tel) 919.513.5082 (Email) qhe2@eos.ncsu.edu. If you feel you have not been treated according to the descriptions in this form, or your rights as a participant in research have been violated during the course of this project, you may contact Dr. Matthew Zingraff, Chair of the NCSU IRB for the Use of Human Subjects in Research Committee, Box 7514, NCSU Campus (919/513-1834) or Mr. Matthew Ronning, Assistant Vice Chancellor, Research Administration, Box 7514, NCSU Campus (919/513-2148)

PARTICIPATION

Your participation in this study is voluntary; you may decline to participate without penalty. If you decide to participate, you may withdraw from the study at any time without penalty and without loss of benefits to which you are otherwise entitled. If you withdraw from the study before data collection is completed your data will be returned to you or destroyed at your request.

CONSENT

“I have read and understand the above information. I have received a copy of this form. I agree to participate in this study with the understanding that I may withdraw at any time.”

Subject's print name _____

Student ID _____ (Not SSN)

Subject's signature _____

Date _____

Investigator's signature _____

Date _____

D.2 ReCAPS Group Assignment Description and Worksheets

Access Control Analysis Assignment

Checklist

CSC 510 Spring 2005
North Carolina State University

You are required to submit **all the following items** in order to be eligible to receive any credit for this assignment:

- _____ (1) The signed NCSU Consent Form for Research.
- _____ (2) Final Access Control Rules and Policies sheet, which contains a list of access control rules and policies.
- _____ (3) A revised version of the software requirements document. You can mark your changes directly on the document. You may also document any additional questions or comments using a blank paper.
- _____ (4) A revised version of the database schema design. You can mark your changes directly on the document. You may also document any additional questions or comments using a blank paper.
- _____ (5) The Inconsistencies between SRS and DB design sheet.
- _____ (6) The Time Effort form you fill in. Your grade will not be determined by the amount of time you spent on the assignment. However, if you fail to document your effort, you will lose 5 points.
- _____ (7) The Questions form with your answers on it.
- _____ (8) All scratch papers.

Access Control Analysis Assignment

Time Effort Form

CSC 510 Spring 2005
North Carolina State University

Date	Task Description ¹	Begin Time	End Time	Total Minutes

Summary of effort:

_____ Minutes on Understanding the assignment (____%)

_____ Minutes on Understanding the problem domain (Step 1) (____%)

_____ Minutes on Step 2 (____%)

_____ Minutes on Step 3 (____%)

_____ Minutes on Other activities (____%)

Total Effort: _____ Person-Hours

¹ Task Description: (1) Understanding the assignment, such as reading project description; (2) Understanding the problem domain, such as reading the SRS and DB schema design; (3) Step 2: deriving access control rules; (4) Step 3: refining rules into policies; (5) Other activities (please specify).

Access Control Analysis Assignment

Project Description

CSC 510 Spring 2005
North Carolina State University

1 Objective

The overall objectives of this assignment are to (1) understand the role of access control analysis in bridging the gap between requirements and design, and (2) derive access control rules (policies) from requirements specifications and database designs using the specified method. Please read this assignment description very carefully to ensure you are able to effectively apply the techniques below to specify your access control policies. Please keep track of the time you spend performing your analysis in the Time Effort Form. Although time is an important metric, the amount of time you spend on this exercise will not affect your grade in any way. This assignment should be completed independently. ***You may not talk with other students about the assignment. You may not read any other material besides this assignment package and class notes.***

2 Overview of Analysis Method

An access control (AC) rule contains four elements: $\langle \text{subject}, \text{object}, \text{action}, \text{condition} \rangle$. A **subject** is a user or a program agent, or any entity that may access objects. An **object** is a data field, a table, a procedure, an application or any entity to which access is restricted. An **action** is a simple operation (e.g. read or write) or an abstract operation (e.g. deposit or withdraw). An ACP may express additional **conditions** that must be satisfied before an access request can be granted. For example, in healthcare applications, the location from which the access request originates might affect the grant/deny decision. If an access request is from the emergency room, then the request may be granted. In this case, we can specify *the location of the request is emergency room* as a condition for the AC rule. In addition to the above four elements, an access control rule can have various **modes** (e.g., permit/deny/oblige/refrain). This assignment requires you to only specify allow and deny rules. **Allow** rules authorize a subject to access a particular object. **Deny** rules explicitly prohibit a subject from accessing a particular object.

For this assignment, two source documents are needed to specify access control policies (ACPs): an **SRS** and a **database design** for a non-profit web portal system. We now detail the analysis process steps listed in Figure 1 using concrete examples from the Security and Privacy Requirements Analysis Tool (SPRAT) project. The SPRAT is a research tool that provides support for analyzing and specifying security and privacy requirements using goals and scenarios. The information stored in the centralized database is proprietary and needs to be protected. Thus, access control is critical in this system.

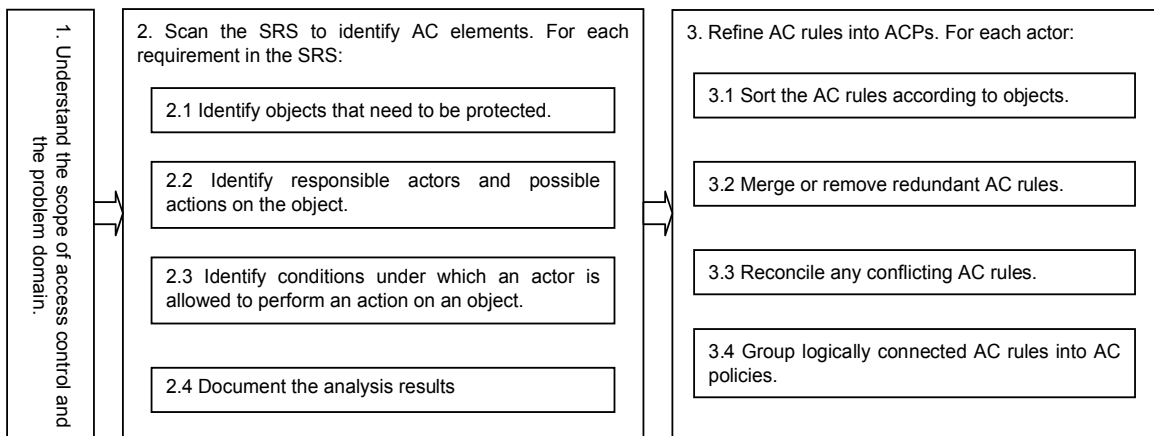


Figure 1. The process of access control analysis

Please use the steps below to guide you through your analysis effort. For each step, heuristics are provided to assist you.

Step 1: Understand the scope of access control and the problem domain

Step 1.1: Scope of access control

Understand what falls within the scope of access control. Consider the following general rules:

- (1) If a requirement involves a **user's** access to some **data** in the system, it is *inside* the scope of access control; otherwise, it is not.
- (2) A requirement that states **physical** security safeguards to control users' access to the building or the system is outside the scope. As are requirements that state something that is **manually** done.
- (3) Access control is not about controlling the system's capabilities. It is about controlling users' access to the system. For example, if a **user** requests some data from the system and the **system** shows the requested information. Then, the user part is within the scope of access control while the system part is outside the scope of access control.

Step 1.2 Read the SRS Introduction section.

An SRS often contains introductory material, which yields a general understanding of the envisioned system's problem domain, the stakeholders and the end users.

Step 2: Scan the entire SRS to identify access control elements

For each requirement, Steps 2.1-2.4 serve as a checklist; you may perform some of these steps in parallel.

Step 2.1 Identify objects that need to be protected.

H1: Objects are nouns that can typically be identified by looking at the nouns that follow verbs.

Consider the following SPRAT requirement FR-GSM-3:

FR-GSM-3: The system shall allow analysts to classify goals.

The noun "goals" follows the verb "classify", thus "goal" is tagged as an object.

H2: Objects are system resources that should only be accessed by authorized actors.

In the case of FR-GSM-3 above, heuristic H2 ensures "goals" is the identified object to be protected instead of "analyst".

H3: Every object identified in the SRS should be mapped to an object (e.g., a table, a column, a row, or a cell) in the database.

Heuristic H3 forces analysts to clearly define what the objects are in the database. For your assignment, every object in your set of derived access control rules should be **an object in the database**. H3 also ensures that the requirements and database design are consistent with one another. Consider requirement FR-UA-1:

FR-UA-1: The system shall allow the administrator to create user groups.

In this case, *user group* was specified in the SRS, but the corresponding data field had not been included in the database design. Therefore, the SRS and DB were inconsistent. You should identify such inconsistencies in your assignment.

Step 2.2 Identify responsible actors and possible actions for each object.

Actions can be classified as either database actions or abstract actions. **Database actions** are direct operations upon a database, such as insert, update, read, delete, whereas **abstract actions** are not, for example, withdraw (money from a banking account). Although both types of actions are important to access control, the objective of this assignment is to specify **database** actions. For this assignment, the allowable database operations are: VIEW (or READ, SELECT), CREATE, UPDATE (write to DB), INSERT, and DELETE.

H4: Given the requirement “The system shall allow <someone> to <do something>”, the actor is <someone> and the action is <do>.

H4.1: When the <do> action is a database operation, specify that action in the derived AC rule.

H4.2: Actions that are not database operations must be decomposed into database operations using scenario analysis.

Consider requirement FR-SSM-1:

FR-SSM-1: The system shall allow analysts to add a scenario in the repository.

The actor is *analyst* and the action is *add (insert)*. Consider requirement FR-GSM-3:

FR-GSM-3: The system shall allow analysts to classify goals.

Classify is an abstract action that needs to be decomposed into DB actions. We can elaborate this action using a scenario (a sequence of events). For example, to classify goals, analysts: (1) retrieve goals from the database to a local client, (2) change some attributes of goals at the local client, and (3) save the changes in the database. Event (1) is an obvious *read* DB operation and event (3) is actually an *update* DB operation. Event (2) is a local action that does not involve DB operations, so there is no need to specify it in access control policies.

H6: When a requirement is vague or when actor/action cannot be easily identified, use scenario analysis to elaborate the requirements.

Consider requirement FR-PM-3:

FR-PM-3: The system shall support multi-user analyst results comparison.

Without clarification it is difficult to understand what this requirement means. Using scenario analysis we identify the main events for this requirement as follows:

Event 1: Analysts classify goals independently according to predefined categories.

Event 2: Project managers select results of analysts whose classifications they wish to compare.

Event 3: The system shall display those goals that are classified differently and how they are different (e.g., by showing the different categories)

The actors identified in this step are specified as *subject* in your final set of ACPs for this assignment (see Table 1).

Step 2.3 Identify conditions under which an actor is allowed to perform an action on an object.

H7: When a requirement specifies constraints (e.g., if, unless, when, during, before, after, etc.), specify the constraints as conditions.

Given the requirement “A doctor cannot access medical records of patients’ whom he/she is not responsible for *unless* the request comes from the emergency room”, *request comes from the emergency room* is a constraint that must be specified in the access control rule.

H8: Define conditions by evaluating whether the rule sufficiently protects the confidentiality and integrity of data.

For example, in the SPRAT, users must be logged in before they can view goals in the repository. Users can only view and update their own personal information. When a user is trying to insert a new goal, a condition for this access rule is that *the goal does not exist in the repository*.

H9: Define conditions by evaluating whether there is any database constraint for the action.

For example, when a user is trying to insert a new goal, a condition for this access rule is that *the goal does not exist in the repository*.

H10: Define conditions by evaluating the privacy concerns of the requirement.

Some data may be used only for certain purposes, which must be specified as a condition in the access control rule. For example, if a requirement states that credit card information can only be used to complete specific transaction, this should be specified as a condition in all the rules that involve credit card information: *purpose is to complete the specific transaction*.

Step 2.4 Document the analysis results and the source requirements of each rule.

For each rule you specify, please document from which requirement the rule was derived. This is important for maintaining traceability between policies and requirements.

Upon completion of Step 2, you will have a set of candidate AC rules to use in Step 3 to specify your final ACPs.

Step 3: Refine AC rules into ACPs

In Step 3, you need to specify ACPs from the AC rules you derived in Step 2. The AC rules produced in Step 2 may be redundant or conflict with one another. Thus, we need to remove redundant rules and resolve any existing conflicts before you can specify the final ACPs as follows:

Step 3.1 Sort AC rules from Step 2 according to subjects and then objects.

Sorting rules in this way allows you to identify redundant and conflicting rules more easily as described in Steps 3.2 and 3.3.

Step 3.2 Merge or remove redundant AC rules.

Some rules are redundant because they use synonymous words. In the SPRAT SRS, for example, “view elements of goals” encompasses “view contexts of goals” because context is a goal element. These two rules can be merged as: “view elements of goals”. Another type of redundancy stems from the mode of an access control rule because an *allow* rule can be specified as a *deny* rule if the conditions are reversed and vice versa. For example, *allow A to do B if condition C is satisfied* may also be specified as *deny A to do B if condition C is not satisfied*. These rules are redundant; thus only one is necessary. You can specify AC rules using the mode that is easiest for you. Sometimes combining both modes makes it easier to specify an access control policy (see Rule 2&3 for Policy 2 in Table 1).

Step 3.3 Reconcile any conflicting AC rules.

If the requirements from which AC rules were derived are in conflict with one another, the resulting AC rules may conflict as well. Thus, any conflicts must be resolved at this stage of the analysis. There are two kinds of conflicts:

- (1) *Modality conflicts*. In one rule the subject is allowed to perform an action on an object, while in another rule, the subject is denied to perform the same action on the same object. This type of conflicts results from requirements and can be resolved by removing one of the rules that is not true.
- (2) *Partial conflicts*. For example, in the SPRAT project, classification is an attribute of goals.

However, classification and goals have different access permissions for Guests as shown below.

Rule 1: Allow Guest to view goals.

Rule 2: Deny Guest to view classification of goals.

We resolve this conflict by grouping both rules as an access control policy. Both rules in the policy must be evaluated during enforcement (see Rule 2&3 for Policy 2 in Table 1).

Step 3.4 Grouping logically connected AC rules into ACPs

Your final ACPs should be a set of logically connected access control rules (see Table 1) as we now discuss. Rule 1 allows Analyst to create a new goal. The object *goals* is a table in the database that stores all the goals. Policy 2 contains two access control rules: a *deny* rule and an *allow* rule (see Step 3.3). They are grouped into one policy because they are logically connected (i.e., the same subject performs the same action on an *object* and on *part* of the object). Either rule alone cannot fully specify the security requirements. Both rules in the policy must be evaluated during enforcement. Policy 3 also contains two access control rules. Rule 5 allows a Project Manager to view table *clOptions*, which stores GoalClassificationResults in the database. Rule 4 denies a Project Manager to view GoalClassificationResults under certain circumstances.

Table 1. Example access control rules and policies (This is not a complete list)

Policy No	Rule No	Mode	Subject	Action	Object	Condition	Sources
1	1	Allow	Role (Analyst)	Insert	goals	user.LoggedIn=true and the goal does not exist in the database	FR-GSM 1
2	2	Deny	Role (Guest)	Read	goals.classification	Null	FR-GSM 11
	3	Allow	Role (Guest)	Read	goals	Null	FR-GSM 11
3	4	Deny	Role (Project Manager)	Read	clOptions	Role (user, Analyst)=true and user.scheduledToClassify=true and user.classifyingFinished=false	FR-PM 3
	5	Allow	Role (Project Manager)	Read	clOptions	user.LoggedIn=true	FR-PM 3

3 Attributes of high quality ACPs

The attributes of a high quality set of access control policies are as follows:

- (1) All possible access control rules are derived from the SRS.
- (2) Each rule is within the scope of access control as defined for this assignment (see Step 1.1).
- (3) Each action is a database operation and each object is an object in the DB (see Step 2.1 and 2.2).
- (4) The conditions for each rule are correctly specified and as completely as possible (see Step 2.3).
- (5) Each rule is traceable to the requirement(s) from which it is derived (see Step 2.4).
- (6) No two rules are redundant (see Step 3.2).
- (7) No two rules conflict with one another (see Step 3.3).
- (8) Logically connected rules are grouped together (see Step 3.4).

Please note that in Table 1, the rules are sorted according to subject and object. The action for each rule is a database action and the object is an object in the database. Each rule can be traced back to the requirement(s) it is derived from. Some rules are grouped into policies.

4 Tasks

This assignment package contains a software requirements document and a database schema design. You need to read this assignment description very carefully before you start. Your task is to derive a set of access control rules/policies from these two sources, specify the policies and rules in the format shown in Table 1, improve both source documents, and finally to answer the questions on page 15.

5 Grading

All of the items listed in the checklist (page 1) must be submitted in order to be eligible for any credit for this assignment. This assignment will count 2 points extra credit towards your final grade in CSC 510. As shown below, the main part of the assignment is the **quality** of access control rules. Be sure to evaluate your own results using the attributes described in Section 3. Your grade will be calculated as follows:

Access control rules: 60%
 Requirements specifications: 10%
 Database design: 5%
 Inconsistencies identified between the SRS and the DB design: 10%
 Effort form: 5%
 Answer questions: 10%

Access Control Analysis Assignment

Inconsistencies between SRS and DB design

CSC 510 Spring 2005
North Carolina State University

SRS	DB schema design	Inconsistency

Access Control Analysis Assignment

Final Access Control Policies and Rules (Page 1)

CSC 510 Spring 2005
North Carolina State University

PLEASE DRAW A HORIZONTAL LINE IN BETWEEN EACH RULE / POLICY

Policy No	Rule No	Mode	Subject	Action	Object	Condition	Sources

Access Control Analysis Assignment

Final Access Control Policies and Rules (Page 2)

CSC 510 Spring 2005
North Carolina State University

Policy No	Rule No	Mode	Subject	Action	Object	Condition	Sources

Access Control Analysis Assignment

Questions

CSC 510 Spring 2005
North Carolina State University

1. Did you find the analysis method you used useful and easy to follow for specifying access control policies? If so, how? If not, what aspect was difficult?
2. Did the heuristics help you perform the analysis efficiently? If so, how? If not, how they can be improved? Which heuristics were most helpful to you and why?
3. Did you develop any additional heuristics?
4. Which step or which part of the method can be improved?
5. Did you read other materials when completing the assignment? If so, please list them.

D.3 Control Group Assignment Description and Worksheets

Access Control Analysis Assignment

Checklist

CSC 510 Spring 2005
North Carolina State University

You are required to submit **all the following items** in order to be eligible to receive any credit for this assignment:

- _____ (1) The signed NCSU Consent Form for Research.
- _____ (2) Final Access Control Rules and Policies sheet, which contains a list of access control rules and policies.
- _____ (3) A revised version of the software requirements document. You can mark your changes directly on the document. You may also document any additional questions or comments using a blank paper.
- _____ (4) A revised version of the database schema design. You can mark your changes directly on the document. You may also document any additional questions or comments using a blank paper.
- _____ (5) The Inconsistencies between SRS and DB design sheet.
- _____ (6) The Time Effort form you fill in. Your grade will not be determined by the amount of time you spent on the assignment. However, if you fail to document your effort, you will lose 5 points.
- _____ (7) The Questions form with your answers on it.
- _____ (8) All scratch papers.

Access Control Analysis Assignment

Time Effort Form

CSC 510 Spring 2005
North Carolina State University

Date	Task Description ²	Begin Time	End Time	Total Minutes

Summary of effort:

_____ Minutes on Understanding the assignment (____%)

_____ Minutes on Understanding the problem domain (____%)

_____ Minutes on ACP analysis (____%)

_____ Minutes on Other activities (____%)

Total Effort: _____ Person-Hours

² Task Description: (1) Understanding the assignment, such as reading project description; (2) Understanding the problem domain, such as reading the SRS and DB schema design; (3) ACP analysis: deriving access control rules; (4) Other activities (please specify).

Access Control Analysis Assignment

Project Description

CSC 510 Spring 2005
North Carolina State University

1 Objective

The overall objectives of this assignment are to (1) understand the role of access control analysis in bridging the gap between requirements and design, and (2) derive access control rules (policies) from requirements specifications and database designs. Please read this assignment description very carefully to ensure you are able to effectively apply the techniques below to specify your access control policies. Please keep track of the time you spend performing your analysis in the Time Effort Form. Although time is an important metric, the amount of time you spend on this exercise will not affect your grade in any way. This assignment should be completed independently. *You may not talk with other students about the assignment. You may not read any other material besides this assignment package and class notes.*

2 Overview of Analysis Method

An access control (AC) rule contains four elements: $\langle \text{subject}, \text{object}, \text{action}, \text{condition} \rangle$. A **subject** is a user or a program agent, or any entity that may access objects. An **object** is a data field, a table, a procedure, an application or any entity to which access is restricted. An **action** is a simple operation (e.g. read or write) or an abstract operation (e.g. deposit or withdraw). An AC rule may also express additional **conditions** that must be satisfied before an access request can be granted.

For this assignment, two source documents are needed to specify access control policies and rules: the **SRS** and **database design** for a non-profit web portal system. This assignment requires you to specify two kinds of rules: allow and deny rules. Logically connected AC rules can be grouped into access control policies (ACPs). **Allow** rules authorize a subject to access a particular object. **Deny** rules explicitly prohibit a subject from accessing a particular object. *Allow* or *deny* is called the **mode** of an AC rule.

The following requirement is taken from the Security and Privacy Requirements Analysis Tool (SPRAT) project. The SPRAT is a research tool that provides support for analyzing and specifying security and privacy requirements using goals and scenarios. The information (goals, scenarios, requirements, etc.) stored in the centralized database is proprietary and needs to be protected. Thus, access control is critical in this system. Consider requirement FR-SSM-1:

FR-SSM-1: The system shall allow analysts to add a scenario in the repository.

The actor is *analyst*, the action is *add (insert)*, and the object is *scenario*. We may derive an AC rule:

AC Rule 1:	
Mode:	Allow
Subject:	Analyst
Action:	Insert
Object:	Scenario
Condition:	Null

You may find ambiguous requirements in the SRS and problems in the database schema design. You may also find inconsistencies between the SRS and the DB design. You can improve both documents by resolving ambiguities and inconsistencies.

The main artifacts that you must produce for this assignment are a set of ACPs and the augmented requirements and database design specifications.

3 Attributes of high quality ACPs

The attributes of a high quality set of access control policies are as follows:

- (1) All possible access control rules are specified.
- (2) Each rule is within the scope of access control.
- (3) Each action is a database operation and each object is an object in the DB.
- (4) The conditions for each rule are correctly specified and as completely as possible.
- (5) Each rule is traceable to the sources from which it was derived.
- (6) No two rules are redundant.
- (7) No two rules conflict with one another.
- (8) Logically connected rules are grouped together.

4 Tasks

This assignment package contains a software requirements document and a database schema design. You need to read this assignment description very carefully before you start. Your task is to derive and specify a set of access control rules/policies from these two sources, improve both source documents, and finally to answer the questions on page 12.

5 Grading

All of the items listed in the checklist (see page 1) must be submitted in order to be eligible for any credit for this assignment. This assignment will count 2 points extra credit towards your final grade in CSC 510. Your grade will be calculated as follows:

Access control rules: 60%

Requirements specifications: 10%

Database design: 5%

Inconsistencies identified between the SRS and the DB design: 10%

Effort form: 5%

Answer questions: 10%

Access Control Analysis Assignment

Inconsistencies between SRS and DB design

CSC 510 Spring 2005
North Carolina State University

SRS	DB schema design	Inconsistency

Access Control Analysis Assignment

Final Access Control Policies and Rules (Page 1)

CSC 510 Spring 2005
North Carolina State University

Access Control Analysis Assignment

Final Access Control Policies and Rules (Page 2)

CSC 510 Spring 2005
North Carolina State University

Access Control Analysis Assignment

Questions

CSC 510 Spring 2005
North Carolina State University

6. Explain how you went about deriving ACPs from the SRS and the DB design. If you developed general guidelines (e.g., when I see x , I know it is y), please list them.
7. Did you read other materials when completing the assignment? If so, please list them.
8. Do you feel that you were efficient in specifying ACPs and identifying inconsistencies?
9. What aspects of specifying ACPs were difficult?
10. Did you find the ACP example useful?
11. Would more guidance be helpful?

Bibliography

- [AE01] A.I. Antón and J.B. Earp. Strategies for Developing Policies and Requirements for Secure Electronic Commerce Systems. In *E-Commerce Security and Privacy*, edited by A. K. Ghosh, Kluwer Academic Publishers, pp. 29-46, 2001.
- [AE04] A.I. Antón and J.B. Earp. A Requirements Taxonomy to Reduce Website Privacy Vulnerabilities. *Requirements Engineering Journal*, Vol. 9 (3), pp. 169-185, 2004.
- [AEB04] A.I. Antón, J.B. Earp, D. Bolchini, Q. He, C. Jensen and W. Stufflebeam. Financial Privacy Policies and the Need for Standardization. *IEEE Security & Privacy*, Vol. 2 (2), pp. 36-45, 2004.
- [AEC03] A.I. Antón, J.B. Earp and R.A. Carter. Precluding Incongruous Behavior by Aligning Software Requirements with Security and Privacy Policies. *Information and Software Technology*, Vol. 45 (14), pp. 967-977, 2003.
- [AEP01] A.I. Antón, J.B. Earp, C. Potts, and T.A. Alspaugh. The role of Privacy and Privacy Values in Requirements Engineering. *IEEE 5th International Symposium on Requirements Engineering (RE'01)*, pp. 138-145, 2001.
- [AEV05] A.I. Antón, J.B. Earp, M.W. Vail, N. Jain, C. Gheen and J.M. Frink. An Analysis of Web Site Privacy Policy Evolution in the Presence of HIPAA. Accepted, to appear in: *IEEE Security & Privacy*, 2005.
- [AHB04] A.I. Antón, Q. He, and D. Baumer. Inside JetBlue's Privacy Policy Violations. *IEEE Security & Privacy*, Vol. 2 (6), pp. 12-18, 2004.
- [AHK03] P. Ashley, S. Hada, G. Karjoth, C. Powers and M. Schunter. Enterprise Privacy Authorization Language (EPAL 1.1) Specification. *IBM Research Report*, 2003. <http://www.zurich.ibm.com/security/enterprise-privacy/epal>
- [Ale03] I. Alexander. Misuse Cases: Use Cases with Hostile Intent. *IEEE Software*, Vol. 20 (1), pp. 58-66, 2003.
- [Als02] T.A. Alspaugh. *Scenario Networks and Formalization for Scenario Management*. PhD thesis, North Carolina State University, Raleigh, NC, 2002.
- [AMP94] A.I. Antón, W.M. McCracken and C. Potts. Goal Decomposition and Scenario Analysis in Business Process Reengineering. *Proc. of the 6th International Conference on Advanced Information Systems Engineering (CAiSE'94)*, pp. 94-104, 1994.
- [ANS01] *American National Standard for Telecommunications: Telecom Glossary 2000*. American National Standard T1.523-2001, American National Standard Institute, 2001.
- [Ant96] A.I. Antón. Goal-Based Requirements Analysis. *Proc. of the 2nd IEEE International Conference on Requirements Engineering (RE'96)*, pp. 136-144, 1996.

- [Ant97] A.I. Antón. *Goal Identification and Refinement in the Specification of Software-Based Information Systems*. PhD thesis, Georgia Institute of Technology, Atlanta, GA, 1997.
- [AP98] A.I. Antón and C. Potts. The Use of Goals to Surface Requirements for Evolving Systems. *Proc. of the 20th International Conference on Software Engineering (ICSE'98)*, pp. 157-166, 1998.
- [AS99] G.-J. Ahn, R. Sandhu. The RSL99 language for role-based separation of duty constraints. *Proc. of the 4th ACM Workshop on Role-based access control (RBAC'99)*, pp. 43-54, 1999.
- [Bas92] V.R. Basili. Software Modeling and Measurement: The Goal Question Metric Paradigm. *Computer Science Technical Report Series*, CS-TR-2956 (UMIACS-TR-92-96), University of Maryland, College Park, MD, 1992.
- [BBF00] E. Bertino, P.A. Bonatti, and E. Ferrari. TRBAC: a temporal role-based access control model. *Proc. of the 5th ACM Workshop on Role-based access control (RBAC'00)*, pp. 21-30, 2000.
- [BCR94] V. Basili, G. Caldiera and D. Rombach. The Goal Question Metric Approach. *Encyclopedia of Software Engineering*. Wiley 1994.
- [Bez98] K. Beznosov. Requirements for Access Control: US Healthcare Domain. *Proc. of the 3rd ACM Workshop on Role-Based Access Control*, pp. 43, 1998.
- [Bib77] K.J. Biba. Integrity considerations for secure computer systems. *Technical Report MTR-3153*, Rev. 1, MITRE Corporation, 1977.
- [BJW02] C. Bettini, S. Jajodia, S. Wang, D. Wijesekera, Provisions and obligations in policy rule management and security applications. *Proc. of the 28th Int'l Conf. on Very Large Data Bases (VLDB'02)*, pp. 502-513, 2002.
- [BKL02] G. Brose, M. Koch, and K.-P. Löhr. Integrating Access Control Design into the Software Development Process. *Proc. of the 6th International Conference on Integrated Design and Process Technology (IDPT)*, 2002.
- [BL73] D.E. Bell and L.J. LaPadula. Secure computer systems: Mathematical foundations. *Technical Report MTR-2547*, Vol. 1, MITRE Corporation, 1973.
- [BL76] D.E. Bell and L.J. LaPadula. Secure computer system: Unified exposition and multics interpretation. *Technical Report MTR-2997*, Rev. 1, MITRE Corporation, 1976.
- [BN89] D.F.C. Brewer and M.J. Nash. The Chinese Wall Security Policy. *Proc. of the 1989 IEEE Symposium on Security and Privacy*, pp. 206-214, 1989.
- [Boe81] B.W. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.

- [Boo94] G. Booch. *Object-Oriented Analysis and Design with Applications*. 2nd ed., Benjamin/Cummings, Redwood City, CA, 1994.
- [Bro87] F. Brooks. No Silver Bullet: Essence and Accidents of Software Engineering. *IEEE Computer*, Vol. 20 (4), pp. 10-19, 1987.
- [Bro02] S. Brown. Uncloaking the Insider Threat. White paper, published at *ITtoolbox Security Knowledge Base*, 2002. <http://security.ittoolbox.com/documents/document.asp?i=2836>
- [CC97] L. Cholvy and F. Cuppens. Analyzing Consistency of Security Policies. *Proc. of the 1997 IEEE Symposium on Security and Privacy*, pp. 103-112, 1997.
- [Cho04] Choicepoint. Electronic Privacy Information Center, 2004. <http://www.epic.org/privacy/choicepoint/>
- [CHO99] S. Clarke, W. Harrison, H. Ossher, and P. Tarr. Subject-Oriented Design: Towards Improved Alignment of Requirements, Design and Code. *Proc. of the 14th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'99)*, pp. 325-339, 1999.
- [Chu93] L. Chung. Dealing with Security Requirements During the Development of Information Systems. *Proc. of the 5th International Conference on Advanced Information Systems Engineering (CAiSE'93)*, C. Rolland, F. Bodat, C. Cauvet (editors), LNCS 685, pp. 234-251, 1993.
- [CIN03] R. Crook, D. Ince, and B. Nuseibeh. Modelling Access Policies Using Roles in Requirements Engineering. *Information and Software Technology*, Vol. 45 (14), pp. 979-991, 2003.
- [CNY00] L. Chung, B.A. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, 2000.
- [Coc97] A. Cockburn. Structuring Use Cases with Goals. *Journal of Object-Oriented Programming*, Vol. 10 (5), pp. 56-62, 1997.
- [COP98] *Children's Online Privacy Protection Act of 1998*. Federal Trade Commission, 1998. <http://www.ftc.gov/ogc/coppa1.htm>
- [Coy96] E.J. Coyne. Role Engineering. *Proc. of the 1st ACM Workshop on Role-Based Access Control (RBAC'96)*, pp. 15-16, 1996.
- [CS96] F. Cuppens and C. Saurel. Specifying a Security Policy: A Case Study. *Proc. of 9th IEEE Computer Security Foundations Workshop*, pp. 123-134, 1996.
- [CS98] R. Chandramouli and R. Sandhu. Role Based Access Control Features in Commercial Database Management Systems. *Proc. of the 21st National Information Systems Security Conference*, 1998.
- [CSI02] The Computer Security Institute, 2002. <http://seclists.org/lists/isn/2002/Mar/0016.html>

- [CSO04] 2004 E-Crime Watch Survey Summary of Findings. Conducted by the CSO magazine, the US Secret Service, and CERT Coordination Center, 2004. <http://www.cert.org/archive/pdf/2004eCrimeWatchSummary.pdf>
- [CW87] D.R. Clark and D.R. Wilson. A Comparison of Commercial and Military Computer Security Policies. *Proc. of the 1987 IEEE Symposium on Security and Privacy*, pp. 184-194, 1987.
- [Dam02] N.C. Damianou. A Policy Framework for Management of Distributed Systems. *PhD Thesis*, Imperial College, London, 2002.
- [DD82] D. E. Denning and P. J. Denning, *Cryptography and Data Security*. Addison-Wesley, 1982.
- [Den76] D.E. Denning. A Lattice Model of Secure Information Flow. *Communications of the ACM*, 19 (5), pp. 236-243, 1976.
- [DL96] A. Dardenne and A. van Lamsweerde. Formal Refinement Patterns for Goal-Driven Requirements Elaboration. *Proc. of the 4th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (SIGSOFT 1996/ FSE-4)*, pp. 179-190, 1996.
- [DLF93] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-Directed Requirements Acquisition. *Science of Computer Programming*, Vol. 20 (1-2), pp. 3-50, 1993.
- [DM89] J.E. Dobson and J.A. McDermid. A Framework for Expressing Models of Security Policy. *Proc. of the 1989 IEEE Symposium on Security and Privacy*, pp. 229-239, 1989.
- [DM92] J. Dobson and M. Martin. Representation of Security Policy for a Telecommunications Application. *Proc. of the 8th International Conference on Software Engineering for Telecommunication Systems and Services*, pp. 87-92, 1992.
- [DPD97] Directive 97/66/EC of the European Parliament and of the Council of 15 December 1997 concerning the processing of personal data and the protection of privacy in the telecommunications sector. European Union, 1997. <http://europa.eu.int/ISPO/legal/en/dataprot/protection.html>
- [DW98] D.F. D'Souza and A.C. Wills. *Objects, Components and Frameworks with UML: The Catalysis Approach*. Addison-Wesley, 1998.
- [Eps02] P. A. Epstein. Engineering of Role/Permission Assignments. *Ph.D. Dissertation*, School of Information Technology and Engineering, George Mason University, Fairfax, VA, 2002.
- [ES01] P. Epstein and R. Sandhu. Engineering of Role/Permission Assignments. *Proc. of the 17th Annual Computer Security Applications Conference (ACSAC 2001)*, pp. 127-136, 2001.

- [ES99] P. Epstein and R. Sandhu. Towards A UML Based Approach to Role Engineering. *Proc. of the 4th ACM Workshop on Role-Based Access Control (RBAC'99)*, pp. 135-143, 1999.
- [FH97] E.B. Fernandez and J.C. Hawkins. Determining Role Rights from Use Cases. *Proc. of the 2nd ACM Workshop on Role-Based Access Control (RBAC'97)*, pp. 121-125, 1997.
- [FIP98] *Fair Information Practice Principles*. Privacy Online: A Report to Congress (Part III). Federal Trade Commission, <http://www.ftc.gov/reports/privacy3/fairinfo.htm>, June 1998.
- [Fis01] S. Fischer-Hübner. IT-Security and Privacy. *Lecture Notes in Computer Science 1958 (LNCS 1958)*, Springer-Verlag, 2001.
- [FK78] R.H. Franke and J.D. Kaul. The Hawthorne experiments: First statistical interpretation. *American Sociological Review*, 43, pp. 623-643, 1978.
- [FKC03] D.F. Ferraiolo, D.R. Kuhn, and R. Chandramouli. *Role-Based Access Control*. Artech House computer security series, 2003.
- [FKP03] A. Fuxman, R. Kazhamiakin, M. Pistore and M. Roveri. Formal Tropos: language and semantics. *University of Trento and IRST*, Trento, Italy, 2003.
- [FM04] I. Fundulaki and M. Marx. Specifying access control policies for XML documents with XPath. *Proc. of the 9th ACM Symposium on Access Control Models and Technologies (SACMAT'04)*, pp. 61-69, 2004.
- [FN93] J.W. Freeman and R.B. Neely. On Security Policy Modeling. *Proc. of the 8th Annual Conference on Computer Assurance: Practical Paths to Assurance (COMPAS'93)*, pp. 61-69, 1993.
- [FNH94] J.W. Freeman, R.B. Neely, M.A. Heckard. A Validated Security Policy Modeling Approach. *Proc. of the 10th Annual Computer Security Applications Conference*, pp. 189-200, 1994.
- [Fon01] P.-J. Fontaine. Goal-Oriented Elaboration of Security Requirements. *Project Dissertation*, Université Catholique de Louvain, Belgium, 2001.
- [GF94] O. Gotel and A. Finkelstein. An Analysis of the Requirements Traceability Problem. *Proc. of the 1st International Conference on Requirements Engineering (ICRE'94)*, pp. 94-101, 1994.
- [GH93] J.V. Guttag and J.J. Horning, with S.J. Garland, K.D. Jones, A. Modet, and J.M. Wing. *Larch: Languages and Tools for Formal Specification*. Springer-Verlag, 1993.
- [GLB01] *Gramm-Leach-Bliley Act: Financial Privacy and Pretexting*. Federal Trade Commission, 2001. <http://www.ftc.gov/privacy/glbact/index.html>

- [GS67] B. Glaser and A.L. Strauss. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine Publishing Company, 1967.
- [HA03] Q. He and A.I. Antón. A Framework for Modeling Privacy Requirements in Role Engineering. *Proc. of the 9th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'03)*, pp. 137-146, 2003.
- [HBM98] R.J. Hayton, J.M. Bacon, and K. Moody. Access control in an open distributed environment. *Proc. of the 1998 IEEE Symposium on Security and Privacy*, pp. 3-14, 1998.
- [HIP96] *The 1996 Health Insurance Portability and Accountability Act (HIPAA)*, <http://www.hhs.gov/ocr/hipaa/>.
- [HMA04] P. Hope, G. McGraw, and A.I. Antón. Misuse and Abuse Cases: Getting Past the Positive. *IEEE Security and Privacy*, Vol. 2 (3), pp. 90-92, 2004.
- [Hol97] G.J. Holzmann. The SPIN Model Checker. *IEEE Transactions of Software Engineering*, Vol. 23 (5), pp. 279-295, 1997.
- [IDE93] Integration Definition for Function Modeling (IDEF0), Draft Federal Information Processing Standards Publication 183, 1993. <http://www.idef.com/pdf/idef0.pdf>
- [IEE93] *IEEE Standards Collection: Software Engineering*. IEEE Standard 610.12-1990, IEEE, 1993.
- [HRU76] M.H. Harrison, W.L. Ruzzo, and J.D. Ullman. Protection in operating systems. *Communications of the ACM*, 19 (8), pp. 461-471, 1976.
- [Jac02] D. Jackson. Alloy: A Lightweight Object Modelling Notation. *ACM Transactions on Software engineering and Methodology*, Vol. 11 (2), pp. 256-290, 2002.
- [JAS04] N. Jain, A.I. Antón, W.H. Stufflebeam, and Q. He. Security and Privacy Requirements Analysis Tool (SPRAT) Software Requirements Specification. *NCSU Computer Science Technical Report TR-2004-7*, April 9, 2004.
- [JBC98] M. Jarke, T.X. Bui and J.M. Carrol. Scenario Management: An Interdisciplinary Approach. *Requirements Engineering Journal*, Vol. 3 (4), pp. 155-173, 1998.
- [JKS00] S. Jajodia, M. Kudo, V.S. Subrahmanian, Provisional Authorizations. *Proc. of the 1st Workshop on Security and Privacy in E-Commerce*, 2000.
- [JSS01] S. Jajodia, P. Samarati, M.L. Sapino, V.S. Subrahmanian. Flexible support for multiple access control policies. *ACM Transactions on Database Systems*, Vol. 26 (2), pp. 214-260, 2001.
- [JSS97] S. Jajodia, P. Samarati, and V.S. Subrahmanian. A Logical Language for Expressing Authorizations. *Proc. of the 1997 IEEE Symposium on Security and Privacy*, pp. 31-42, 1997.

- [Kai95] H. Kaindl. An Integration of Scenarios with their Purposes in Task Modeling. *Proc. of the 1995 Symposium on Designing Interactive Systems: Processes, Practices, Methods, and Techniques (DIS'95)*, pp. 227-235, ACM, Aug. 1995.
- [Kai00] H. Kaindl. A Design Process Based on a Model Combining Scenarios with Goals and Functions. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 30 (5), pp. 537-551, 2000.
- [Kav02] E. Kavakli. Goal-Oriented Requirements Engineering: A Unifying Framework. *Requirement Engineering Journal*, Vol. 6 (4), pp. 237-251, 2002.
- [KG03] D. Kulak and E. Guiney. *Use Cases: Requirements in Context*. 2nd edition, Addison-Wesley, 2003
- [KKC02] A. Kumar, N. Karnik, and G. Chafle. Context Sensitivity in Role-based Access Control. *ACM SIGOPS Operating Systems Review*, pp. 53-66, July 2002.
- [KS02] G. Karjoth and M. Schunter. A Privacy Policy Model for Enterprises. *Proc. of the 15th IEEE Computer Security Foundations Workshop*, pp. 271-281, 2002.
- [Lam01] A. van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. *Proc. of the 5th International Symposium on Requirements Engineering (RE'01)*, pp. 249-262, 2001.
- [Lam04] A. van Lamsweerde. Elaborating Security Requirements by Construction of Intentional Anti-Models. *Proc. of the 26th International Conference on Software Engineering (ICSE'04)*, pp. 148-157, 2004.
- [Lam74] B.W. Lampson. Protection. *Proc. of the 5th Princeton Symposium on Information Science and Systems*, pp. 437-443, 1971. Reprinted in *ACM Operating Systems Review*, Vol. 8 (1), pp. 18-24, 1974.
- [LDL98] A. van Lamsweerde, R. Darimont and E. Letier. Managing Conflicts in Goal-Driven Requirements Engineering. *IEEE Transactions on Software Engineering*, Vol. 24 (11), pp. 908-925, 1998.
- [LDM95] A. van Lamsweerde, R. Darimont and P. Massonet. Goal-directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt. *Proc. of the 2nd IEEE International Symposium on Requirements Engineering (RE'95)*, pp. 194-203, 1995.
- [LL00] A. van Lamsweerde and E. Letier. Handling Obstacles in Goal-Oriented Requirements Engineering. *IEEE Transactions on Software Engineering*, Vol. 26 (10), pp. 978-1005, 2000.
- [LL02a] E. Letier and A. van Lamsweerde. Agent-Based Tactics for Goal-Oriented Requirements Elaboration. *Proc. of the 24th International Conference on Software Engineering (ICSE'02)*, pp. 83-93, 2002.

- [LL02b] E. Letier and A. van Lamsweerde. Deriving Operational Software Specifications from System Goals. *Proc. of the 10th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (SIGSOFT 2002/ FSE-10)*, pp. 119-128, 2002.
- [LYM02] L. Liu, E. Yu, and J. Mylopoulos. Analyzing Security Requirements as Relationships Among Strategic Actors. *Proc. of the 2nd Symposium on Requirements Engineering for Information Security (SREIS'02)*, 2002.
- [LYM03] L. Liu, E. Yu and J. Mylopoulos. Security and Privacy Requirements Analysis within a Social Setting. *Proc. of the 11th International Requirements Engineering Conference (RE'03)*, pp. 151-161, 2003.
- [MCN92] J. Mylopoulos, L. Chung, B. Nixon. Representing and Using Nonfunctional Requirements: A Process-Oriented Approach. *IEEE Transactions on Software Engineering*, Vol. 18 (6), pp. 483-497, 1992.
- [MHN04] J.D. Moffett, C.B. Haley, and B. Nuseibeh. Core Security Requirements Artefacts. *Technical Report Number 2004/23*, Department of Computing, Open University, UK, 2004.
- [Mof99] J.D. Moffett. Requirements and Policies. *Proc. of Policy Workshop*, HP-Laboratories, Bristol, UK, 1999.
- [MS93] J.D. Moffett and M.S. Sloman. Policy Hierarchies for Distributed Systems Management. *IEEE Journal on Selected Areas in Communications*, Vol. 11 (9), pp. 1404-1414, 1993.
- [NCS88] National Computer Security Center (NCSC), *Glossary of Computer Security Terms*, N C S C - T G - 0 0 4 , V e r s i o n - 1 , O c t o b e r 1 9 8 8 . <http://www.radium.ncsc.mil/tpep/library/rainbow/NCSC-TG-004.txt>
- [NCS92] National Computer Security Center (NCSC), *A Guide to Understanding Security Modeling in Trusted Systems*, NCSC-TG-010, Version-1, J Williams primary author, October 1992. <http://www.radium.ncsc.mil/tpep/library/rainbow/NCSC-TG-010.txt>
- [NE00] B.A. Nuseibeh and S.M. Easterbrook. Requirements Engineering: A Roadmap. In A.C.W. Finkelstein (ed.) *The Future of Software Engineering*. (Companion Volume to *Proc. of the 22nd International Conference on Software Engineering*), pp. 35-46, 2000.
- [NS02] G. Neumann and M. Strembeck. A Scenario-driven Role Engineering Process for Functional RBAC Roles. *Proc. of the 7th ACM Symp. on Access Control Models and Technologies (SACMAT'02)*, pp. 33-42, 2002.
- [OAS05] *OASIS eXtensible Access Control Markup Language (XACML)*. Version 2.0, February 1, 2005. <http://www.oasis-open.org/committees/xacml>

- [OEC80] *OECD Guidelines on the Protection of Privacy and Transborder Flows of Personal Data*. Organization of Economic Cooperation and Development (OECD), 1980, <http://www1.oecd.org/publications/e-book/9302011E.PDF>.
- [P3P05] *The Platform for Privacy Preferences 1.1 (P3P1.1) Specification*. The World Wide Web Consortium (W3C), 4 January 2005. <http://www.w3.org/TR/2004/WD-P3P11-20040210/>
- [Pet78] L. Peters. Relating Software Requirements and Design. *Proc. of the Software Quality Assurance Workshop on Functional and Performance Issues*, pp. 67-71, 1978. (Also published in *ACM SIGSOFT Software Engineering Notes*, Vol. 3 (5), pp. 67-71, 1978.)
- [PP02] C.P. Pfleeger and C.L. Pfleeger. *Security in Computing*. 3rd ed., Prentice Hall, 2002.
- [Pre05] R.S. Pressman. *Software Engineering: A Practitioner's Approach*. 6th edition, McGraw-Hill, 2005.
- [PS02] J. Park and R. Sandhu. Towards Usage Control Models: Beyond Traditional Access Control. *Proc. of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT'02)*, pp. 57-64, 2002.
- [PS04] J. Park and R. Sandhu. The UCONABC Usage Control Model. *ACM Transactions on Information and Systems Security*, Vol. 7 (1), pp. 128-174, 2004.
- [PTA94] C. Potts, K. Takahashi and A.I. Antón. Inquiry-Based Requirements Analysis. *IEEE Software*, Vol. 11 (2), pp. 21-32, 1994.
- [RJB99] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.
- [Rob73] C. Robson. *Experiment Design and Statistics in Psychology*. Penguin Books Ltd., Harmondsworth, Middlesex, England, 1973.
- [RSA98] C. Rolland, C. Souveyet, and C. B. Achour. Guiding goal modeling using scenarios. *IEEE Transactions on Software Engineering*, Vol. 24 (12), pp. 1055-1071, 1998.
- [RSW00] H. Roeckle, G. Schimpf, and R. Weidinger. Process-Oriented Approach for Role-Finding to Implement Role-Based Security Administration in a Large Industrial Organization. *Proc. of the 5th ACM Workshop on Role-Based Access Control (RBAC'00)*, pp. 103-110, 2000.
- [RZF01] C. N. Ribeiro, A. Zuquete, P. Ferreira, and P. Guedes. SPL: An Access Control Language for Security Policies with Complex Constraints. *Proc. of Network and Distributed System Security Symposium (NDSS'01)*, pp. 89-107, 2001.
- [SAA03] W. Stufflebeam, A.I. Antón, and T.A. Alspaugh. SMaRT - Scenario Management and Requirements Tool. *Proc. of the 11th IEEE International Requirements Engineering Conference (RE'03)*, pp. 351, 2003.

- [SB99] Rini van Solingen and Egon Berghout. *The Goal/Question/Metric Method: A Practical Guide For Quality Improvement of Software Development*. McGraw-Hill, 1999.
- [SCF96] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, C.E. Youman. Role-Based Access Control Models. *IEEE Computer*, Vol. 29 (2), pp. 38–47, 1996.
- [Sch00] G. Schimpf. Role-Engineering Critical Success Factors for Enterprise Security Administration. *Proc. of the 16th Annual Computer Security Applications Conference (ACSAC'00)*, 2000.
- [Sha01] M. Shaw. The Coming-of-Age of Software Architecture Research. *Proc. of the 23rd International Conference on Software Engineering (ICSE 2001)*, pp. 657-664, 2001.
- [Sie73] S. Siegel. *Nonparametric Statistics for the Behavioral Sciences*. McGraw-Hill Kogakusha Ltd., 1973.
- [SL02] M. Sloman and E. Lupu. Security and Management Policy Specification. *IEEE Network*, Vol. 16 (2), pp. 10-19, 2002
- [SMJ01] A. Schaad, J. Moffett, J. Jacob. The Role-Based Access Control System of a European Bank: A Case Study and Discussion. *Proc. of the 6th ACM Symposium on Access Control Models & Technologies (SACMAT'01)*, pp. 3-9, 2001.
- [SO00] G. Sindre and A.L. Opdahl. Eliciting Security Requirements by Misuse Cases. *Proc. of the 37th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS-37'00)*, pp. 120-131, 2000.
- [Spi92] J.M. Spivey. *The Z Notation: A Reference Manual*. 2nd edition, Prentice-Hall, Englewood Cliffs, NJ, 1992.
- [SSH97] Site Security Handbook. RFC 2196, Network Working Group, Internet Engineering Task Force (IETF), 1997. <http://www.ietf.org/rfc/rfc2196.txt>
- [SV01] P. Samarati, S. De Capitani di Vimercati. Access Control: Policies, Models, and Mechanisms. *IFIP WG 1.7 International School on Foundations of Security Analysis and Design (FOSAD 2000)*, LNCS 2171, pp. 137-196, 2001.
- [UML05] The Unified Modeling Language Specification, Version 1.5. Object Management Group, 2005. <http://www.omg.org/technology/documents/formal/uml.htm>
- [WC98] K.M. Walker and L.C. Cavanaugh. *Computer Security Policies and SunScreen Firewalls*. Sun Microsystems Press, 1998.
- [Wes67] A. Westin. *Privacy and Freedom*. New York, 1967.
- [WK99] J. Warmer and A. Kleppe, *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley, 1999.

- [WPJ98] K. Weidenhaupt, K. Pohl, M. Jarke and Peter Haumer. Scenarios in System Development: Current Practice. *IEEE Software*, Vol. 15 (2), pp. 34-45, 1998.
- [YC02] E. Yu and L. Cysneiros. Designing for Privacy and Other Competing Requirements. *Proc. of the 2nd Symposium on Requirements Engineering for Information Security (SREIS'02)*, 2002.
- [YC03] E. Yu and L.M. Cysneiros. Designing for Privacy in a Multi-Agent World. In: *Trust, Reputation and Security: Theories and Practice*, R. Falcone, S. Barber, L. Korba and M. Singh (editors), LNCS 2631, Springer-Verlag, pp. 209-223, 2003.
- [YL01] E. Yu and L. Liu. Modelling Trust for System Design Using the i* Strategic Actors Framework. In: *Trust in Cyber-Societies - Integrating the Human and Artificial Perspectives*, R. Falcone, M. Singh, Y.-H. Tan (editors), LNCS 2246, Springer-Verlag, pp. 175-194, 2001.
- [Yu93] E. Yu. Modeling Organizations for Information Systems Requirements Engineering. *Proc. of the 1st IEEE International Symposium on Requirements Engineering (RE'93)*, pp. 34-41, 1993.
- [Yu97] E. Yu. Towards Modelling and Reasoning Support for Early-phase Requirements Engineering. *Proc. of the 3rd IEEE International Symposium on Requirements Engineering (RE'97)*, pp. 226-235, 1997.
- [Zav97] Pamela Zave. Classification of Research Efforts in Requirements Engineering. *ACM Computing Surveys*, Vol. 29 (4), pp. 315-321, 1997.
- [ZWC02] J. Zao, H. Wee, J. Chu, and D. Jackson. *RBAC Schema Verification Using Lightweight Formal Model and Constraint Analysis*. Massachusetts Institute of Technology, 2002.