HAL
open science

Spécification et animation de modèles de conception de la sécurité avec Z

Muhammad Nafees Qamar

To cite this version:

Muhammad Nafees Qamar. Spécification et animation de modèles de conception de la sécurité avec Z. Autre [cs.OH]. Université de Grenoble, 2011. Français. NNT : 2011GRENM057 . tel-00716404

HAL Id: tel-00716404

https://theses.hal.science/tel-00716404v1

Submitted on 10 Jul 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministériel : 7 août 2006

Présentée par

Muhammad Nafees QAMAR

Thèse dirigée par **Prof. Yves LEDRU** et
codirigée par **Dr. Akram IDANI**

préparée au sein du **Laboratoire LIG / INRIA Rhône Alpes**
dans **l'École Doctorale Mathématique, Science et Technologie de l'Information, Informatique**

Spécification et animation de modèles de conception de la sécurité avec Z

Thèse soutenue publiquement le **02 Décembre 2011**,
devant le jury composé de:

Mme, Régine, LALEAU Professeur à l'Université Paris-Est Créteil,	Rapporteur
M, Jean-Michel, BRUEL Professeur à l'Université de Toulouse,	Rapporteur
M, Jean Claude, FERNANDEZ Professeur à l'Université Joseph Fourier,	Président
Mme, Nicole, LEVY Professeur au CNAM,	Examinateur
M, Yves, LEDRU Professeur à l'Université Joseph Fourier,	Directeur de thèse
M, Akram, IDANI MCF à Grenoble INP,	Co-directeur de thèse

Acknowledgment

During the course of this research I benefited a lot from the expertise and guidance of my advisor, Professor Yves Ledru. I owe my deepest gratitude to him for his regular feedback on the scientific work. His clear research vision helped me enormously to achieve research objectives. I thank him for all his support when I needed the most. Without his mentorship, I would have not been able to finish this dissertation.

I am also thankful to my co-advisor, Dr. Akram Idani, for his involvement in the work and constructive feedback. It has been a real pleasure to work with him and learning pedagogical skills.

I consider myself fortunate to have worked with them. I hope we can continue to collaborate in the future.

Second, I would like to express sincere thanks to my readers, Professor Régine Laleau, Professor Jean-Michel Bruel, for this painstaking work. Special thanks go to the examiners, Professor Nicole Lévy, and Professor Jean-Claude Fernandez. I commend their invaluable suggestions and interesting perspectives on the work.

Next, I wish to thank all my colleagues in the VASCO team for their moral support. I also thank all of my friends for their assistance and good friendships. I am grateful to my Pakistani friends in Grenoble who shared some joyful moments with me in the Cricket ground, BBQs, and during other gatherings.

Financial support for the major part of my thesis was a research fellowship from INRIA Rhône-Alpes, and partly by ANR TASCCC project. I am their grateful for providing me this opportunity to reach the goal.

I would like to express my deepest gratitude to my parents whose prayers and continuous support is my source of success. I especially wish to thank my elder brother Idrees Zafar who played a vital role in my life and helped me everywhere. It is because of him I have attained this position.

In particular, I would like to express heartiest gratitude to my wife, Sahar; whose love, affection, and care played a pivotal role and gave me the strength to accomplish this task.

The best part of my life is my world, my daughter; Aaifa, whose smiling face always faded away the homesickness and worries, and pushed me to smile with her even when I did not want. I dedicate this thesis to her.

Finally, I thank Allah Almighty for His divine help which enabled me to achieve this milestone.

Specification and Animation of Security Design Models using Z

Abstract: Specifying security-critical software urges to develop techniques that allow early bugs detection and prevention. This is aggravated by the fact that massive cost and time are spent during product validation and verification (V&V). There exists a multitude of formal and informal techniques striving to confront the challenge of specifying and validating specifications. Our approach mainly concerns validating the security specifications by animating the formal models, which adds a new dimension to the state-of-the-art.

Secure system engineering dedicated to tackle security features offers security-design models to sketch secure applications. Generally for these, Unified Modeling Language (UML) is considered a de facto standard along with a few extensions such as SecureUML and Object Constraint Language (OCL). OCL tends to add precision in design but yet it remains far from obtaining bugs free specifications. One reason to that is the inability of the OCL-based techniques to animate models before proceeding to an implementation.

Combining formal languages such as Z with UML allows applying to apply animation techniques enabling early validation of software design. The RoZ tool is capable of translating UML models into the Z specifications which further can be verified or validated. But RoZ is lacking to provide similar features for secure applications. In this thesis, we have upgraded this tool using an underlying security kernel backed up by Role Based Access Control (RBAC). Our approach not only allows validating the specifications but can animate the formal models. The animation also takes into account both the static and the dynamic aspects (i.e., session management) of RBAC-based security policies. Our unified approach and toolset involves a systematic usage and linkage of UML, SecureUML, RBAC, RoZ, Z, and the Just Another Z Animator (Jaza) tool. Using Jaza, the sort of validation we perform allows enumerate user defined scenarios to determine if the specification describes the intended reality. We emphasize on simultaneous consideration of functional and non-functional properties and consider functional models as contextual constraints over the security models. From a user viewpoint, our proposed approach can arbitrarily be composed with any functional model to examine an RBAC-based security policy.

Spécification et animation de modèles de conception de la sécurité avec Z

Résumé: L'écriture de spécifications pour des logiciels en général et en particulier pour des applications sécurisées demande de développer des techniques qui facilitent la détection et la prévention des erreurs de conception, dès les premières phases du développement. Ce besoin est motivé par les coûts et délais des phases de vérification et validation. De nombreuses méthodes de spécification, tant formelles qu'informelles ont été proposées et, comme nous le verrons dans cette thèse, les approches formelles donnent des spécifications de meilleure qualité.

L'ingénierie des systèmes sécurisés propose l'utilisation de modèles de conception de la sécurité pour représenter les applications sécurisées. Dans de nombreux cas, ces modèles se basent sur les notations graphiques d'UML avec des extensions, sous forme de profils comme SecureUML, pour exprimer la sécurité. Néanmoins, les notations d'UML, même étendues avec des assertions OCL, sont insuffisantes pour garantir la correction de ces modèles. Ceci est notamment du aux limites des outils d'animation utilisés pour valider des modèles UML étendus en OCL.

Nous proposons de combiner des langages formels comme Z avec UML pour valider des applications en animant leurs spécifications, indépendamment de futurs choix d'implémentation. Le but de cette thèse est de présenter une approche pour analyser par animation des modèles de conception de la sécurité. Nous utilisons un outil pré-existant, RoZ, pour traduire les aspects fonctionnels du modèle UML en Z. Cependant, RoZ ne couvre pas la modélisation des aspects sécuritaires. Dans cette thèse, nous avons complété l'outil RoZ en l'associant à un noyau de sécurité qui spécifie les concepts du modèle RBAC (Role Based Access Control). Nous utilisons l'animation pour explorer dynamiquement et ainsi valider les aspects sécuritaires de l'application.

Notre approche et les outils qui la supportent intègrent UML, SecureUML (un langage de modélisation de la sécurité), RBAC, RoZ, Z et Jaza, un animateur pour le langage Z. L'animation des spécifications prend la forme de scénarios définis par l'utilisateur qui permettent de se convaincre que la spécification décrit correctement ses besoins. Notre approche permet une validation dès la phase de spécification, qui prend en considération l'interaction entre les modèles fonctionnel et sécuritaire, et qui fait abstraction des choix de l'implémentation. Les éléments du modèle fonctionnel peuvent être utilisés comme contexte dans la définition des permissions du modèle de sécurité. Notre approche ne met pas de contrainte sur ce modèle fonctionnel ce qui permet de l'utiliser pour une vaste gamme d'applications.

Contents

1 Introduction **1**
1.1 Research Context . . . 3
1.2 Motivations and Accomplishments . . . 5
1.2.1 Research Motivations . . . 5
1.2.2 Research Contributions . . . 5
1.2.3 Publications . . . 6
1.3 Dissertation Roadmap . . . 7
1.3.1 Part I: State of the Art . . . 7
1.3.2 Part II: Contributions . . . 8

I STATE OF THE ART **9**

2 Z and RoZ: An Introduction **11**
2.1 The Z Language . . . 11
2.1.1 The Z Basics . . . 13
2.1.2 Set Constructors . . . 14
2.1.3 Operations on Sets . . . 16
2.1.4 Closure . . . 16
2.1.5 State Constraints . . . 17
2.1.6 Schema Calculus . . . 18
2.2 RoZ Tool . . . 20
2.2.1 The Meeting SCHEDULER . . . 22
2.2.2 RoZ Translation Process . . . 23
2.2.3 RoZ Types . . . 24
2.2.4 RoZ Classes . . . 24
2.2.5 Translation of Relations . . . 25
2.2.6 Generation of Elementary Operations Using RoZ . . . 27
2.3 Using Jaza to Animate the Model . . . 31
2.4 Discussion . . . 31

3 Access Control Mechanisms & SecureUML **35**
3.1 Access Controls Mechanisms . . . 36
3.1.1 Mandatory Access Control (MAC) . . . 37
3.1.2 Discretionary Access Control (DAC) . . . 39

3.1.3 Role Based Access Control (RBAC) 40
3.2 Functional Specification Packages 44
3.2.1 Summarizing Access Control Mechanisms 46
3.3 Data Security Properties 47
3.3.1 Availability 47
3.3.2 Confidentiality 47
3.3.3 Integrity 48
3.4 Why to Choose RBAC over MAC and DAC? 48
3.5 UML-based RBAC specifications 49
3.5.1 SecureUML 49
3.6 Summary 52

4 Evaluating RBAC Supported Techniques and their Validation and Verification 53
4.1 Introduction 54
4.2 RBAC Coverage as Evaluation Criteria 55
4.2.1 RBAC Functions 55
4.2.2 RBAC Separation of Duties (SoD) Constraints 56
4.2.3 Other RBAC Variants 56
4.2.4 Verification and Validation Tools 56
4.3 Semi-formal Techniques 56
4.3.1 A Summary of Semi-formal Techniques 59
4.4 Formal Languages and RBAC Constraints 59
4.4.1 Alloy-based Approaches 61
4.4.2 Z-based Related Work 61
4.4.3 A Summary of Formal Techniques 62
4.5 Conclusion & Lessons Learned 62

II CONTRIBUTIONS 65

5 Validation of Security-Design Models Using Z 67
5.1 Introducing the Proposed Z-based Toolset 68
5.2 Illustrative Example : Medical Information System 69
5.3 Translating the Functional Model into Z 71
5.4 The Security Kernel 73
5.4.1 Permissions 74
5.4.2 Role Hierarchy 74
5.4.3 Action Hierarchy 75
5.4.4 Roles, Users and Sessions 77
5.4.5 Putting it All Together 78
5.5 Linking Functional and Security Models 80
5.6 Validating and Animating Secure Operations 81
5.6.1 Normal Behavior 81

5.6.2 Analyzing a Malicious Behavior . . . 82
5.7 Summary . . . 83

6 A Z-based Toolset For the Validation of Security Policies 85
6.1 Overview . . . 86
6.2 State of the Art Tools . . . 87
6.2.1 RBAC and SecureUML . . . 87
6.2.2 USE for the Validation of Security Policies . . . 87
6.2.3 SecureMOVA . . . 88
6.3 The Need for Dynamic Analyses . . . 89
6.4 Applying Toolset to Meeting SCHEDULER Example . . . 90
6.4.1 Input Models . . . 90
6.4.2 Diagrams for the Security Model . . . 91
6.4.3 Linking both Formal Models . . . 94
6.5 Animation of the Specification . . . 95
6.5.1 Queries on the Security Model . . . 96
6.5.2 Dynamic Analyses : Nominal Behaviors . . . 97
6.5.3 Further dynamic analyses . . . 100
6.5.4 Studying an Attack Scenario . . . 100
6.6 Conclusion . . . 102

7 A Set of Validation Queries 105
7.1 Formal Queries . . . 105
7.1.1 Authorized Roles for an Atomic Action . . . 105
7.1.2 Available Actions Against Roles . . . 107
7.1.3 Access to Resources . . . 108
7.1.4 Permissions Against Atomic Action and Role . . . 109
7.1.5 Finding Duplicate Roles . . . 109
7.1.6 Atomic Action Accessed by All . . . 110
7.1.7 Atomic Action Access by Nobody . . . 111
7.2 RBAC Supporting System Functions . . . 111
7.3 Summary . . . 113

8 Conclusion and Future Outlook 115
8.1 Conclusion . . . 115
8.2 Suggestions for Future Work . . . 116

A Appendix - Complete Formal Specification of Security Kernel 119
A.1 Abstract and atomic actions . . . 119
A.2 List of employed roles . . . 119
A.3 Resources . . . 119
A.4 Types . . . 120
A.5 Basic RBAC and other sets . . . 120

B Appendix - Formal Specifications of Health care Information System 125
B.1 RoZ Types 125
B.2 RoZ data 125
B.3 RoZ Operations 127
B.4 Secure Operations 137

C Appendix - Secure Operations of Meeting SCHEDULER Example 141
C.1 Z types 141
C.2 RoZ generated operations 141
C.3 Secure Operations of Meeting Scheduler Example 146
C.4 RoZ Data 150

Bibliography 153

List of Figures

2.1 The approach of RoZ tool 21
2.2 Use cases for the meeting scheduler 23
2.3 Meeting SCHEDULER Case Study 24
2.4 Jaza tool 33

3.1 MAC mechanism applied to the meeting SCHEDULER 38
3.2 Core RBAC [Ferraiolo *et al.* 2001] 40
3.3 Hierarchical RBAC [Ferraiolo *et al.* 2001] 43
3.4 SSD within Hierarchical RBAC [Ferraiolo *et al.* 2001] 44
3.5 Dynamic Separation of Duty relations [Ferraiolo *et al.* 2001] 44
3.6 Methodology for creating functional packages [Ferraiolo *et al.* 2001] . 45
3.7 SecureUML metamodel [Basin *et al.* 2009] 50
3.8 Class diagram and permissions for the meeting scheduler 51

5.1 Bird's eye view of Z-based toolset 68
5.2 Security policy model using SecureUML 69
5.3 Object diagram for the functional model produced from the output of Jaza 70

6.1 Users, roles and separation of duty for the meeting scheduler 92
6.2 A subset of the *perm_Assignment* table 94
6.3 Sessions with their users and roles 97
6.4 Object diagram for the meeting scheduler 97
6.5 Another Object diagram for the meeting scheduler 101

List of Tables

3.1 DAC mechanism applied to the meeting SCHEDULER 40
3.2 RBAC Configurations 42
3.3 RBAC Configurations: [C:Create, U:Update, D:Delete: R:Read] . . . 42

4.1 Evaluation table for semi-formal techniques 57
4.2 Evaluation table for formal techniques 60

5.1 Three sessions 78

Chapter 1

Introduction

Contents

1.1 Research Context **3**

1.2 Motivations and Accomplishments **5**

1.2.1 Research Motivations 5

1.2.2 Research Contributions 5

1.2.3 Publications 6

1.3 Dissertation Roadmap **7**

1.3.1 Part I: State of the Art 7

1.3.2 Part II: Contributions 8

Building reliable and correct information systems is inordinately hard despite of several standardized Software Development Life Cycle (SDLC) processes, because, software systems are often complex and mission critical [Keyes 2002]. Another qualified reason is that most software projects are started by quickly moving to code and subsequent debugging. The people do not carry out a proper requirements specification paperwork beforehand [Boehm 1984]. Specification is defined as the process of describing a system and its desired properties [Clarke & Wing 1996].

In spite of such a critical prerequisite, 80% of requirements specifications are English text documents, and the rest is probably whiteboard [Barjaktarovic 1998]. They are the biggest source of system engineering problems and impact all aspects of software development [Barker 2000]. It yet yields another bottleneck that 50% of the total software development budget is incurred during product Validation and Verification (V&V) [Barjaktarovic 1998]. The terms validation and verification are defined as follows:

- *Verification*: is intended to answer the question: did we build our system right? i.e., conformant to specifications. Its main purpose tends to ensure that the software fully satisfies all the expressed requirements.

- *Validation*: is according to ANSI/IEEE Standard [IEEE 1982], the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies the user needs. In other words, did we build the right system? i.e., conformance to the actual customer requirements.

One important and yet challenging task is the development of secure information systems which are intrinsically more complex than simple business applications due to additional data security challenges, and repercussions, e.g., data loss of a bank may result in the loss of customers. Equivalently, a tedious task can be to specify a valid and working security policy in a complex, distributed application environment or to verify its behavior. So the specifications may get more complex when dealing with secure systems, since it may involve ensuring the integrity or confidentiality of a secure software system. In this thesis, we confine ourselves to the study of such systems.

UML being a *de facto* standard is often employed to model requirements of a secure application on top of textual descriptions. But graphical models do not follow strong semantic rules and therefore their meaning can be ambiguous and subject to interpretation [Martin 2002]. This may end up in implementing some undesirable functionality by the developer. Such a perception led the design of OCL, being used to minimize the ambiguity of graphical models [Warmer & Kleppe 1998]. Alike, formal specification languages can also be used to set up the precision in specifications. Z [Spivey 1992], B [Abrial 1996] and Alloy [Jackson 2006] are alternative specification languages intended to be used for formal specification. Nonetheless, their coupling with graphical models has been an interesting area of research in the past decade.

The focal point of this dissertation is to propose an improved technique for formal specification and validation of secure applications. Our approach is placed on top of graphical models and is intended to avoid their multiple interpretations. It also provides implementation-independent validation of a security policy. In a nutshell, our intention is to graphically model, formally specify, and to validate the security properties of a system. The second distinguishing feature of our unified approach is to specify and validate security policies before proceeding to their implementations, because, implementation technology of a secure system may be by a choice. Similarly, a secure application can be a military system, web system or some banking system. Thus, enforcement of security rules may also vary in a wide variety of settings. For instance, in a library system an element of trust and the limited consequences of a security violation/breach may lead to flexible rules but one can not afford such a risk in military systems. So it becomes of interest to tackle security policies in an implementation-independent manner.

There are many security mechanisms available such as Mandatory Access Control (MAC) [Bell & LaPadula 1975], Discretionary Access Control (DAC) [Latham 1983] and Role Based Access Control (RBAC) [Ferraiolo *et al.* 2001] being used for secure application. We choose RBAC due to a number of advantages over MAC and DAC in terms of organizational productivity, enhanced systems security and integrity, and simplified regulatory compliance [Gallaher *et al.* 1996].

In the past, our group developed RoZ, a tool that transforms a UML class diagram, annotated with Z assertions, into a Z specification [Dupuy *et al.* 2000]. This tool can alleviate the ambiguity of graphical diagrams. But the tool is only limited to specify and verify the functional properties of an application. We extend

this tool to formally specify and validate security properties of an application. We emphasize on secure system engineering that requires a simultaneous consideration of functional and security properties. We consider elements of functional models as contextual constraints over the security models and magnify the important connections between them. It is to be noted that designing a software with secure features is indeed of great importance since it considered to be one of ten best practices towards secure software development [Paul 2009].

We propose a Z-based security kernel constituting the RBAC features that can arbitrarily be used with any functional model to examine a security policy. Using Jaza, the sort of validation we perform allows enumerating user defined scenarios to determine if the specification describes the intended reality. Jaza is a tool that can execute a large number of Z constructs.

The introduction is structured as follows: Section 1.1 narrowly explains the research context of the dissertation; Section 1.2 abstractly details motivations (and problem statement) with our accomplishments and contributions; Section 1.3 gives roadmap to the succeeding chapters. Overall, the approach taken in this thesis uses existing body of work where possible and develops new concepts when necessary.

1.1 Research Context

This work is situated in the areas of software engineering and formal methods, particularly the software design and the usage of formal specification languages to unambiguously specify and validate security-design models. UML and extended security profiles are becoming the de facto modeling languages in software industry [Toahchoodee *et al.* 2009]. UML can mainly be divided into structural and behavioral diagrams corresponding to static and dynamic views, respectively. The wide variety of diagrams includes Use Case Diagram, Class Diagram, Interaction diagrams (further divided into Sequence and Collaboration diagrams), State and Activity Diagrams, and Physical Diagrams constituting Component and Deployment diagrams. However, software practitioners do not like to have overly designed applications [Martin 2002] and thus pick a subset of these, essentially the class diagrams. An obvious reason to this approach is to avoid spending extra time and the money on software design.

Class diagrams have become indispensable as they provide a sufficient amount of detail on application classes, packages and objects, and their relationship depicted as containment, inheritance, associations, and others. These models also help design-teams to mutually collaborate and brainstorm [Barker 2000]. As an advantage, Barker further recalls that requirements models reduce cost as well as probability of overruns, in addition to help with support and sustainment of the product.

Software design is also expected to be free from bugs, and bugs should not be proliferated in the implemented systems: yet this is not promised by class diagrams. Research community have already devised solutions [Richters & Gogolla 2000] to validate and verify UML models [Lilius & Paltor 1999]. Verification of UML mod-

els is needed to avoid behavior not expected by the designer [Lilius & Paltor 1999]. For example, vUML [Lilius & Paltor 1999] converts UML models into PROMELA specification and invokes a model checker named SPIN to verify the models. Similarly, UML-based Specification Environment (USE) tool validates UML models augmented with OCL constraints [Gogolla *et al.* 2007].

Often, making a system entirely reliable is economically and technically impractical (except for avionics or some other safety critical systems); nonetheless one can characterize solutions contributing reasonably toward reliable systems. One possible way of achieving such reliability is the lightweight use of formal methods since formal methods are precise and unambiguous. Formal specification languages have well-defined syntax and semantics which avoid having multiple interpretations of the specification. For example, Z focuses on specifying the functional aspects of sequential systems using mathematical structures such as sets, relations, and functions; state transitions are given in terms of a predicate linking the initial and final states. The non-functional aspects of sequential systems may include performance, real-time constraints, security policies, and architectural design.

We particularly consider the non-functional aspects such as security policies by the use of Z in this thesis. Using Z with UML design is a so-called synonym to incorporate the precision of formal methods into intuitive graphical models. We can denote it as a lightweight application of formal methods due to number of reasons i.e., not having proofs but animation of models, and not having verification but validation. Actually the Lightweight formal methods do not entail a deep application of the technology. For example, formalism need not be applied in full depth to all components of an entire product and through all stages of their development [Woodcock *et al.* 2009].

Secure system engineering is a *concept* based on system and security engineering – that means functional and non-functional aspects of software systems should not be taken orthogonally. These days, such approaches are indispensable to rigorously address threats and malicious attacks in secure applications. Obviously, it seems sensible to address security right from the beginning. Recent advancements [Jürjens 2005] in security critical systems also show that security properties should not be retrofitted. We will discover later in the dissertation that functional aspects may influence non-functional aspects – which mean independent development of these concepts may originate serious risks to the secure assets.

Formal specification languages such as Z, B, and Alloy have been used for various industrial case studies. The essence of this use stems from the fact that specifications should be unambiguous. Hitherto, the mathematical skills required to use these languages are a genuine reason for their large-scale inadaptability. Having executable specifications (our Z formal specification are completely executable by tools such as Jaza.) on top of design can be interesting solution toward achieving a right blend of formal methods with design artifacts. Although, formal methods can also be used at any stage of a program development suchlike requirements capture, specification, validation test-case generation, refinement, analysis, verification, runtime checking – but their use is increasing at earlier stages such as specification and

design [Woodcock *et al.* 2009].

1.2 Motivations and Accomplishments

This section elaborates upon our research motivations, and subsequently discusses the corresponding accomplishments made to this end in the dissertation.

1.2.1 Research Motivations

In secure software systems, specifications include functional aspects, describing how information is processed, and security aspects, which describe who may access the functionalities of the information system. Separation of concerns tends to separate functional and security models. But since the security model refers to elements of the functional one, it is necessary to integrate them to fully address security concerns and perform more complete security analyses. Such models can adequately support the study of malicious attacks and threats, which if left unaddressed, can be confronted in the installed systems.

With the advent of new challenges from secure information systems several proposals have been made for security-design models [Basin *et al.* 2009]. There are not so many works except [Basin *et al.* 2009] that combinedly specify functional and security properties of an application. Our work is mainly influenced by this direction.

Further to this, till to-date V&V of security-design models based on RBAC have only discussed the static part of the software design and paid no attention to animate the dynamic aspects. The static aspects of RBAC-based systems include data structures that do not change over time while dynamic aspects include session management (Chapter 4 will describe these aspects in detail). However, it should be extended to cover V&V of dynamic aspects of these RBAC based security-designs models.

In addition, state of the art approaches enable querying the model but do not offer playing a sequence of actions over an implementation-independent specification of an application, which is definitely worth studying, because, it can simulate the steps of an attack scenario. We are also particularly interested to study how evolving states of a model leave behind security flaws potentially exploitable by the attackers. This way of analyzing and reasoning over models can be helpful to avoid systems to enter into the unauthorized states.

1.2.2 Research Contributions

The following points provide a summary of our published contributions to the specification, validation of security-design models by the animation technique:

- **Specification of security-design models:** As for SecureUML, we propose to express functional and security models as UML diagrams. These diagrams are then translated into a single formal specification, expressed in the Z language. RoZ tool is used to generate functional specification of an application.

We have devised new rules to generate security specifications of an application and link them to the functional specifications. The resulting Z specifications can be animated/executed using Jaza. Validation is performed by asking queries about the access control rules, as done in the SecureMOVA tool [Basin *et al.* 2009]. It is also achieved by playing scenarios, which lead the system through several state changes and involve both the security and the functional model. Such dynamic scenarios can exhibit security flaws, which cannot be detected by static queries. Jaza is a Z animation tool based on a combination of proof (simplification, rewriting) and search (generate and test) techniques. It covers a wide range of Z constructs and supports some level of non-determinism in the specifications (provided the search space is not too large).

- **Application level use of Z for RBAC:** As stated, Our security model is based on RBAC [Ferraiolo *et al.* 2001] and SecureUML [Basin *et al.* 2009]. Several attempts [Abdallah & Khayat 2006], [Yuan *et al.* 2006] have already been placed to specify RBAC in Z but most of them are the specifications of RBAC meta-model. To our knowledge, none of these has been used in conjunction with an animator in order to validate a given security policy. Our approach ensures a feedback on the blocked operations found in the specifications, which then can be used to improve the quality of a software application.

- **Validation/verification of dynamic aspects of RBAC based security policies:** Currently, several tools exploit OCL in order to validate RBAC or SecureUML specifications. Sohr et al [Sohr *et al.* 2008] have adapted the USE OCL tool [Sohr *et al.* 2005] for the analysis of security policies. SecureMOVA [Basin *et al.* 2009] is a tool dedicated to SecureUML, to validate RBAC based security policies. This tool allows querying the security policy, and can also evaluate which actions are permitted for a given role in a given context, depicted as an object diagram. Still, both tools only address the functional model statically, i.e., they do not animate the operations of the functional model and also partly cover the dynamic aspects of RBAC. We have particularly made progress toward this type of analysis. However, Sohr does animation of the administrative functions of RBAC. We will evaluate all such techniques in detail in Chapter 4 including the ones using Alloy such as [Hu & Ahn 2008], [Schaad & Moffett 2002], [Ahn & Hu 2007], [Zao *et al.* 2003] that speak about verifying and validating RBAC-base security policies.

1.2.3 Publications

- **N. Qamar**, Y. Ledru and A. Idani: Security Design Models: Formal Specification and Validation using Z, Formal Methods and Software Engineering - 13th International Conference on Formal Engineering Methods (ICFEM 2011), 25th-28th October 2011, Durham, United Kingdom.

- **N. Qamar**, Y. Ledru and A. Idani: Evaluating RBAC Supported Techniques and Their Validation and Verification, in the Proceedings of Fifth International Workshop on Secure Software Engineering (SecSE'11) in conjunction with the 6th IEEE International Conference on Availability, Reliability and Security (ARES'11), Vienna, Austria, August 22nd - 26th, 2011.

- Y. Ledru, **N. Qamar**, A. Idani, J. Richier and M. Labiadh: Validation of Security Policies by the Animation of Z Specifications, in Proceedings of the 16th ACM Symposium on Access Control Models and Technologies (SACMAT'11), Innsbruck, Austria, June 15-17, 2011.

- Y. Ledru, A. Idani1, J. Milhau, **N. Qamar**, R. Laleau, J. Richier and M. Labiadh: Taking into account functional models in the validation of IS security policies, in International Workshop on Information Systems Security Engineering (WISSE'11), associated with CAiSE'11, London, UK, June 2011., Publisher: Springer (LNBIP).

1.3 Dissertation Roadmap

The thesis has the following structure (excluding the Chapter 1): Part I includes chapters 2, 3, and 4 that actually refer to state of the art. Part II is based on chapters 5, 6, 7. Chapter 5 will first cover the underlying security kernel with the help of an example from medical domain, whereas a comparison of our toolset with SecureMOVA and USE will be discussed in Chapter 6. Chapter 7 will detail our offered set of formal queries.

1.3.1 Part I: State of the Art

- Chapter 2: mainly covers two things: it includes theoretical details on the Z formal specification language including fundamental concepts such as sets, relations, schema calculus besides some other important notions. It also details the RoZ tool which is used to generate formal specifications of functional models (modeled in UML). This chapter will also describe the Jaza tool.

- Chapter 3: The notions of access control mechanisms such as RBAC, Mandatory Access Control (MAC), Discretionary Access Control (DAC) are introduced in Chapter 3. This chapter also explains SecureUML, a UML profile with authorization constraints. We take an example of a meeting SCHEDULER to illustrate the underlying idea of these access control mechanisms.

- Chapter 4: This chapter provides an evaluation of RBAC supported verification and validation techniques with the help of proposed criteria based on RBAC features. It also details numerous state-of-the-art techniques by giving an overview of each of them.

1.3.2 Part II: Contributions

- Chapter 5: First, we proceed to expose the underlying work of our approach which is of formal nature using Z specification languages. This explains the security kernel that has been integrated with RoZ enabling a formal analysis of security policies and helps reporting bugs to the security engineer.

- Chapter 6: This is devoted to compare our work with SecureMOVA and USE tools and illustrates on the benefits and contributions we provide. The Z-based toolset is explained in detail and we highlight the importance of sequencing operations of an application to discover security flaws in them.

- Chapter 7: This chapter is actually a description of formal queries that we use in chapter 5 to elaborate our approach.

Conclusion and Suggestions for Future Work

- Chapter 8: This chapter shares some conclusions based on the work carried out in this thesis along with future perspectives.

Part I

STATE OF THE ART

Chapter 2

Z and RoZ: An Introduction

Contents

2.1 The Z Language **11**
2.1.1 The Z Basics 13
2.1.2 Set Constructors 14
2.1.3 Operations on Sets 16
2.1.4 Closure 16
2.1.5 State Constraints 17
2.1.6 Schema Calculus 18
2.2 RoZ Tool **20**
2.2.1 The Meeting SCHEDULER 22
2.2.2 RoZ Translation Process 23
2.2.3 RoZ Types 24
2.2.4 RoZ Classes 24
2.2.5 Translation of Relations 25
2.2.6 Generation of Elementary Operations Using RoZ 27
2.3 Using Jaza to Animate the Model **31**
2.4 Discussion **31**

This chapter combinedly presents the Z formal language, and the RoZ, essential to understand the approach proposed in this thesis. In Section 2.1, we first discuss the fundamental notions of Z to be used later in the dissertation. The translation (and annotation) process of RoZ from UML class diagrams to Z is introduced subsequently in Section 2.2 using the meeting SCHEDULER example.

2.1 The Z Language

The ISO standardized (2002) formal language Z has its foundations on first-order logic and Zermelo-Fraenkel (ZF) set theory. Z, pronounced as Zed, has abundant research literature [Spivey 1992], [Davies & Woodcock 1996] and case studies [Bowen 1996]. The Z language offers an extensive set of concepts and constructs from first-order logic and set theory. Several tools [Saaltink 1997], [Jia 2002] (to mention but two) can then be used to parse and interpret these Z notations.

The Z model of computation is based on a non-deterministic state transition machine. The model itself comprises of state variables and transitions, and transitions

are regarded as the changes of the variable values. The abstract Z specifications focus to describe what a system does rather than how it does it. The language has been successfully applied to various industrial projects for formal modeling and development [Bowen 2003].

There are two possible descriptions with Z, i.e., State Descriptions and Operation Descriptions:

- ***State Descriptions*** – State descriptions involve a set of variables that can have basic types such as integer and more complex data types like relations and functions. The state of variables presents the state of a system at any instant. So a state space of an application can be modeled using state schemas. Schema is a notation (kind of a box) that the Z notation includes for structuring specifications.

- ***Operation descriptions*** – Operation descriptions are considered a particular way of Z to depict affects on the state variables. The operations, similar to any programming language, may have precondition(s), but Z does not distinguish between pre- and postconditions. Operation schemas may or may not exert changes to the state. A simple output operation will have no affect on the state while an operation intended to modify some contents can update the state.

Z is a typed language which means that every variable in Z has a specific type (i.e., set from which it is drawn). These types allow a certain amount of machine checking of specifications to avoid 'obvious' errors using tools such as ZTC [Jia 2002]. It also helps avoid nonsense specifications [Bowen 2003]. On such given types, constraints or predicates can be applied. Schemas are the main structuring primitive in Z. A schema is divided into two parts i.e., signature part including variables and types where the second part is called predicate. Each schema is given a title in order to be called in further schemas or representing a particular component of the system state or an operation. Let's consider a standard example of Z specification borrowed from [Spivey 1992]:

$$
\begin{array}{|l}
\hline
\textit{BirthdayBook} \\
\textit{known} : \mathbb{P}\, \textit{NAME} \\
\textit{birthday} : \textit{NAME} \rightarrowtail \textit{DATE} \\
\hline
\textit{known} = \mathrm{dom}\, \textit{birthday} \\
\hline
\end{array}
$$

The schema BirthdayBook is simply a state schema that defines a state space of birthday records: known is the set of names with birthdays recorded, birthday is a function which, when applied to certain names, gives the birthdays associated with them. The part below the line gives a predicate/assertion which is true in every state of the system and is maintained by every operation on it. It states that the set known is the same as the domain of the function birthday. This is an invariant of the system.

FindBirthday

$\Xi BirthdayBook$
$name? : NAME$
$date! : DATE$

$name? \in known$
$date! = birthday(name?)$

An operation FindBirthday is used to find the date of birth of a particular person. It is also expressed as a schema. This schema illustrates two new notations. The declaration ΞBirthdayBook shows that this is an operation in which state does not change: the values known' and birthday' of the observations after the operation are same as values of known and birthday beforehand. The operators (?) and (!) denote input/output parameters, respectively. FindBirthday operation takes a name as input and yields the corresponding birthday as output. The precondition of the operation illustrates that name? is one of the names known to the system. If this is the case, the output date! is the value of the birthday function at argument name?.

2.1.1 The Z Basics

This section further explains needed background to understand Z specifications. Let's revisit the underlying structure (or skeleton) of BirthdayBook schema, which is actually comprised of D (declaration(s)) and P (Predicate(s)), as follows:

Schema_Name

D

P

P is also sometimes referred to as body of the schema. However, schemas can be defined in another format as convenient:

Schema_Name == [D | P]

A schema can contain a collection of variables and predicates depending upon the system to be modeled. A few notions associated to schemas are described in the following sections.

2.1.1.1 State Components

Constant sets represent the types in Z. As stated, the declaration part has state components which are introduced as under:

$var : Type$

2.1.1.2 Types in Z

Every set in Z specifications must be drawn from some basic type. This also applies to the empty set, i.e., there is a different empty set for each type. A basic type can be either enumerated type or given type. A given type is declared as [NAME] where NAME is a type. These types are further used to build complicated specifications using axiomatic definitions, state, or operation schemas. Z has a single predefined type integers($\mathbb{Z}$), and predefined operations like subtraction and addition. An enumerated type is defined as follows:

$$\begin{aligned} &\mathit{Weekdays} ::= \mathit{Monday} \mid \mathit{Tuesday} \mid \mathit{Wednesday} \mid \mathit{Thursday} \mid \\ &\qquad \mathit{Friday} \mid \mathit{Saturday} \mid \mathit{Sunday}. \end{aligned}$$

2.1.2 Set Constructors

Using Z, several set constructions are possible defining the relevant structures to model the system. These range from set extensions, power sets, set comprehension, Cartesian products, to relations and functions. These structures are then used to model sophisticated aspects of the real world systems. Some of them are given here:

2.1.2.1 Set Extension

We can introduce a new typed set extension of working days as following:

$$\mathit{Workings_Days} == \{\mathit{Monday}, \mathit{Tuesday}, \mathit{Wednesday}, \mathit{Thursday}, \mathit{Friday}\}$$

Workings_Days is a subset of the basic type Weekdays. It means that if a subset is sufficiently small, we can define it by an extension.

2.1.2.2 Set Comprehension

The set comprehension is a compact way to avoid cumbersome set specifications. Set comprehension is a general way of specifying sets. Following is the general form of this:

$$\{x : \mathit{Type} \mid \mathit{Predicate}(x)\} \text{ or } \{\mathit{Signature} \mid \mathit{Predicate}\}$$

where the signature may include several variables. An example is:

$$\{x : \mathbb{N} \mid x \bmod 2 = 0\} = (\{0, 2, 4, 6, ...\}).$$

In Z, it takes the following form as well:

$$\{\mathit{Signature} \mid \mathit{Predicate} \bullet \mathit{Expression}\}$$

where expression is any valid expression. This notation allows us to write more complex sets. For example, the set of squares of even numbers may be defined as:

$$\{x : \mathbb{N} \mid x \bmod 2 = 0 \bullet x * x\} = (\{0, 4, 16, 36, ...\}).$$

The expression $x * x$ works as the defining term for the set.

2.1.2.3 Power Set

A defined set in the Z specification may itself be a set of sets. For this we use a special notation, $\mathbb{P}\, S$ that denotes the set of all subsets of S. For example, power set of a singleton is: $P\{b\} = \{\varnothing, \{b\}\}$.

2.1.2.4 Cartesian Products

In order to build more complex types, Cartesian products can be used. If M and N are types then their Cartesian product is given as follows.

$M \times N$ denotes the type of ordered pairs (m, n) with $m : M$ and $n : N$. Formally:

$$M \times N == \{m : M;\ n : N \mid (m, n)\}$$

2.1.2.5 Relations

A relation is a subset of the Cartesian product of two sets. The type of all relations between M and N is denoted as:

$$(M \leftrightarrow N) \text{ and is the set } (\mathbb{P}(M \times N))$$

A relation can be given a name: $R : M \leftrightarrow N$. Each relation has a domain and a co-domain, defined formally as:

$$\operatorname{dom} R = \{x : M \mid (\exists\, y : N \bullet (x, y) \in R)\}$$
$$\operatorname{ran} R = \{y : N \mid (\exists\, x : M \bullet (x, y) \in R)\}$$

2.1.2.6 Functions

Mathematical functions are a special kind of relations. The Z toolkit has defined several function types such as partial injections, total injections, partial surjections and so on. A frequently used type of function is partial function which is formally specified as:

$$M \nrightarrow N == \{f : M \leftrightarrow N \mid (\forall\, m : M;\ n1, n2 : N \bullet$$

$$(m \mapsto n1) \in f \wedge (m \mapsto n2) \in f \Rightarrow n1 = n2)\}$$

Informally speaking, if a function relates an element from M to two elements of N, these two elements must be the same.

If every element of a set is related to one and only one element of another set, then the relation between the two sets is called a total function $f : M \rightarrow N$. Formally:

$$M \rightarrow N == \{f : M \leftrightarrow N \mid \forall\, m : M \bullet \exists_1\, n : N \bullet (m, n) \in f\}$$

2.1.3 Operations on Sets

Z provides a number of useful operations on sets and relations. The widely known sets operations are equality ($=$), membership ($\in$), subset ($\subseteq$), union ($\cup$), intersection ($\cap$), and set difference ($\setminus$).

Relation operations include composition of relations, range, domain and override etc. Hereunder, we just discuss the forward composition of relations using the standard definition from Z toolkit:

$$x \mapsto z \in R \fatsemi S \Leftrightarrow \exists\, y : Y \bullet x \mapsto y \in R \wedge y \mapsto z \in S$$

Two relations may only be composed into one if the target type of one relation matches the source type of the other relation.

2.1.4 Closure

Closure is a useful means of computing interesting properties from the relations having domain and range of the same type. Based on some preliminary information from relations, this helps adding maplets to a relation (R) until some meaningful property is achieved. For example, the information obtained from all finite iterations of R may be combined to form the relation R^+, where

$$R^+ = \bigcup\{n : N \mid n \geq 1 \bullet R^n\}$$

This is a transitive relation. For any positive natural number n, we may write R^n to denote the composition of n copies of R. Here, R is a homogeneous relation and transitive closure is denoted as R^+. The relation is said to be homogeneous if its source and target have the same type. An important homogeneous relation is the identity relation, defined by

$$\mathrm{id}\, X == \{x : X \bullet x \mapsto x\}.$$

R is a homogeneous relation from X to X, R^r is written to denote its reflexive closure:

$$R^r = R \cup \mathrm{id}\, X$$

This way, we can also consider the reflexive transitive closure of a homogeneous relation. If R is a relation of type $X \rightarrow X$, then we write R^* to denote the smallest relation containing R that is both reflexive and transitive.

$$R^* = R^+ \cup \mathrm{id}\, X$$

2.1.5 State Constraints

The components in the declaration part of a schema may be bound using constraints or invariants. These constraints can be expressed using Propositional Calculus or Predicate Calculus. A schema must always respect these constraints.

2.1.5.1 Propositional Calculus

Propositional logic is used to deal with alleged [Bowen 2003] facts in terms of statements that should either be true or false but not both simultaneously. The operators negation ($\neg$), conjunction ($\wedge$), disjunction ($\vee$), implication ($\Rightarrow$), and equivalence ($=$) are generally used to construct these propositions. Simple propositions may be turned into complex propositions that help cleanly specifying constraints of a system being modeled. Note that, all the notations including Propositional logic from Z have well-versed semantics.

2.1.5.2 Predicate Calculus

Quantification is one of the desired property when specifying complex systems. Z, with its toolkit offers these quantifying constructs. They can be summarized mainly into two categories:

- Universal Quantifier ($\forall$): Using this notation, we can specify that a property is true for all the objects of a particular set. For example, we can write a predicate that every natural number is greater than -1:

  $$\forall\, x : \mathbb{N} \bullet x > -1$$

- Existential Quantifier ($\exists$): Such quantifications help state that a property is true for at least one member of a particular set. For example, the following quantified expression states that there is some natural number *min* such that every natural number *num* must be greater than or equal to *min*:

$$\exists\, min : \mathbb{N} \bullet \forall\, num : \mathbb{N} \bullet num \geq min$$

2.1.6 Schema Calculus

Z specifications can be organized using the schema calculus. A number of operations are available to structure the specifications, for example, schemas can be included in each other. Schema inclusion also facilitates the combination of predicates. We take an example:

```
┌─ A ──────────────────────────────────────
│ x : T1
│ y : T2
├──────────
│ P(x)
│ Q(y)
└──────────────────────────────────────────
```

T1 and T2 are some types and P(x), Q(y) are the predicates of the schema A. In a Z schema, two predicates appearing on different lines are implicitly conjuncted or joined. Here, the predicate part of schema A is thus equivalent to $P(x) \wedge Q(y)$. The schema A can also be included in schema B.

```
┌─ B ──────────────────────────────────────
│ A
│ z : T3
├──────────
│ R(x, z)
│ S(y, z)
└──────────────────────────────────────────
```

The schema B is equivalent to the following schema C:

```
┌─ C ──────────────────────────────────────
│ x : T1
│ y : T2
│ z : T3
├──────────
│ P(x)
│ Q(y)
│ R(x, z)
│ S(y, z)
└──────────────────────────────────────────
```

The schema inclusion allows to merge the declarations and to conjoin the predicates of both schemas.

2.1.6.1 Operations and Δ & Ξ Notations

In Z, an operation is also represented as a schema. An operation schema specifies the state of components (i.e., in the declaration part) before the operation takes place, and the state after. In fact, the Z specification describes the changes made by the operation on the state of the abstract machine by stating a predicate that is satisfied by this pair of states (before and after). For example:

In schema Abc, x is defined and this schema has been included to operation Xyz (where x is incremented by 1).

Abc

$x : \mathbb{N}$

The operator ($'$) represents the variable x after the operation has taken place, while x corresponds to the state before the operation.

Xyz

Abc
Abc′

$x' = x + 1$

State' stands for State where all variables have been decorated with a (x'). The schema $\Delta State$ is a combination of State and State' schemas, where State is some schema.

$\Delta State$

State
State′

The Δ and Ξ notations are used to indicate the expected change/no change in a schema, respectively.

*Xyz*2

ΔAbc

$x' = x + 1$

Z provides a shortcut which comes in the form of Δ as a prefix operator with the schema being included. So the schema Xyz2 is an alternate form. In schema Xyz2, ΔAbc means that both states before and after of the schema Abc will be changed.

It is clear that when operations are made, a schema is added twice, with and without dashed variables (shown examples are: ΔState, Xyz). The notation Ξ works oppositely that leaves the variables of a included schema unchanged. For example, in schema Abc it did not occur any change thus x = x'.

$$\begin{array}{|l}
\hline
\Xi Abc \\
\Delta Abc \\
\hline
x = x' \\
\hline
\end{array}$$

$$\begin{array}{|l}
\hline
Z \\
\Xi X \\
\Delta Y \\
\hline
\end{array}$$

The schema Z shows that this operation schema will make no change to the variables of schema X while the schema Y will be updated.

So far, we have familiarized with fundamental notions of the Z formal language. Such a formal notation can be extremely suitable to specify software systems especially security-critical applications due to the fact that specifications would be precise enough. Also, there is less chance of occurring a misunderstanding over a software product among the development teams.

2.2 RoZ Tool

Z is a powerful and expressive language with precise syntax and semantics that has leveraged its benefits at different phases of software life-cycle. Z is often well-suited to data modeling due to offering constructs for structuring and compositionally building data-oriented specifications – with the help of schemas (state and operation), I/O parameters, and schema calculus – to compose sub-specifications. In this context, RoZ [Dupuy *et al.* 2000] brings the use of Z to specification phase on the basis of UML models. This section will explain the underlying concepts of RoZ.

RoZ targets structural elements of UML such as class diagrams. For structuring and composing data-oriented specifications, Z offers schemas notation which is effectively used in RoZ. RoZ generated models can then be verified (using Z-EVES [Saaltink 1997]) or validated using animation techniques [Boehm 1984] by tools like Jaza [Utting 2005]. The main intuitions behind the RoZ are:

- The expressiveness of UML graphical notations is limited and it becomes difficult to express constraints on the classes attributes (unless OCL or specification languages such as Z or Alloy is used);
- One may wish to have detailed specification of operations of a given model, implicitly contributing to the completeness of a model;
- The RoZ translation process encourages to use graphical models since these models are an easy way to obtain software design; yet the formal specification is generated from those models. In this spirit, graphical models can be a meaningful start to use formal notations, which somehow promotes the use of formal notation in industrial applications.

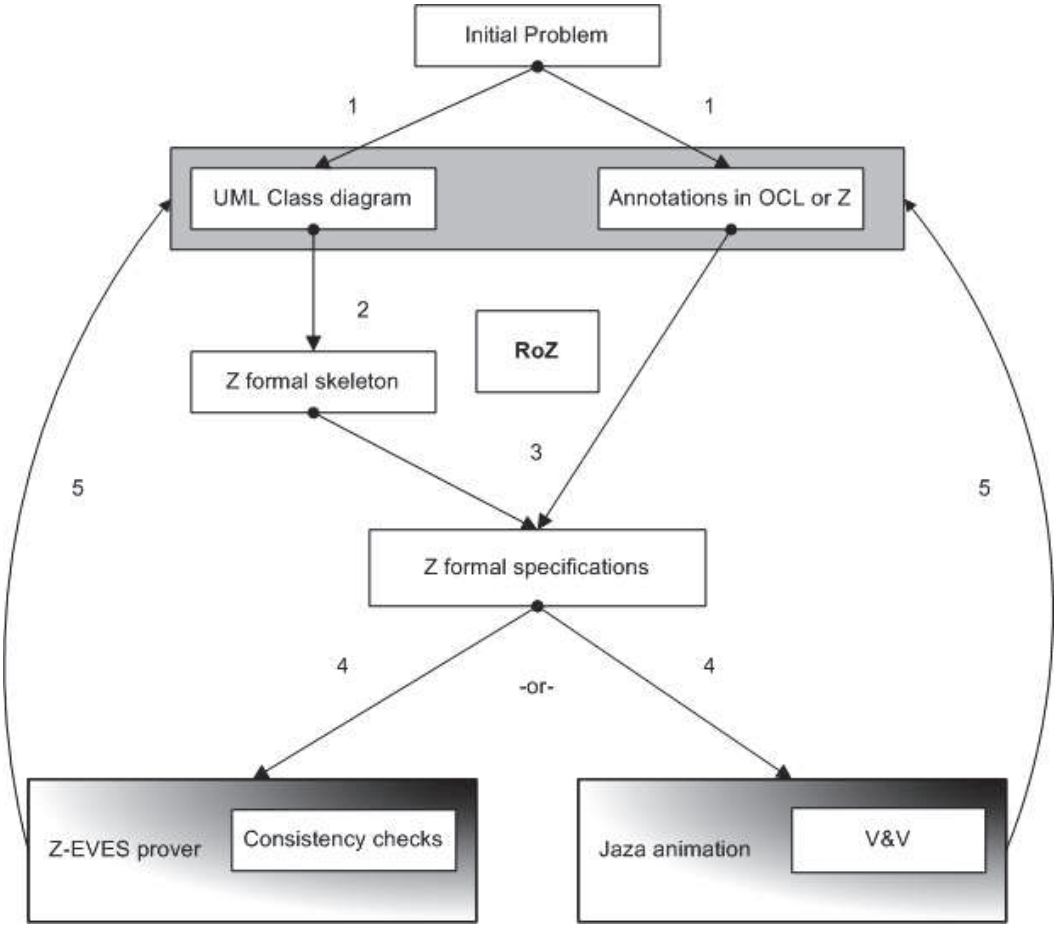

Figure 2.1: The approach of RoZ tool

RoZ uses the Rational Rose [IBM] environment (a current evolution uses TOPCASED)[1]. Conceivably, class diagrams can range from simple associations to complex constructs like specializations. RoZ has the ability to translate most constructs of class diagrams: class, operation, association, aggregation, composition, whilst inheritance to some degree. The underlying rules of RoZ (i.e., the prerequisites for a model to be accepted by it) help automated generation of formal specifications e.g., each class attribute must have a type and operations should have their pre-post predicates.

The RoZ process (Fig. 2.1) is composed of a three-step strategy integrating UML and Z:

– ***Generate Z specifications***: Corresponding to the step 1, we input to RoZ UML class diagrams and OCL/Z based annotations. In step 2, formal skeletons are obtained from UML class diagrams with their respective annotations.

[1] http://www.topcased.org

In step 3, we can obtain the augmented Z skeleton with the annotations, and denoted in the figure as Z formal specifications. The generated formal specifications are of Latex style and are saved in a separate file, which is easy to parse using Z tools;

- ***Generate elementary operations***: It generates some generic and fundamental operations that concrete classes may have. For instance, operations modifying the attributes can be common to several classes in an application. Note that, this is included in step 3 when the formal specifications are generated. This point is thoroughly detailed in Section 2.2.6;

- ***Generate theorems to validate operation guards*** [Ledru 1998]: After generating the basic operations, there can be some other constraints (missing during translation process) on data to be respected. Actually these constraints are implicit and included from schemas. So the tool helps to make them explicit. RoZ proposes to design guards to be evaluated before operation execution and so an operation fails if the constraints are violated when the operation is called. This helps achieving data integrity of an application. Afterwards, for each guard a theorem is generated. The theorems demonstration uses a semi-automatic tool Z-EVES [Saaltink 1997].

To illustrate our work, we consider a meeting SCHEDULER example used by Basin *et* al. to illustrate SecureUML and SecureMOVA tools [Basin *et al.* 2009]. We first elaborate upon this example which will be followed by the RoZ translation in Section 2.2.2.

2.2.1 The Meeting SCHEDULER

The meeting SCHEDULER helps users plan a "meeting" involving several "persons". Basically, the information system records information about persons, meetings, and the links between these. These links are (a) the ownership of a meeting by a person who organizes it, and (b) the participation of a given user in a meeting. Fig. 2.2 gives the major use cases of this system and the related actors. The major kind of actor is the system user. System users can create and cancel meetings, modify the meeting's information (e.g., change the time or duration of the meeting), add participants to a meeting, and notify the participants about the meeting (which performs some side-effecting operation such as sending a mail to the participants). The system administrator is another actor. Basically, he is responsible of managing information about the persons, i.e., the potential owners of and participants to the meetings. Supervisors are thus introduced as a specialization of system users. Another kind of actor is the Director, who is both a user and an administrator. The Fig. 2.3 is the class diagram used for the meeting SCHEDULER application.

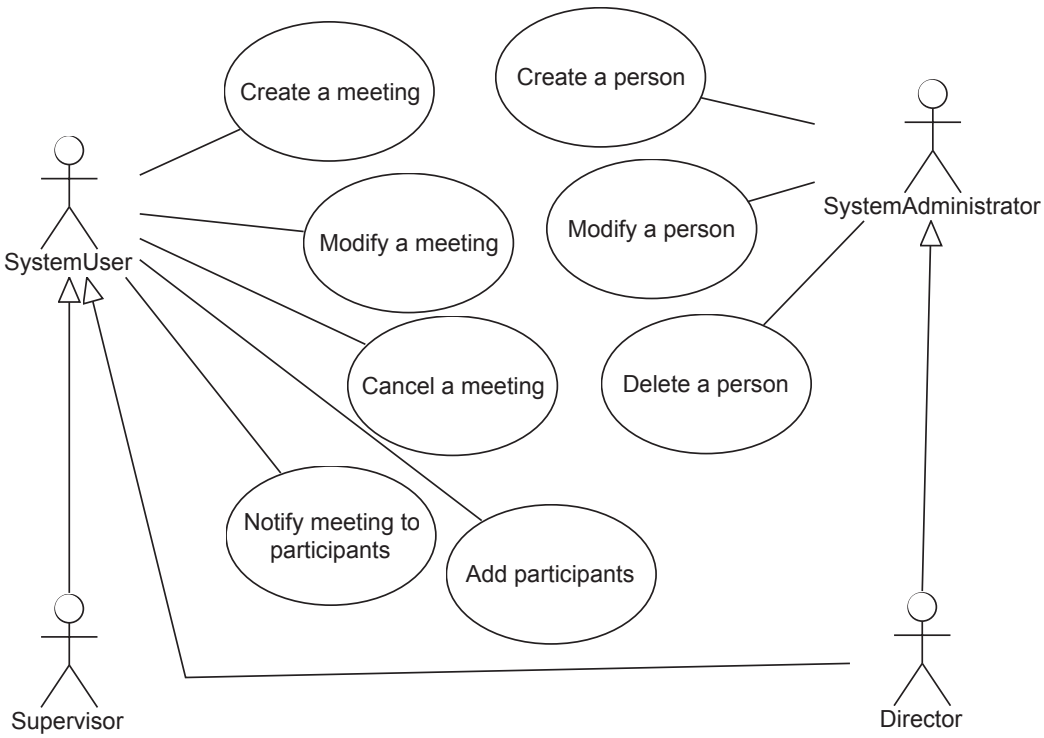

Figure 2.2: Use cases for the meeting scheduler

2.2.2 RoZ Translation Process

The translation process of RoZ is intended to generate annotated Z skeletons from UML constructs. The RoZ translation is quite systematic that can be summed up in the following steps:

- Specification of classes:

  a. identification of the useful types for the corresponding attributes;

  b. attributes of the classes;

  c. extensions of the corresponding sets.

- Specification of the relations between classes;

- System specifications as a whole.

The aforementioned steps are easy to follow. We have two classes named MEETING and PERSON. The first step is to list meaningful types for the attributes of these classes.

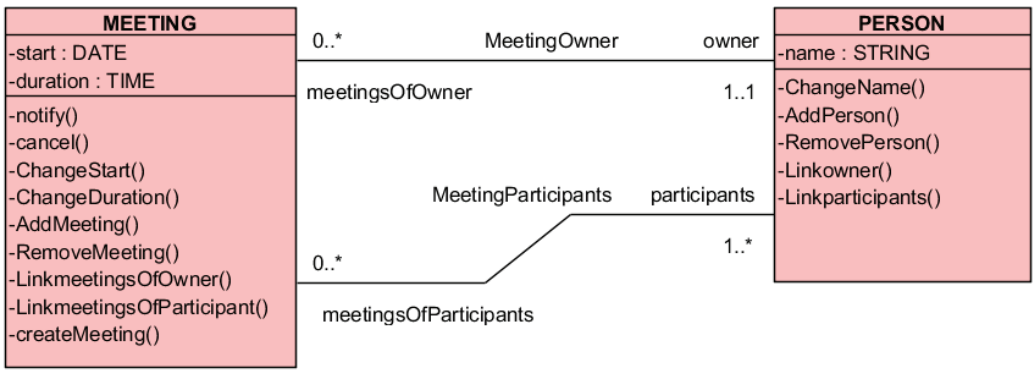

Figure 2.3: Meeting SCHEDULER Case Study

2.2.3 RoZ Types

For the given application, three types are introduced by the user for the attributes of MEETING and PERSON classes. For each attribute, a type must be chosen which can be a given type, an enumerated type or $\mathbb{Z}$, the set of integers. The chosen types of the example are given below:

$$[STRING]$$

$$DATE == \mathbb{Z}$$

$$TIME == \mathbb{Z}$$

STRING is a given type. *DATE* and *TIME* are a renaming of the integer type. Dates and time are thus the amount of time or date units passed since reference date/time.

2.2.4 RoZ Classes

RoZ generates class intension and extension for each of the classes. Considering each class, the "intension" is the set of all possible entities of this kind, which may often be an infinite set of elements. Contrarily, "extension" is the set of entities stored in the information system at a given time. This is considered a variable set and its contents may evolve with time since entities may be added/removed from the information system. For class MEETING RoZ represents them as MEETING (intension) and MeetingExt (extension), respectively.

$$\begin{array}{|l}
\hline
\textit{MEETING} \\
start : DATE \\
duration : TIME \\
\hline
\end{array}$$

MEETING schema is an intension with the given attributes start and duration as state components. MeetingExt states the extension of MEETING in the set Meeting. In this case, MEETING schema is considered as a type to define the extension. This extension has no constraint.

MeetingExt

$Meeting : \mathbb{F}\, MEETING$

Similarly, PersonExt states the extension of PERSON. This extension has also no constraint.

PERSON

$name : STRING$

PersonExt

$Person : \mathbb{F}\, PERSON$

2.2.5 Translation of Relations

At this point, we have defined the two classes (meeting, person) and their associated attributes. Their linkage in the system specifications is as follows:

- There is a relation (MeetingOwner) one to zero/many between persons and meetings. Informally, a person may have zero to many meetings. Conversely, a meeting must be owned by a person.

- The relation (MeetingParticipants) specifies that at least one person must be listed as participant of a meeting, and meetings of participants can be many or none.

The relations between meetings and persons involve two roles which will be translated as two functions: owner is a function from meetings to persons while meetingsOfOwner is from persons to meetings. Both roles are translated as partial functions, associating a set of meetings to each person. This is translated into a schema MeetingOwnerRel in Z. In the predicate part, domain and range of owner function are constrained. meetingsOfOwner and owner compute meetings of owner from owner, and owner from meetings of owner. They express that one role can be deduced from the other one. The distributed union ($\bigcup$) offers a generalization to calculate a distributed union of all the sets in the relation owner.

$$
\begin{array}{l}
\textit{MeetingOwnerRel} \\
\hline
\textit{PersonExt};\ \textit{MeetingExt} \\
\textit{owner} : \textit{MEETING} \nrightarrow \textit{PERSON} \\
\textit{meetingsOfOwner} : \textit{PERSON} \nrightarrow \mathbb{F}\,\textit{MEETING} \\
\hline
\mathrm{dom}\,\textit{owner} = \textit{Meeting} \\
\mathrm{ran}\,\textit{owner} \subseteq \textit{Person} \\
\textit{meetingsOfOwner} = \{\textit{person} : \mathrm{ran}\,\textit{owner} \bullet \textit{person} \mapsto \\
\qquad \{\textit{meeting} : \mathrm{dom}\,\textit{owner} \mid \textit{owner}(\textit{meeting}) = \textit{person} \bullet \textit{meeting}\}\} \\
\textit{owner} = \bigcup\{\textit{person} : \mathrm{dom}\,\textit{meetingsOfOwner} \bullet \\
\qquad \{\textit{meeting} : \textit{meetingsOfOwner}(\textit{person}) \bullet \textit{meeting} \mapsto \textit{person}\}\}
\end{array}
$$

The MeetingParticipantsRel schema also includes two partial functions participants and meetingsOfParticipant. This relation features to other roles between Meeting and Person. The constraints specify that the domain of participants is equal to Meeting i.e., that all meetings have at least one participant, and participants are persons from meetings. The two partial functions participants and meetingsOfParticipant compute the participants of meeting and meetings of participants, respectively.

$$
\begin{array}{l}
\textit{MeetingParticipantsRel} \\
\hline
\textit{PersonExt};\ \textit{MeetingExt} \\
\textit{participants} : \textit{MEETING} \nrightarrow \mathbb{F}\,\textit{PERSON} \\
\textit{meetingsOfParticipant} : \textit{PERSON} \nrightarrow \mathbb{F}\,\textit{MEETING} \\
\hline
\mathrm{dom}\,\textit{participants} = \textit{Meeting} \\
\bigcup(\mathrm{ran}\,\textit{participants}) \subseteq \textit{Person} \\
\textit{participants} = \{\textit{meeting} : \bigcup(\mathrm{ran}\,\textit{meetingsOfParticipant}) \bullet \\
\qquad \textit{meeting} \mapsto \{\textit{person} : \mathrm{dom}\,\textit{meetingsOfParticipant} \mid \textit{meeting} \\
\qquad\quad \in \textit{meetingsOfParticipant}(\textit{person}) \bullet \textit{person}\}\} \\
\textit{meetingsOfParticipant} = \{\textit{person} : \bigcup(\mathrm{ran}\,\textit{participants}) \bullet \\
\qquad \textit{person} \mapsto \{\textit{meeting} : \mathrm{dom}\,\textit{participants} \mid \textit{person} \\
\qquad\quad \in \textit{participants}(\textit{meeting}) \bullet \textit{meeting}\}\}
\end{array}
$$

These are just a few glimpses how RoZ translates the relations between the classes from a UML class diagram. Then, the schema GlobalView groups all data structures we have in the SCHEDULER application. This includes a predicate which describes that the owner of the meeting is one of the participants. This uses Z schema inclusion to group the relevant schemas. This ends up the modeling of the global state of the meeting SCHEDULER application. We can see that the constraints added in GlobalView shows the possibility of adding constraints explicitly to the formal specifications of UML models.

$$
\begin{array}{l}
\textit{GlobalView} \\
\hline
\textit{MeetingOwnerRel} \\
\textit{MeetingParticipantsRel} \\
\hline
\forall\, m : \textit{Meeting} \bullet \textit{owner}(m) \in \textit{participants}(m)
\end{array}
$$

2.2.5.1 Arbitrary Queries

Operations can also be defined to perform arbitrary queries on the system specification contents. meetingnotify is an example to that. This operation returns the participants of a meeting while meeting? should be one of the existing meetings in the system specifications. meetingnotify describes no impact (Ξ) on MeetingParticipantsRel. We will experience extensive use of similar queries in Chapter 5 and Chapter 6 of the thesis which will demonstrate its usefulness in the process of security policy evaluation. These chapters will help understand the nature of querying a security policy model, as here the arbitrary queries are discussed only in terms of functional models.

$$
\begin{array}{|l}
\hline
\textit{meetingnotify} \\
\Xi MeetingParticipantsRel \\
meeting? : MEETING \\
x! : MEETING \times (\mathbb{F}\, PERSON) \\
\hline
x! = (meeting?, participants(meeting?)) \\
meeting? \in Meeting \\
\hline
\end{array}
$$

2.2.6 Generation of Elementary Operations Using RoZ

In information systems, elementary operations enable creation or deletion of an object, a link, or to modify an attribute value. They often appear as regular part of many classes in an application. RoZ works on the same principle to generate such elementary operations of the UML class diagrams. For instance, a class meeting from meeting SCHEDULER example contains several operations which will be translated into Z and are demonstrated in next sections. The generation of the operations does not consider the eventual constraints on a diagram. However, to check that an operation is consistent with the constraints, one needs to validate its guard by proving a theorem generated by RoZ. This process is out of scope and not being elaborated in the dissertation. Interested readers are directed to [Ledru 1998] and [Dupuy *et al.* 2000] for a detailed account.

Note that a UML class diagram groups operations on a single instance of the class (e.g., ChangeDuration), operations which impact the extension of the class (e.g., RemoveMeeting), and operations (e.g., LinkmeetingsOfParticipant) on the relations which involve the class. RoZ helps precise the scope and the semantics of these operations since graphical notations mainly specify the syntax (i.e., the signature) of operations but do not provide much details about their semantics except if we use OCL. Operations which modify the system specifications content are structured into the following categories by RoZ:

- Operations on instances of classes;
- Operations on the extension of a class;
- Operations on relations between classes;

– Operations on the global system specification.

An excerpt of RoZ generated operations of meeting SCHEDULER example are used to describe each of above operations category. A few generated operations will be described below in order to share a flavor of the underlying technique. The complete set of RoZ generates operations of meeting SCHEDULER application can be found in Appendix C.

2.2.6.1 Operations on Instances of Classes

MEETINGChangeStart is an example of an operation intended to address an instance of the class MEETING. This operation takes a date as input (newstart?) and modifies the start of the meeting with the new input value. The duration remains unchanged. MEETINGChangeDuration takes into account the change of duration of a meeting.

MEETINGChangeStart

$\Delta MEETING$
$newstart? : DATE$

$start' = newstart?$
$duration' = duration$

MEETINGChangeDuration

$\Delta MEETING$
$newduration? : TIME$

$duration' = newduration?$
$start' = start$

2.2.6.2 Operations on the Extension of a Class

The typical operations on the extension of the classes correspond to the addition and deletion of an element to the set. The first schema MeetingAddMeeting adds a meeting to the extension of the class.

MeetingAddMeeting is an operation schema to add a meeting to the extension of meeting class. It takes a new meeting as input (meeting?) and computes the new meeting set i.e., Meeting'.

MeetingAddMeeting

$\Delta MeetingExt$
$meeting? : MEETING$

$Meeting' = Meeting \cup \{meeting?\}$

The schema MeetingRemoveMeeting is intended to remove a meeting from the set of meetings. It takes a meeting input (meeting?) to be deleted and updates

the meetings. This schema will be used in another user defined operation named meetingcancel in next section.

$$
\begin{array}{|l}
\hline
\textit{MeetingRemoveMeeting} \\
\Delta MeetingExt \\
meeting? : MEETING \\
\hline
Meeting' = Meeting \setminus \{meeting?\} \\
\hline
\end{array}
$$

Z provides a great degree of modularization using schema inclusion. Complex modularization is also possible by the use of promotion or framing. It permits to compose and factor specifications. This mechanism is also adapted by RoZ in the generated Z specifications.

The basic motivation of these operation stems from the fact that operation such as MeetingChangeStart and MeetingChangeDuration are not sufficient since their scope is limited to one object of the meeting class. In order to perform promotion, a general purpose schema ChangeMeeting is introduced. ChangeMeeting modifies two kinds of variables: MeetingExt and an instance of the schema type MEETING, corresponding to the input parameter meeting?. This instance is the one modified by an operation such as MEETINGChangeStart. Z allows to describe the initial and final states of Meeting, a component of MeetingExt. But it is necessary to get a control on the instance of MEETING in order to speak about its initial and final states. This is the intent of the operator θ used in the predicate part of this schema. θMEETING is a variable of type MEETING, and θMEETING' denotes its final state e.g., after the execution of MEETINGChangeStart. The first constraint in the predicate part is a precondition which states that the given input meeting (meeting?) should be one of the existing meetings. This precondition cannot be deduced from the imported schemas so this is an additional precondition. Thus, this schema provides a link to promote operations given as MeetingChangeStart.

$$
\begin{array}{|l}
\hline
\textit{ChangeMeeting} \\
\Delta MeetingExt \\
\Delta MEETING \\
meeting? : MEETING \\
\hline
meeting? \in Meeting \\
\theta MEETING = meeting? \\
Meeting' = Meeting \setminus \{meeting?\} \cup \{\theta MEETING'\} \\
\hline
\end{array}
$$

MeetingChangeStart is described by ChangeMeeting and MEETINGChangeStart. We can put these two operations into another grand operation. This way, one can significantly modularize the formal descriptions. So, MeetingChangeStart is defined as a combination of ChangeMeeting and MEETINGCHANGEStart.

$$
\begin{aligned}
MeetingChangeStart == (&ChangeMeeting \wedge MEETINGChangeStart) \\
&\setminus (start, start') \setminus (duration, duration')
\end{aligned}
$$

The ($\backslash$) operator is used to hide variables. However, the other interesting parameters (unhidden, here meeting? and newstart?) must be supplied with inputs. Usually general purpose operations can be combined with other operations. A detailed description on the idea of promotion can be found in [Spivey 1992].

2.2.6.3 Operations on the Relations

These operations work at the level of relations between classes. Actually, the modification of meetings also involves the modification of the components of the relation between meeting owner and meeting participants relations.

$$
\begin{array}{|l}
\hline
\textit{meetingcreateMeeting} \\
\textit{MeetingAddMeeting} \\
\Xi \textit{PersonExt} \\
\Delta \textit{MeetingOwnerRel} \\
\Delta \textit{MeetingParticipantsRel} \\
\textit{owner?} : \textit{PERSON} \\
\hline
\textit{owner}' = \textit{owner} \oplus \{\textit{meeting?} \mapsto \textit{owner?}\} \\
\textit{participants}' = \textit{participants} \oplus \{\textit{meeting?} \mapsto \{\textit{owner?}\}\} \\
\hline
\end{array}
$$

The schema meetingcreateMeeting performs the promotion of MeetingAddMeeting (one of the RoZ generated operation discussed previously). It modifies the variables of MeetingOwnerRel and MeetingParticipantsRel schemas, and access the PersonExt in read-only mode. In the predicate part, the definitions of two functions owner and participants are updated.

The meetingcancel is another example of working at the relations level. It includes PersonExt in read-only mode but performs changes to the relation schemas MeetingOwnerRel and MeetingParticipantsRel. It also promotes the schema MeetingRemoveMeeting.

$$
\begin{array}{|l}
\hline
\textit{meetingcancel} \\
\textit{MeetingRemoveMeeting} \\
\Xi \textit{PersonExt} \\
\Delta \textit{MeetingOwnerRel} \\
\Delta \textit{MeetingParticipantsRel} \\
\hline
\textit{owner}' = \{\textit{meeting?}\} \lhd \textit{owner} \\
\textit{participants}' = \{\textit{meeting?}\} \lhd \textit{participants} \\
\hline
\end{array}
$$

The operator $\lhd$ is used to subtract the meeting meeting? from the domain of participants function. Note that, meetingcancel is one of the user defined operations to conform to the particular operations of our borrowed example.

2.2.6.4 Operations on the System Specifications

A first operation that takes place at the highest level is the initialization of the whole system specifications. It's mandatory to specify initial state of the abstract

machine in Z. Here, InitGlobalView does the same job. The constraints express that the extensions of the classes are empty in the initial state and so the respective relations specified earlier.

$$
\begin{array}{|l}
\hline
\textit{InitGlobalView} \\
\hline
MeetingExt' \\
PersonExt' \\
MeetingOwnerRel' \\
MeetingParticipantsRel' \\
\hline
Meeting' = \varnothing \\
Person' = \varnothing \\
owner' = \varnothing \\
meetingsOfOwner' = \varnothing \\
participants' = \varnothing \\
meetingsOfParticipant' = \varnothing \\
\hline
\end{array}
$$

2.3 Using Jaza to Animate the Model

Jaza [Utting 2005] can be used to validate functional properties of an application. It can animate a large subset of constructs of the Z specification language. Its salient features encompass concepts like combination of proofs (simplification, rewriting) and search (generate and test) techniques. It determines whether an operation can be executed on the current state. Using a current "system state", it results into a new state with operation execution. Similarly, if the invariants fail in the initial or final state, the tool returns "No Solutions" that can further be queried by (Why) command in order to find out the reasons. When we initialize schema InitGlobalView, it iterates over the constraints and applies respective invariants.

In Fig. 2.4 a screen shot of the Jaza tool has been provided. Initially the tool requires us to load all the Z files including the types, generated operations, and the data (besides any additional files that we may need). Based on this input information, we can execute each of the operations given in the Z specifications. For example a few operations such as ;PersonAddPerson, ;meetingcreateMeeting are visible in the command prompt area of the shown figure.

2.4 Discussion

In this chapter we have tersely described the Z language, RoZ translation process, and the Jaza tool. One may think that the RoZ annotation process is also achievable by the use of OCL. Although it seems true, OCL tools are not sufficient to animate a class diagram and its operations. We will come back to this point in Chapter 5 in detail. We believe that a better formal analysis of UML diagrams is possible by the Z language with available tool support such as Z-EVES. However, Jaza is used to animate the specifications in this thesis. The demonstration of using RoZ with

Jaza has also been explained in [Ledru 2006]. The actual outcome of RoZ comes in the form of analyzing and debugging formal specifications. It also contributes to the understandability of some constructs such as aggregation and composition of UML models. A detailed account of the tool can be found in [Dupuy *et al.* 2000].

Note that, the basic operations generated by RoZ enable an unrestricted access to the resources of an application which is indeed not desirable in a secure information system. This motivates us to move a step further in order to upgrade RoZ for secure applications.

```
C:\Windows\system32\cmd.exe - Jaza

C:\ztcwin\M>Jaza
Welcome to Jaza, version 1.1.  June 2005
Copyright(C) 1999-2005 Mark Utting (marku@cs.waikato.ac.nz).
Jaza comes with ABSOLUTELY NO WARRANTY (see file COPYING).
This is free software, and you are welcome to redistribute
it under certain conditions (see file COPYING).

Type `help' to see the available commands.

JAZA> reset
Specification is now empty.
JAZA> load ZTypes.zed
Loading 'ZTypes.zed' ...
Added 3 definitions.
JAZA> load rozdata.zed
Loading 'rozdata.zed' ...
Added 8 definitions.
JAZA> load rozops.zed
Loading 'rozops.zed' ...
Added 27 definitions.
JAZA> do InitGlobalView
\lblot Meeting'==\{\}, Person'==\{\}, meetingsOfOwner'==\{\},
       meetingsOfParticipant'==\{\}, owner'==\{\},
       participants'==\{\} \rblot
JAZA> ; PersonAddPerson
    Input person? = \lblot name=="Alice" \rblot
\lblot Meeting'==\{\}, Person'==\{\lblot name=="Alice" \rblot\},
       meetingsOfOwner'==\{\}, meetingsOfParticipant'==\{\}, owner'==\{\},
       participants'==\{\} \rblot
JAZA> ; PersonAddPerson
    Input person? = \lblot name=="Bob" \rblot
\lblot Meeting'==\{\},
       Person'==\{\lblot name=="Alice" \rblot, \lblot name=="Bob" \rblot\},
       meetingsOfOwner'==\{\}, meetingsOfParticipant'==\{\}, owner'==\{\},
       participants'==\{\} \rblot
JAZA> ; meetingcreateMeeting
    Input meeting? = \lblot start == 1, duration == 10 \rblot
    Input owner? = \lblot name=="Alice" \rblot
\lblot Meeting'==\{\lblot duration==10, start==1 \rblot\},
       Person'==\{\lblot name=="Alice" \rblot, \lblot name=="Bob" \rblot\},
       meetingsOfOwner'==\{\{\lblot name=="Alice" \rblot,
                            \{\lblot duration==10, start==1 \rblot\}\}\},
       meetingsOfParticipant'==\{\{\lblot name=="Alice" \rblot,
                                  \{\lblot duration==10,
                                        start==1 \rblot\}\}\},
       owner'==\{\{\lblot duration==10, start==1 \rblot,
                   \lblot name=="Alice" \rblot\}\},
       participants'==\{\{\lblot duration==10, start==1 \rblot,
                          \{\lblot name=="Alice" \rblot\}\}\} \rblot
JAZA> ; meetingLinkmeetingsOfParticipant
    Input person? = \lblot name=="Bob" \rblot
    Input meeting? = \lblot start==1, duration==10 \rblot
\lblot Meeting'==\{\lblot duration==10, start==1 \rblot\},
       Person'==\{\lblot name=="Alice" \rblot, \lblot name=="Bob" \rblot\},
       meetingsOfOwner'==\{\{\lblot name=="Alice" \rblot,
                            \{\lblot duration==10, start==1 \rblot\}\}\},
       meetingsOfParticipant'==\{\{\lblot name=="Alice" \rblot,
                                  \{\lblot duration==10,
                                        start==1 \rblot\}\},
                                 (\lblot name=="Bob" \rblot,
                                  \{\lblot duration==10,
                                        start==1 \rblot\}\}\},
       owner'==\{\{\lblot duration==10, start==1 \rblot,
                   \lblot name=="Alice" \rblot\}\},
       participants'==\{\{\lblot duration==10, start==1 \rblot,
                          \{\lblot name=="Alice" \rblot,
                           \lblot name=="Bob" \rblot\}\}\} \rblot
JAZA>
```

Figure 2.4: Jaza tool

Chapter 3

Access Control Mechanisms & SecureUML

Contents

3.1 Access Controls Mechanisms **36**
3.1.1 Mandatory Access Control (MAC) 37
3.1.2 Discretionary Access Control (DAC) 39
3.1.3 Role Based Access Control (RBAC) 40
3.2 Functional Specification Packages **44**
3.2.1 Summarizing Access Control Mechanisms 46
3.3 Data Security Properties **47**
3.3.1 Availability 47
3.3.2 Confidentiality 47
3.3.3 Integrity 48
3.4 Why to Choose RBAC over MAC and DAC? **48**
3.5 UML-based RBAC specifications **49**
3.5.1 SecureUML 49
3.6 Summary **52**

An enterprise which is regarded as a *system* consists of products or components, operating systems, applications, IT staff, internal users and management, customers, external users and the surrounding environment [Bishop 2003]. Enforcing and managing such enterprise-wide security is often considered a complex and tedious task. More importantly, computer systems need special security treatments against *soft* attacks and threats. Several access control mechanisms (a.k.a authorization mechanisms) can be used to provide a shield against these challenges. So we can limit individual's access to valuables. A secure system has several important components: it should be backed up by a security policy, security mechanism, and a security model. These notions are described below by adapting the definitions from [Bishop 2003].

– *Security policy* is a statement of what is, and what is not, allowed; or it is a statement that partitions the states of the system into a set of authorized, or secure, states and a set of unauthorized, or non secure states.

- *Security mechanism* is a method, tool, or procedure for enforcing a security policy.

- Security model is a model that represents a particular policy or set of policies.

- Secure system is a system that starts in an authorized state and cannot enter an unauthorized state.

Security policies are generally context-dependent or enterprise-specific which are expressed as security models. However, to realize the implementation of security policies, security mechanisms are utilized e.g., MAC [Bell & LaPadula 1975], DAC [DOD 5200.28-STD 1985], RBAC [Ferraiolo *et al.* 2001]. The MAC and DAC predate RBAC.

The principle subject of our research is RBAC access control mechanism, and its use in graphical modeling of security policies, formal specifications and the subsequent validation.

This chapter is dedicated to the description of access control mechanisms and the SecureUML security modeling language. The first main section 3.1 introduces MAC, DAC and RBAC. Section 3.2 discusses the logical approach of using RBAC; while data security properties are discussed in Section 3.3 and in Section 3.4 we provide a brief comparison of MAC, DAC, and RBAC. Section 3.5 states SecureUML that helps specifying security design models of secure applications with an example, along with its metamodel in Section 3.5.1.1.

3.1 Access Controls Mechanisms

The use of multi-user computer systems requires enforcing access control in a right way so that the users could perform allowable actions, and only these actions. One can imagine the following list of actions experienced everyday in the office:

- A user can exercise program permissions such as the right to execute a program on an application server;

- The permissions linked to a file like create, read, delete, and edit on a file server;

- Updating and retrieving information from a database.

All the access control mechanisms (i.e., mainly MAC, DAC, and RBAC) designed for computer security require to deal with similar issues. Their application rules are clearly disparate from each other. We briefly discuss below their major differences and the underlying concepts individually.

3.1.1 Mandatory Access Control (MAC)

Definition 1. When a system mechanism controls access to an object and an individual user cannot alter that access, the control is a mandatory access control (MAC), occasionally called a rule-based access control (not to be confused with RBAC).

Bell-LaPadula's MAC [Bell & LaPadula 1975] is an access control mechanism and perhaps the oldest one as a standard. The administrator is in charge to define permissions in MAC. In MAC, the operating system restricts the ability of a subject (also known as initiator) to perform operations on an object or a target. A subject may also be regarded as a process or a thread. Objects can be constructs, e.g., directories, files and segments of the shared memory. Subjects and objects have clearance properties and security properties, respectively. An operating system's security kernel examines the relevant access control properties and thus access is granted as specified by authorization rules. Authorization rules come into play to determine if the operation is permitted. This means that MAC enforces the system to remain in a secure state regardless of user actions. Labels are introduced on the objects to secure. The labels help state who is authorized to access these objects. Similarly, the security label of an object can only be changed by the administrator. The set of classification level is ordered by a $<$ relationship.

Unclassified $<$ *Confidential* $<$ *Secret* $<$ *TopSecret*

There are three important notions to discuss here:

- Clearance level: indicates the level of trust given to a person with a security clearance. It may also be a computer that processes classified information as well as an area physically secured for storing classified information.
- Classification level: determines the level of sensitivity associated with some information, for example, in a computer file or a document.
- Security level: indicates to a generic term for either a classification level or a clearance level.

Nonetheless, the clearance and security labels are designed in this way:

(classification level, {categorization code(s)})

Security levels have a hierarchical and non-hierarchical component. The hierarchical components include unclassified (U), confidential (C), secret (S), and top-secret (TS) whilst non-hierarchical components include NATO and NUCLEAR. The permissions over the documents of a user are granted by the system instead of users. But a few exceptions include UNIX and Windows operating systems, which give

control to the creator of a document to specify permissions. Examples of user objects in Windows operating system include printers, services, and files. MAC has wide applications in the real world scenarios such as database management systems. It should be noted that MAC only deals with the confidentiality property with no read up and no write down.

We can revisit our meeting SCHEDULER example in previous chapter (section 2.2.1) in order to find out the feasibility of structuring it using MAC. In our example, we have two resources i.e., Person and Meeting to secure; each of them can have multiple instances. A meeting is owned by a person who has the authority to delete or modify his own meeting, and others can read that of his listed meeting. The supervisor actually inherits the permissions of a system user. The class of persons is managed by the system administrator using some operations. We can create a mapping between all these objects and perhaps tags can be associated to protect the information. However, this looks clumsy and more implementation oriented.

We show the MAC-based design of SCHEDULER example with simple read and write permissions. We have specified some users, their clearance level, and similarly objects with their classification level.

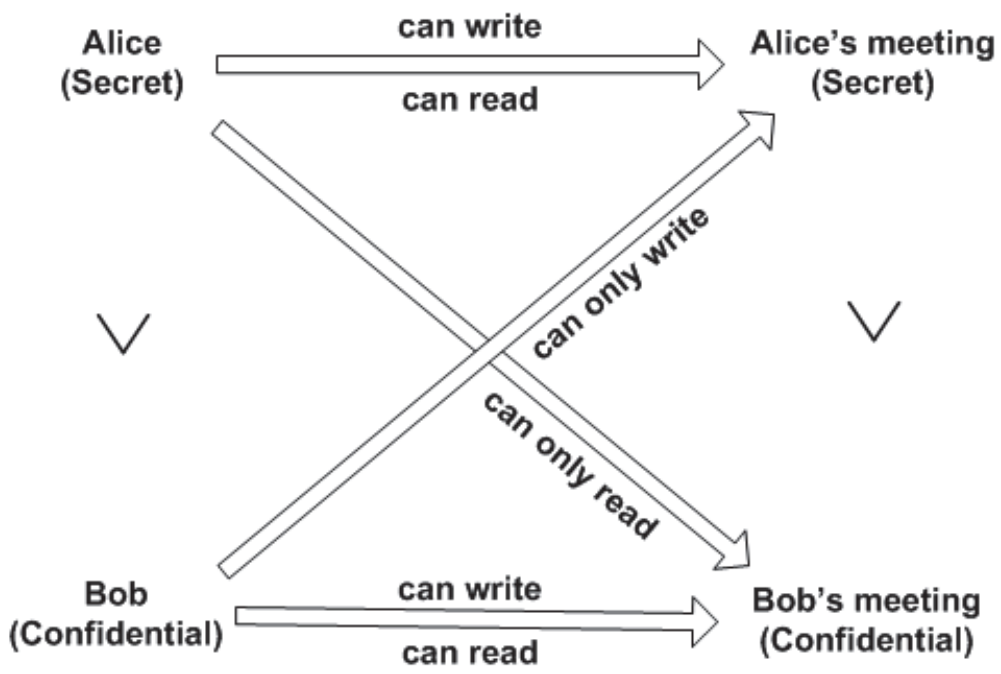

Figure 3.1: MAC mechanism applied to the meeting SCHEDULER

In 3.1 a subject is permitted to have read access to an object if the subject's security level dominates the security level of the object. This is also called *simple*

security property [Ferraiolo *et al.* 2007]. Alice can both read and write to her meeting, and the same is true for Bob. Their respective operations are possible since their classification level and the corresponding clearance level are the same.

In MAC a subject is permitted write access to an object if the object's classification level dominates the clearance level of the subject, also named as *star property* [Ferraiolo *et al.* 2007]. When designing our subjected example using MAC, we witness that Bob can only write to Alice's meeting and Alice can only read Bob's meeting. Nonetheless, the requirement was that the users should only be able to read each other's meetings. Thus, we find it difficult to implement the example as it is using MAC. We can see that Bob can write to Alice's meeting.

MAC allows devising a mechanism for the read and writing operations, which prevents users from being able to read information that dominates their clearance level. But it is pertinent to state that it only deals with the confidentiality property of a secure system.

3.1.2 Discretionary Access Control (DAC)

Definition 2. In DAC access control mechanism, an individual user has the authority to allow or deny the access to an object. This is also called an identity-based access control (IBAC).

Restricting dissemination of information has always been a challenging task. DAC is another choice to secure objects in a system. Contrary to MAC, an access control mechanism is called DAC, if an individual manages the allowing/forbidding access control decision to an object. This scheme is also referred as Identity Based Access Control (IBAC) [Andersson 2001]. Access to resources is based on user's identity. An Access Control List (ACL) of users is updated if a user is given access to a resource, and also associated with a resource. Access Control Entry (ACE) is called as an entry on a resource's ACL. A user may also be a group, a set of users. The DAC model is based on the concept of resource ownership.

It is pertinent to explain that the end users in industry and civilian organizations don't "own" the information to which they are allowed access as is assumed by DAC policies [Ferraiolo *et al.* 2007]. In that case, the actual "owner" of system objects is the corporation or the parent agency.

Table 3.1 shows that a user can allow access to another user on an object, for example, Bob allows Alice to perform read operation on his meeting Meeting1. Alice has created a meeting named Meeting2 on which Bob is revoked to read the meeting. One can imagine millions of records granting and revoking permissions to them. This is purely a discretionary activity. In these circumstances, Harrison, Ruzzo, Ullman have proved that undecidability result applies to DAC ([Ferraiolo *et al.* 2007], page – 27). They argue that safety is inherently undecidable in a conventional access matrix view of security. It means that it is impossible to know whether a given configuration considered safe with respect to some security requirement would remain safe [Ferraiolo *et al.* 2007]. If the system is started with a set of access rights to objects, it would be impossible to judge that the system will not eventually grant

Table 3.1: DAC mechanism applied to the meeting SCHEDULER

Users	Objects	Permission	Status
Bob	Meeting1	to Alice (read)	Granted
Alice	Meeting2	to Bob (read)	Revoked

access rights that are not in the original matrix.

3.1.3 Role Based Access Control (RBAC)

Definition 3. Role-based access control (RBAC) is a method of regulating access to computer or network resources based on the roles of individual users within an enterprise. In this context, access is the ability of an individual user to perform a specific task, such as view, create, or modify a file. Roles are defined according to job competency, authority, and responsibility within the enterprise.

The NIST (National Institute of Standards and Technology) has standardized RBAC [Ferraiolo *et al.* 2001]. It provides the means to define access control mechanisms to inhibit unauthorized use. The notion of role is a permanent organizational identity that has certain allowable actions associated to it. RBAC follows a generic principle of access control policy that makes it adaptable to any organizational structure as well as flexible *wrt.* the application implementation. The main distinguishing feature of RBAC is its assignments of roles to users and the fact that roles have certain predetermined privileges and rights.

3.1.3.1 Core RBAC

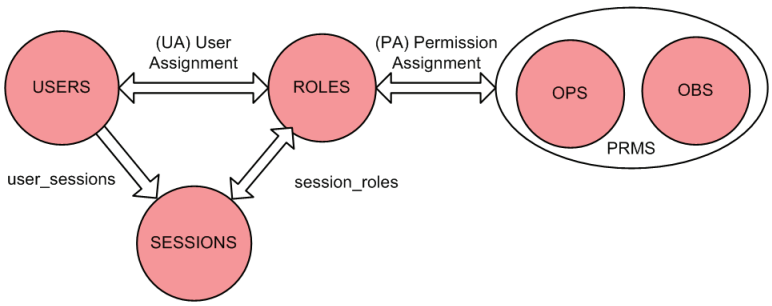

Figure 3.2: Core RBAC [Ferraiolo *et al.* 2001]

The data model of RBAC (Fig. 3.2) is based on five data types: users (USERS),

roles (ROLES), objects (OBS), permissions (PRMS) and executable operations (OPS) by users on objects. A sixth data type, session (SESSIONS), is used to associate roles temporarily to users. Sessions correspond to the dynamic aspect of RBAC. RBAC differentiates between users and roles. A role is considered as a permanent position in an organization whereas a given user might be switched with another user for that role. Thus, rights are offered to roles instead of users. Roles are assigned to permissions that can later be exercised by users playing these roles. Modeled objects (OBS) in RBAC are potential resources to protect. Operations (OPS) are viewed as application-specific user functions. UA is user assignment, RH is role hierarchy and PA is permission assignment. Here, we very briefly outline the RBAC constructs:

- **Users**: a person who uses a system or an automated agent;
- **Role**: is an organization entity or a permanent position in an enterprise. Each role may have an allowable set of actions according to access the control policy. This way, access to computational resources is made via roles;
- $UA \subseteq USERS \times ROLES$, a many-to-many mapping between users and roles; UA specifies which roles can be played by a given user;
- $PA \subseteq PRMS \times ROLES$, a many-to-many mapping permission-to-role; PA expresses which roles may be granted a given permission;
- $user_sessions(u : USERS) \rightarrow 2^{SESSIONS}$, the mapping of user u onto a set of sessions; it lists the current sessions of a given user;
- $session_roles(s : SESSIONS) \rightarrow 2^{ROLES}$, the mapping of session s onto a set of roles; it lists the current roles of a given user in a given session;
- $PRMS = 2^{(OPS \times OBS)}$, the set of permissions. Permissions are regarded as an approval to perform operations on RBAC protected objects. An executable image of a program is considered as an operation, which upon invocation executes some function for the user. The type of the implemented system specifies the types of operations and objects to be controlled by RBAC. For example, within a database management system, operations might include insert, delete, append, and update; within a file system, operations might include read, write, and execute.

The core RBAC (Fig. 3.2) embodies only includes users, roles, permissions, operations, and objects entities – and relations as types and functions. Core RBAC is a mandatory part of any RBAC system. A user can be assigned to several roles and a single role can have many users. The variants (e.g., SSD and DSD) are orthogonally constructed and implemented in a RBAC computer system. In core RBAC, role activation is given as part of a user's session with a computer system. The role activation/deactivation allows the selective exercise of some permission during a session.

Table 3.2: RBAC Configurations

Users	Role
Alice	SystemUser
Bob	SystemUser
John	Supervisor
Mike	SystemAdministrator

Table 3.3: RBAC Configurations: [C:Create, U:Update, D:Delete: R:Read]

Role	Permission	Object	Operations
SystemUser	UserMeeting	Meeting	C, U, R, D
Supervisor	OwnerMeeting/UserMeeting	Meeting	C, U, D, R
SystemAdministrator	UserManagement	Person	C, U, D

We now move to design our meeting SCHEDULER example by the use of RBAC. This represents a general idea about its working structure. Table 3.2 shows that first each user is assigned with a role. We assume some roles i.e., System User, Supervisor, and System Administrator. Permissions include Owner Meeting (owner of a meeting), User Meeting (a participant of a meeting), Supervisor Cancel (a privilege to the supervisor to cancel the meeting), and User Management (right of some director to add, delete persons). In table 3.3, roles have been assigned with the corresponding permissions, objects to secure, and the operations offered by them (e.g., C:Create, U:Update, D:Delete: R:Read). Note that, these roles and permissions will always remain the same but users can be changed, which is a sort of flexibility offered by RBAC.

3.1.3.2 RBAC Variants

The variants of RBAC can be used to specify a secure system in terms of simple relations (e.g., PA and UA) as well as SSD (static separation of duty) and DSD (dynamic separation of duty) constraints (stated in sections 3.1.3.4, 3.1.3.5). A RBAC supported system need not to include all variants of RBAC but it can be customized accordingly. RBAC standardized by NIST has several building blocks consisting of separation of duties, roles, roles hierarchy, role activation, and constraints on user/role membership and the activation of role set [Ferraiolo *et al.* 2001]. Below we provide a brief description about the variants of RBAC.

3.1.3.3 Hierarchical Role Based Access Control

Hierarchies are used as a natural way of structuring roles to state the line of authority and responsibility of an organization. Fig. 3.3 sketches the Hierarchical RBAC. Role

hierarchies reflect on inheritance relation among roles. The RH is defined as:

– $RH \subseteq R \times R$, a partially ordered role hierarchy; a senior role may inherit the permissions from its junior roles;

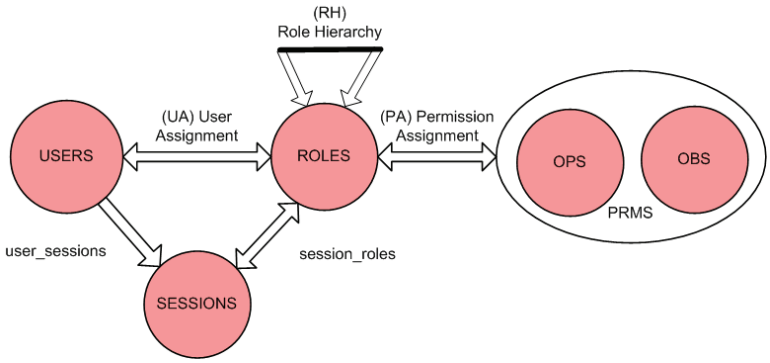

Figure 3.3: Hierarchical RBAC [Ferraiolo *et al.* 2001]

The term inheritance refers to inheriting a permission which means *r1* "inherits" role *r2* provided that all rights of *r2* are also rights of *r1*, where *r1, r2* are some roles. For instance, in the meeting SCHEDULER example, the role supervisor can inherit the permissions OwnerMeeting and UserMeeting from the role system user.

3.1.3.4 Role Based Access Control with SSD

SSD (Static Separation of Duty) constraint (Fig. 3.4) offers the means to address conflict of interest among roles. For example, a user may be assigned to two roles, let's say a Programmer and a Tester, which are supposed to create a conflict if exercised at once so we can use SSD constraints in order to forbid using both roles simultaneously. For this, SSD constraints are applied on assignment of users to roles and thus, UA is restricted during sessions. By this, if a user is assigned to a role, the user can never take the prohibited role. SSD can be applied not only to colluding users but to groups, which is a collection of users. Permissions can be associated with both users and groups.

SSD constraints are specified over UA assignments as pairs of roles. Although NIST RBAC terms it very restrictive in two aspects: the size of the set of roles in the SSD and then combining roles in the set for which UA is restricted. This restrictive SSD constraint is understandable since the security policy of an organization may state that no one user may be assigned to four of the five roles representing some accounting functions. We can see that this will result in a different type of SSD constraint impossible to impose using SSD of RBAC, and is definitely one of the limitations of RBAC.

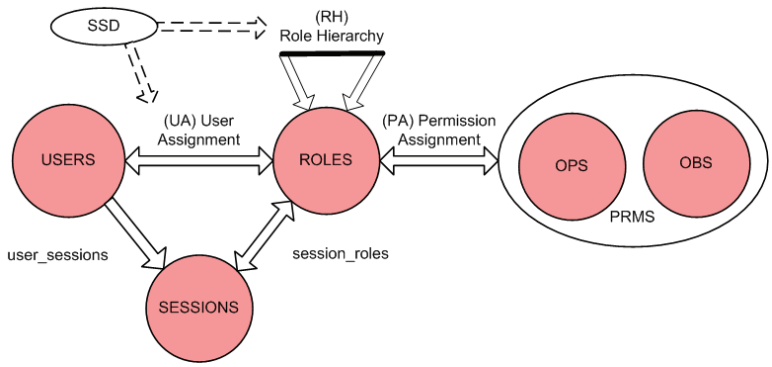

Figure 3.4: SSD within Hierarchical RBAC [Ferraiolo *et al.* 2001]

3.1.3.5 Role Based Access Control with DSD

Aside from SSD, DSD (Dynamic Separation of Duty) is the second kind of constraint offered by RBAC (Fig. 3.5). These constraints are intended to limit the permissions that are available to a user, whilst SSD constraints reduce the number of potential permissions that can be made available to a user by placing constraints on the users that can be assigned to a set of roles. The main difference between SSD and DSD constraints lies in the context in which they are used. SSD are imposed on user's total permission space but DSD restricts the users to activate the roles within or across a user's sessions. For example, a user Bob may have been assigned with two roles i.e., Supervisor and System User but he may not exercise the permissions of both roles in the same session.

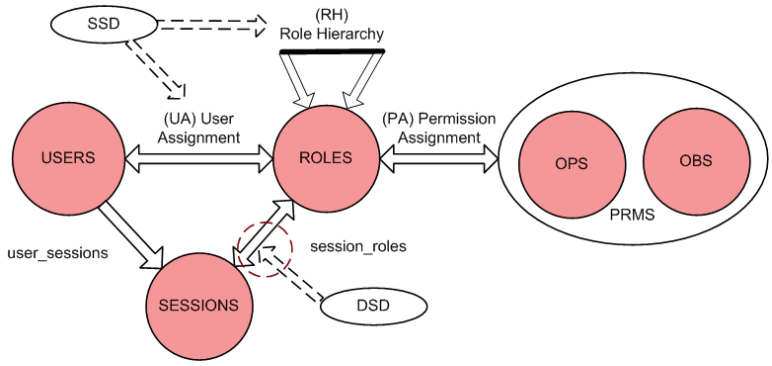

Figure 3.5: Dynamic Separation of Duty relations [Ferraiolo *et al.* 2001]

3.2 Functional Specification Packages

The RBAC Functional specification specifies administrative review functions for performing administrative queries; administrative operations for the creation and maintenance of RBAC element sets and relations; and system functions for creating

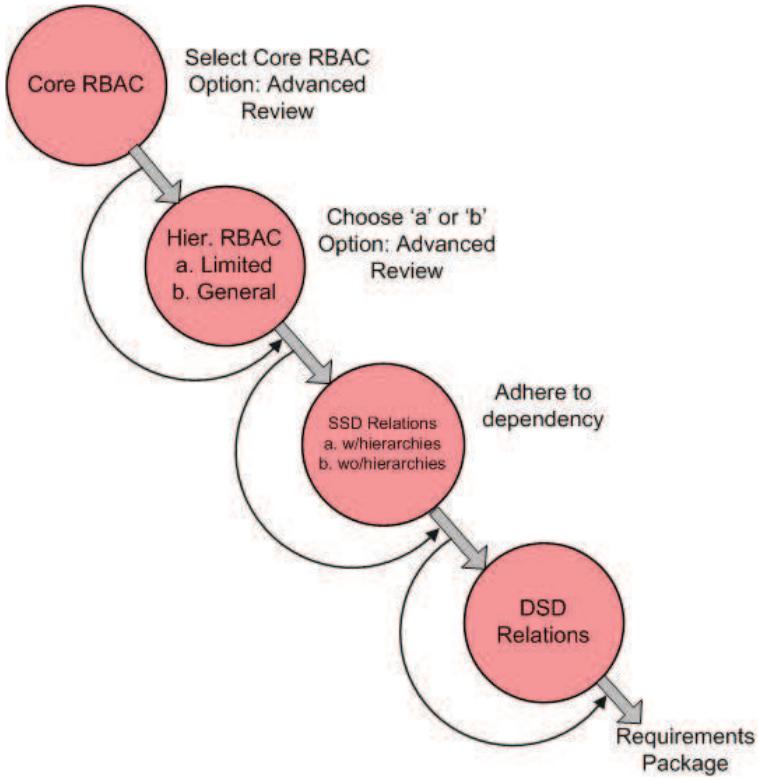

Figure 3.6: Methodology for creating functional packages [Ferraiolo *et al.* 2001]

and managing RBAC attributes on user sessions and making access control decisions.

As is seen in the previous sections, RBAC provides a diverse set of access control management features, compactly depicted in Fig. 3.6. This is a logical approach for defining packages of functional components where every package may pertain to a different threat environment and/or market segment. Ferraiolo states that each component can optionally be selected for inclusion into a package with an exception that Core RBAC must be introduced as a part of all packages. In a nutshell, RBAC offers four functional components: core RBAC, Hierarchical RBAC, Static Separation of Duty Relations, and Dynamic Separation of Duty Relations. Each functional component embodies three sections: administrative operations for the creation and maintenance of RBAC sets and corresponding relations, the review functions to understand/query a access control policy, and session management operations that involve activation of roles to a user's session. Within Core RBAC, advance review functions for permission-role assignment is also an optional feature, same as in Hierarchical RBAC.

Hierarchy is realized by defining an arbitrary partial order to serve as the role hierarchy. It's often the case that roles have overlapping capabilities or general permissions may exist within an organization, available to everybody. Oppositely, limited role hierarchies are subject to certain constraints such as simple structure trees or inverted trees.

Using Hierarchical RBAC, it offers the flexibility to go for multiple inheritance of permissions and user membership among roles. A limited hierarchy is essentially a tree (or inverted tree) structure. For a limited hierarchy, a function used by RBAC AddInheritance is constrained to a single ascendant (or descendant) role.

The two subcomponents of SSD relations are with or without hierarchies. If Hierarchical RBAC is chosen as a security implementation scheme, then SSD relations must adhere to the dependencies and should include the SSD with hierarchies. This looks quite intuitive since absence of such obligation can lead to catastrophic flaws in the security policies. For example, a role may obtain an associated permission when inheriting another role due to some insufficient SSD constraints. The last component of RBAC is DSD relations that is free from dependency relations except to Core RBAC. Addressing DSD with limited role hierarchies let most of the RBAC remain intact except that a number of supporting system functions are redefined such as CreateSession and AddActiveRole. Also, some administrative functions of RBAC are also modified like AddInheritance. For a detailed description of each of the modified functions, readers are referred to [Ferraiolo *et al.* 2001].

3.2.1 Summarizing Access Control Mechanisms

The MAC was originally developed for military systems where the documents could be labeled. DAC purely involves a discretionary activity to exercise access rights. RBAC is a natural evolution of these access control mechanism that tries to address their shortcomings. In software applications, the notion of roles has been used for nearly 25 years [Ferraiolo *et al.* 2001]. RBAC offered its use as a proper model for software security equally mature to MAC or DAC. In a decade or so, RBAC has emerged as one of the standard practices in computer security.

Presently, it is considered as a single access control model despite its multiple versions. It evolved in nearly a decade as a customizable, flexible, sophisticated and practical nature in that it can easily be implemented in organizations. Osborn [Osborn *et al.* 2000] have shown that RBAC can also be configured to enforce MAC and DAC, which demonstrates its expressiveness. RBAC is a centrally administrated access control scheme, but role permissions are not handled centrally if it needs to manage distributed RBAC implementations, which is a kind of flexibility.

However, RBAC has some limitations. The number of roles and users can be tens or thousands which looks a great challenge for RBAC administration. ARBAC97 [Sandhu *et al.* 1999] (happened to be decentralized administration scheme) is a solution toward that kind of problems. However, it's without some important features such as organization. The concept of *organization* works as a context of the roles by which they are employed. This feature has been later added by Oh et al [Oh *et al.* 2006] and the new model emerged as RBAC02. RBAC02 thoroughly explains the intricacies of ARBAC97 [Sandhu *et al.* 1999]. The main listed problem was to control the scope/boundary of administrative roles. Oh et al [Oh *et al.* 2006] keep the main features of ARBAC97 but add the notion of organizations to facilitate better administration of roles and users when the number is larger. ORBAC

[Abou El Kalam *et al.* 2003] also explored the organizational aspects of access control mechanism and discusses organization as a context. Likewise, some other papers have dealt with similar kinds of issues densely in [Strembeck & Neumann 2004], [Li & Tripunitara 2006], [Ferraiolo *et al.* 1999].

3.3 Data Security Properties

Access control mechanisms are intended to yield three basic data security properties availability, confidentiality, and integrity. Computer security is designed to cover these basic properties. For the sake of clarity, we only give their short explanations and definitions adapted from [Bishop 2003]. Except RBAC, MAC and DAC do not reward us with all the three security properties upon their implementation.

3.3.1 Availability

Definition 4. Let X be a set of entities and let I be a resource. Then I has the property of availability with respect to X if all members of X can access I.

The kind of availability property offered by RBAC deals with granting permissions which will ensure that the resource is available to X. Utilization of a particular service is handled by the availability property. For example, at certain time a sub-office tries to access an employee's information from the head-office. However, availability property, if compromised, can give rise to unpleasant situations, for example, attacks on an information system of an organization can overload a server and thus genuine users will no longer access to their resources. RBAC can allow only the legitimate users to access the resources. In other words, RBAC avoids to occur undesirable states in which an untrusted user gets access to a resource and a state in which a user who is entitled to an access permission does not get it.

3.3.2 Confidentiality

Definition 5. Let X be a set of entities and let I be some information. Then I has the property of confidentiality with respect to X if no member of X can obtain information about I.

Confidentiality of the data is an equally important aspect of computer security. In this context, data protection mechanisms often restrict the users to access only their data. For example, medical records of an hospital are of highly confidential nature and must be protected. Any leakage of information not only affects the repute of an organization but can leave behind irreversible damage to the patients whose records are unrevealed. RBAC deals with confidentiality by giving access to confidential data only to the authorized users.

3.3.3 Integrity

Definition 6. Let X be a set of entities and let I be some information or a resource. Then I has the property of integrity with respect to X if all members of X trust I.

Security policies, mechanisms, and procedures always rest on some assumptions. For example, a system administrator needs to install a security patch and the certain assumption about its correctness may be such as the patch was not tampered in the transit and the patch is thoroughly tested [Bishop 2003]. Bishop alludes to the point that any security policy, mechanism, or procedure is based on assumption that, if incorrect, it will destroy the superstructure on which it is built. Thus, the role of trust is crucial to the secure system development, which plays a vital role to ensure integrity property.

Integrity is more challenging than availability and the confidentiality. To avoid masquerading information is one of the internal as well as external challenges enterprise-wide. These attacks happen when an authorized user of some set of information, may exercise unauthorized attempts. Schneier [Schneier 2000] mentions a scenario for trading with stock. An employee from ParGain Technologies managed to post fake announcements which looked like from Bloomberg news service. This ended up in a worst effect since the stock went up by 30 percent before the truth was revealed. RBAC deals with integrity by controlling who may modify the protected data.

3.4 Why to Choose RBAC over MAC and DAC?

MAC, DAC, and RBAC contribute to ensure the above data security properties in one or another way. The feature that makes RBAC distinguishing from its counterparts is its ability to address enterprise-specific access control policies and to delicately handle the authorization management process of an organization, as stated in [Ferraiolo *et al.* 2001]. RBAC is also called non-discretionary access control mechanism since the owner of documents does not decide the accessibility. Another serious advantage of using RBAC is its flexibility and details of control as compared to the MAC and DAC standards. DAC is user oriented mechanism where a user has the authority to allow/disallow the permission. MAC is an inverse of that and is controlled by the system.

One of the limitations of MAC is its invisible user and administrator interactions with the model. Using MAC, it also appears infeasible to implement practical and cost-effective solution. Actually, this limitation applies to both MAC and DAC. Thus, MAC and DAC can't be fitted well in commercial business applications. Such restrictions have motivated the researchers to design RBAC.

RBAC has definite advantages over other access control mechanisms especially due to the concept of *roles* which is much conformed to the organizational structure. Thus it provides a coherent way to lay out authorization constraints. For example, a clinic may have the roles of doctors, nurses, surgeons and administrators. Applying RBAC will naturally enforce a mechanism in the form of roles which actually

addresses the needs of that organization.

A vital difference between military security policy and commercial security policy is that the former is primarily developed to provide confidentiality and latter is intended to ensure integrity. This determines which security mechanism be used to meet specific requirements. For example, a general consensus about RBAC is its applications in the domain of commercial systems while MAC is often regarded as a solution towards military systems. Their further prioritization and feature-wise comparison is beyond the scope of this thesis.

3.5 UML-based RBAC specifications

RBAC can also be used to model secure applications. To this end, there are a number of RBAC representations available in the literature (detailed in Chapter 4). One of the underlying motivations of using RBAC in designing secure applications is its easiness and practicality. SecureUML (and its tool SecureMOVA) is one of those techniques that adapts the principles of RBAC and offers a way to model and specify an access control system. SecureUML is a security profile that employs UML with some extensions, and OCL to express authorization constraints over structural diagrams. UMLSec [Jürjens 2005], USE [Sohr *et al.* 2005], are some of the other modeling languages capable of modeling a secure system. We have borrowed SecureUML to model the design of secure application. By and large, SecureUML appears a better choice to us since it has most of the concepts needed to sketch a security policy using RBAC. Thus, Section 3.5.1 is dedicated to introduce SecureUML.

3.5.1 SecureUML

SecureUML [Basin *et al.* 2009] offers a separation of functional and security aspects to model systems to secure. Functional part is expressed as UML diagrams while security part is based on RBAC basic elements such as users, roles, actions, permissions, objects, and the respective relations. It provides the way to declare static structure of system by using UML profile extensions like stereotypes, tagged values, and constraints. The semantics are then specified using set-theoretic semantics. In SecureUML, *security-design models* are based on system-designs + security model, and have concrete syntax (or notation) and abstract syntax. The goal of this language is to automatically generate system architectures embodying access control infrastructures from security-design models. The SecureUML can be combined with any design language (e.g., ComponentUML) both syntactically and semantically. SecureUML slightly extends RBAC in its proposed meta-model with fairly interesting concepts like authorization constraints using OCL, atomic and composite action, and dividing subject (user) into two categories i.e., group user and a single user.

3.5.1.1 SecureUML Metamodel

The SecureUML metamodel in Figure 3.7 defines actions against resources to protect (from the right side). It leaves out the exact definition of protected resources. Composite actions are actually applied to group lower-level operations to specify permissions for sets of actions. Atomic actions represent operations from the real modeled system. As SecureUML embeds RBAC into its approach, it customizes Core RBAC to model a system without using *sessions*.

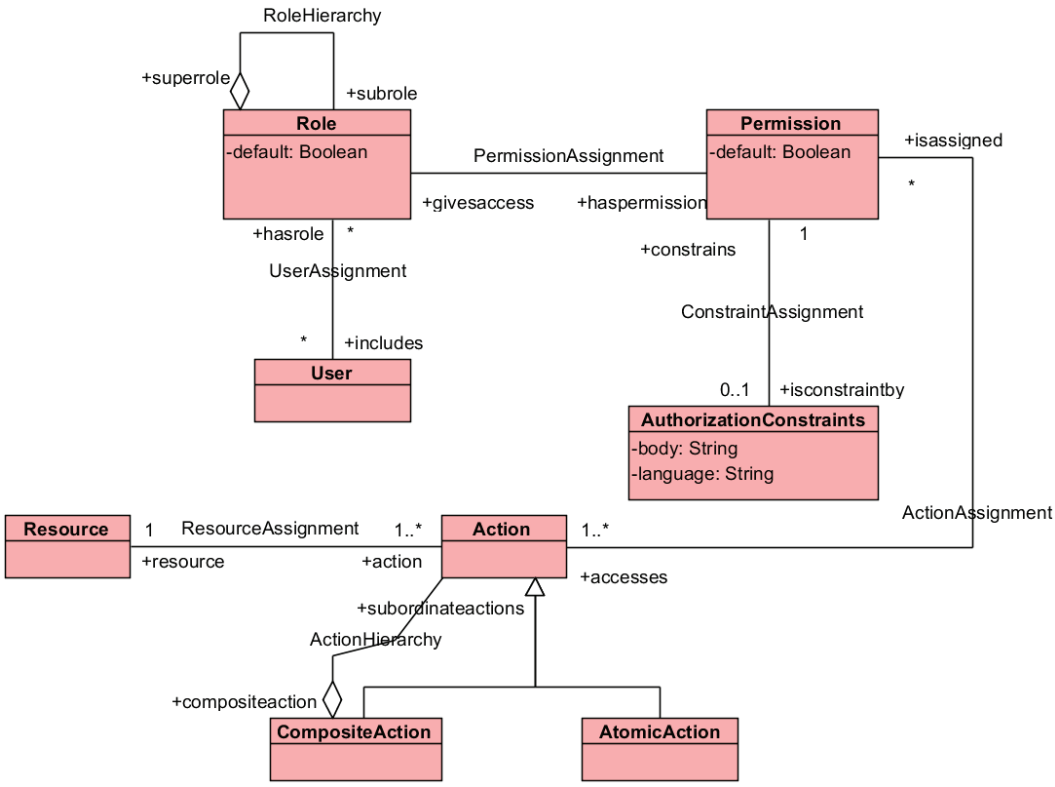

Figure 3.7: SecureUML metamodel [Basin *et al.* 2009]

The RBAC roles are given in the metamodel on the left side. Role inheritance is addressed along with roles to users assignment. SecureUML supports two types of access control decisions:

- The assignment of users and permissions to roles depending on the RBAC configuration, that is called *declarative access control decisions*.
- *Programmatic access control* are those which takes care of authorization constraints in the current system. Against these authorization constraints, the general approach for analyzing models is based on evaluating OCL expressions on snapshots of the metamodel. A snapshot can be considered as an object diagram from UML.

We have modeled two examples using SecureUML. The first is a meeting SCHEDULER application and second one is from the health care domain detailed in Chapter 5. The metamodel of SecureUML is given in Fig. 3.7. Two important notions which are detailed by SecureUML are the composite actions and atomic actions. CompositeAction is of five types: EntityFullAccess, EntityUpdate, EntityRead, AttributeFullAccess, and AssociationEndFullAccess. Whereas, Atomic actions are AtomicUpdate, AtomicRead, AtomicCreate, AtomicDelete, and AtomicExecute.

The atomic actions are intended to map directly onto actual operations of the modeled system. The composite actions are used to hierarchically group more lower-level ones and are used to specify permissions for sets of actions.More detail about them can be found in [Basin *et al.* 2009]. We now model our example using SecureUML.

The center of Fig. 3.8 gives the class diagram for the meeting SCHEDULER already given in 2.3. It includes two classes (Meeting and Person) and two associations. A meeting is characterized by its starting date and its duration; a person is simply characterized by his/her name. Most operation specifications have been created automatically using RoZ. They correspond to operations to create and delete objects, update object attributes, and create links between objects. Three operations are user-defined: notify and cancel are specific to the application, createMeeting creates of an object of class meeting and simultaneously links it to its owner and first participant. This operation is necessary in order to satisfy the arity constraints related to both associations: a meeting has at least one owner and one participant.

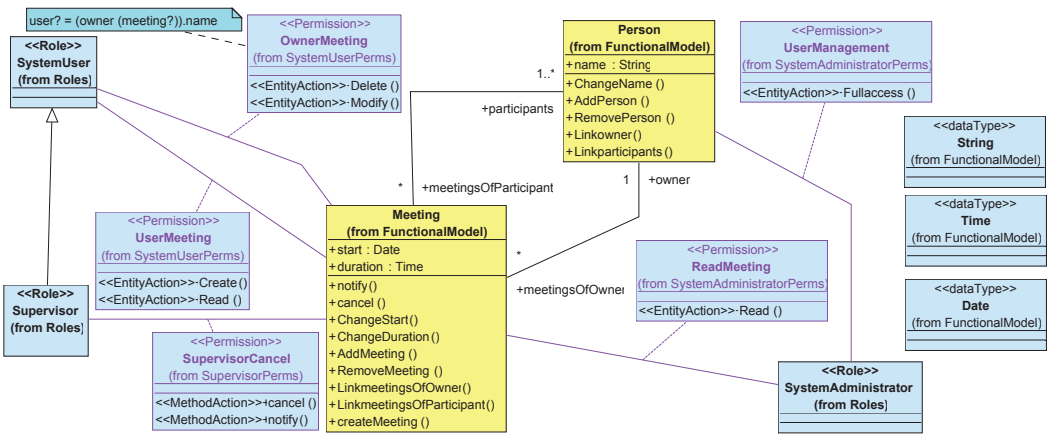

Figure 3.8: Class diagram and permissions for the meeting scheduler

The basic access control rules are: users are in charge of their meetings and system administrators manage the persons. An important security property is related to the integrity of information about meetings. It is expressed by the following rules: (1) a meeting may only be modified or canceled by its owner, and (2) supervisors have the privilege to modify or cancel meetings they do not own.

In Fig. 3.8 meeting SCHEDULER model is specified using SecureUML. This

model contains functional features as well as access control rules. We can see that each of the permissions is stereotyped between a role and a resource. For example, on meeting resource, we have linked systems users and the supervisor to exercise permissions such as OwnerMeeting, UserMeeting, and SupervisorCancel, respectively. The role supervisor also inherits the permissions from system users. Each of the permissions has an associated set of actions. There is another permission named UserManagement that authorizes a system administrator to manage persons using several operations such as ChangeName, AddPerson etc. He has full access which means that the administrator qualifies to perform all types of operations. The corresponding data types are given which are actually used during the translation to Z.

We remind that the RoZ translation of the functional part of this diagram 3.8 has been shown in Section 2.2.2 of the Chapter 2. However, the security part which involves the constructs from RBAC will be detailed in Chapter 6.

3.6 Summary

In this chapter, we detailed access control mechanisms targeted toward secure systems development. MAC and DAC are mandatory and discretionary access control mechanisms, respectively. RBAC is role-based or non-discretionary access control structure. RBAC variants include Hierarchical RBAC, with SSD and DSD along with the basic Core RBAC. We also showed how SecureUML uses RBAC to model an application. The design mechanism of SecureUML has also been exemplified using the SCHEDULER application in this chapter. The choice of using RBAC over MAC and DAC is also made clear when designing business applications. The conformance between an organization structure and RBAC configurations is one of the main reasons to that end. However, for designing military systems it may be appropriate to employ access control mechanisms such as MAC.

Chapter 4

Evaluating RBAC Supported Techniques and their Validation and Verification

Contents

4.1 Introduction **54**
4.2 RBAC Coverage as Evaluation Criteria **55**
4.2.1 RBAC Functions 55
4.2.2 RBAC Separation of Duties (SoD) Constraints 56
4.2.3 Other RBAC Variants 56
4.2.4 Verification and Validation Tools 56
4.3 Semi-formal Techniques **56**
4.3.1 A Summary of Semi-formal Techniques 59
4.4 Formal Languages and RBAC Constraints **59**
4.4.1 Alloy-based Approaches 61
4.4.2 Z-based Related Work 61
4.4.3 A Summary of Formal Techniques 62
4.5 Conclusion & Lessons Learned **62**

This chapter evaluates the security specification techniques that employ RBAC variants. The overall management of a RBAC supported system is made through its administrative, review and supporting system functions. In this chapter, a summary of semi-formal and formal techniques employing RBAC is provided along with their benefits and limitations. Here, semi-formal techniques refer to UML+OCL while formal ones are based on Alloy. This chapter may guide through the process of selecting an appropriate technique to specify security rules. This is done by analyzing the degree of coverage of RBAC including some extensions like SoD and role hierarchy. We also investigate the use of validation and verification tools in these techniques. We find that formal techniques are more amenable to automated analysis as compared to semi-formal ones. Semi-formal techniques are rich in specifying RBAC variants but have prototypic tools. Session based dynamic aspects of RBAC have been partly covered in both techniques. In this chapter, we also describe Z based approaches covering RBAC that are mostly at metamodel level. Most of the contents of this chapter have been reproduced from our paper [Qamar *et al.* 2011a]

Our surveying method is based on a search from various access control conferences and journals. The main keywords used are as follows: RBAC, validation and verification, UML, Z and Alloy languages. The rest of the chapter is structured as follows: Section 4.2 describes the evaluation criteria; Section 4.3 discusses the semi-formal techniques; Section 4.4 reviews the formal techniques. Section 4.4.2 details on other Z based works. Section 4.5 shares the learned lessons with some future work.

4.1 Introduction

The specifications may contain simple functional requirements as well as complex non-functional requirements such as security requirements, expressed as access control rules. These rules not only require robust specification methods but also validation and verification techniques to protect systems against malicious attacks. To this end, a number of such techniques have been presented in the literature. However, their strengths and limitations need to be analyzed. This chapter reviews two types of security specification techniques that use validation and verification approaches:

- Security Design Models are the first kind of techniques. They address access control rules in UML diagrams. These approaches favor a combination of UML modeling elements with OCL constraints. In turn, one can specify and validate a (sub)set of access control rules. A few example studies are [Yu *et al.* 2009], [Basin *et al.* 2006a], [Kim *et al.* 2004], [Basin *et al.* 2009], [Yu *et al.* 2008]. In the sequel, we call these semi-formal techniques.
- The second kind of techniques, [Schaad & Moffett 2002], [Ahn & Hu 2007], [Zao *et al.* 2003][Hu & Ahn 2008] to verify access control rules are based on formal specification languages (e.g., Z [Spivey 1992], Alloy [Jackson 2002]). These will be denoted as formal techniques.

As described in Chapter 3, RBAC [Ferraiolo *et al.* 2001] is intended to support access control properties such as confidentiality, integrity, availability – and consequently, it is mostly used as the underlying technique for the above classified approaches. In this context, we provide a survey of two kinds of aforementioned techniques and analyze their ability to verify or validate access control rules. The analysis criteria are:

1 Coverage of RBAC functions and constraints;

2 Coverage of static and dynamic aspects of RBAC;

3 Use of verification and validation tools (see Section 4.2.4).

The second criterion refers to the static data structure of RBAC and SSD that does not change over time; whereas dynamic constraint relations such as DSD are maintained during sessions. Based on the criteria, answers to the following questions are sought in this chapter:

- How much in depth coverage of RBAC variants is provided by an approach?
- To what extent, an approach is amenable to automated analysis for its respective validation or verification?
- How does each approach scale up to address reasonable size of problems?
- Does an approach entirely cover the administrative, review and supporting system functions of RBAC?
- Is an approach capable of addressing static and/or dynamic aspects of RBAC?

4.2 RBAC Coverage as Evaluation Criteria

The evaluation criteria take into account RBAC and its variants to analyze the extent to which an approach covers it. Here, we expand the three aspects of our evaluation criteria as follows.

4.2.1 RBAC Functions

RBAC functions have mainly three types: administrative functions, supporting system functions and review functions. This criterion will measure how the proposed technique covers the RBAC functions. A brief overview of each of these is given below:

- ***Administrative Functions*** involve creation and maintenance of basic sets elements. Of these sets are USERS, ROLES, OPS and OBS. Additionally, constructing relations among the sets is also covered by administrative functions such as UA and PA assignments.
- ***Supporting System Functions*** are useful in access control decisions and session management. These functions concern adding and dropping of active roles and other auxiliary functions, for instance: CreateSession, CheckAccess, AddActiveRole, DropActiveRole. For example, a user may have a fixed set of roles that can be used by him. The composition of this default set can be altered by supporting system functions.
- ***Review Functions*** help querying the data structures such as UA and PA assignments. The administrator may view the contents of specified relations through review functions. Through this feature, we can perform queries such as knowing the assigned users to a role, permissions of a role and allowed roles in a session. In the RBAC standard, some of the review functions are mandatory like querying the assigned users and assigned roles while a function such as querying permissions of a role, is an optional one. Therefore, not all RBAC implementations provide all the review functions.

4.2.2 RBAC Separation of Duties (SoD) Constraints

These constraints are an optional construct of RBAC and are used to address conflict of interest among roles.

- ***SoD*** constraints consist of two categories i.e., SSD and DSD. The former takes care of conflict of interest and ensures that it forbids a user to take some conflicting roles even in different sessions. DSD requires that a given user may not take conflicting roles simultaneously in the same session or in the life-cycle of a given object.

4.2.3 Other RBAC Variants

Besides RBAC functions and SoD constraints, there are additional constructs of RBAC that are considered here as a part of the evaluation process. Their brief overview is as follows:

- ***Role Hierarchy*** is based on the typical concept of inheritance that helps RBAC implementations to avoid repeated definitions of permissions. A hierarchy is mathematically a partial order through which one can define seniority relations. In RBAC senior roles acquire the permissions of juniors roles. Similarly RH can be combined with SSD or DSD.

- ***Session*** is a traditional way of communicating information between a user and a computer during a given time interval. Session management in RBAC deals with functions such as session creation for users including role activation/deactivation, enforcing constraints (e.g., DSD) on role activation. An obligatory part of DSD constraints is the use of sessions.

4.2.4 Verification and Validation Tools

Availability of verification or validation tools may grade the usability and scalability of an approach. In addition to the criteria listed above, we also study the availability of verification or validation tools in the surveyed techniques. The validation tools check the intended behavior of a system whereas verification ensures the consistency of the specification.

4.3 Semi-formal Techniques

Semi-formal methods develop a combination of UML+OCL, that can help graphically model and validate or verify RBAC rules of a secure system. Table 4.1 provides an assessment of RBAC supported features in each of these semi-formal techniques. In this context, in [Basin *et al.* 2009], Basin *et* al. use UML to specify both the functional aspects of an information system, and its security rules, using a UML profile named SecureUML and a tool SecureMOVA. The security part is based on RBAC concepts such as users, roles, actions, permissions, objects and relations UA

Table 4.1: Evaluation table for semi-formal techniques

Authors	A.F	R.F	S.S.F	SSD	DSD	RH	Sessions	V/V	Tools
[Basin *et al.* 2006b][Basin *et al.* 2009]	N	Y	N	N	N	Y	N	Y	SecureMOVA
[Sohr *et al.* 2005]	Y	Y	P	Y	Y	Y	P	Y	USE
[Ç. Cirit & Buzluca 2009]	P	P	P	Y	Y	Y	Y	N	N
[Yu *et al.* 2008]	P	N	N	Y	Y	N	N	Y	Y
[Yu *et al.* 2009]	P	P	P	Y	Y	Y	Y	Y	N
[Ray *et al.* 2004]	Y	N	N	Y	Y	Y	Y	Y	N
[Kim *et al.* 2004]	Y	P	Y	Y	Y	Y	Y	Y	N
[Ahn & Hu 2007]	P	Y	P	Y	Y	Y	Y	Y	Y
[Alghathbar 2007]	P	N	N	Y	Y	N	N	Y	AuthUML

[a][p=partial, y=yes, n=no], A.F = Administrative Functions, R.F = Review Functions, S.S.F = Supporting System Functions, SSD = Static Separation of Duty, DSD = Dynamic Separation of Duty, RH = Role Hierarchy

and RA. Inspired by RBAC review functions, SecureUML first graphically models a policy and then it allows to check roles assigned to a permission. But SecureUML only covers static aspects of RBAC without taking into account sessions. Although SecureUML is designed to be combined with any modeling language, tool support is proposed only for the combination of SecureUML+ComponentUML. The SecureMOVA tool [Basin *et al.* 2009] allows to create a functional diagram, i.e., a class diagram, and to relate it to permission rules.

Sohr *et* al. [Sohr *et al.* 2005] take advantage of the USE tool to validate administrative, supporting system and review functions of RBAC on a given object diagram. Constraints (invariants) are associated to classes of a diagram. Moreover, pre- and post conditions of operations can be evaluated if the object diagram represents the initial or final state of some operation. USE covers more RBAC constructs than SecureMOVA. The covered elements are sessions (but supporting system functions are partially detailed in USE), SSD, DSD in addition to basic elements such as users and roles.

Cirit and Buzluca [Ç. Cirit & Buzluca 2009] propose a UML profile for the RBAC authorization constraints. It introduces the UML profile stereotypes like user, resource, operation, role, permission, session for modeling a system. It also supports to model DSD and SSD constraints. OCL based validation is proposed in order to measure the well-formedness (syntax) and meaning (semantics) of information models against the RBAC constraints. However, this technique relies on general CASE-tools support and does not present any particular insight into the automation part of verification or validation.

Yu *et* al. [Yu *et al.* 2009] propose a technique to uncover violations in security rules specified by using RBAC. Their technique implements role activation and SoD constraints and role hierarchy using UML and OCL. In their approach, scenarios are used to analyze security violations of a given security policy where a scenario is a sequence of operation calls. Based on a generated tree (limited depth, limited number of objects, and small domain) of various invocations, scenarios are generated. Afterwards, it is possible to verify legal (intended) and illegal (unintended) scenarios in the design. Using this technique, one can analyze role activation constraints and SoD constraints. Yu *et* al., [Yu *et al.* 2008] also present another technique in which a class model is transformed into a static model of behavior, called a *Snapshot Model* (where a snapshot describes an application state). A snapshot model is comprised of sequences of snapshots. A snapshot is an application state. A sequence of operation invocations (described as scenario) is verified against the snapshot model. Both Yu's techniques are very closely related except that in [Yu *et al.* 2008] they demonstrate the use of tools such as USE or Object Constraint Language Environment (OCLE) for verifying structural properties of the models. Such techniques can be regarded as visualization techniques to uncover security rules violations.

Shin and Ahn [Shin & Ahn 2000] state UML based representation of RBAC in terms of static view, functional view and dynamic view. Static modeling provides a structural view of RBAC base systems using class diagrams; functional view is taken as use cases and actors of the systems; in dynamic aspects, they refine use

cases to show interactions among the objects participating in each use case using collaboration diagrams. In another paper, Ahn and Shin [Ahn & Shin 2001] also attempt the use of OCL for RBAC constraints. However, validation or verification of the properties remains unaddressed in both of these papers.

Ray *et* al. [Ray *et al.* 2004] discuss SoD constraints by using object diagrams and try to alleviate the complexity of OCL by using templates. The RBAC constraints that are checked include SoD, prerequisite constraints and cardinality constraints. Kim *et* al. [Kim *et al.* 2004] also present UML templates to specify static structure of RBAC along with SSD and DSD. They examine violation pattern occurrences in a security policy.

Ahn and Hu [Ahn & Hu 2007] explain an approach using UML class diagrams, a language for specifying role-based authorization constraints (RCL2000) and OCL to validate SoD constraints. In this approach, it is checked whether a current state violates authorization constraints. Snapshots based on object diagrams are used to determine violated constraints. They cover most of the RBAC constructs with a tool support. Alghathbar [Alghathbar 2007] describes an approach to specify access control policies into use case diagrams. This approach tends to analyze access control policies at early stages of secure software development. It must be noted that UMLSec [Jürjens 2005] is another attempt to address security in UML, but it focuses on the security model (in particular cryptographic aspects) and does not address its interaction with the functional model. It is also not based on RBAC so we do not review this approach in this thesis.

4.3.1 A Summary of Semi-formal Techniques

The semi-formal techniques are recapitulated and analyzed in Table 4.1 against the defined criteria. The values specified in evaluation against the criteria can be interpreted as: P (Partial), Y (Yes), and N (No). Our study shows that none of the approaches fully covers RBAC variants in terms of administrative, review and supporting system functions except [Ahn & Hu 2007] that covers most of the RBAC variants. It shows that almost half of the approaches do not provide automation for their proposals. We believe that practical use of an approach may be possible only through availability of automated tools. SecureMOVA and USE significantly contribute towards automated coverage of RBAC variants. Tool support can aid to improve the usability and scalability of an approach. So these two techniques have quite sufficient implementation of their concepts. Yet, SecureMOVA does not cover dynamic aspects in terms of sessions while USE addresses it partially.

4.4 Formal Languages and RBAC Constraints

Together with semi-formal techniques, formal languages, for instance, Z [Spivey 1992] and the Alloy [Jackson 2002] are elegant specification methods that can be applied to verify or validate software specifications. Alloy and Z are much adopted languages to study RBAC variants. As compared to Z, Alloy has more mature tools such as

Table 4.2: Evaluation table for formal techniques

Authors	A.F	R.F	S.S.F	SSD	DSD	RH	Sessions	Verification	Tools
[Zao *et al.* 2003]	Y	N	N	Y	N	Y	Y	Y	A-Analyzer
[Schaad & Moffett 2002]	Y	P	N	Y	Y	Y	Y	Y	A-Analyzer
[Ahn & Hu 2007]	Y	P	P	Y	Y	Y	Y	P	A-Analyzer
[Hu & Ahn 2008]	Y	N	N	Y	Y	Y	Y	Y	A-Analyzer
[Toahchoodee *et al.* 2009]	N	N	N	Y	N	N	N	Y	A-Analyzer

[a] [p=partial, y=yes, n=no], A.F = Administrative Functions, R.F = Review Functions, S.S.F = Supporting System Functions, SSD = Static Separation of Duty, DSD = Dynamic Separation of Duty, RH = Role Hierarchy

Alloy Analyzer. Moreover, Z has mostly been used to specify RBAC concepts at the abstract level. On the contrary, Alloy is applied to specify a given security policy, and its verification tools can be used to analyze this particular policy. An added advantage of Alloy over other languages (e.g., OCL, Z) is that Alloy Analyzer can search instances satisfying complex set of predicates [Power *et al.* 2010]. Alloy is used to generate counterexamples against specifications that help verifying the system.

4.4.1 Alloy-based Approaches

Simulating a system using Alloy involves individual transitions or properties of sequences of transitions. However, use of Alloy for dynamic modeling of security policies has been a scant subject so far. Most of the proposed approaches merely focus on the static analysis where Alloy is used for generating counterexamples against specifications. As an added advantage of Alloy over other languages (e.g., OCL and UML), it is reported [Power *et al.* 2010] as more amenable to automatic analysis. Alloy offers two kinds of automated analysis i.e., *simulation* and *checking*. In simulation, operations are interpreted to compute resulting states, and check that they conform to invariant properties. In checking, Alloy attempts to generate instances of a data structure up to a given (small) maximum size, and can identify counterexamples which do not satisfy a given property. The types of answers that Alloy provides are: *"this property always holds for problems up to size X"* or *"this property does not always hold, and here is a counter example'.*

Regarding the analysis of security models, especially RBAC with SoD constraints, significant amount of work has been carried out using Alloy. Zao [Zao *et al.* 2003] proposes a technique to verify algebraic characteristics of RBAC schema using Alloy. Alloy is used as a constraint analyzer to check the inconsistencies among roles and SSD constraints, and provide a counter-example when some inconsistency is found. However, the authors do not report on DSD but only SSD constraints. Schaad *et* al. [Schaad & Moffett 2002] and Ahn *et* al. [Ahn & Hu 2007] also discuss SoD constraints. The former [Schaad & Moffett 2002] thoroughly discuss decentralized administration of RBAC and arbitrary changes to a initially stated model that may result in conflicting policies over time *w.r.t* SoD constraints. They argue that SoD constraints may introduce implicit security policies flaws because of role hierarchies. Counterexamples are used to examine a specified security policy. [Toahchoodee *et al.* 2009] have discussed a translation from UML to Alloy and then a verification is run to verify UML models. This study mainly involved the analysis of contextual information such as location and time, for making access decisions in real-world dengue decision support (DDS).

4.4.2 Z-based Related Work

Hall [Hall 1994] used Z to specify a formal security policy model for an industrial project. Likewise, ISO standardized RBAC has widely been described by researchers

using Z. A few notable propositions are [Abdallah & Khayat 2006],[Yuan *et al.* 2006] that offer generic formal representation of RBAC. Yet, these works focus on abstract model foundations of RBAC.

Various validation and verification of security properties based on RBAC are given in [Morimoto *et al.* 2007], [Boswell 1995]. Abdallah, [Abdallah & Khayat 2006] defines a security administration using access monitor for core RBAC and distinguishes among various concepts of RBAC. Boswell [Boswell 1995], describes a security policy model in Z, for NATO Air Command and Control System (ACCS). The work targeted to develop a model for both mandatory and discretionary access controls based on the Bell-LaPadula approach.The author shares learned lessons from manual validation of this large, distributed, and multi-level-secure system. This too questions manual versus automated validation/verification and creates room for tools like Jaza. Morimoto *et* al., [Morimoto *et al.* 2007] chose a common-criteria security functional requirements taken from ISO/IEC-15408 and proposed a process to verify Z specifications by the Z/EVES theorem prover.

4.4.3 A Summary of Formal Techniques

During the recent years Z has been used to write precise software specifications of numerous software applications. The standard RBAC has also been specified using Z but its use at application level is not widely available.

The other formal techniques reviewed in this chapter take the full advantage of Alloy Analyzer to verify specifications. We compare Alloy with semi-formal approaches because Alloy is relatively lightweight modeling system. Underlying shortcomings of OCL [Vaziri & Jackson 2000] have motivated the design of Alloy. One limitation of Alloy is its response time against verifying specifications and memory crashes. These can be avoided if the specifications are rightly decomposed into small components. The discussed formal techniques mainly cover SSD constraints and thus, stick to static aspects of RBAC. Dynamic aspects based on DSD are partially covered in some techniques using an unaltered set of sessions. Table 4.2 summarizes the analysis of the techniques.

Considering the fact that Alloy is a fairly expressive language with its robust tools, it can be used to investigate the unaddressed aspects of RBAC such as supporting system functions. RH combined with SSD and DSD can be interesting to study using Alloy. Another interesting direction is the translation of UML functional models to Alloy. [Anastasakis *et al.* 2010] discusses this direction but does not offer translation of security profiles. An extended work of such attempts can be helpful to exploit Alloy Analyzer for secure software development.

4.5 Conclusion & Lessons Learned

In this chapter we have surveyed and analyzed techniques that tend to validate or verify RBAC supported security rules. The first type of reviewed approaches is semi-formal that adopt UML+OCL to help design a security policy. The second

type uses Alloy for similar purposes. Based on this survey, we recall our raised questions in Section I, and provide their answers:

- *Regarding the coverage of RBAC variants:* Semi-formal techniques have covered more RBAC variants than formal ones. USE and SecureMOVA provide sufficient implementations and RBAC coverage to specify and validate security specifications.

- *Amenability to automated analysis:* Formal techniques are more rigorous since Alloy Analyzer is pretty mature for design verification problems [Power *et al.* 2010]. Alloy, being a declarative language can help construct partial models that can be built and analyzed easily. The 4.2 shows that all the formal techniques considered in this study use Alloy Analyzer. On part of semi-formal techniques, USE and SecureMOVA can be of great help if the UML diagrams are aimed to be analyzed and are of quite formal nature. In these tools, OCL is chosen as a natural partner of UML.

- *About the scalability of the techniques:* We believe that the scalability of a technique depends on the automation level it offers. Out of the studied approaches, Alloy Analyzer, USE and SecureMOVA are relatively mature tools and have the ability to automatically handle RBAC supported systems.

- *Coverage of entire RBAC functions:* There is no systematic coverage in both semi-formal and formal techniques of administrative, review and supporting functions. However, this is not a definite shortcoming since these techniques can be extended to cover the remaining features of RBAC.

- *Coverage of static and dynamic aspects:* Both semi-formal and formal techniques have significantly dealt with static structure of RBAC; however, dynamic aspects in terms of sessions are not widely studied. Besides DSD constraints, Yu *et* al. [Yu *et al.* 2009], [Yu *et al.* 2008] study sequences of snapshots, where a snapshot denotes an application state. Such studies involving states of a designed application can give further perspectives on dynamic analysis of security specifications and therefore, are worth to explore.

This chapter presented an analysis of RBAC supported techniques. Another survey of security design techniques [Talhi *et al.* 2009] employs different criteria such as expressitivity, tool support, verifiability and complexity but does not focus on RBAC. However, a paper [Drouineaud *et al.* 2004] have discussed the formalisms and methods for the validation of RBAC policies. [Sohr *et al.* 2005] has proposed protecting clinical information systems to overcome risks by using first-order LTL supported by Isabelle/HOL for formal verification of security policy for RBAC. Our survey shows that semi-formal techniques have been applied both ways for validation and verification on security specifications. Formal techniques focus on a sort of verification using Alloy Analyzer that aim at finding counterexamples. Alloy Analyzer has been continuously improved by adding new functionalities. Its

latest version can be found on its web-page [1]. Comparatively, other tools analyzing graphical models have not been incrementally improved in either case graphically or feature-wise. SoD constraints, one of the RBAC properties are commonly studied in both of the techniques. Generally, all the semi-formal techniques have prototypic tools. In small systems, it is relatively easy to apply prototypic tools but their large scale use needs further industrialization.

[1] (http://alloy.mit.edu/community/)

Part II

CONTRIBUTIONS

Chapter 5

Validation of Security-Design Models Using Z

Contents

5.1 Introducing the Proposed Z-based Toolset **68**
5.2 Illustrative Example : Medical Information System **69**
5.3 Translating the Functional Model into Z **71**
5.4 The Security Kernel **73**
5.4.1 Permissions 74
5.4.2 Role Hierarchy 74
5.4.3 Action Hierarchy 75
5.4.4 Roles, Users and Sessions 77
5.4.5 Putting it All Together 78
5.5 Linking Functional and Security Models **80**
5.6 Validating and Animating Secure Operations **81**
5.6.1 Normal Behavior 81
5.6.2 Analyzing a Malicious Behavior 82
5.7 Summary **83**

This chapter presents our translation of functional security models into Z and how these can be validated, using the Jaza animator. Our approach has the following goals: (1) to start from an intuitive graphical specification which features both functional and security models, 2) to systematically construct a formal specification of the integrated system from the graphical model, and 3) to use queries and animation to validate the integrated model. It must be noted that we currently focus on validation, i.e., confront our model to the user, and ensure that it exhibits the expected behavior. Although we use formal methods, we do not address verification (i.e., prove that the system is right) at this stage.

The content of this chapter is structured in the following order: in Sec. 5.1 we outline our approach. Sect. 5.2 introduces an illustrative example. Sect. 5.3 recalls the principles of the translation of the functional model, while Sect. 5.4 features the specification of the security kernel. The integration of both models is described in Sect. 5.5. Sect. 5.6 features the validation activities, based on animation. Finally, the conclusion of this chapter will be detailed. Most of the contents of this chapter

have been adapted from our paper [Qamar *et al.* 2011b]. In the first instance, we overview our approach presented in Fig. 5.1.

5.1 Introducing the Proposed Z-based Toolset

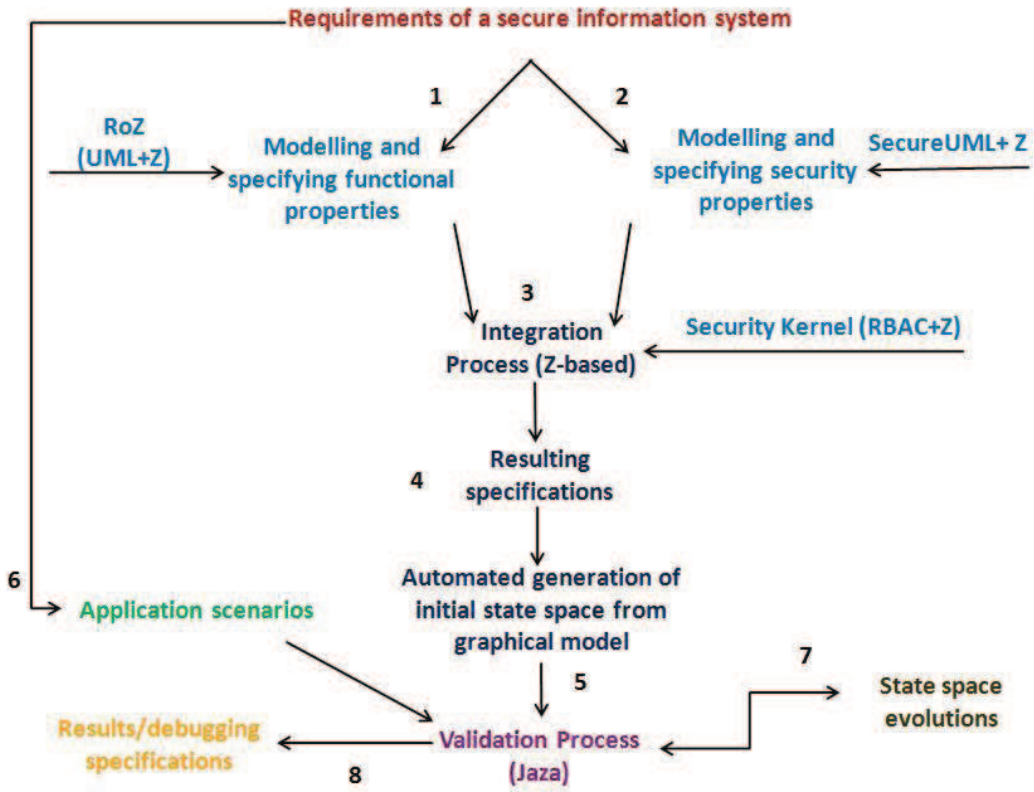

Figure 5.1: Bird's eye view of Z-based toolset

- The first step is to model an application using UML on which RoZ is applied to have Z annotated skeleton. We can consider the example shown in Section 2.2.2 (Fig. 2.3), meeting SCHEDULER, which has been successfully transformed into Z. This translation respects the rules of RoZ and generates the data structure as well as the basic operations which can also be animated.

- The second step involves the use of SecureUML splittable into two parts: a) A modeled application in SecureUML is translated to Z; b) An independently developed security kernel on top of RBAC access control model specified in Z. 'a' and 'b' are mutually interlinked. Chapter 3 has explained RBAC and SecureUML in detail.

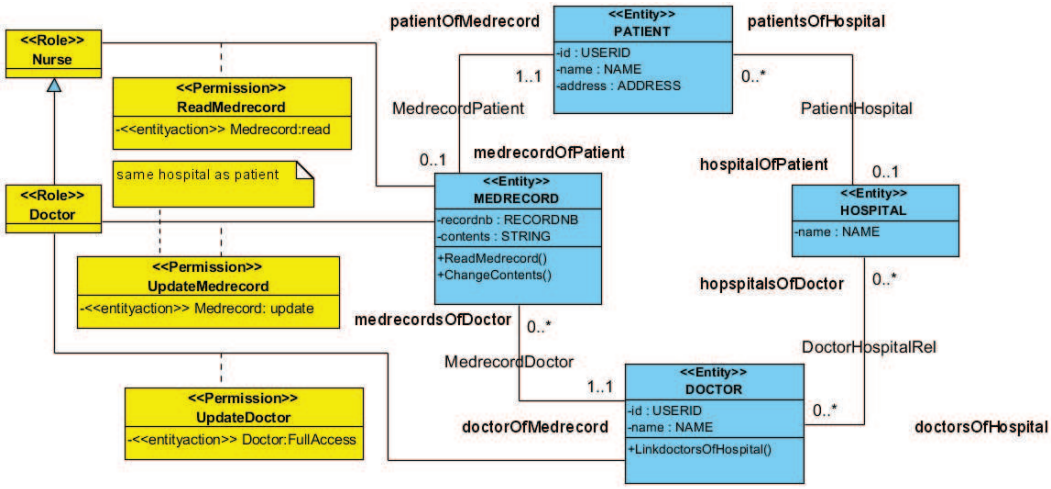

Figure 5.2: Security policy model using SecureUML

- At now, using step 1 & 2, we have obtained Z specifications of both functional and security parts. The next step is to merge them in order to proceed with the validation process. This covers step numbers 3, 4 & 5.

- As we start from a graphical model, the graphical model is automatically translated into an initialization state by our toolset. Nonetheless, specification of the secure versions of the functions must be done manually.

- The next step is to create application-specific scenarios which could have the ability to investigate the security and functional properties of the modeled system. Scenarios can be simple as well as complex. A scenario may be a single operation or a set of operations.

- Lastly, the designed scenarios are validated against the application's initial state. In turn, specifications may be debugged if some discrepancies are found. The state space is explored and its evolutions are studied to determine the loopholes in specifications.

5.2 Illustrative Example : Medical Information System

Fig. 5.2 models a simple medical information system using SecureUML. The figure has two sides where functional features on the right, are decoupled from security features on the left. The functional part describes four classes: patients, doctors, hospitals and medical records. Each medical record corresponds to exactly one patient. Its field `contents` stores confidential information whose integrity must be preserved. The functional part also records the current hospital hosting the patient,

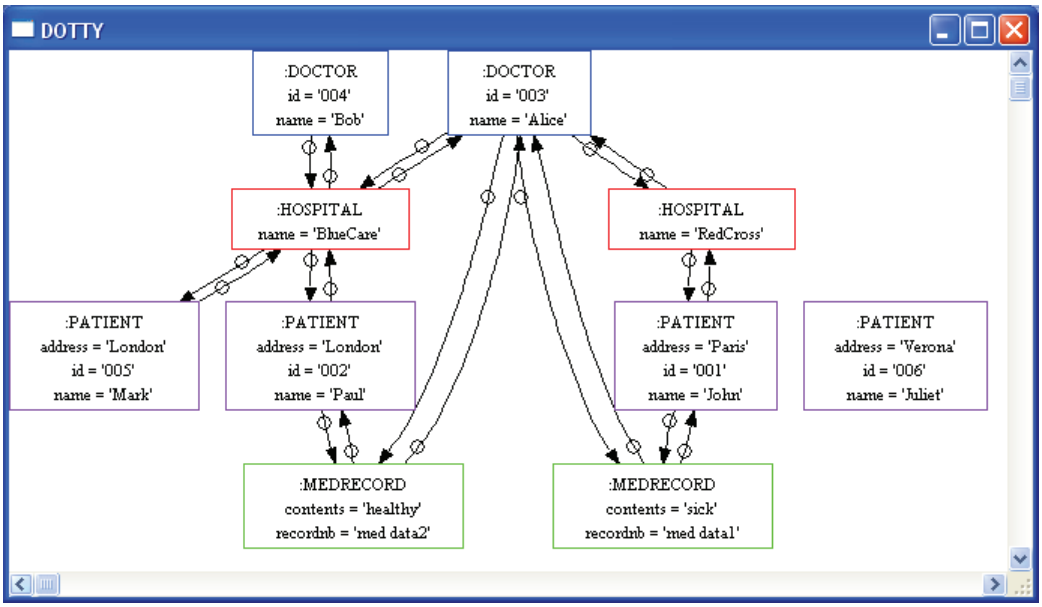

Figure 5.3: Object diagram for the functional model produced from the output of Jaza

the doctors working in this hospital, and the one responsible for the patient's medical record.

Fig. 5.3 gives an object diagram corresponding to this functional model. It features 4 patients, 2 doctors (Alice and Bob), 2 medical records, and 2 hospitals. Alice is linked to both hospitals, while Bob only works for one of them.

The left part of Fig. 5.2 describes the access control rules of the information system. In SecureUML and RBAC, users of the system are abstracted into roles, and permissions are granted to roles. Fig. 5.2 features two roles : Nurse and Doctor. An inheritance relation links Doctor to Nurse, expressing that doctors inherit all permissions of nurses. Confidentiality and integrity must be ensured for medical records. Two permissions rule the access to class MEDRECORD. Permission ReadMedRecord is granted to nurses (and inherited by doctors). It expresses that nurses and doctors have read access to medical records. It refers to entity action: read which designates operations accessing the class without modifying it. Although the security part of Fig.5.2 uses the graphical syntax of UML (classes and association classes), elements stereotyped as roles or permissions only make sense for security concerns. For example, there will not be objects of type Nurse in our animations, but there will be users playing this role. Similarly, the associative class ReadMedrecord is simply a graphical notation to express the permissions associated to nurses and related to medical records.

Permission UpdateMedrecord grants additional rights to doctors, who may update medical records. Constraint "Same hospital as patient" restricts this permission to the doctors linked to the same hospital as the patient. In Fig. 5.3, it means that

only Alice may modify the medical record of John, numbered "med data1", because she is the only doctor linked to RedCross hospital. In SecureUML, such constraints are expressed in OCL; here, they will be expressed in the Z language. In Fig. 5.2, a third permission named UpdateDoctor grants to all doctors full access, i.e., read and update access, to objects of class DOCTOR.

Validation studies normal and malicious behaviors. In this chapter, besides some nominal scenarios, we study the following attack: Bob, a malicious doctor, wants to corrupt the integrity of John's medical record. Since Bob is not working for RedCross hospital, the access control rules should forbid this modification.

5.3 Translating the Functional Model into Z

The following sections describe how the integrated graphical model of Fig. 5.2 is translated into a Z specification. First, the RoZ tool automatically translates the functional model, corresponding to the right part of Fig. 5.2. An optional feature of the tool also generates basic operations such as setters and getters for the attributes and the associations. These basic operations are often implicitly included in a class diagram; so the tool avoids the analyst to manually specify each of these operations. One may fear that, in a security context, these additional operations augment the "attack surface" of the application. In such a case, the analyst can choose to disable their optional generation, or to exclude them from the set of operations linked to the access control model (Sect. 5.5). Here are some elements of the formal specification generated from the functional diagram of the medical record information system. First, the types of the class attributes are introduced as given types.

$$[NAME, USERID, STRING, RECORDNB]$$

$$\begin{array}{|l}
\hline
\textit{MEDRECORD} \\
\textit{recordnb} : \textit{RECORDNB} \\
\textit{contents} : \textit{STRING} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\hline
\textit{MedrecordExt} \\
\textit{Medrecord} : \mathbb{F}\, \textit{MEDRECORD} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\hline
\textit{DOCTOR} \\
\textit{id} : \textit{USERID} \\
\textit{name} : \textit{NAME} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\hline
\textit{DoctorExt} \\
\textit{Doctor} : \mathbb{F}\, \textit{DOCTOR} \\
\hline
\end{array}$$

Every class is translated into two Z schemas. The first one, a schema type, describes the type of the elements of the class. This schema corresponds to the class

intent and lists the class attributes. Schemas MEDRECORD and DOCTOR describe the intent of the corresponding classes. A second schema describes the extension of the class, i.e., the set of objects belonging to the class. Schemas MedrecordExt and DoctorExt correspond to these extensions; each of these includes a finite set of objects corresponding to the type of the class.

During a Jaza animation, each object is represented as a list of pairs attribute == value. The list is enclosed between $\langle\!| \ldots |\!\rangle$. Here is the Jaza representation of sets Doctor and Medrecord corresponding to the state of Fig. 5.3.

$$
\begin{aligned}
&Doctor' == \{\langle\!| \, id == \text{"003"}, name == \text{"}Alice\text{"} \, |\!\rangle, \langle\!| \, id == \text{"004"}, name == \text{"}Bob\text{"} \, |\!\rangle\} \\
&Medrecord' == \{\langle\!| \, contents == \text{"}healthy\text{"}, recordnb == \text{"}meddata2\text{"} \, |\!\rangle, \\
&\qquad\qquad \langle\!| \, contents == \text{"}sick\text{"}, recordnb == \text{"}meddata1\text{"} \, |\!\rangle\}
\end{aligned}
$$

UML associations are translated by RoZ as a pair of functions corresponding to both roles of the association. For example, functions hospitalsOfDoctor and doctorsOfHospital describe the association between doctors and hospitals. Their domain and range are constrained by predicates of the schema. Additional predicates express that the inverse role can be constructed from the direct one.

$$
\begin{array}{|l}
\hline
\textit{DoctorHospitalRel} \\
HospitalExt;\ DoctorExt \\
hospitalsOfDoctor : DOCTOR \nrightarrow \mathbb{F}\, HOSPITAL \\
doctorsOfHospital : HOSPITAL \nrightarrow \mathbb{F}\, DOCTOR \\
\hline
\mathrm{dom}\, hospitalsOfDoctor \subseteq Doctor \\
\bigcup(\mathrm{ran}\, hospitalsOfDoctor) \subseteq Hospital \\
hospitalsOfDoctor = \{doctor : \bigcup(\mathrm{ran}\, doctorsOfHospital) \bullet doctor \mapsto \\
\qquad \{hospital : \mathrm{dom}\, doctorsOfHospital \mid doctor \in \\
\qquad\quad doctorsOfHospital(hospital) \bullet hospital\}\} \\
doctorsOfHospital = \{hospital : \bigcup(\mathrm{ran}\, hospitalsOfDoctor) \bullet \\
\qquad hospital \mapsto \{doctor : \mathrm{dom}\, hospitalsOfDoctor \mid hospital \\
\qquad\quad \in hospitalsOfDoctor(doctor) \bullet doctor\}\} \\
\hline
\end{array}
$$

Here is how Jaza represents role doctorsOfHospital corresponding to Fig. 5.3.

$$
\begin{aligned}
&doctorsOfHospital' == \\
&\qquad \{(\langle\!| \, name == \text{"}BlueCare\text{"} \, |\!\rangle, \quad \{\langle\!| \, id == \text{"003"}, name == \text{"}Alice\text{"} \, |\!\rangle, \\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad \langle\!| \, id == \text{"004"}, name == \text{"}Bob\text{"} \, |\!\rangle\}), \\
&\qquad (\langle\!| \, name == \text{"}RedCross\text{"} \, |\!\rangle, \quad \{\langle\!| \, id == \text{"003"}, name == \text{"}Alice\text{"} \, |\!\rangle\})\}
\end{aligned}
$$

Finally, we give several specifications of operations. MRChangeContents is a setter for field contents. This operation, which works on the type of medical records, must be "promoted" to impact the actual contents of the class extension, and to modify the related associations. MRChangeContentsP, the promoted operation takes an additional input x? designating the object to modify[1].

[1]For more information on operation promotion please refer to Wordsworth's text [Wordsworth 1992] page 137.

$$
\begin{array}{|l}
\hline
\textit{MEDRECORDChangeContents} \\
\Delta MEDRECORD \\
newcontents? : STRING \\
\hline
contents' = newcontents? \wedge recordnb' = recordnb \\
\hline
\end{array}
$$

$$
\begin{aligned}
&MedrecordChangeContents == (ChangeMedrecord \\
&\quad \wedge MEDRECORDChangeContents) \setminus (recordnb, recordnb') \setminus (contents, contents')
\end{aligned}
$$

Please note that the operation MRChangeContentsP is actually MedrecordChange-ContentsandRels given in the appendix B.

MRChangeContentsP ==

$$
\begin{aligned}
&MedrecordChangeContentsandRels == (ChangeMedrecord \wedge \\
&\qquad MEDRECORDChangeContents \wedge SubstituteMedrecordInRels) \\
&\qquad \setminus (recordnb, recordnb') \setminus (contents, contents')
\end{aligned}
$$

Operation DRLinkDoctors creates a link between a doctor and an hospital. Its predicates distinguish between the case where a first doctor is linked to the hospital, and the case where doctors were already linked to this hospital.

$$
\begin{array}{|l}
\hline
\textit{DRLinkDoctors} \\
\Xi HospitalExt;\ \Xi DoctorExt;\ \Delta DoctorHospitalRel \\
hospital? : HOSPITAL;\ doctor? : DOCTOR \\
\hline
(hospital? \notin \mathrm{dom}\ doctorsOfHospital) \Rightarrow (doctorsOfHospital' = \\
\qquad doctorsOfHospital \oplus \{hospital? \mapsto \{doctor?\}\}) \\
(hospital? \in \mathrm{dom}\ doctorsOfHospital) \Rightarrow (doctorsOfHospital' = \\
\qquad doctorsOfHospital \oplus \{hospital? \mapsto \\
\qquad\quad (doctorsOfHospital(hospital?) \cup \{doctor?\})\}) \\
\hline
\end{array}
$$

These operations are sufficiently detailed to animate the model with Jaza. After several steps, one may end up with a state corresponding to Fig. 5.3. Nevertheless, these operations don't take into account the access control rules. In particular, they are not aware of which user is executing them. This will be the responsibility of the security kernel described in the next section.

5.4 The Security Kernel

The translation process proceeds with the security part of Fig. 5.2. Our approach is based on a reusable security kernel which specifies the main concepts of RBAC in Z. This security kernel is instantiated with the roles, permissions, resources and operations of the SecureUML diagram.

5.4.1 Permissions

A permission assignment links a role to an operation on a given class, also called the protected resource. These four types are introduced in Z as given types or as enumerated types. When considering enumerated types, the values of the type must be extracted from the UML diagram in order to instantiate the security kernel. Here are the type declarations corresponding to Fig. 5.2. Schema Sets includes sets of values corresponding to each of these types.

$[PERMISSION]$
$ROLE ::= Doctor \mid Nurse$
$RESOURCE ::= Medrecords \mid Patients \mid Doctors \mid Hospitals$
$ABS_ACTION ::= EntityRead \mid EntityUpdate \mid EntityFullAccess$

Sets

$role : \mathbb{F}\, ROLE$
$resource : \mathbb{F}\, RESOURCE$
$permission : \mathbb{F}\, PERMISSION$
$abs_action : \mathbb{F}\, ABS_ACTION$

Below are the values of these variables during the Jaza animation, as they appear after the initialisation step.

$$\begin{array}{ll} abs_action' == & \{EntityFullAccess, EntityRead, EntityUpdate\}, \\ permission' == & \{"ReadMedrecord", "UpdateDoctor", "UpdateMedrecord"\}, \\ resource' == & \{Doctors, Hospitals, Medrecords, Patients\}, \\ role' == & \{Doctor, Nurse\}, \end{array}$$

Schema ActionAssignment links roles to a tuple made of the name of the permission, the abstract action allowed by the permission and the kind of resource associated to this permission.

ActionAssignment

$action_Assignment : ROLE \leftrightarrow (PERMISSION \times ABS_ACTION \times RESOURCE)$

The permissions of Fig. 5.2 are stored during a Jaza session as:

$$\begin{array}{ll} action_Assignment' == & \{(Doctor, ("UpdateDoctor", EntityFullAccess, Doctors)), \\ & (Doctor, ("UpdateMedrecord", EntityUpdate, Medrecords)), \\ & (Nurse, ("ReadMedrecord", EntityRead, Medrecords))\} \end{array}$$

5.4.2 Role Hierarchy

RBAC allows us to define hierarchical relations between roles. This is captured by schema RoleInherits in Fig. 3.3 of the Chapter 3. The predicates forbid circularity in the role hierarchy, and forbid the use of roles not declared in set role.

$$
\begin{array}{l}
\textit{RoleInherits} \\
\hline
\textit{Sets} \\
\textit{role_Inherits} : \textit{ROLE} \leftrightarrow \textit{ROLE} \\
\hline
\textit{role_Inherits}^{+} \cap \mathrm{id}\, \textit{role} = \varnothing \\
\mathrm{dom}\, \textit{role_Inherits} \subseteq \textit{role} \wedge \mathrm{ran}\, \textit{role_Inherits} \subseteq \textit{role} \\
\hline
\end{array}
$$

Fig. 5.2 features a simple role hierarchy, where role Doctor inherits all permissions of Nurse. This is expressed in Jaza as:

$$\textit{role_Inherits}' == \{(\textit{Doctor}, \textit{Nurse})\},$$

Schema InheritAssignment computes comp_Assignment which is action_Assignment combined with the inherited permissions.

$$
\begin{array}{l}
\textit{InheritAssignment} \\
\hline
\textit{RoleInherits} \\
\textit{ActionAssignment} \\
\textit{comp_Assignment} : \textit{ROLE} \leftrightarrow \\
\qquad\qquad (\textit{PERMISSION} \times \textit{ABS_ACTION} \times \textit{RESOURCE}) \\
\hline
\textit{comp_Assignment} = \{r : \mathrm{dom}\, \textit{action_Assignment};\ x : \textit{role}; \\
\qquad a : \mathrm{ran}\, \textit{action_Assignment} \mid ((x \mapsto r) \in ((\textit{role_Inherits}^{+}) \\
\qquad\quad \cup (\mathrm{id}\, \textit{role}))) \wedge ((r \mapsto a) \in \textit{action_Assignment}) \bullet (x \mapsto a)\} \\
\hline
\end{array}
$$

In our example, permission ReadMedrecord is inherited by doctors from nurses. Relation comp_Assignment is initialized by Jaza as:

$$
\begin{array}{ll}
\textit{comp_Assignment}' == & \{(\textit{Doctor}, (\text{"}\textit{ReadMedrecord}\text{"}, \textit{EntityRead}, \textit{Medrecords})), \\
& (\textit{Doctor}, (\text{"}\textit{UpdateDoctor}\text{"}, \textit{EntityFullAccess}, \textit{Doctors})), \\
& (\textit{Doctor}, (\text{"}\textit{UpdateMedrecord}\text{"}, \textit{EntityUpdate}, \textit{Medrecords})), \\
& (\textit{Nurse}, (\text{"}\textit{ReadMedrecord}\text{"}, \textit{EntityRead}, \textit{Medrecords}))\},
\end{array}
$$

5.4.3 Action Hierarchy

The notion of abstract action is similar to composing some lower actions into one abstract action. Atomic actions are those actions which appear on a class diagram of an operation. Permissions of Fig. 5.2 refer to abstract actions, such as read or update. These must be linked to their concrete counterparts. Our security kernel expresses this link in action_Relation, as well as an action hierarchy (action_Hierarchy), defining abstract actions in terms of other abstract actions (e.g., EntityFullAccess includes EntityUpdate and EntityRead). These relations are expressed in schema ActionsRelation. We first introduce the enumerated type of atomic actions, corresponding to the methods of PATIENT and MEDRECORD in Fig. 5.2.

$$ATM_ACTION ::= MRReadMedrecord1 \mid DRLinkDoctors1 \mid MRChangeContentsP1$$

ActionsRelation

$$\begin{array}{l}
Sets \\
action_Hierarchy : ABS_ACTION \leftrightarrow ABS_ACTION \\
atm_action : \mathbb{F}\, ATM_ACTION \\
action_Relation : ABS_ACTION \leftrightarrow (ATM_ACTION \times RESOURCE) \\
\hline
action_Hierarchy^{+} \cap \mathrm{id}\, abs_action = \varnothing \\
\mathrm{dom}\, action_Hierarchy \subseteq abs_action \land \\
\qquad \mathrm{ran}\, action_Hierarchy \subseteq abs_action \\
\mathrm{dom}\, action_Relation \subseteq abs_action \land \\
\qquad \mathrm{ran}\, action_Relation \subseteq (atm_action \times resource)
\end{array}$$

It must be noted that the correspondence between abstract and concrete actions takes into account the class on which the abstract action is performed. For example, concrete operation MRReadMedrecord1 only makes sense for medical records. These relations are instantiated as follows in our example.

$$\begin{array}{ll}
action_Hierarchy' == & \{(EntityRead, EntityFullAccess), \\
& \quad (EntityUpdate, EntityFullAccess)\}, \\
action_Relation' == & \{(EntityRead, (MRReadMedrecord1, Medrecords)), \\
& \quad (EntityUpdate, (MRChangeContentsP1, Medrecords)), \\
& \quad (EntityUpdate, (DRLinkDoctors1, Doctors))\},
\end{array}$$

abstract_Assignment unfolds the hierarchy of abstract actions in comp_Assignment. Then concrete_Assignment replaces abstract actions by their concrete counterparts for the given kind of resource.

ComputeAssignment

$$\begin{array}{l}
InheritAssignment;\ ActionsRelation \\
abstract_Assignment : ROLE \leftrightarrow \\
\qquad (PERMISSION \times ABS_ACTION \times RESOURCE) \\
concrete_Assignment : ROLE \leftrightarrow \\
\qquad (PERMISSION \times ATM_ACTION \times RESOURCE) \\
\hline
abstract_Assignment = \{r : \mathrm{dom}\, comp_Assignment;\ p : permission; \\
\qquad asup, asub : abs_action;\ rsrc : resource \mid (r \mapsto (p, asup, rsrc)) \\
\qquad\quad \in comp_Assignment \land ((asub \mapsto asup) \in ((action_Inherits^{+}) \\
\qquad\qquad \cup(\mathrm{id}\, abs_action))) \bullet (r \mapsto (p, asub, rsrc))\} \\
concrete_Assignment = \{r : \mathrm{dom}\, comp_Assignment;\ p : permission; \\
\qquad aa : abs_action;\ atm : atm_action;\ rsrc : resource \\
\qquad\quad \mid (r \mapsto (p, aa, rsrc)) \in abstract_Assignment \land \\
\qquad\qquad (aa \mapsto (atm, rsrc)) \in comp_Actions \bullet (r \mapsto (p, atm, rsrc))\}
\end{array}$$

$$
\begin{aligned}
&concrete_Assignment' == \\
&\quad \{(Doctor, (\text{"}ReadMedrecord\text{"}, MRReadMedrecord1, Medrecords)), \\
&\quad\ (Doctor, (\text{"}UpdateDoctor\text{"}, DRLinkDoctors1, Doctors)), \\
&\quad\ (Doctor, (\text{"}UpdateMedrecord\text{"}, MRChangeContentsP1, Medrecords)), \\
&\quad\ (Nurse, (\text{"}ReadMedrecord\text{"}, MRReadMedrecord1, Medrecords))\},
\end{aligned}
$$

For example, this table tells us that nurses may call MRReadMedrecord on class MEDRECORD.

5.4.4 Roles, Users and Sessions

Users of the security kernel are linked to roles through sessions. Schema RoleAssignment introduces a set of users and relation role_Assignment corresponding to Fig. 3.2 lists the roles a user can take. SessionRoles defines sessions and user ids. Type USERID already appeared in the functional model and is used to make a link between users taking a role featured in the security part of the model (e.g., Doctor), and the classes representing these users in the functional model (e.g., DOCTOR). Injective function accessRights links user ids to users. Function session_User links a session to some user, who has activated a set of roles, recorded in session_Role. These roles must correspond to roles allowed to this particular user in role_Assignment. Several predicates, associated to these schemas, check the consistency between these variables. Table 5.1 features several sessions with associated users, roles and ids.

$$[USER, SESSION]$$

$$
\begin{array}{|l}
\hline
\textit{RoleAssignment} \\
Sets \\
user : \mathbb{F}\, USER \\
roles_SSD : ROLE \leftrightarrow ROLE \\
role_Assignment : USER \leftrightarrow ROLE \\
\hline
\mathrm{dom}\, roles_SSD \subseteq role \\
\mathrm{ran}\, roles_SSD \subseteq role \\
\mathrm{dom}\, role_Assignment \subseteq user \\
\mathrm{ran}\, role_Assignment \subseteq role \\
\forall\, u : user \bullet \forall\, i, j : role \mid ((u \mapsto i) \\
\qquad \in role_Assignment) \wedge ((u \mapsto j) \\
\qquad\quad \in role_Assignment) \bullet ((i, j) \notin roles_SSD) \\
\hline
\end{array}
$$

Session	User	Role	User Id
sess1	Alice	Doctor	003
sess2	Bob	Doctor	004
sess3	Jeck	Nurse	007

Table 5.1: Three sessions

SessionRoles

$RoleAssignment$
$uid : \mathbb{F}\, USERID;$
$session : \mathbb{F}\, SESSION$
$accessRights : USERID \rightarrowtail\!\!\!\!\rightarrow USER$
$session_User : SESSION \nrightarrow USER$
$session_Role : ROLE \leftrightarrow SESSION$

$\forall\, r : role \bullet \forall\, s : session \bullet (r, s) \in session_Role$
$\quad \Rightarrow (session_User(s), r) \in role_Assignment$
$\forall\, s : session \bullet \forall\, i, j : role \mid ((i, s) \in session_Role)$
$\quad \wedge ((j, s) \in session_Role) \bullet ((i, j) \notin roles_DSD)$

5.4.5 Putting it All Together

Schema PermissionAssignment computes an entire table of the graphical model given in Fig. 5.2. It constructs a relation between user identity, user, role and the respective permissions, atomic actions, and the resources. This is achieved using the concrete_Assignment relation and linking roles to their users and user ids.

PermissionAssignment

$SessionRoles;\ RoleAssignment;\ ComputeAssignment$
$perm_Assignment : (USERID \times USER \times ROLE) \leftrightarrow$
$\quad (PERMISSION \times ATM_ACTION \times RESOURCE)$

$perm_Assignment = \{uid : \mathrm{dom}\, accessRights;\ u : \mathrm{dom}\, role_Assignment;$
$\quad r : \mathrm{ran}\, role_Assignment;\ b : \mathrm{ran}\, concrete_Assignment \mid$
$\quad (uid, u) \in accessRights \wedge (u, r) \in role_Assignment$
$\quad \wedge (r, b) \in concrete_Assignment \bullet ((uid, u, r) \mapsto b)\}$

In our example, perm_Assignment is initialized as follows:

$$
\begin{aligned}
&perm_Assignment' == \\
&\{((\text{"003"}, \text{"}Alice\text{"}, Doctor), (\text{"}ReadMedrecord\text{"}, MRReadMedrecord1, Medrecords)), \\
&((\text{"003"}, \text{"}Alice\text{"}, Doctor), (\text{"}UpdateDoctor\text{"}, DRLinkDoctors1, Doctors)), \\
&((\text{"003"}, \text{"}Alice\text{"}, Doctor), (\text{"}UpdateMedrecord\text{"}, MRChangeContentsP1, Medrecords)), \\
&((\text{"004"}, \text{"}Bob\text{"}, Doctor), (\text{"}ReadMedrecord\text{"}, MRReadMedrecord1, Medrecords)), \\
&((\text{"004"}, \text{"}Bob\text{"}, Doctor), (\text{"}UpdateDoctor\text{"}, DRLinkDoctors1, Doctors)), \\
&((\text{"004"}, \text{"}Bob\text{"}, Doctor), (\text{"}UpdateMedrecord\text{"}, MRChangeContentsP1, Medrecords)), \\
&((\text{"007"}, \text{"}Jeck\text{"}, Nurse), (\text{"}ReadMedrecord\text{"}, MRReadMedrecord1, Medrecords))\},
\end{aligned}
$$

We can now use this table, and the information about sessions, to specify the basis for secure operations. SecureOperation actually does nothing: it does neither update the state nor computes a result. It simply states preconditions to allow user?, with id uid?, acting in a given role?, during a given session? to perform a given action? on a resource?, as stated by permission?.

$$
\begin{array}{|l}
\hline
\textit{SecureOperation} \\
\Xi SessionRoles;\ \Xi PermissionAssignment \\
session? : SESSION;\ resource? : RESOURCE;\ atm_action? : ATM_ACTION \\
role? : ROLE;\ user? : USER;\ uid? : USERID;\ permission? : PERMISSION \\
\hline
(session?, user?) \in session_User \\
(role?, session?) \in session_Role \\
((uid?, user?, role?), (permission?, atm_action?, resource?)) \in perm_Assignment \\
\hline
\end{array}
$$

SecureOperation will be used in the next section in combination with operations of the functional model.

Another use of tables perm_Assignment and concrete_Assignment is to perform queries on the access control policy. In Chapter 6, we feature six such queries, inspired by SecureMOVA [Basin *et al.* 2009].

Queries are a meaningful way of investigating the UML models properties and understand conceivably subtle consequences of the security policies they define. For example, query EvaluateActionsAgainstRoles returns a table listing all roles allowed to perform a given action, and the corresponding permission.

$$
\begin{array}{|l}
\hline
\textit{EvaluateActionsAgainstRoles} \\
\Xi Sets;\ \Xi ComputeAssignment \\
atm_action? : ATM_ACTION \\
z_roles! : ROLE \leftrightarrow (PERMISSION \times ATM_ACTION \times RESOURCE) \\
\hline
z_roles! = \{r : \mathrm{dom}\ comp_Assignment;\ p : permission;\ rsrc : resource \mid \\
\qquad (r \mapsto (p, atm_action?, rsrc)) \in concrete_Assignment \\
\qquad\quad \bullet (r \mapsto (p, atm_action?, rsrc))\} \\
\hline
\end{array}
$$

This can be evaluated using Jaza. For example, the following query questions about the permissions to call MRChangeContentsP1. The answer tells us that only role doctor is allowed to perform this action on medical records.

$; \ EvaluateActionsAgainstRoles[atm_action? := MRChangeContentsP1]$
$\ldots$
$z_roles! == \{(Doctor, (\text{"}UpdateMedrecord\text{"}, MRChangeContentsP1, Medrecords))\}$

5.5 Linking Functional and Security Models

SecureOperation is meant to be included, as a precondition, in the secured version of the operations of the functional model. For example, let us consider the setter method for contents, named MedrecordChangeContentsandRels. A secured version of this operation includes the schema of the operation and SecureOperation. Schemas PatientHospitalRel and DoctorHospitalRel are also included to get read access to the associations between hospitals, patients and doctors.

$$
\begin{array}{|l}
\hline
\textit{SecureMRChangeContentsP} \\
\hline
SecureOperation \\
MedrecordChangeContentsandRels \\
\Xi PatientHospitalRel;\ \Xi DoctorHospitalRel \\
\hline
atm_action? = MRChangeContentsP1 \land resource? = Medrecords \\
\exists\, hospital : Hospital \mid hospitalOfPatient(patientOfMedrecord(x?)) = hospital \bullet \\
\qquad \exists\, doctor : Doctor \mid accessRights^{\sim}(session_User(session?)) \\
\qquad\quad = doctor.id \bullet doctor \in doctorsOfHospital(hospital) \\
\hline
\end{array}
$$

The first predicate links this operation to the corresponding atomic action and resource in the security model. It can be generated automatically. The other predicate expresses constraint Same hospital as patient: "the medical record may only be updated by a doctor working in the current hospital of the patient". It retrieves hospital, the hospital corresponding to the patient of medical record x?. Then it retrieves the DOCTOR object corresponding to the id of the user of the current session. Finally, it checks that this doctor works for hospital. This constraint, expressed informally in Fig. 5.2 must be added manually by the analyst.

This operation inherits all input parameters of schema SecureOperation. Most of these parameters can be deduced by Jaza once session? has been fixed. Therefore, we define a new version of the schema hiding these parameters.

$$
\begin{aligned}
&SecureMRChangeContentsP2 == SecureMRChangeContentsP \backslash \\
&\qquad (uid?, user?, abs_action?, atm_action?, resource?, permission?, role?)
\end{aligned}
$$

Secure versions of ReadMedicalRecord and LinkDoctors are defined similarly.

Constraint "Same hospital as patient" links information from the security model (the id of the current user) to the state of the functional model (the hospital of the patient). Its evaluation depends on the states of both models and can thus evolve if any of these states evolves. As we will see in the following section, this makes the analyses and validation of the security policy more complex.

5.6 Validating and Animating Secure Operations

Graphical models such as Fig. 5.2 remain rather abstract. Moreover, complex interactions between functional and security models may either forbid one to play a nominal behavior, or allow an attack to succeed. This would reveal that the detailed specifications do not model the user's intent. Animation can help convince the user that the model corresponds to his intent. Our validation of security properties uses the Jaza tool. Jaza can animate a large subset of constructs of the Z language. It uses a combination of rewriting and constraint solving to find a final state and outputs from a given initial state and inputs. If the initial state and inputs don't satisfy the precondition of the operation, the tool returns "No Solutions". The tool can be further queried to find out which constraint could not be satisfied.

In the sequel, we start from the state of Fig. 5.3 and Table 5.1. We first show that normal behaviors are permitted by the security model. We then investigate the attempts of a malicious doctor to corrupt the integrity of a medical record.

5.6.1 Normal Behavior

Our first tests play nominal scenarios. Their success will show that the combination of security and functional models allows normal behaviors to take place.

Scenario I: *A doctor reads a medical record.*

$; \ SecureMRReadMedrecord2$
$Input\ session? = "sess1"$
$Input\ r? = "meddata2"$

This first scenario tests whether a doctor, here Alice using sess1, may read medical record meddata2. This tests the inheritance of permission ReadMedrecord from nurses to doctors. Jaza animation succeeds and gives the following result:

$x! == \{\langle\!|\ contents == "healthy", recordnb == "meddata2"\ |\!\rangle\}$

Scenario II: *A doctor updates the medical record of a patient in the same hospital.* In this scenario, doctor Alice wants to update some medical record. Since Alice belongs to the same hospital as the patient, this modification is allowed.

$; \ SecureMRChangeContentsP2$
$Input\ x? = \langle\!|\ contents == "healthy", recordnb == "meddata2"\ |\!\rangle$
$Input\ newcontents? = "severe"$
$Input\ session? = "sess1"$

The output shows that the medical record's contents have changed to *"severe"*.

$$Medrecord' == \{\langle\!|\ contents == "severe", recordnb == "meddata2"\ |\!\rangle, \langle\!|\ contents == "sick", recordnb == "meddata1"\ |\!\rangle\}$$

These two examples show that the security kernel does not block licit operations. They can be shown to stakeholders of the information system to validate that the right behavior was captured.

5.6.2 Analyzing a Malicious Behavior

Security analysis must also evaluate the system's ability to block unauthorized behavior. Here, let us consider a malicious doctor, Bob, who tries to corrupt the integrity of medical record med data1, calling operation MRChangeContentsP1.

As we have seen in Sect. 5.4.5, a query tells us that only doctors are allowed to perform this operation. Still, animations go beyond the results of queries presented in Sect. 5.4.5, because queries don't take into account constraints such as Same hospital as patient which may restrict the access to some operations. We will thus try a scenario where Bob attempts to modify medical record med data1.

Scenario III.A: *A doctor attempts to update the medical record of a patient of another hospital.*

$; \ SecureMRChangeContentsP2$
$Input\ x? = \langle\!|\ contents == "sick", recordnb == "meddata1"\ |\!\rangle$
$Input\ newcontents? = "cured"$
$Input\ session? = "sess2"$

Hopefully, Jaza answers that this execution is not allowed by the model.

No Solutions

A closer look at the constraints tells us that Bob's hospital is not the same as the one of the patient. The query tool told us that only doctors are allowed to change the contents of a medical record. But Jaza animation also confirmed that a constraint requires the doctor to work in the same hospital as the patient. Since Bob does not work in the same hospital, there are two ways for him to change the outcome of this constraint. Either he moves the patient to his hospital, or he joins the hospital of the patient. Let us study the latter solution, and query the model about which roles are allowed to change the affiliation of a doctor.

$; \ EvaluateActionsAgainstRoles[atm_action? := DRLinkDoctors1]$
$\ldots$
$z_roles! == \{(Doctor, ("UpdateDoctor", DRLinkDoctors1, Doctors))\}$

The query tells us that doctors are allowed to call this operation. Let us try it!

Scenario III.B: *The doctor first attempts to change his hospital association using one of the class methods and he succeeds in his attempt.*

$; \ SecureDRLinkDoctors2$
$Input\ session? = "sess2"$
$Input\ hospital? = \langle\!|\ name == "RedCross"\ |\!\rangle$
$Input\ doctor? = \langle\!|\ id == "004", name == "Bob"\ |\!\rangle$

The output tells us that Bob is now working for both hospitals.

$$
\begin{aligned}
&doctorsOfHospital' == \\
&\quad \{(\langle\!|\ name == "BlueCare"\ |\!\rangle, \ \{\langle\!|\ id == "003", name == "Alice"\ |\!\rangle, \\
&\qquad\qquad \langle\!|\ id == "004", name == "Bob"\ |\!\rangle\}), \\
&\quad (\langle\!|\ name == "RedCross"\ |\!\rangle, \ \{\langle\!|\ id == "003", name == "Alice"\ |\!\rangle, \\
&\qquad\qquad \langle\!|\ id == "004", name == "Bob"\ |\!\rangle\})\}
\end{aligned}
$$

Scenario III.C: *The doctor makes the malicious changes to the medical record*

; $SecureMRChangeContentsP2$
$Input\ x? = \langle\!|\ contents == "sick", recordnb == "meddata1"\ |\!\rangle$
$Input\ newcontents? = "cured"$
$Input\ session? = "sess2"$

Bob did succeed and compromised the integrity of the medical record.

$$Medrecord' == \{\langle\!|\ contents == "cured", recordnb == "meddata1"\ |\!\rangle, \langle\!|\ contents == "severe", recordnb == "meddata2"\ |\!\rangle\},$$

It means that the current access control rules allow any doctor to join the hospital of any patient. Constraint "Same hospital as patient" is thus useless!

Our approach supports three kinds of validation activities: (a) answering standard queries about the access rules (leaving out the constraints), (b) checking that a given operation may be performed by a given user in a given state, (c) sequencing several operations for given users from a given state. Our scenarios show that the three kinds of activities are useful. State of the art tools such as SecureMOVA or OCL/USE only allow (a) and (b), which are mainly of static nature. Our tool covers (c), adding a dynamic character to validation activities and allowing exploring attack scenarios. Constructing a sufficiently complete set of scenarios is essential to perform a suitable validation. This construction is outside the scope of the thesis that focuses on making such scenarios animatable.

5.7 Summary

We have presented an approach to validate security design models using Z assertions. Our proposal goes through three steps: (a) automated generation of functional specifications using RoZ, (b) the use of a generic security kernel, instantiated by the security model, and specified in Z, and (c) the link between the kernel and the operations of the functional model. Animation of the specifications makes it possible to check that normal behaviors are authorized by the security model and to analyze potential attacks. This is based on the evaluation of standard queries about the security policy and the animation of user-defined scenarios. Using Jaza brings a dynamic dimension to these analyses which are not covered by state of the art tools such as SecureMOVA and USE. In next chapter, we will thoroughly explain the differences between our toolset when compared to SecureMOVA and USE.

Chapter 6

A Z-based Toolset For the Validation of Security Policies

Contents

6.1 Overview **86**
6.2 State of the Art Tools **87**
6.2.1 RBAC and SecureUML 87
6.2.2 USE for the Validation of Security Policies 87
6.2.3 SecureMOVA 88
6.3 The Need for Dynamic Analyses **89**
6.4 Applying Toolset to Meeting SCHEDULER Example **90**
6.4.1 Input Models 90
6.4.2 Diagrams for the Security Model 91
6.4.3 Linking both Formal Models 94
6.5 Animation of the Specification **95**
6.5.1 Queries on the Security Model 96
6.5.2 Dynamic Analyses : Nominal Behaviors 97
6.5.3 Further dynamic analyses 100
6.5.4 Studying an Attack Scenario 100
6.6 Conclusion **102**

In this chapter, we consider the security policy for the meeting SCHEDULER. This case study also includes constraints for the separation of duty, as well as contextual constraints. Contextual constraints use information about the state of the functional model of the application to grant permissions to users. These constraints add flexibility to the security policy, but make its validation more difficult. We first review two tools, USE and SecureMOVA, which can be used to analyze and validate a security policy. These tools focus on analyses of static aspects of the secured system. We use the tool presented in Chapter 5, based on the Z formal language, which uses animation of the specification to validate the static as well as dynamic aspects of the security policy, taking into account possible evolutions of the state of the functional model. Most of the contents of this chapter are reproduced from our paper [Ledru *et al.* 2011].

6.1 Overview

Constraints can be associated to access control models. They allow to express Separation of Duty properties [Clark & Wilson 1987], and other properties on roles (e.g. precedence, see Sect. 6.2.2). Constraints may also link permissions to contextual information, such as the current state of the information system. This is one of the interesting features of SecureUML which groups UML diagrams of the application with security information describing the access control rules. In the remainder, we will refer to the UML diagrams of the application as the *functional model*. The term *security model* will refer to the access control model. Constraints give flexibility to describe security policies, but result in complex descriptions which need tool support for their verification and validation. Verification checks that the description is consistent. In particular, it must check that constraints are not contradictory, which would result in unsatisfiable policies. Validation checks that the policy corresponds to the user's requirements. Our work focuses on validation.

With such complex models, validation can become a difficult task. The separation between the functional model and the security model is an interesting solution based on separation of concerns. However, existing works [Jürjens 2005], [Sohr *et al.* 2008] are mainly interested in the security part. They propose techniques to verify the consistency of an access control policy without taking into account the impact of the functional part. Although it is definitely useful to analyze both models in isolation, interactions between these models must also be taken into account. Such interactions result from the fact that constraints expressed in the security model also refer to information of the functional model. Hence, evolutions of the functional state influence the security behavior. Conversely, security constraints can impact the functional behavior. For example, it is important to consider both security and functional models in order to check liveness properties on the information system. Indeed, it can be the case that security constraints are too strong and block the system. Only a few tools have been proposed to support validation of RBAC models. They focus on static analysis of the model. In this chapter, we propose a toolset which supports both static and dynamic analyses, allowing to study nominal and malicious behaviors of the secure system.

In Sect. 2.2.1 of Chapter 2, we present the meeting scheduler example which will be reused here. In Sect. 6.2, we review the features of two tools, USE and SecureMOVA, which are representative of the current state of the art. In Sect. 6.3, we discuss the interest of leading dynamic analyses of security policies. Sect. 6.4 discusses the translation of security and functional diagrams into a Z specification. Sect. 6.5 details the dynamic analyses that can be performed on our specification. Finally Sect. 6.6 draws the conclusions of this chapter.

6.2 State of the Art Tools

6.2.1 RBAC and SecureUML

As shown in Chapter 3, SecureUML provides a functional model of the application, and the stereotyped elements define the security model. It includes the concepts of RBAC and the possibility to associate permissions with contextual constraints. These constraints involve elements of both security and functional models and restrict the applicability of the permission to the cases where the constraint is verified. In the meeting SCHEDULER, such a constraint is associated to the permission of system users to modify or cancel a meeting. The constraint restricts this permission to the owner of the meeting. Information about the owner of the meeting will be retrieved from the functional class diagram, while information about the user performing the action is related to the security model.

Contextual constraints give much flexibility to express a security policy, but their validation must take into account both functional and security models. Therefore, they require adequate tools. In the next sections, we briefly review two tools which support the validation of role-based security policies with constraints. In both cases, the constraints are written in OCL, a language based on first order logic predicates over the constructs of an UML class diagram.

6.2.2 USE for the Validation of Security Policies

The USE tool [Gogolla *et al.* 2007] allows to evaluate OCL constraints on a given object diagram. These constraints are usually invariants associated to the classes of the diagram, but can also stand for pre- or post-conditions if the object diagram represents the initial or final state of some operation. The tool also allows programming a random generator for object diagrams, and to program sequences of object diagrams.

Sohr *et* al [Sohr *et al.* 2008] have adapted this tool for the analysis of security policies. Their work focuses on the security model, i.e., users, roles, sessions and permissions, constrained by OCL assertions. This allows to express properties such as:

- Cardinality: a given role has at most n users.
- Precedence: u may be assigned to role r_2 only if u is already member of r_1.
- Separation of Duty: roles r_3 and r_4 are conflicting.
- Separation of Duty for Colluding Users, e.g. two brothers may not take conflicting roles.
- Context-dependent permissions, e.g. a meeting may only be modified by its owner.

The last two properties cannot be expressed on a pure security model. It must be augmented with functional information, e.g. some attribute ownedMeetings should be added to the users. Another possibility is to explicitly include this information in the constraints, e.g. in [Sohr *et al.* 2008] all sets of colluding users are listed as OCL rules. Both cases correspond to extensions of the RBAC + constraint model which do not really scale up. Such information definitely belongs to the functional model.

Sohr et al [Sohr *et al.* 2008] report on two kinds of validation activities. An object diagram can be given to the tool, and the tool will check which constraints are violated. The object diagram can be user-defined, randomly generated, or member of a programmed sequence. This allows to detect unsatisfiable constraints, i.e. constraints which are always false. They have also developed a tool named authorization editor, which implements the administrative, system and review functions of the RBAC standard. The tool is connected to the API of USE so that the constraints of the security policy are checked after each operation. It detects erroneous dynamic behaviors of the security policy. For example, if two roles are constrained both by a precedence and a conflict relations, it is impossible to find a sequence of RBAC administrative and system operations which leads to create the second role.

6.2.3 SecureMOVA

SecureMOVA allows to create a functional diagram, i.e., a class diagram, and to relate it to permission rules. Constraints can be attached to permissions and these constraints may refer to the elements of the functional diagram.

SecureMOVA allows to evaluate queries about the security policy. The tool provides an extensive set of queries over a given model, possibly associated with a given initial state. In [Basin *et al.* 2009], Basin *et* al. list the queries that are supported by the tool. A first set of queries explores the relations between roles and actions.

- Given a role, what are the atomic actions that a user in this role can perform?
- Given an atomic action, which roles can perform this action?
- Given a role and an atomic action, under which circumstances can a user in this role perform this action?

Other queries ask more general questions to analyze the security policy. They help identify redundant roles or permissions.

- Are there two roles with the same set of atomic actions?
- Given an atomic action, which roles allow the least set of actions, including the atomic action?
- Do two permissions overlap?

– Are there atomic actions that every role, except the default role, may perform?

With SecureMOVA it is also possible to ask questions about a current state, i.e., a given object diagram. Such queries return the actions authorized for a given role, or to a given user in the current context.

– Given a functional and a security state, can a given user in a given role perform a given action on a given resource?

– Given a user and a state, what are all actions that this user can perform?

– Given a state, which users may perform a given action on a given resource?

– Given a state, which role should take a given user to perform a given action on a given resource?

This extensive set of supported queries is of great help to analyze and validate a security policy. In particular, the last set of queries, which involve a given functional state, can be very useful studying the impact of contextual constraints. Nevertheless, all reported examples [Basin *et al.* 2009] are of static nature, i.e., they do not allow to sequence actions (either administrative or functional) and check that a given sequence is permitted by the combination of the security and functional models.

6.3 The Need for Dynamic Analyses

In the sequel, we emphasize to use animation techniques to further validate security policies. Animation allows to play sequences of actions from a given state. USE and SecureMOVA only report whether the first action of the sequence can be executed from the given state. Animation of sequences of actions is useful to investigate two kinds of behaviors: nominal behaviors, corresponding to the requirements of the system, and malicious behaviors, corresponding to attacks against the secure system.

In both cases, the corresponding behavior may involve several steps, and it is not sufficient to investigate whether a given action can be performed in a given state. It is also necessary to check that the given state can be reached from the initial state, and when sequences of actions are considered, to compute the resulting state and check that the next action can be performed from this resulting state. Animation tools allow to perform a sequence of actions, starting from an initial state and to compute all intermediate states.

Such dynamic analyses require the availability of executable models. Security policies based on RBAC can easily be made executable, as demonstrated by Sohr in his authorization editor [Sohr *et al.* 2008]. Executability of the functional model can be achieved in two ways: either by providing an implementation of the model which can interface with the contextual constraints of the security model, or by providing an executable model. Providing an implementation makes sense in a context where

the functional system is designed first, without considering security aspects, and where a security policy must be designed later for this application. It also makes sense during a maintenance phase where a given implemented security policy must evolve. Some prototypes of RBAC can be coupled with an existing implementation. For example, the MotOrBAC tool provides an API between its security engine and the application [Autrel *et al.* 2008].

Instead of working at the implementation level, our approach favors early validation at the abstract level of a PIM (Platform-Independent Model). The other way is to get an executable functional model. In the case of USE or SecureMOVA, the model is expressed as a class diagram combined with OCL predicates. In order to turn UML methods into executable ones, one needs to provide an implementation of the methods. Actually, USE allows defining a body for each method using an imperative language based on OCL. It seems that this feature was not explored in [Sohr *et al.* 2008] and might be interesting to investigate. Another way is to animate the methods based on their pre- and post-conditions. We do not know of tools which support this approach for OCL, but they exist for formal languages such as Z [ISO 2002], B [Abrial 1996], or Alloy [Jackson 2006]. In [Toahchoodee *et al.* 2009], functional and security models are merged into a single UML model which is translated into Alloy. Alloy can then be used to find a state which breaks a given property. The properties described in [Toahchoodee *et al.* 2009] are mainly of static nature, i.e. they focus on the search for a state which breaks a property, and don't search for sequences of actions leading to such a state. Nevertheless, Alloy can take into account the behavior of the actions of the model, and we believe it has the potential to perform such dynamic analyses.

6.4 Applying Toolset to Meeting SCHEDULER Example

We propose to translate the functional and security models into a Z specification, and then to use the Jaza animator to analyze this specification, using animation and queries. Several attempts have already specified RBAC in Z [Ferraiolo *et al.* 2001], but these were not aimed to be the input of an animator.

6.4.1 Input Models

Our toolset takes as input: (a) a class diagram of the functional application, possibly annotated in Z, and (b) several security diagrams, including diagrams stating the permissions, and a diagram assigning users to roles. Security diagrams are completed by a description of an action hierarchy linking abstract actions to concrete ones. From these inputs, our toolset computes a Z specification of the system which can be animated with Jaza (see sect. 6.5).

The functional model has been described in Chapter 2 and Chapter 3. Section 3.5.1 of Chapter 3 sketches the diagram whereas in Section 2.2 of the Chapter 2

RoZ translation has been provided which will be used as the input of functional model in this chapter.

6.4.2 Diagrams for the Security Model

6.4.2.1 Permissions

The security model involves several diagrams. The main diagram (Fig. 3.8) expresses the permissions related to each role. In accordance with the use cases of Fig. 2.2 of Chapter 2, users may only access meetings. A first permission, UserMeeting, allows them to create and read objects of the class meeting. A second permission, OwnerMeeting, details the rights to update an existing meeting, i.e., to modify it or to delete it. This permission is associated to a constraint, written in the Z language, which states that the user must have the same name as the owner of the meeting.

Similar permissions are expressed for Supervisor and SystemAdministrator. Permission SupervisorCancel grants to supervisors the right to perform operations cancel and notify on any meeting. UserManagement grants to administrators full access to the class Person, and ReadMeeting grants them the right to read class Meeting.

It must be noted that these permissions refer to abstract operations (e.g. Read or Fullaccess) and that a link must be established between these abstract operations and their concrete counterparts. This will be explained in Sect. 6.4.2.3.

6.4.2.2 Roles and Users

An additional diagram (Fig. 6.1) declares the roles of the application, and links them to users. In this diagram, the roles correspond to the actors of the use case diagram: SystemUser, Supervisor, SystemAdministrator and Director. Four users are declared and assigned to these roles through UA (User Assignment) links. These user assignments list the roles that a user can take in a session. Yet, the user may choose to perform the session using a subset of his possible roles. The diagram also declares some separation of duty constraints between roles. Fig. 6.1 features one static separation of duty (SSD) between Supervisor and Administrator, and one dynamic separation of duty (DSD) between Director and SystemUser. It can be visually checked that the SSD constraint is respected by the user assignments. The DSD constraint, which will be enforced during a session, may only be violated by Mark who may use both roles of the DSD.

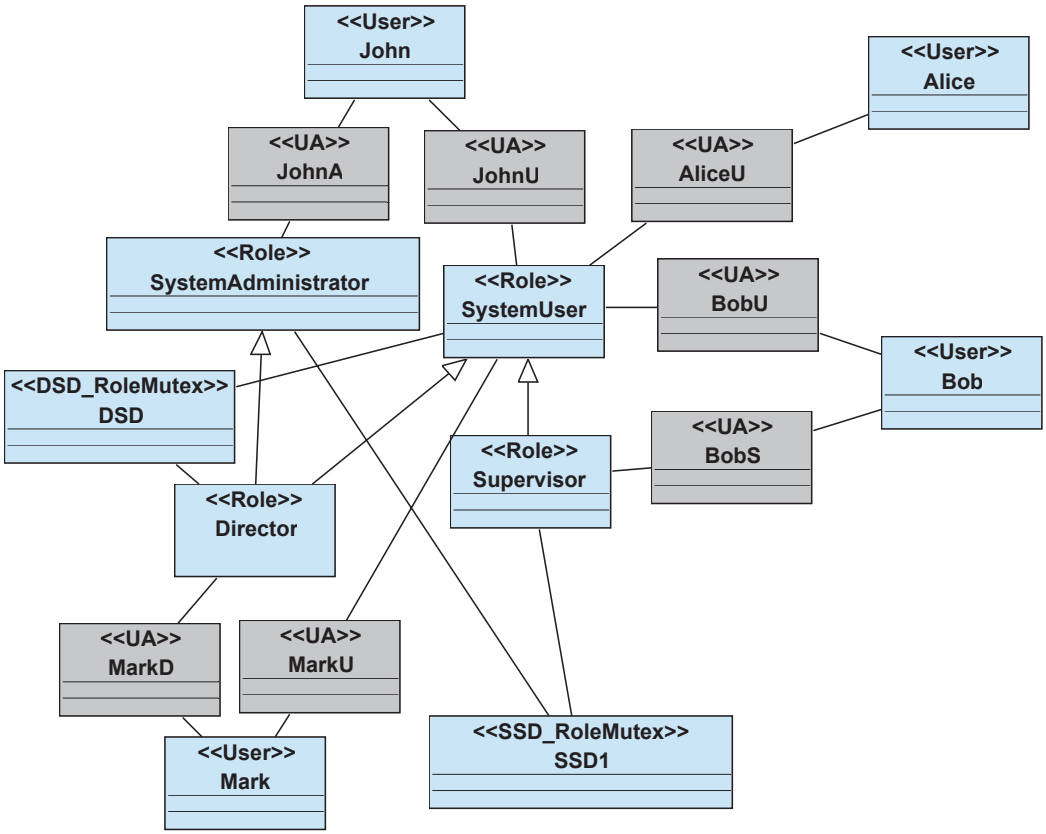

Figure 6.1: Users, roles and separation of duty for the meeting scheduler

$$
\begin{aligned}
&action_Relation = \\
&\quad \{(EntityDelete \mapsto (Cancel1, Meetings)), \\
&\quad\ (EntityDelete \mapsto (RemovePerson1, Persons)), \\
&\quad\ (EntityRead \mapsto (Notify1, Meetings)), \\
&\quad\ (EntityCreate \mapsto (AddMeeting1, Meetings)), \\
&\quad\ (EntityCreate \mapsto (CreateMeeting1, Meetings)), \\
&\quad\ (EntityCreate \mapsto (AddPerson1, Persons)), \\
&\quad\ (EntityUpdate \mapsto (ChangeStart1, Meetings)), \\
&\quad\ (EntityUpdate \mapsto (ChangeDuration1, Meetings)), \\
&\quad\ (EntityUpdate \mapsto (ChangeName1, Persons)), \\
&\quad\ (AssocEndUpdate \mapsto (Linkowner1, Persons)), \\
&\quad\ (AssocEndUpdate \mapsto (Linkparticipants1, Persons)), \\
&\quad\ (AssocEndUpdate \mapsto (LinkmeetingsOfOwner1, Meetings)), \\
&\quad\ (AssocEndUpdate \mapsto (LinkmeetingsOfParticipant1, Meetings)), \\
&\quad\ (NotifyExecute \mapsto (Notify1, Meetings)), \\
&\quad\ (CancelExecute \mapsto (Cancel1, Meetings)), \}
\end{aligned}
$$

6.4.2.3 Action Hierarchy

As mentioned earlier, the permissions of Fig. 3.8 refer to abstract actions. A link must be established between these and the actual operations defined in the classes. Currently, our toolset does not provide a graphical notation expressing this link. It must be defined directly using the Z syntax. The following table, *action_Relation* expresses how abstract actions are instantiated in each class. For example, action EntityDelete corresponds to Cancel in class Meeting and to RemovePerson in class Person. To avoid name conflicts in the Z specification, operation names are suffixed with "1" and class names with "s". It must be noted that the previous table does not explain what FullAccess stands for. This is because FullAccess corresponds to several abstract operations. This is detailed in *action_Inherits*. The table also defines AssocEndUpdate as a special case of EntityUpdate.

$$\begin{aligned} action_Inherits = \ & \{(EntityRead \mapsto EntityFullAccess), \\ & (EntityUpdate \mapsto EntityFullAccess), \\ & (EntityCreate \mapsto EntityFullAccess), \\ & (EntityDelete \mapsto EntityFullAccess), \\ & (AssocEndUpdate \mapsto EntityUpdate)\} \end{aligned}$$

6.4.2.4 Z Translation of the Security Model

The security diagrams are prepared with the TopCased tool[1]. A meta-model and a UML profile have been defined to support the edition of these models. The graphical security models of figures 3.8 and 6.1 are translated into Z using Acceleo[2], a MDA based code generator. The original RoZ [Dupuy *et al.* 2000] was designed for the Rational Rose tool[3]; a new version is currently developed for TopCased, which integrates both security and functional models into the same environment.

The Z specification of the security model is based on the specification of a Z security kernel already presented in Section 5.4 of Chapter 5, independent of a specific application, which specifies the main RBAC data structures (user assignment to roles, role hierarchy, definition of permissions, action hierarchy, session management, Static and Dynamic separation of duty) and computes a table, named perm_Assignment which links user ids, users, roles, permissions, actions and resources.

The translation of the security diagrams and the action hierarchy of Sect. 6.4.2.3 are used to instantiate these data structures, and the associated enumerated types. Using the Jaza animator, we can compute perm_Assignment for our example. Fig. 6.2 gives a subset of this table. For example, the first line tells us that Alice, whose user id is 001, acting as System User, may cancel a meeting due to permission OwnerMeeting. It also tells us that Bob, acting as a supervisor, has two ways to cancel a meeting, using either permission OwnerMeeting or permission Supervisor-

[1] http://www.topcased.org/
[2] http://www.acceleo.org/pages/home/en
[3] http://www.ibm.com/software/rational/

$$
\begin{array}{l}
perm_Assignment == \\
\quad \{ \quad ((\text{"}001\text{"}, \text{"}Alice\text{"}, SystemUser), (\text{"}OwnerMeeting\text{"}, Cancel1, Meetings)), \\
\qquad ((\text{"}001\text{"}, \text{"}Alice\text{"}, SystemUser), (\text{"}OwnerMeeting\text{"}, ChangeDuration1, Meetings)), \\
\qquad ((\text{"}001\text{"}, \text{"}Alice\text{"}, SystemUser), (\text{"}OwnerMeeting\text{"}, ChangeStart1, Meetings)), \\
\qquad ((\text{"}001\text{"}, \text{"}Alice\text{"}, SystemUser), (\text{"}UserMeeting\text{"}, CreateMeeting1, Meetings)), \\
\qquad ((\text{"}001\text{"}, \text{"}Alice\text{"}, SystemUser), (\text{"}UserMeeting\text{"}, Notify1, Meetings)), \\
\ldots \\
\qquad ((\text{"}002\text{"}, \text{"}Bob\text{"}, Supervisor), (\text{"}OwnerMeeting\text{"}, Cancel1, Meetings)), \\
\qquad ((\text{"}002\text{"}, \text{"}Bob\text{"}, Supervisor), (\text{"}SupervisorCancel\text{"}, Cancel1, Meetings)), \\
\qquad ((\text{"}002\text{"}, \text{"}Bob\text{"}, Supervisor), (\text{"}UserMeeting\text{"}, Notify1, Meetings)), \\
\ldots \\
\qquad ((\text{"}003\text{"}, \text{"}John\text{"}, SystemAdministrator), (\text{"}UserManagement\text{"}, AddPerson1, Persons)), \\
\qquad ((\text{"}003\text{"}, \text{"}John\text{"}, SystemAdministrator), (\text{"}UserManagement\text{"}, Linkowner1, Persons)), \\
\ldots
\end{array}
$$

Figure 6.2: A subset of the *perm_Assignment* table

Cancel. It must be noted that this table does not refer to contextual constraints. Its information is thus partial.

The security kernel defines a generic operation, named SecureOperation, which takes as arguments a user, its user id, a role, a session, a permission, an atomic action and a resource and checks that (a) the user is logged in the session with the given role, and (b) that table perm_Assignment authorizes this action for the user in the given role. This definition of SecureOperation is actually a precondition that must be satisfied for the action to take place.

$$
\begin{array}{|l}
\hline
\textit{SecureOperation} \\
\Xi SessionRoles;\ \Xi PermissionAssignment \\
session? : SESSION;\ resource? : RESOURCE;\ atm_action? : ATM_ACTION \\
role? : ROLE;\ user? : USER;\ uid? : USERID;\ permission? : PERMISSION \\
\hline
(session?, user?) \in session_User \\
(role?, session?) \in session_Role \\
((uid?, user?, role?), (permission?, atm_action?, resource?)) \in perm_Assignment \\
\hline
\end{array}
$$

6.4.3 Linking both Formal Models

The last step in the preparation of the Z specification links the Z specifications of both models. First, one must relate the types appearing in both models. Here, the constraint on OwnerMeeting compares the name of the owner to the user performing the cancel operation. This requires that name and user have compatible types. In our example, this is done by redefining type USER as a STRING.

$$USER == STRING$$

At this point, secure versions of the functional operations can be defined. For example, the secure version of meetingcancel includes SecureOperation and meetingcancel (given in Sect. 2.2.6.3). What the operation actually does is completely defined in the functional operation (i.e., meetingcancel). So the secure operation simply adds several checks to allow the operation to take place. These checks take the form of additional preconditions. These require the atomic action to be Cancel1 and the resource to be Meetings. They also require that input parameter meeting? corresponds to an existing meeting, which allows to retrieve its owner. The last condition includes the contextual constraint (user is owner). Since this constraint only applies for OwnerMeeting and not for SupervisorCancel, it only applies if the role is not Supervisor.

$$
\begin{array}{|l}
\hline
\textit{Securemeetingcancel} \\
\textit{SecureOperation} \\
\textit{meetingcancel} \\
\hline
\textit{atm_action?} = \textit{Cancel1} \\
\textit{resource?} = \textit{Meetings} \\
\textit{meeting?} \in \textit{Meeting} \\
\textit{role?} \neq \textit{Supervisor} \Rightarrow (\textit{user?} = (\textit{owner}(\textit{meeting?})).\textit{name}) \\
\hline
\end{array}
$$

Securemeetingcancel has a large number of input parameters. Many of these parameters can be deduced from a subset of the input parameters (here session? and meeting?) and the preconditions of the operation. Operation Securemeetingcancel2 actually hides the useless parameters. In Z, the hide operation ($\backslash$) existentially quantifies the hidden variables. This means that the Z animator will have to find a value for each of the hidden parameters.

$$
\begin{aligned}
&\textit{Securemeetingcancel2} == \textit{Securemeetingcancel} \setminus (\textit{userid?}, \\
&\qquad \textit{user?}, \textit{atm_action?}, \textit{resource?}, \textit{permission?}, \textit{role?})
\end{aligned}
$$

Currently, secure operations are defined manually. Still this definition is completely systematic and a significant part can be automated, using additional Acceleo transformations of the diagram of Fig. 3.8.

6.5 Animation of the Specification

Based on the resulting Z specification, we can use the Jaza animator to perform static (queries) and dynamic analyses (animations) of the security policy.

Jaza can execute an operation whose input parameters are fully instantiated. It checks the preconditions, computes the resulting state and checks that the resulting state is in accordance with all postconditions of the operation and with the state invariants. The user may also omit some input parameters, using the hiding operator. In that case, Jaza searches for values which will satisfy the pre-conditions of the operation and chooses one of these. This requires the search space to be finite,

and small enough. In the sequel we will exploit both features of Jaza to analyze the Z specification.

6.5.1 Queries on the Security Model

We start our analysis by asking some queries, inspired by the ones of SecureMOVA [Basin *et al.* 2009](see Sect. 6.2.3). These queries are mainly based on the perm_Assignment table (Fig. 6.2).

- What are the atomic actions associated to a given role?
- Which roles can perform a given atomic action?

For each of these queries, a corresponding Z operation has been defined. Since the queries don't depend on the application, the Z operations are also reusable. For example, let us query which roles may perform the cancel operation using schema EvaluateActionsAgainstRole which was presented in Section 5.4.5. Jaza answers that three roles can perform this action and reports on the associated permissions. A closer look at the diagrams reveals that one of these permissions is associated to a constraint.

$$; \ EvaluateActionsAgainstRoles[atm_action? := Cancel1]$$

$$\begin{array}{l} z_roleAction! == \\ \{ \ (Director, (\text{"}OwnerMeeting\text{"}, Cancel1, Meetings)), \\ \quad (Supervisor, (\text{"}OwnerMeeting\text{"}, Cancel1, Meetings)), \\ \quad (Supervisor, (\text{"}SupervisorCancel\text{"}, Cancel1, Meetings)), \\ \quad (SystemUser, (\text{"}OwnerMeeting\text{"}, Cancel1, Meetings))\} \end{array}$$

A second series of queries consider the whole set of rules. They help identify generic flaws in the security policy.

- Are there duplicate roles, i.e., two roles with the same set of atomic actions?
- Do two permissions overlap?
- Is there an atomic action that every role may perform?
- Is there an atomic action that nobody may perform?

For example, the following query reports that Supervisor and SystemUser are duplicate roles. It means that they have the same privileges in table perm_Assignment. Still a closer look at the diagrams shows that a contextual constraint restricts the rights of SystemUser, which justifies the existence of both roles.

$$; \ FindDuplicateRoles$$

$$z_role1! == Supervisor, z_role2! == SystemUser$$

The schema FindDuplicateRoles will be presented in Section 7.1.5 of Chapter 7. The following query looks for operations that are always blocked by the security policy. It reveals that RemoveMeeting is not accessible. Actually, RemoveMeeting is meant to be used as a part of meetingcancel. So it is normal that no role has access to this operation.

; *AccessNobody*

$z_action! == RemoveMeeting1$

It must be noted that the same queries are supported by SecureMOVA, but that it only answers "yes" or "no". We found it useful to provide witnesses when the answer is positive, because it speeds up the debugging process.

6.5.2 Dynamic Analyses : Nominal Behaviors

The queries of the previous section are of static nature and do not take into account the contextual constraints associated to permissions. So they don't benefit from our integration of the functional and security models. In this section, we will perform dynamic queries, animating sequences of actions which correspond either to nominal behaviors or to possible attacks.

All animations of this section rely on an initial state where some sessions are predefined. Fig. 6.3 gives information about these sessions.

Session	User	Roles
sess1	Alice	SystemUser
sess2	Bob	Supervisor, SystemUser
sess3	John	SystemAdministrator, SystemUser
sess4	Mark	Director

Figure 6.3: Sessions with their users and roles

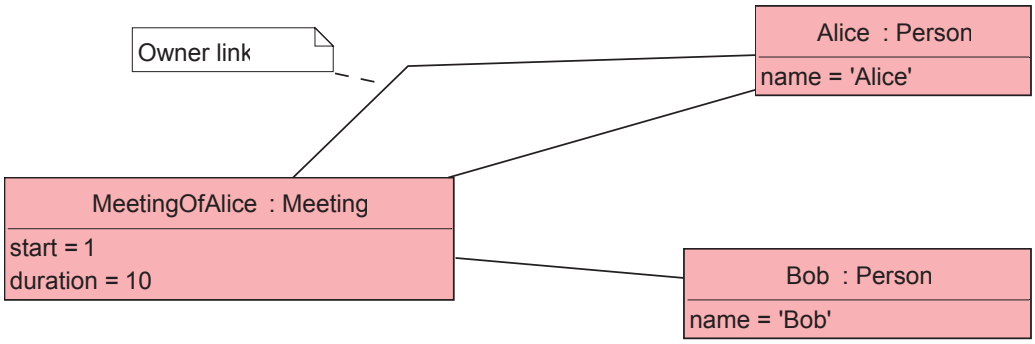

Figure 6.4: Object diagram for the meeting scheduler

First, we explore nominal behaviors. Our first goal is to find a sequence of ac-

tions which will lead us to the functional state depicted in Fig. 6.4. This requires to create two persons, one meeting, and three links. Persons must be created by the system administrator (i.e., John in session 3), then the meeting and its links will be created by Alice (session 1). This corresponds to the following Jaza animation.

$$
\begin{array}{l}
; \ SecurePersonAddPerson2[session? := "sess3", \\
\quad person? := \langle\!| \ name == "Alice" \ |\!\rangle] \\
; \ SecurePersonAddPerson2[session? := "sess3", \\
\quad person? := \langle\!| \ name == "Bob" \ |\!\rangle] \\
; \ SecuremeetingcreateMeeting2[session? := "sess1", \\
\quad meeting? := \langle\!| \ start == 1, duration == 10 \ |\!\rangle, \\
\quad owner? := \langle\!| \ name == "Alice" \ |\!\rangle] \\
; \ SecuremeetingLinkmeetingsOfParticipant2[\ \text{session? := "sess1"}, \\
\quad meeting? := \langle\!| \ start == 1, duration == 10 \ |\!\rangle, \\
\quad person? := \langle\!| \ name == "Bob" \ |\!\rangle]
\end{array}
$$

This animation proceeds with success. Actually it covers a nominal behavior which includes several use cases of Fig. 2.2: create a person, create a meeting, add participants.

We proceed by trying to cancel the meeting. This will validate the contextual constraint. First, we use the session of John to perform this attempt. Since John is neither supervisor nor the owner of the meeting, this attempt should fail. And this is exactly what happens.

$$
\begin{array}{l}
; \ Securemeetingcancel2[session? := "sess3", \\
\quad meeting? := \langle\!| \ start == 1, duration == 10 \ |\!\rangle]
\end{array}
$$

No solutions

We then try the same operation, using the session of Alice, the owner of the meeting. This time, the operation succeeds and the set of meetings is empty after the operation.

$$
\begin{array}{l}
; \ Securemeetingcancel2[session? := "sess1", \\
\quad meeting? := \langle\!| \ start == 1, duration == 10 \ |\!\rangle]
\end{array}
$$

$Meeting' == \{\}, \ldots$

These animations increase our confidence that we expressed the right rule and the right constraint. Another nominal behavior is to delete some person. Let us consider that Alice has left the company and that we must delete object Alice, starting from the state of Fig. 6.4. Only system administrators are allowed to remove a person, so this will be performed by John in session 3.

$$
\begin{array}{l}
; \ SecurePersonRemovePerson2[session? := "sess3", \\
\quad person? := \langle\!| \ name == "Alice" \ |\!\rangle]
\end{array}
$$

No solutions

Jaza reports that the operation failed. Actually, this is due to the fact that deleting Alice leads to have a meeting without owner, which is forbidden by the class diagram (every meeting has one and only one owner). So the functional model requires to first cancel Alice's meeting and then remove Alice. Since John is administrator, he has no right to cancel Alice's meeting. Since Alice has left the company, we need the help of a supervisor, here Bob in session 2. Now the following sequence of operations will succeed.

$$
\begin{array}{l}
; \ Securemeetingcancel2[session? := "sess2", \\
\quad meeting? := \langle\!\langle \ start == 1, duration == 10 \ \rangle\!\rangle] \\
; \ SecurePersonRemovePerson2[session? := "sess3", \\
\quad person? := \langle\!\langle \ name == "Alice" \ \rangle\!\rangle]
\end{array}
$$

This animation convinces us that it was useful to create role Supervisor in our security policy, otherwise, the security rule would make it impossible to remove a user who has left the company. One may wonder whether role Director could be used to cancel the meeting and then remove the person. But the animator reports that the Director, who is neither supervisor nor the meeting owner, may not cancel Alice's meeting. This may suggest to modify the definition of Director and make him inherit from Supervisor (but this will conflict with the SSD constraint).

Other analyses of nominal behaviors can test SSD and DSD constraints. For example, the following animation shows that the SSD constraint works as expected

$$
\begin{array}{l}
; \ AddRole[user? := "Mark", role? := SystemAdministrator] \\
; \ AddRole[user? := "Mark", role? := Supervisor]
\end{array}
$$

No solutions

and the following one gives a similar result for DSD

$$
\begin{array}{l}
; \ NewSession[session? := "sess5", \\
\quad user? := "Mark", role? := Director] \\
; \ AddSessionRole[session? := "sess5", \\
\quad user? := "Mark", role? := SystemUser]
\end{array}
$$

No solutions

The current tool supports an elementary definition of DSD which forbids simultaneous use of conflicting roles in the same session. More elaborate versions of DSD, which consider non-simultaneous use during the same session or during the life-cycle of an object will be studied in future work.

Please note that we will come back to the schemas AddRole, NewSession, and AddSessionRole in Chapter 7 to provide their underlying details.

6.5.3 Further dynamic analyses

In [Basin *et al.* 2009], SecureMOVA is used evaluating queries which depend on a given context. "Given a state, which role should take a given user to perform a given action on a given resource?" For example, which role should take Bob to cancel Alice's meeting?

This result does not only depend on perm_Assignment, but also on the current state of the data. We can ask a similar query in Z, by defining the following operation.

$$RoleNeededForMeetingCancel == (NewSession \setminus (role?)) \fatsemi (Securemeetingcancel2)$$

RoleNeededForMeetingCancel first creates a new session, and then uses this session to cancel the meeting. It hides input parameter role? so that Jaza must find a role which satisfies the preconditions of both operations. When we call this operation, acting as user Bob, it actually leads to a resulting state where the set of meetings is empty. A closer look at the state shows that session 6 was created with Bob as user, and in the role of Supervisor. This answers our question: Bob may cancel Alice's meeting if he logs in as a supervisor.

$$\begin{aligned}
&; RoleNeededForMeetingCancel[session? := "sess6", \\
&\quad user? := "Bob", \\
&\quad meeting? := \langle\!| \, start == 1, duration == 10 \, |\!\rangle] \\
&Meeting' == \{\}, \ldots \\
&session_Role' == \{\ldots, (Supervisor, "sess6")\}, \\
&session_User' == \{\ldots, ("sess6", "Bob")\},
\end{aligned}$$

6.5.4 Studying an Attack Scenario

Integrity of meetings is an important security property we want to enforce on our information system. Let us now focus on user John, who may play the role of SystemAdministrator and SystemUser. For some malicious reason, John wants to cancel the meeting of Alice. Since John may play two different roles, we can ask which role he should use to cancel the meeting (as we did for Bob in the previous section).

$$\begin{aligned}
&; RoleNeededForMeetingCancel[session? := "sess7", \\
&\quad user? := "John", \\
&\quad meeting? := \langle\!| \, start == 1, duration == 10 \, |\!\rangle]
\end{aligned}$$

No solutions

As expected, the system answers that John is not allowed, in any of his roles to perform this action. In Sect. 6.5.1, we already queried which roles allow to perform action Cancel1 (using EvaluateActionsAgainstRoles), and found that it requires roles SystemUser, Supervisor, or Director. John may only use role SystemUser to cancel

the meeting, but a closer look at Fig. 3.8 tells us that permission "OwnerMeeting" requires John to be the owner of the meeting. This explains why he is not allowed to cancel the meeting. This also suggests that John may get this permission if he becomes owner of the meeting. This requires a more elaborate attack where John first becomes owner of the meeting and then cancels it. The functional model provides two methods to change the owner of the meeting (see Fig. 3.8): LinkmeetingsOfOwner in class Meeting and Linkowner in class Person.

Let us check which roles may use these operations:

$$
\begin{array}{l}
; \ EvaluateActionsAgainstRoles[\\
\quad atm_action? := LinkmeetingsOfOwner1] \\
\\
z_roleAction! == \\
\{ \quad (Director, \\
\qquad (\text{"}OwnerMeeting\text{"}, LinkmeetingsOfOwner1, Meetings)), \\
\quad (Supervisor, \\
\qquad (\text{"}OwnerMeeting\text{"}, LinkmeetingsOfOwner1, Meetings)), \\
\quad (SystemUser, \\
\qquad (\text{"}OwnerMeeting\text{"}, LinkmeetingsOfOwner1, Meetings))\}
\end{array}
$$

None of these permissions apply for John, because he may only take the role SystemUser in this list, and in that case, he must be the owner of the meeting. Operation `Linkowner` corresponds to the other end of the association. A similar query may be performed.

$$
\begin{array}{l}
; \ EvaluateActionsAgainstRoles[atm_action? := Linkowner1] \\
\\
z_roleAction! == \\
\{ \quad (Director, (\text{"}UserManagement\text{"}, Linkowner1, Persons)), \\
\quad (SystemAdministrator, \\
\qquad (\text{"}UserManagement\text{"}, Linkowner1, Persons))\}
\end{array}
$$

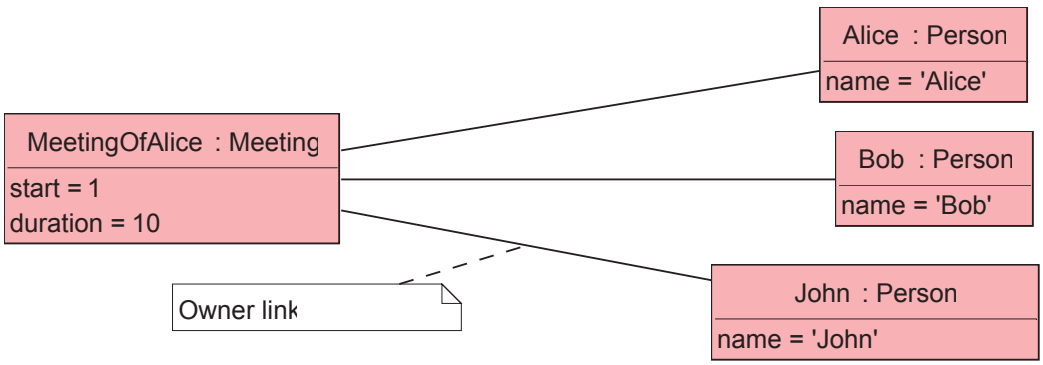

Figure 6.5: Another Object diagram for the meeting scheduler

So John may perform action Linkowner as SystemAdministrator. This action requires to first create an object of class Person corresponding to John. John being system administrator, he may create this object, using session sess3. Note that the constraint owner is a participant was not included in our Z specifications. The current Fig. 6.5 does not fulfill the constraint "owner is a participant" given in Chapter 2.

$$; \ SecurePersonAddPerson2[session? := "sess3", \\ person? := \langle\!| \ name == "John" \ |\!\rangle] \\ ; \ SecurepersonLinkowner2[session? := "sess3", \\ person? := \langle\!| \ name == "John" \ |\!\rangle, \\ meeting? := \langle\!| \ start == 1, duration == 10 \ |\!\rangle]$$

John is now the owner of the meeting, as shown in Fig. 6.5. Being the owner, he may now cancel the meeting.

$$; \ Securemeetingcancel2[session? := "sess3", \\ meeting? := \langle\!| \ start == 1, duration == 10 \ |\!\rangle]$$

$$Meeting' == \{\} \ldots$$

The attack of John has succeeded! This may be considered as a flaw of the security policy. The meeting scheduler example was discussed in several articles, and defined independently of our research team. To the best of our knowledge, this problem was never reported before. We foresee that similar problems will happen in SecureUML descriptions which use contextual constraints.

The problem is that SystemAdministrator has full access to class `Person`, which includes the right to modify association ends. One solution is to add a SSD constraint between SystemAdministrator and SystemUser. Hence, John will still be able to become owner of the meeting, but will not be able to log in as SystemUser in order to delete it.

Currently, finding out attack scenarios relies on the analyst's skills. Regarding contextual constraints, we believe that this study can become quite systematic using our query tools to review how, and by which role, the functional state can be modified to change the outcome of these constraints.

6.6 Conclusion

This chapter has addressed the validation of security policies expressed as RBAC rules with contextual constraints based on a new case study. Such constraints refer to elements of both security and functional models, using the state of the functional model as a context to grant access rights. Separation of concerns suggests treating the functional and security models in isolation. Unfortunately, when constraints establish a link between these models, validation must take both models into account.

Also we presented our formal queries similar to SecureMOVA tool. One feature of SecureMOVA remains difficult to support. SecureMOVA is able to report the text

of the conditions that are associated to a permission, due to the reflexive character of the UML model. Our Z specification does not allow such reflexivity mechanisms, and can only evaluate the condition in a given state. Our toolset includes a large number of the queries supported by USE and SecureMOVA; it can be extended to support most of the remaining ones.

The animation techniques for the validation of models is not new, it has been applied to B, event B, and Z formal models. [Barden *et al.* 1996] have cited several benefits of animating formal models:

- Errors in formal specifications can take two forms, either owing to the mathematics, or to errors in the requirements capture process. Such errors can be revealed using animation techniques.
- Animating specifications ensures more confidence that the requirements have been captured well. This is beneficial for both the specifiers and for customers who experiment with the animation.
- Generally the customers do not know their own requirements. Animation techniques can play a vital role to help them know what they want. In this way, one can also differentiate between the undesired and the desired properties of a system from a user perspective.

More examples of animation include [Aït-Sadoune & Ameur 2008] that offers an animation of event B formal models. Similarly another tool to animate a subset of Z specifications named ZANS approach is proposed in [Jia 1995]. In their approach a proof activity is carried out using animation technique. Not surprisingly, there are several other papers studying the animation of formal specifications adding several benefits to the use of formal methods. For example, to animate a model, it requires us to execute the model which implicitly contributes to the large scale applicability of formal languages. Our shown approach is a sequel of such endeavors aimed at integrating formal notations at design phase of software development.

Chapter 7

A Set of Validation Queries

Contents

7.1 Formal Queries **105**
7.1.1 Authorized Roles for an Atomic Action 105
7.1.2 Available Actions Against Roles 107
7.1.3 Access to Resources 108
7.1.4 Permissions Against Atomic Action and Role 109
7.1.5 Finding Duplicate Roles 109
7.1.6 Atomic Action Accessed by All 110
7.1.7 Atomic Action Access by Nobody 111
7.2 RBAC Supporting System Functions **111**
7.3 Summary **113**

The use of formalized validation queries has been demonstrated in Chapter 5 and Chapter 6. In this chapter, we present the formalization of the queries that contributed to the validation of security design models in previous two chapters. Subsequently, we will explain a subset of session management functions of RBAC that we have used before.

7.1 Formal Queries

7.1.1 Authorized Roles for an Atomic Action

The first query is EvaluateRoleAuthorizedAtomicAction that computes the set of atomic actions against a given role. For example, we want to know the set of atomic actions assigned to a role in a security policy.

$$
\begin{array}{l}
\textit{EvaluateRoleAuthorizedAtomicAction} \\
\hline
\Xi Sets \\
role? : ROLE \\
\Xi ComputeAssignment \\
z_atomicActions! : ROLE \leftrightarrow \\
\qquad (PERMISSION \times ATOMIC_ACTION \times RESOURCE) \\
\hline
role? \in \operatorname{dom} concrete_Assignment \\
z_atomicActions! = \{prm : \operatorname{ran} concrete_Assignment \mid \\
\qquad (role?, prm) \in concrete_Assignment \bullet (role? \mapsto prm)\}
\end{array}
$$

The declaration part of EvaluateRoleAuthorizedAtomicAction includes the state schema Sets and an input variable role? of type ROLE. The set being computed z_atomicActions! is a relation which has a type of cross product of role with associated permissions, atomic actions, and resources. The predicate part of the operation schema checks that the input role (role?) is actually from the domain of relation concrete_Assignment, defined in ComputeAssignment. The relation z_atomicActions! is a cross product of role with permission, atomic action and resource. Thus the output (z_atomicActions!) would be the set of all possible values associated to a particular role (i.e.,role?) within the system specification. Below is an example of executing this schema where against the role Supervisor: Jaza has detailed all the atomic actions authorized to it.

$JAZA >;\ EvaluateActionsAgainstRoles$
$Input\ atm_action? = Supervisor$
$z_atomicActions! == \{(Supervisor,$
$(" OwnerMeeting", Cancel1, Meetings)),$
$(Supervisor,$
$(" OwnerMeeting", ChangeDuration1, Meetings)),$
$(Supervisor, (" OwnerMeeting", ChangeStart1, Meetings)),$
$(Supervisor, (" OwnerMeeting", LinkmeetingsOfOwner1, Meetings)),$
$(Supervisor,$
$(" OwnerMeeting", LinkmeetingsOfParticipant1, Meetings)),$
$(Supervisor,$
$(" SupervisorCancel", Cancel1, Meetings)),$
$(Supervisor, (" SupervisorCancel", Notify1, Meetings)),$
$(Supervisor, (" UserMeeting", AddMeeting1, Meetings)),$
$(Supervisor, (" UserMeeting", CreateMeeting1, Meetings)),$
$(Supervisor, (" UserMeeting", Notify1, Meetings))\}$

Sets

$role : \mathbb{F}\ ROLE$
$user : \mathbb{F}\ USER$
$userid : \mathbb{F}\ USERID$
$session : \mathbb{F}\ SESSION$
$resource : \mathbb{F}\ RESOURCE$
$permission : \mathbb{F}\ PERMISSION$
$atm_action : \mathbb{F}\ ATOMIC_ACTION$
$abs_action : \mathbb{F}\ ABSTRACT_ACTION$

Since the state schema Sets and ComputeAssignment are being repeatedly included in every operation schema of this chapter, we just reproduced them here. Note that the schema Sets has several components. In Chapter 5 it has been decomposed into several state schemas just for the sake of clarity.

ComputeAssignment

$Sets$
$RoleInherits$
$ComputeActions$
$ActionsRelation$
$ActionAssignment$
$comp_Assignment : ROLE \leftrightarrow$
$\quad (PERMISSION \times ABSTRACT_ACTION \times RESOURCE)$
$abstract_Assignment : ROLE \leftrightarrow$
$\quad (PERMISSION \times ABSTRACT_ACTION \times RESOURCE)$
$concrete_Assignment : ROLE \leftrightarrow$
$\quad (PERMISSION \times ATOMIC_ACTION \times RESOURCE)$

$comp_Assignment = \{r : \mathrm{dom}\, action_Assignment;\ x : role;$
$\quad a : \mathrm{ran}\, action_Assignment \mid ((x \mapsto r) \in ((role_Inherits^{+})$
$\quad \cup(\mathrm{id}\, role))) \wedge ((r \mapsto a) \in action_Assignment)$
$\quad \bullet (x \mapsto a)\}$
$abstract_Assignment = \{r : \mathrm{dom}\, comp_Assignment;\ p : permission;$
$\quad asup, asub : abs_action;\ rsrc : resource \mid (r \mapsto (p, asup, rsrc))$
$\quad \in comp_Assignment \wedge ((asub \mapsto asup) \in ((action_Inherits^{+})$
$\quad \cup(\mathrm{id}\, abs_action))) \bullet (r \mapsto (p, asub, rsrc))\}$
$concrete_Assignment = \{r : \mathrm{dom}\, comp_Assignment;\ p : permission;$
$\quad aa : abs_action;\ atm : atm_action;\ rsrc : resource$
$\quad \mid (r \mapsto (p, aa, rsrc)) \in abstract_Assignment \wedge$
$\quad (aa \mapsto (atm, rsrc)) \in comp_Actions \bullet (r \mapsto (p, atm, rsrc))\}$

7.1.2 Available Actions Against Roles

The operation schema EvaluateActionsAgainstRoles works exactly opposite to the operation schema EvaluateRoleAuthorizedAtomicAction and against a given atomic action, it returns the list of all associated roles (along with resources name and permissions) to perform that action.

EvaluateActionsAgainstRoles

$\Xi Sets$
$\Xi ComputeAssignment$
$atm_action? : ATOMIC_ACTION$
$z_roleAction! : ROLE \leftrightarrow$
$\quad (PERMISSION \times ATOMIC_ACTION \times RESOURCE)$

$z_roleAction! = \{r : \mathrm{dom}\, comp_Assignment;\ p : permission;\ rsrc : resource \mid$
$\quad (r \mapsto (p, atm_action?, rsrc)) \in concrete_Assignment \bullet$
$\quad (r \mapsto (p, atm_action?, rsrc))\}$

The input (atm_action?) is of the set type ATOMIC_ACTION. The output z_roleAction! has the same type as given in the previous schema.

The set z_roleAction! retrieves the allowed roles to perform an atomic action. Note that, we also keep the linked permissions and resources with each obtained role since it's pretty useful to analyze them together. Here is an example to this schema query. We provided an atomic action named Cancel1 and it has returned the corresponding information.

$JAZA >;\ EvaluateActionsAgainstRoles$
$Input\ atm_action? = Cancel1$
$z_roleAction! == \{(Director, (\text{"}OwnerMeeting\text{"}, Cancel1, Meetings)),$
$(Director, (\text{"}UserManagement\text{"}, Cancel1, Persons)),$
$(Supervisor, (\text{"}OwnerMeeting\text{"}, Cancel1, Meetings)),$
$(Supervisor,$
$(\text{"}SupervisorCancel\text{"}, Cancel1, Meetings)),$
$(SystemAdministrator,$
$(\text{"}UserManagement\text{"}, Cancel1, Persons)),$
$(SystemUser,$
$(\text{"}OwnerMeeting\text{"}, Cancel1, Meetings))\}$

7.1.3 Access to Resources

It's equally important to know of the resources within the system that can be accessed by some roles. EvaluateResourcesAccess is used for this purpose. Against a given resource? it returns the pairs of atomic actions associated to that particular resource.

$$
\begin{array}{l}
\textit{EvaluateResourcesAccess} \\
\hline
\Xi Sets \\
\Xi ComputeAssignment \\
resource? : RESOURCE \\
resourcesAccess! : ROLE \leftrightarrow \\
\qquad (PERMISSION \times (ATOMIC_ACTION \times RESOURCE)) \\
z_action_resource_set! : \mathbb{F}(ATOMIC_ACTION \times RESOURCE) \\
\hline
resourcesAccess! = \{r : \operatorname{dom} comp_Assignment;\ p : permission; \\
\qquad atm : atm_action \mid (r \mapsto (p, atm, resource?)) \\
\qquad \in concrete_Assignment \bullet (r \mapsto (p, (atm, resource?)))\} \\
z_action_resource_set! = \{x : \operatorname{ran} resourcesAccess! \bullet second(x)\} \\
\hline
\end{array}
$$

This operation also takes an input resource? of the type RESOURCE and computes the related roles and atomic actions of that resource. z_action_resource_set! ensures that only the atomic actions corresponding to the resources are retrieved. This schema is exemplified below where the input is resource Meetings and the result produced by Jaza is followed by it.

$JAZA >;\ EvaluateResourcesAccess$
$Input\ resource? = Meetings$
$z_action_resource_set! == \{(AddMeeting1, Meetings),$
$(Cancel1, Meetings),$
$(ChangeDuration1, Meetings),$
$(ChangeStart1, Meetings),$
$(CreateMeeting1, Meetings),$
$(LinkmeetingsOfOwner1, Meetings),$
$(LinkmeetingsOfParticipant1, Meetings),$
$(Notify1, Meetings)\}$

7.1.4 Permissions Against Atomic Action and Role

The operation schema FindPermissions is intended to query the permissions for both a given atomic action and a role.

$$
\begin{array}{l}
\textit{FindPermissions} \\
\hline
\Xi Sets \\
\Xi ComputeAssignment \\
atm_action? : ATOMIC_ACTION \\
role? : ROLE \\
z_perms! : ROLE \leftrightarrow \\
\qquad\qquad (PERMISSION \times ATOMIC_ACTION \times RESOURCE) \\
\hline
z_perms! = \{p : permission;\ rsrc : resource \mid \\
\qquad\qquad (role? \mapsto (p, atm_action?, rsrc)) \in concrete_Assignment \bullet \\
\qquad\qquad (role? \mapsto (p, atm_action?, rsrc))\} \\
\hline
\end{array}
$$

This schema has two input parameters i.e., atm_action? and role? of the types ATOMIC_ACTION and ROLE, respectively. The predicate computes the set of permissions for the input role and the atomic action. As a result, z_perms! will return the set of all associated permissions against the input values. We need to give as input the atomic action along with role, and it will return the permissions linked to them as follows.

$JAZA >;\ FindPermissions$
$Input\ atm_action? = AddPerson1$
$Inputrole? = SystemAdministrator$
$z_perms! == \{(SystemAdministrator,$
$(\text{"}UserManagement\text{"}, AddPerson1, Persons))\}$

7.1.5 Finding Duplicate Roles

The schema FindDuplicateRoles allows to search for duplicate roles. This query is useful to determine whether two roles have same privileges in a secure system.

$$
\begin{array}{l}
\textit{FindDuplicateRoles} \\
\hline
\Xi Sets \\
\Xi ComputeAssignment \\
z_role1! : ROLE \\
z_role2! : ROLE \\
z_aSet1!, z_aSet2! : \mathbb{F}\, ATOMIC_ACTION \\
\hline
z_role1! \in role \\
z_role2! \in role \\
z_role1! \neq z_role2! \\
z_aSet1! = \{p : permission;\ a : ATOMIC_ACTION;\ rsrc : resource \mid \\
\qquad (z_role1! \mapsto (p, a, rsrc)) \in concrete_Assignment \bullet a\} \\
z_aSet2! = \{p : permission;\ a : ATOMIC_ACTION;\ rsrc : resource \mid \\
\qquad (z_role2! \mapsto (p, a, rsrc)) \in concrete_Assignment \bullet a\} \\
z_aSet1! = z_aSet2! \\
\hline
\end{array}
$$

The predicate of this schema specifies that for any two unique roles, it will iterate over the available actions of the system. It then performs an equality check to determine if the resulted sets for two different roles are the same. If it is the case, Jaza will return those roles which have the same rights in the system. The following query reports that Supervisor and SystemUser are duplicate roles, more precisely they have the permissions to perform the same actions.

; *FindDuplicateRoles*

$z_role1! == Supervisor, z_role2! == SystemUser$

But a closer look reveals that constraints are linked to some of the associated permissions (given in Chapter 6.

7.1.6 Atomic Action Accessed by All

The operation schema AccessAll returns the accessible atomic operations by all the roles of a system.

$$
\begin{array}{l}
\textit{AccessAll} \\
\hline
\Xi Sets \\
\Xi ComputeAssignment \\
z_action! : ATOMIC_ACTION \\
\hline
\forall\, r : role \bullet \exists\, p : permission;\ rsrc : resource \bullet \\
\qquad (r \mapsto (p, z_action!, rsrc)) \in concrete_Assignment \\
\hline
\end{array}
$$

The declaration part includes an output component z_action!. The given predicate returns the atomic action accessible by all the roles in the system.

JAZA >; *AccessAll*
NoSolution

7.1.7 Atomic Action Access by Nobody

The operation schema AccessNobody returns the atomic action which is completely inaccessible by all the roles.

$$
\begin{array}{|l}
\hline
\textit{AccessNobody} \\
\Xi Sets \\
\Xi ComputeAssignment \\
z_action! : ATOMIC_ACTION \\
\hline
\neg\, (\exists\, r : role \bullet \exists\, p : permission;\ rsrc : resource \bullet \\
\qquad\qquad (r \mapsto (p, z_action!, rsrc)) \in concrete_Assignment) \\
\hline
\end{array}
$$

It includes an output variable z_action! as a type of atomic actions. The predicate part checks for the atomic actions accessible by all and applies a negation, which reveals the inaccessible atomic action.

$JAZA >;\ AccessNobody$
$z_action! == RemoveMeeting1$

7.2 RBAC Supporting System Functions

RBAC has numerous session management functions, we only implement a limited number of functions as required. Using these functions we analyze DSD and DSD constraints in our security policies.

$$
\begin{array}{|l}
\hline
\textit{AddSessionRole} \\
\Xi Sets \\
role? : ROLE \\
session? : SESSION \\
\Xi AccessRights \\
\Delta SessionRoles \\
\hline
role? \in role \\
session? \in session \\
accessRights' = accessRights \\
session_User' = session_user \\
session_Role' = session_Role \cup \{role? \mapsto session?\} \\
roles_DSD' = roles_DSD \\
roles_SSD' = roles_SSD \wedge role_Assignment' = role_Assignment \\
\hline
\end{array}
$$

The declaration part includes the schemas Sets and SessionRoles and includes two input parameters session? and role? of the types SESSION, and ROLE, respectively. The first precondition states that role is one of the existing roles and second illustrates that session is also one of the existing sessions. accessRights, session_User,

roles_SSD, and role_Assignment remain unchanged. However, both session_Role' have been updated with the new given values.

AddRole

$\Xi Sets$
$\Delta RoleAssignment$
$user? : USER$
$role? : ROLE$

$user? \in USER$
$role? \in role$
$roles_SSD' = roles_SSD$
$role_Assignment' = role_Assignment \cup \{user? \mapsto role?\}$

The schema AddRole adds a new role to the existing against a specific user. The two inputs are the role? and user? for which the role_Assignment' is updated in the predicate part while roles_SSD remains unchanged. In other words, it means this does not impact static SSD constraint in the specifications.

NewSession

$\Delta Sets$
$\Delta SessionRoles$
$session? : SESSION$
$user? : USER$
$role? : ROLE$

$user? \in user$
$session? \notin session$
$session' = session \cup \{session?\}$
$session_User' = session_User \cup \{session? \mapsto user?\}$
$session_Role' = session_Role \cup \{role? \mapsto session?\}$
$roles_DSD' = roles_DSD$
$role' = role$
$user' = user$
$userid' = userid$
$resource' = resource$
$permission' = permission$
$accessRights' = accessRights$
$atm_action' = atm_action$
$abs_action' = abs_action$
$roles_SSD' = roles_SSD \wedge role_Assignment' = role_Assignment$

The operation schema NewSession adds a new session to the set of existing sessions. The declaration and the predicate are much of the same as given in previous schema; however, we need to specify that the set of roles, user, resources, permissions, atomic, and abstract actions, are unchanged.

$$
\begin{array}{|l}
\hline
\textit{DropSessionRole} \\
\Xi Sets \\
\Delta SessionRoles \\
session? : SESSION \\
role? : ROLE \\
\hline
session? \in session \\
accessRights' = accessRights \\
session_User' = session_User \\
session_Role' = session_Role \setminus \{role? \mapsto session?\} \\
roles_DSD' = roles_DSD \\
roles_SSD' = roles_SSD \wedge role_Assignment' = role_Assignment \\
\hline
\end{array}
$$

One of the session management functions of RBAC is to drop a session role. The operation schema DropSessionRole serves this purpose. It alters the set of session roles based on two taken inputs session? and role?. In the predicate part, we ensure that the session being modified is one of the existing sessions and thus, session_Role' is changed by deleting the pair role? $\mapsto$ session?.

7.3 Summary

This chapter presented the underlying formalism of validation queries. These queries were executed using Jaza that has similarity with the RBAC review functions as also done by SecureMOVA tool. Such queries are a useful means of evaluating security policies from customers as well as from security engineers viewpoint.

Chapter 8

Conclusion and Future Outlook

Contents

8.1 Conclusion 115
8.2 Suggestions for Future Work 116

8.1 Conclusion

This thesis is mainly composed of two parts i.e., state-of-the-art and the contributions. Part-I of the thesis presented the essential theoretical foundations to comprehend our work given in Part-II. We recapitulate them hereunder:

Chapter 2 detailed the Z formal notation briefly, and RoZ tool that also employs Z to formally specify and validate (or possibly verify) functional properties of an application, purely in an implementation-independent manner. Chapter 3 presented various access control mechanisms consisting of MAC, DAC, and RBAC and the data security properties. SecureUML borrowed as a modeling notation in our approach is also detailed in this chapter using an example. The next chapter 4 establishes RBAC based criteria in order to evaluate existing works. Our proposed criteria evaluate each of the techniques addressing the specification, verification, and validation of RBAC-based security policies.

Chapter 5 begins the contribution part by exposing the underlying security kernel with a case study from health care domain. We have presented an approach to validate security design models using Z assertions. In Chapter 6, we compared our Z based toolset for specification and validation of security policies with SecureMOVA and USE. We particularly highlighted our contributions from a user viewpoint. This chapter has addressed the validation of security policies expressed as RBAC rules with contextual constraints. This chapter mentions another case study named meeting SCHEDULER to share our insights comparing to the state-of-the-art tools. A number of queries have been used to this end. Chapter 7 elaborated upon the formalized validation queries set used in Chapter 5 & Chapter 6. The formal queries presented in this chapter bring our proposed security kernel to use for the validation of a security policies.

In this dissertation, we have presented a toolset based on a variant of SecureUML, RBAC, RoZ, and the Z specification language. It allows to perform static analyses, as done by the SecureMOVA tool, and dynamic analyses, playing

sequences of actions. Such sequences of actions correspond to expected behaviors, and to attacks against the secure system. We presented these tools on a classical example, the meeting SCHEDULER, addressed in the presentation of SecureMOVA. We identified a potential attack against the integrity of the information system that requires a sequence of actions to allow evolutions of the functional state. We believe that it is easier to analyze this sequence of actions with animation tools, than with static analyses only. We also have validated our approach on another case study taken from health care domain.

As stated, several works attempt to specify RBAC in Z. Most of them specify the RBAC meta-model. As far as we know, none of these has been used in conjunction with an animator in order to validate a given security policy. So our goal was not to model or validate RBAC itself, but to validate security policies expressed as RBAC rules in the context of a functional specification. Several tools exploit OCL in order to validate RBAC rules. Sohr *et* al., [Sohr *et al.* 2008] have adapted the USE OCL tool for the analysis of security policies. SecureMOVA [Basin *et al.* 2009], a tool dedicated to SecureUML, allows one to query the security policy, and to evaluate which actions are permitted for a given role in a given context, depicted as an object diagram. Still, both tools don't animate the operations of the functional model, making it difficult to evaluate how evolutions of the functional state can impact authorization rules.

Contextual constraints have been introduced in this thesis. These constraints refer to elements of both security and functional models, using the state of the functional model as a context to grant access rights. We find that separation of concerns suggests treating the functional and security models in isolation. Unfortunately, when constraints establish a link between these models, validation must take both models into account. Our work tries to incorporate the precision of formal languages into intuitive graphical models which leads the system through several state changes and involve both security and functional models. Such dynamic scenarios can exhibit security flaws, which cannot be detected by static queries.

8.2 Suggestions for Future Work

There are a number of possible research directions which can be followed as a sequel of this thesis. As we have seen that there reside several techniques to capture the design of an RBAC base system; yet there is no systematic support for all RBAC features as given by NIST.

A limited part of our translation from diagrammatic models to Z specifications is currently performed manually. Still, this manual translation is systematic. Our short term goal can be to integrate the translation of both security and functional parts into the Topcased tool. Finally, we did not evaluate the capability of our tool to scale up, and only used it on small models, with acceptable response times (a few seconds). Further work is needed to experiment it on real-size models and, if needed, to optimize its calculations. Also, the security kernel can be improved to

take into account additional concepts such as delegation or organization.

An adequate choice of nominal and attack scenarios is essential to guarantee the quality of the validation activities. Perspectives include the definition of metrics for the coverage of the model by these scenarios, and the automated generation of scenarios that systematically explore the model. This could benefit from the use of verification techniques like model-checking.

Animation is not the only way performing dynamic analysis. Model-checking provides an interesting alternative. In this chapter, we showed a sequence of actions which compromises the integrity of the information system. Our tools help identify such sequences, but model-checking could help find a sequence of actions which leads from a given initial state to some unwanted state. Model-checking tools are not available for the Z language, but Pro-B [Leuschel & Butler 2008] provides such a tool for the B language, which is close to Z. This gives an interesting perspective for future work.

There are numerous approaches discussing RBAC but one of another lacking areas is the unavailability of tool support to perform an early validation and verification. If one wishes to experience a formal language other than Z, Alloy can be a good option since it comes with sufficient technical and implementation details to perform specification and V&V of security policies. Alloy has emerged in the recent years as a simplification of OCL or Z which offers compact tool support essential to manage large scale systems.

SecureUML is a security profile that has formally specified a number of RBAC constructs both syntactically and semantically. Yet, some constraints such as SSD and DSD do not appear on its metamodel and its subsequent dialect named ComponentUML. Hence, it would be interesting to provide an extension of SecureUML for the remaining constructs of RBAC both syntactically as well as semantically.

Appendix A

Appendix - Complete Formal Specification of Security Kernel

Contents

A.1 Abstract and atomic actions **119**

A.2 List of employed roles **119**

A.3 Resources **119**

A.4 Types **120**

A.5 Basic RBAC and other sets **120**

A.1 Abstract and atomic actions

$$ABSTRACT_ACTION ::= EntityFullAccess \mid AssociationEndUpdate \mid EntityCreate \mid EntityRead \mid EntityUpdate \mid EntityDelete \mid CancelExecute \mid NotifyExecute$$

$$ATOMIC_ACTION ::= Notify1 \mid Cancel1 \mid ChangeStart1 \mid ChangeDuration1 \mid AddMeeting1 \mid RemoveMeeting1 \mid LinkmeetingsOfOwner1 \mid LinkmeetingsOfParticipant1 \mid CreateMeeting1 \mid ChangeName1 \mid AddPerson1 \mid RemovePerson1 \mid Linkowner1 \mid Linkparticipants1$$

A.2 List of employed roles

$$ROLE ::= SystemUser \mid Supervisor \mid SystemAdministrator \mid Director$$

A.3 Resources

$$RESOURCE ::= Meetings \mid Persons$$

A.4 Types

$USER == STRING$

$[USERID, SESSION, PERMISSION]$

A.5 Basic RBAC and other sets

Sets

$role : \mathbb{F}\, ROLE$
$user : \mathbb{F}\, USER$
$userid : \mathbb{F}\, USERID$
$session : \mathbb{F}\, SESSION$
$resource : \mathbb{F}\, RESOURCE$
$permission : \mathbb{F}\, PERMISSION$
$atm_action : \mathbb{F}\, ATOMIC_ACTION$
$abs_action : \mathbb{F}\, ABSTRACT_ACTION$

AccessRights

$Sets$
$accessRights : USERID \rightarrowtail\!\!\!\!\!\rightarrow USER$

$\mathrm{dom}\, accessRights \subseteq userid$
$\mathrm{ran}\, accessRights \subseteq user$

RoleAssignment

$Sets$
$roles_SSD : ROLE \leftrightarrow ROLE$
$role_Assignment : USER \leftrightarrow ROLE$

$role \neq \varnothing$
$\mathrm{dom}\, roles_SSD \subseteq role$
$\mathrm{ran}\, roles_SSD \subseteq role$
$\mathrm{dom}\, role_Assignment \subseteq user$
$\mathrm{ran}\, role_Assignment \subseteq role$
$\forall\, u : user \bullet \forall\, i, j : role \mid ((u \mapsto i) \in role_Assignment) \wedge ((u \mapsto j) \in role_Assignment) \bullet ((i, j) \notin roles_SSD)$

AddSSD

$\Xi Sets$
$\Delta RoleAssignment$
$role1? : ROLE$
$role2? : ROLE$

$roles_SSD' = roles_SSD \cup \{role1? \mapsto role2?\}$
$role_Assignment' = role_Assignment$

SessionRoles

$Sets$
$RoleAssignment$
$session_User : SESSION \nrightarrow USER$
$session_Role : ROLE \leftrightarrow SESSION$
$roles_DSD : ROLE \leftrightarrow ROLE$

$\mathrm{dom}\ roles_DSD \subseteq role$
$\mathrm{ran}\ roles_DSD \subseteq role$
$\mathrm{dom}\ session_User \subseteq session$
$\mathrm{ran}\ session_User \subseteq user$
$\mathrm{dom}\ session_Role \subseteq role$
$\mathrm{ran}\ session_Role \subseteq session$
$\forall\, r : role \bullet \forall\, s : session \bullet (r, s) \in session_Role$
$\quad \Rightarrow (session_User(s), r) \in role_Assignment$
$\forall\, s : session \bullet \forall\, i, j : role \mid ((i, s) \in session_Role)$
$\quad \wedge ((j, s) \in session_Role) \bullet ((i, j) \notin roles_DSD)$

ActionAssignment

$Sets$
$action_Assignment : ROLE \leftrightarrow$
$\quad (PERMISSION \times ABSTRACT_ACTION \times RESOURCE)$

$\mathrm{dom}\ action_Assignment \subseteq role$
$\mathrm{ran}\ action_Assignment \subseteq$
$\quad (permission \times abs_action \times resource)$

RoleInherits

$Sets$
$role_Inherits : ROLE \leftrightarrow ROLE$

$role_Inherits^{+} \cap \mathrm{id}\ role = \varnothing$
$\mathrm{dom}\ role_Inherits \subseteq role$
$\mathrm{ran}\ role_Inherits \subseteq role$

ActionsRelation

Sets

$action_Relation : ABSTRACT_ACTION \leftrightarrow (ATOMIC_ACTION \times RESOURCE)$

$\mathrm{dom}\ action_Relation \subseteq abs_action$

$\mathrm{ran}\ action_Relation \subseteq (atm_action \times resource)$

ActionInherits

Sets

$action_Inherits : ABSTRACT_ACTION \leftrightarrow ABSTRACT_ACTION$

$action_Inherits^{+} \cap \mathrm{id}\ abs_action = \varnothing$

$\mathrm{dom}\ action_Inherits \subseteq abs_action$

$\mathrm{ran}\ action_Inherits \subseteq abs_action$

ComputeActions

Sets

ActionInherits

ActionsRelation

$comp_Actions : ABSTRACT_ACTION \leftrightarrow (ATOMIC_ACTION \times RESOURCE)$

$$
\begin{aligned}
comp_Actions = \{ & i : \mathrm{dom}\ action_Relation;\ j : abs_action; \\
& k : \mathrm{ran}\ action_Relation \mid ((i \mapsto j) \in \\
& \quad ((action_Inherits^{+}) \cup (\mathrm{id}\ abs_action))) \\
& \quad \wedge ((i \mapsto k) \in action_Relation) \\
& \quad \bullet (j \mapsto k)\}
\end{aligned}
$$

ComputeAssignment

$Sets$
$RoleInherits$
$ComputeActions$
$ActionsRelation$
$ActionAssignment$
$comp_Assignment : ROLE \leftrightarrow$
$\quad (PERMISSION \times ABSTRACT_ACTION \times RESOURCE)$
$abstract_Assignment : ROLE \leftrightarrow$
$\quad (PERMISSION \times ABSTRACT_ACTION \times RESOURCE)$
$concrete_Assignment : ROLE \leftrightarrow$
$\quad (PERMISSION \times ATOMIC_ACTION \times RESOURCE)$

$comp_Assignment = \{r : \mathrm{dom}\, action_Assignment;\ x : role;$
$\quad a : \mathrm{ran}\, action_Assignment \mid ((r \mapsto x) \in ((role_Inherits^{+})$
$\quad\quad \cup(\mathrm{id}\, role))) \wedge ((r \mapsto a) \in action_Assignment)$
$\quad\quad\quad \bullet (x \mapsto a)\}$
$abstract_Assignment = \{r : \mathrm{dom}\, comp_Assignment;\ p : permission;$
$\quad asup, asub : abs_action;\ rsrc : resource \mid (r \mapsto (p, asup, rsrc))$
$\quad\quad \in comp_Assignment \wedge ((asub \mapsto asup) \in ((action_Inherits^{+})$
$\quad\quad\quad \cup(\mathrm{id}\, abs_action))) \bullet (r \mapsto (p, asub, rsrc))\}$
$concrete_Assignment = \{r : \mathrm{dom}\, comp_Assignment;\ p : permission;$
$\quad aa : abs_action;\ atm : atm_action;\ rsrc : resource$
$\quad\quad \mid (r \mapsto (p, aa, rsrc)) \in abstract_Assignment \wedge$
$\quad\quad\quad (aa \mapsto (atm, rsrc)) \in comp_Actions \bullet (r \mapsto (p, atm, rsrc))\}$

PermissionAssignment

$Sets$
$AccessRights$
$RoleAssignment$
$ComputeAssignment$
$perm_Assignment : (USERID \times USER \times ROLE) \leftrightarrow$
$\quad (PERMISSION \times ATOMIC_ACTION \times RESOURCE)$

$perm_Assignment = \{uid : \mathrm{dom}\, accessRights;\ u : \mathrm{dom}\, role_Assignment;$
$\quad r : \mathrm{ran}\, role_Assignment;\ b : \mathrm{ran}\, concrete_Assignment$
$\quad\quad \mid (uid, u) \in accessRights \wedge (u, r) \in role_Assignment$
$\quad\quad\quad \wedge (r, b) \in concrete_Assignment$
$\quad\quad\quad\quad \bullet ((uid, u, r) \mapsto b)\}$

SecureOperation

$\Xi Sets$; $\Xi SessionRoles$; $\Xi ActionsRelation$
$\Xi AccessRights$; $\Xi RoleAssignment$;
$\Xi PermissionAssignment$
$role? : ROLE$
$user? : USER$
$userid? : USERID$
$session? : SESSION$
$resource? : RESOURCE$
$permission? : PERMISSION$
$atm_action? : ATOMIC_ACTION$
$abs_action? : ABSTRACT_ACTION$

$permission? \in permission$
$(userid?, user?) \in accessRights$
$(user?, role?) \in role_Assignment$
$(session?, user?) \in session_User$
$(role?, session?) \in session_Role$
$(userid?, user?, role?) \in \mathrm{dom}\, perm_Assignment$
$(abs_action?, (atm_action?, resource?)) \in action_Relation$
$((userid?, user?, role?), (permission?, atm_action?, resource?)) \in perm_Assignment$

Appendix B

Appendix - Formal Specifications of Health care Information System

Contents

B.1 RoZ Types 125
B.2 RoZ data 125
B.3 RoZ Operations 127
B.4 Secure Operations 137

B.1 RoZ Types

$$[NAME, USERID, STRING, RECORDNB, ADDRESS, SPECIALTY]$$

B.2 RoZ data

MEDRECORD

$recordnb : RECORDNB$
$contents : STRING$

MedrecordExt

$Medrecord : \mathbb{F}\, MEDRECORD$

$true$

PATIENT

$id : USERID$
$name : NAME$
$address : ADDRESS$

PatientExt

$Patient : \mathbb{F}\, PATIENT$

$true$

HOSPITAL

$name : NAME$

HospitalExt

$Hospital : \mathbb{F}\, HOSPITAL$

DOCTOR

$id : USERID$
$name : NAME$

DoctorExt

$Doctor : \mathbb{F}\, DOCTOR$

MedrecordPatientRel

$PatientExt;\ MedrecordExt$
$patientOfMedrecord : MEDRECORD \nrightarrow PATIENT$
$medrecordOfPatient : PATIENT \nrightarrow MEDRECORD$

$\mathrm{dom}\, patientOfMedrecord = Medrecord$
$\mathrm{ran}\, patientOfMedrecord \subseteq Patient$
$medrecordOfPatient = \{patient : \mathrm{ran}\, patientOfMedrecord;\ medrecord :$
$\quad \mathrm{dom}\, patientOfMedrecord \mid (medrecord, patient)$
$\quad\quad \in patientOfMedrecord \bullet patient \mapsto medrecord\}$
$patientOfMedrecord = \{patient : \mathrm{ran}\, medrecordOfPatient;$
$\quad medrecord : \mathrm{dom}\, medrecordOfPatient \mid (medrecord, patient)$
$\quad\quad \in medrecordOfPatient \bullet patient \mapsto medrecord\}$

MedrecordDoctorRel

$MedrecordExt;\ DoctorExt$
$medrecordsOfDoctor : DOCTOR \nrightarrow \mathbb{F}\, MEDRECORD$
$doctorOfMedrecord : MEDRECORD \nrightarrow DOCTOR$

$\mathrm{dom}\, doctorOfMedrecord = Medrecord$
$\mathrm{ran}\, doctorOfMedrecord \subseteq Doctor$
$medrecordsOfDoctor = \{doctor : \mathrm{ran}\, doctorOfMedrecord \bullet doctor \mapsto$
$\quad \{medrecord : \mathrm{dom}\, doctorOfMedrecord \mid$
$\quad\quad doctorOfMedrecord(medrecord) = doctor \bullet medrecord\}\}$
$doctorOfMedrecord = \bigcup\{doctor : \mathrm{dom}\, medrecordsOfDoctor \bullet$
$\quad \{medrecord : medrecordsOfDoctor(doctor) \bullet medrecord \mapsto doctor\}\}$

PatientHospitalRel

$HospitalExt;\ PatientExt$
$hospitalOfPatient : PATIENT \nrightarrow HOSPITAL$
$patientsOfHospital : HOSPITAL \nrightarrow \mathbb{F}\, PATIENT$

$\mathrm{dom}\, hospitalOfPatient \subseteq Patient$
$\mathrm{ran}\, hospitalOfPatient \subseteq Hospital$
$patientsOfHospital = \{hospital : \mathrm{ran}\, hospitalOfPatient \bullet hospital \mapsto$
$\quad \{patient : \mathrm{dom}\, hospitalOfPatient \mid$
$\quad\quad hospitalOfPatient(patient) = hospital \bullet patient\}\}$
$hospitalOfPatient = \bigcup\{hospital : \mathrm{dom}\, patientsOfHospital \bullet$
$\quad \{patient : patientsOfHospital(hospital) \bullet patient \mapsto hospital\}\}$

DoctorHospitalRelRel

$HospitalExt;\ DoctorExt$
$hospitalsOfDoctor : DOCTOR \nrightarrow \mathbb{F}\, HOSPITAL$
$doctorsOfHospital : HOSPITAL \nrightarrow \mathbb{F}\, DOCTOR$

$\mathrm{dom}\, hospitalsOfDoctor \subseteq Doctor$
$\bigcup(\mathrm{ran}\, hospitalsOfDoctor) \subseteq Hospital$
$hospitalsOfDoctor = \{doctor : \bigcup(\mathrm{ran}\, doctorsOfHospital) \bullet doctor \mapsto$
$\quad \{hospital : \mathrm{dom}\, doctorsOfHospital \mid doctor \in$
$\quad\quad doctorsOfHospital(hospital) \bullet hospital\}\}$
$doctorsOfHospital = \{doctor : \bigcup(\mathrm{ran}\, hospitalsOfDoctor) \bullet$
$\quad doctor \mapsto \{hospital : \mathrm{dom}\, hospitalsOfDoctor \mid doctor$
$\quad\quad \in hospitalsOfDoctor(hospital) \bullet hospital\}\}$

GlobalView

$MedrecordPatientRel$
$MedrecordDoctorRel$
$PatientHospitalRel$
$DoctorHospitalRelRel$

B.3 RoZ Operations

ChangeMedrecord

$\Delta MedrecordExt$
$\Delta MEDRECORD$
$x? : MEDRECORD$

$x? \in Medrecord$
$\theta MEDRECORD = x?$
$Medrecord' = Medrecord \setminus \{x?\} \cup \{\theta MEDRECORD'\}$

$SubstituteMedrecordInRels$

$\Delta MedrecordPatientRel$
$\Delta MedrecordDoctorRel$
$\Xi PatientExt$
$\Xi DoctorExt$
$\Delta MEDRECORD$

$patientOfMedrecord' = (\{x : Medrecord \setminus \{\theta MEDRECORD\}$
$\quad \bullet x \mapsto x\} \cup \{\theta MEDRECORD' \mapsto$
$\quad \theta MEDRECORD\}) \fatsemi patientOfMedrecord$
$doctorOfMedrecord' = (\{x : Medrecord \setminus \{\theta MEDRECORD\}$
$\quad \bullet x \mapsto x\} \cup \{\theta MEDRECORD' \mapsto$
$\quad \theta MEDRECORD\}) \fatsemi doctorOfMedrecord$

$MEDRECORDChangeRecordnb$

$\Delta MEDRECORD$
$newrecordnb? : RECORDNB$

$recordnb' = newrecordnb?$
$contents' = contents$

$MedrecordChangeRecordnb == (ChangeMedrecord \wedge MEDRECORDChangeRecordnb)$
$\quad \backslash (recordnb, recordnb') \setminus (contents, contents')$

$MedrecordChangeRecordnbandRels == (ChangeMedrecord \wedge$
$\quad MEDRECORDChangeRecordnb \wedge SubstituteMedrecordInRels)$
$\quad \backslash (recordnb, recordnb') \setminus (contents, contents')$

$MEDRECORDChangeContents$

$\Delta MEDRECORD$
$newcontents? : STRING$

$contents' = newcontents?$
$recordnb' = recordnb$

$MedrecordChangeContents == (ChangeMedrecord \wedge$
$\quad MEDRECORDChangeContents) \setminus (recordnb, recordnb')$
$\quad \backslash (contents, contents')$

$MedrecordChangeContentsandRels == (ChangeMedrecord \wedge$
$\quad MEDRECORDChangeContents \wedge SubstituteMedrecordInRels)$
$\quad \backslash (recordnb, recordnb') \setminus (contents, contents')$

MedrecordAddMedrecord

$\Delta MedrecordExt$
$medrecord? : MEDRECORD$

$Medrecord' = Medrecord \cup \{medrecord?\}$

MedrecordRemoveMedrecord

$\Delta MedrecordExt$
$medrecord? : MEDRECORD$

$Medrecord' = Medrecord \setminus \{medrecord?\}$

medrecordLinkmedrecordOfPatient

$\Xi PatientExt;\ \Xi MedrecordExt$
$\Delta MedrecordPatientRel$
$patient? : PATIENT$
$medrecord? : MEDRECORD$

$patientOfMedrecord' = patientOfMedrecord \oplus \{medrecord? \mapsto patient?\}$

medrecordLinkmedrecordsOfDoctor

$\Xi MedrecordExt;\ \Xi DoctorExt$
$\Delta MedrecordDoctorRel$
$medrecord? : MEDRECORD$
$doctor? : DOCTOR$

$doctorOfMedrecord' = doctorOfMedrecord \oplus \{medrecord? \mapsto doctor?\}$

medrecordAddMedicalRecord

$\Delta MedrecordPatientRel$
$\Delta MedrecordDoctorRel$
$\Xi PatientExt$
$\Xi DoctorExt$
$m? : MEDRECORD$
$p? : PATIENT$
$d? : DOCTOR$

$Medrecord' = Medrecord \cup \{m?\}$
$patientOfMedrecord' = patientOfMedrecord \cup \{m? \mapsto p?\}$
$doctorOfMedrecord' = doctorOfMedrecord \cup \{m? \mapsto d?\}$
$m? \notin Medrecord$

medrecordReadMedrecord

$\Xi MedrecordExt$
$r? : RECORDNB$
$x! : \mathbb{F}\, MEDRECORD$

$x! = \{m : Medrecord \mid m.recordnb = r?\}$

medrecordRemoveMedicalRecordAndLinks

$\Delta MedrecordPatientRel$
$\Delta MedrecordDoctorRel$
$\Xi DoctorExt$
$\Xi PatientExt$
$m? : MEDRECORD$

$Medrecord' = Medrecord \setminus \{m?\}$
$patientOfMedrecord' = patientOfMedrecord \setminus$
$\quad \{m? \mapsto patientOfMedrecord(m?)\}$
$doctorOfMedrecord' = doctorOfMedrecord \setminus$
$\quad \{m? \mapsto doctorOfMedrecord(m?)\}$

ChangePatient

$\Delta PatientExt$
$\Delta PATIENT$
$x? : PATIENT$

$x? \in Patient$
$\theta PATIENT = x?$
$Patient' = Patient \setminus \{x?\} \cup \{\theta PATIENT'\}$

SubstitutePatientInRels

$\Delta MedrecordPatientRel$
$\Delta PatientHospitalRel$
$\Xi MedrecordExt$
$\Xi HospitalExt$
$\Delta PATIENT$

$medrecordOfPatient' = (\{x : Patient \setminus \{\theta PATIENT\}$
$\quad \bullet\, x \mapsto x\} \cup \{\theta PATIENT' \mapsto$
$\quad\quad \theta PATIENT\}) \fatsemi medrecordOfPatient$
$hospitalOfPatient' = (\{x : Patient \setminus \{\theta PATIENT\}$
$\quad \bullet\, x \mapsto x\} \cup \{\theta PATIENT' \mapsto \theta PATIENT\})$
$\quad\quad \fatsemi hospitalOfPatient$

$$
\begin{array}{|l}
\hline
\textit{PATIENTChangeId} \\
\Delta PATIENT \\
newid? : USERID \\
\hline
id' = newid? \\
name' = name \\
address' = address \\
\hline
\end{array}
$$

$$
\begin{aligned}
&PatientChangeId == (ChangePatient \wedge PATIENTChangeId) \\
&\qquad \backslash (id, id') \backslash (name, name') \backslash (address, address')
\end{aligned}
$$

$$
\begin{aligned}
&PatientChangeIdandRels == (ChangePatient \wedge PATIENTChangeId \\
&\qquad \wedge SubstitutePatientInRels) \backslash (id, id') \\
&\qquad \backslash (name, name') \backslash (address, address')
\end{aligned}
$$

$$
\begin{array}{|l}
\hline
\textit{PATIENTChangeName} \\
\Delta PATIENT \\
newname? : NAME \\
\hline
name' = newname? \\
id' = id \\
address' = address \\
\hline
\end{array}
$$

$$
\begin{aligned}
&PatientChangeName == (ChangePatient \wedge PATIENTChangeName) \\
&\qquad \backslash (id, id') \backslash (name, name') \backslash (address, address')
\end{aligned}
$$

$$
\begin{aligned}
&PatientChangeNameandRels == (ChangePatient \wedge \\
&\qquad PATIENTChangeName \wedge SubstitutePatientInRels) \\
&\qquad \backslash (id, id') \backslash (name, name') \backslash (address, address')
\end{aligned}
$$

$$
\begin{array}{|l}
\hline
\textit{PATIENTChangeAddress} \\
\Delta PATIENT \\
newaddress? : ADDRESS \\
\hline
address' = newaddress? \\
id' = id \\
name' = name \\
\hline
\end{array}
$$

$$
\begin{aligned}
&PatientChangeAddress == (ChangePatient \wedge PATIENTChangeAddress) \\
&\qquad \backslash (id, id') \backslash (name, name') \backslash (address, address')
\end{aligned}
$$

$$
\begin{aligned}
&PatientChangeAddressandRels == (ChangePatient \wedge \\
&\qquad PATIENTChangeAddress \wedge SubstitutePatientInRels) \\
&\qquad \backslash (id, id') \backslash (name, name') \backslash (address, address')
\end{aligned}
$$

PatientAddPatient

$\Delta PatientExt$
$patient? : PATIENT$

$Patient' = Patient \cup \{patient?\}$

PatientRemovePatient

$\Delta PatientExt$
$patient? : PATIENT$

$Patient' = Patient \setminus \{patient?\}$

patientLinkpatientOfMedrecord

$\Xi PatientExt;\ \Xi MedrecordExt$
$\Delta MedrecordPatientRel$
$patient? : PATIENT$
$medrecord? : MEDRECORD$

$patientOfMedrecord' = patientOfMedrecord \oplus \{medrecord? \mapsto patient?\}$

patientLinkpatientsOfHospital

$\Xi HospitalExt;\ \Xi PatientExt$
$\Delta PatientHospitalRel$
$hospital? : HOSPITAL$
$patient? : PATIENT$

$hospitalOfPatient' = hospitalOfPatient \oplus \{patient? \mapsto hospital?\}$

patientReadPatient

$\Xi GlobalView$
$name? : NAME$
$x! : \mathbb{F}\, PATIENT$

$x! = \{p : Patient \mid p.name = name? \bullet p\}$

patientUnlinkpatientsOfHospital

$\Xi HospitalExt;\ \Xi PatientExt$
$\Delta PatientHospitalRel$
$hospital? : HOSPITAL$
$patient? : PATIENT$

$hospitalOfPatient' = hospitalOfPatient \setminus \{patient? \mapsto hospital?\}$

$$patientClosePatientAndRecord == medrecordRemoveMedicalRecordAndLinks \fatsemi patientUnlinkpatientsOfHospital \fatsemi PatientRemovePatient$$

ChangeHospital

$$\Delta HospitalExt$$
$$\Delta HOSPITAL$$
$$x? : HOSPITAL$$

$$x? \in Hospital$$
$$\theta HOSPITAL = x?$$
$$Hospital' = Hospital \setminus \{x?\} \cup \{\theta HOSPITAL'\}$$

SubstituteHospitalInRels

$$\Delta PatientHospitalRel$$
$$\Delta DoctorHospitalRelRel$$
$$\Xi PatientExt$$
$$\Xi DoctorExt$$
$$\Delta HOSPITAL$$

$$patientsOfHospital' = (\{x : Hospital \setminus \{\theta HOSPITAL\} \bullet x \mapsto x\} \cup \{\theta HOSPITAL' \mapsto \theta HOSPITAL\}) \fatsemi patientsOfHospital$$
$$doctorsOfHospital' = (\{x : Hospital \setminus \{\theta HOSPITAL\} \bullet x \mapsto x\} \cup \{\theta HOSPITAL' \mapsto \theta HOSPITAL\}) \fatsemi doctorsOfHospital$$

HOSPITALChangeName

$$\Delta HOSPITAL$$
$$newname? : NAME$$

$$name' = newname?$$

$$HospitalChangeName == (ChangeHospital \wedge HOSPITALChangeName) \setminus (name, name')$$

$$HospitalChangeNameandRels == (ChangeHospital \wedge HOSPITALChangeName \wedge SubstituteHospitalInRels) \setminus (name, name')$$

HospitalAddHospital

$$\Delta HospitalExt$$
$$hospital? : HOSPITAL$$

$$Hospital' = Hospital \cup \{hospital?\}$$

HospitalRemoveHospital

$\Delta HospitalExt$
$hospital? : HOSPITAL$

$Hospital' = Hospital \setminus \{hospital?\}$

hospitalLinkhospitalOfPatient

$\Xi HospitalExt;\ \Xi PatientExt$
$\Delta PatientHospitalRel$
$hospital? : HOSPITAL$
$patient? : PATIENT$

$hospitalOfPatient' = hospitalOfPatient \oplus \{patient? \mapsto hospital?\}$

hospitalLinkhospitalsOfDoctor

$\Xi HospitalExt;\ \Xi DoctorExt$
$\Delta DoctorHospitalRelRel$
$hospital? : HOSPITAL$
$doctor? : DOCTOR$

$(hospital? \in \mathrm{dom}\, doctorsOfHospital) \Rightarrow$
$\quad (doctorsOfHospital' = doctorsOfHospital \oplus$
$\quad\quad \{hospital? \mapsto (doctorsOfHospital(hospital?)$
$\quad\quad\quad \cup \{doctor?\})\})$
$(hospital? \notin \mathrm{dom}\, doctorsOfHospital) \Rightarrow$
$\quad (doctorsOfHospital' = doctorsOfHospital \oplus$
$\{hospital? \mapsto \{doctor?\}\})$

ChangeDoctor

$\Delta DoctorExt$
$\Delta DOCTOR$
$x? : DOCTOR$

$x? \in Doctor$
$\theta DOCTOR = x?$
$Doctor' = Doctor \setminus \{x?\} \cup \{\theta DOCTOR'\}$

SubstituteDoctorInRels

$\Delta MedrecordDoctorRel$
$\Delta DoctorHospitalRelRel$
$\Xi MedrecordExt$
$\Xi HospitalExt$
$\Delta DOCTOR$

$medrecordsOfDoctor' = (\{x : Doctor \setminus \{\theta DOCTOR\}$
$\quad \bullet x \mapsto x\} \cup \{\theta DOCTOR' \mapsto$
$\quad \theta DOCTOR\}) \fatsemi medrecordsOfDoctor$
$hospitalsOfDoctor' = (\{x : Doctor \setminus \{\theta DOCTOR\}$
$\quad \bullet x \mapsto x\} \cup \{\theta DOCTOR' \mapsto$
$\quad \theta DOCTOR\}) \fatsemi hospitalsOfDoctor$

DOCTORChangeId

$\Delta DOCTOR$
$newid? : USERID$

$id' = newid?$
$name' = name$

$DoctorChangeId == (ChangeDoctor \wedge DOCTORChangeId)$
$\quad \backslash (id, id') \setminus (name, name')$

$DoctorChangeIdandRels == (ChangeDoctor \wedge DOCTORChangeId$
$\quad \wedge SubstituteDoctorInRels) \setminus (id, id') \setminus (name, name')$

DOCTORChangeName

$\Delta DOCTOR$
$newname? : NAME$

$name' = newname?$
$id' = id$

$DoctorChangeName == (ChangeDoctor \wedge DOCTORChangeName)$
$\quad \backslash (id, id') \setminus (name, name')$

$DoctorChangeNameandRels == (ChangeDoctor \wedge DOCTORChangeName$
$\quad \wedge SubstituteDoctorInRels) \setminus (id, id') \setminus (name, name')$

DoctorAddDoctor

$\Delta DoctorExt$
$doctor? : DOCTOR$

$Doctor' = Doctor \cup \{doctor?\}$

DoctorRemoveDoctor

$\Delta DoctorExt$
$doctor? : DOCTOR$

$Doctor' = Doctor \setminus \{doctor?\}$

doctorLinkdoctorOfMedrecord

$\Xi MedrecordExt;\ \Xi DoctorExt$
$\Delta MedrecordDoctorRel$
$medrecord? : MEDRECORD$
$doctor? : DOCTOR$

$doctorOfMedrecord' = doctorOfMedrecord \oplus$
$\quad \{medrecord? \mapsto doctor?\}$

doctorLinkdoctorsOfHospital

$\Xi HospitalExt;\ \Xi DoctorExt$
$\Delta DoctorHospitalRelRel$
$hospital? : HOSPITAL$
$doctor? : DOCTOR$

$(hospital? \in \operatorname{dom} doctorsOfHospital) \Rightarrow$
$\quad (doctorsOfHospital' = doctorsOfHospital \oplus$
$\quad\quad \{hospital? \mapsto (doctorsOfHospital(hospital?)$
$\quad\quad\quad \cup \{doctor?\})\})$
$(hospital? \notin \operatorname{dom} doctorsOfHospital) \Rightarrow$
$\quad (doctorsOfHospital' = doctorsOfHospital \oplus$
$\quad\quad \{hospital? \mapsto \{doctor?\}\})$

InitGlobalView

$HospitalExt'$
$PatientExt'$
$DoctorExt'$
$MedrecordExt'$
$MedrecordPatientRel'$
$MedrecordDoctorRel'$
$PatientHospitalRel'$
$DoctorHospitalRelRel'$

$Hospital' = \varnothing$
$Patient' = \varnothing$
$Doctor' = \varnothing$
$Medrecord' = \varnothing$
$patientOfMedrecord' = \varnothing$
$medrecordOfPatient' = \varnothing$
$medrecordsOfDoctor' = \varnothing$
$doctorOfMedrecord' = \varnothing$
$hospitalOfPatient' = \varnothing$
$patientsOfHospital' = \varnothing$
$hospitalsOfDoctor' = \varnothing$
$doctorsOfHospital' = \varnothing$

CheckGlobalInvariant

$\Xi GlobalView$

B.4 Secure Operations

SecurePatientAddPatient

$SecureOperation$
$PatientAddPatient$

$atm_action? = AddPatient1$
$resource? = Patients$

$$SecurePatientAddPatient2 == SecurePatientAddPatient \backslash (uid?, user?, abs_action?, atm_action?, resource?, permission?, role?)$$

SecureDoctorAddDoctor

$SecureOperation$
$DoctorAddDoctor$

$atm_action? = AddDoctor1$
$resource? = Doctors$

$SecureDoctorAddDoctor2 == SecureDoctorAddDoctor \backslash$
$(uid?, user?, abs_action?, atm_action?, resource?,$
$permission?, role?)$

$SecuremedrecordAddMedicalRecord$

$SecureOperation$
$medrecordAddMedicalRecord$

$atm_action? = AddMedicalRecord1$
$resource? = Medrecord$

$SecuremedrecordAddMedicalRecord2 == SecuremedrecordAddMedicalRecord$
$\backslash(uid?, user?, abs_action?, atm_action?,$
$resource?, permission?, role?)$

$SecuremedrecordReadMedrecord$

$SecureOperation$
$medrecordReadMedrecord$

$atm_action? = ReadMedrecord1$
$resource? = Medrecord$
$role? = Patient \Rightarrow (user? = (owner'(meeting?)).name)$

$SecuremedrecordReadMedrecord2 == SecuremedrecordReadMedrecord$
$\backslash(uid?, user?, abs_action?, atm_action?,$
$resource?, permission?, role?)$

$SecureMedrecordChangeContentsandRels$

$SecureOperation$
$MedrecordChangeContentsandRels$

$atm_action? = MedrecordChangeContentsandRels1$

$SecureMedrecordChangeContentsandRels2 ==$
$SecureMedrecordChangeContentsandRels \setminus (uid?, user?,$
$abs_action?, atm_action?, resource?, permission?, role?)$

$SecurepatientReadPatient$

$SecureOperation$
$patientReadPatient$

$atm_action? = ReadPatient1$
$resource? = Patients$

$SecurepatientReadPatient2 == SecurepatientReadPatient$
$\backslash(uid?, user?, abs_action?, atm_action?,$
$resource?, permission?, role?)$

$SecurePatientChangeAddressandRels$

$SecureOperation$
$PatientChangeAddressandRels$

$atm_action? = PatientChangeAddressandRels1$

$SecurePatientChangeAddressandRels2 ==$
$SecurePatientChangeAddressandRels \backslash (uid?, user?,$
$abs_action?, atm_action?, resource?, permission?, role?)$

$SecuremedrecordRemoveMedicalRecordAndLinks$

$SecureOperation$
$medrecordRemoveMedicalRecordAndLinks$

$atm_action? = medrecordRemoveMedicalRecordAndLinks1$

$SecuremedrecordRemoveMedicalRecordAndLinks2 ==$
$SecuremedrecordRemoveMedicalRecordAndLinks \backslash (uid?,$
$user?, abs_action?, atm_action?, resource?, permission?, role?)$

$SecurePatientRemovePatient$

$SecureOperation$
$PatientRemovePatient$

$atm_action? = RemovePatient1$
$resource? = Patients$

$SecurePatientRemovePatient2 == SecurePatientRemovePatient$
$\backslash(uid?, user?, abs_action?, atm_action?,$
$resource?, permission?, role?)$

$SecurepatientClosePatientAndRecord$

$SecureOperation$
$patientClosePatientAndRecord$

$atm_action? = patientClosePatientAndRecord1$

$SecurepatientClosePatientAndRecord2 ==$
$SecurepatientClosePatientAndRecord \backslash (uid?, user?,$
$abs_action?, atm_action?, resource?, permission?, role?)$

Init

$Sets'$; $MedrecordExt'$; $PatientExt'$; $DoctorExt'$;
$SessionRoles'$; $MedrecordPatientRel'$; $MedrecordDoctorRel'$;
$AccessRights'$; $RoleAssignment'$; $ActionAssignment'$;
$RoleInherits'$; $ComputeAssignment'$; $PermissionAssignment'$

$Patients' = \{\}$
$Doctor' = \{\}$
$Medrecord' = \{\}$
$patientOfMedrecord' = \{\}$
$medrecordOfPatient' = \{\}$
$medrecordsOfDoctor' = \{\}$
$doctorOfMedrecord' = \{\}$

Appendix C

Appendix - Secure Operations of Meeting SCHEDULER Example

Contents

C.1 Z types 141
C.2 RoZ generated operations 141
C.3 Secure Operations of Meeting Scheduler Example 146
C.4 RoZ Data 150

C.1 Z types

$[STRING]$

$DATE == \mathbb{Z}$

$TIME == \mathbb{Z}$

C.2 RoZ generated operations

ChangeMeeting

$\Delta MeetingExt$
$\Delta MEETING$
$meeting? : MEETING$

$meeting? \in Meeting$
$\theta MEETING = meeting?$
$Meeting' = Meeting \setminus \{meeting?\} \cup \{\theta MEETING'\}$

SubstituteMeetingInRels

$\Delta MeetingOwnerRel$
$\Delta MeetingParticipantsRel$
$\Xi PersonExt$
$\Xi PersonExt$
$\Delta MEETING$

$owner' = (\{x : Meeting \setminus \{\theta MEETING\} \bullet x \mapsto x\}$
$\quad \cup \{\theta MEETING' \mapsto \theta MEETING\}) \fatsemi owner$
$participants' = (\{x : Meeting \setminus \{\theta MEETING\} \bullet x \mapsto x\}$
$\quad \cup \{\theta MEETING' \mapsto \theta MEETING\}) \fatsemi participants$

MEETINGChangeStart

$\Delta MEETING$
$newstart? : DATE$

$start' = newstart?$
$duration' = duration$

$MeetingChangeStart == (ChangeMeeting \wedge MEETINGChangeStart)$
$\quad \backslash (start, start') \setminus (duration, duration')$

$MeetingChangeStartandRels == (ChangeMeeting \wedge MEETINGChangeStart$
$\quad \wedge SubstituteMeetingInRels) \setminus (start, start') \setminus (duration, duration')$

MEETINGChangeDuration

$\Delta MEETING$
$newduration? : TIME$

$duration' = newduration?$
$start' = start$

$MeetingChangeDuration == (ChangeMeeting \wedge MEETINGChangeDuration)$
$\quad \backslash (start, start') \setminus (duration, duration')$

$MeetingChangeDurationandRels == (ChangeMeeting \wedge MEETINGChangeDuration$
$\quad \wedge SubstituteMeetingInRels) \setminus (start, start') \setminus (duration, duration')$

MeetingAddMeeting

$\Delta MeetingExt$
$meeting? : MEETING$

$Meeting' = Meeting \cup \{meeting?\}$

MeetingRemoveMeeting

$\Delta MeetingExt$
$meeting? : MEETING$

$Meeting' = Meeting \setminus \{meeting?\}$

meetingLinkmeetingsOfOwner

$\Xi PersonExt;\ \Xi MeetingExt$
$\Delta MeetingOwnerRel$
$person? : PERSON$
$meeting? : MEETING$

$owner' = owner \oplus \{meeting? \mapsto person?\}$

meetingLinkmeetingsOfParticipant

$\Xi PersonExt;\ \Xi MeetingExt$
$\Delta MeetingParticipantsRel$
$person? : PERSON$
$meeting? : MEETING$

$(person? \in \operatorname{dom} meetingsOfParticipant) \Rightarrow$
$\quad (meetingsOfParticipant' = meetingsOfParticipant \oplus$
$\quad\quad \{person? \mapsto (meetingsOfParticipant(person?) \cup \{meeting?\})\})$
$(person? \notin \operatorname{dom} meetingsOfParticipant) \Rightarrow$
$\quad (meetingsOfParticipant' = meetingsOfParticipant \oplus$
$\quad\quad \{person? \mapsto \{meeting?\}\})$

meetingnotify

$\Xi MeetingParticipantsRel$
$meeting? : MEETING$
$x! : MEETING \times (\mathbb{F}\, PERSON)$

$x! = (meeting?, participants(meeting?))$
$meeting? \in Meeting$

meetingcancel

$MeetingRemoveMeeting$
$\Xi PersonExt$
$\Delta MeetingOwnerRel$
$\Delta MeetingParticipantsRel$

$owner' = \{meeting?\} \lhd owner$
$participants' = \{meeting?\} \lhd participants$

$$
\begin{array}{|l}
\hline
\textit{meetingcreateMeeting} \\
\textit{MeetingAddMeeting} \\
\Xi \textit{PersonExt} \\
\Delta \textit{MeetingOwnerRel} \\
\Delta \textit{MeetingParticipantsRel} \\
\textit{owner?} : \textit{PERSON} \\
\hline
\textit{owner}' = \textit{owner} \oplus \{\textit{meeting?} \mapsto \textit{owner?}\} \\
\textit{participants}' = \textit{participants} \oplus \{\textit{meeting?} \mapsto \{\textit{owner?}\}\} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{ChangePerson} \\
\Delta \textit{PersonExt} \\
\Delta \textit{PERSON} \\
\textit{person?} : \textit{PERSON} \\
\hline
\textit{person?} \in \textit{Person} \\
\theta \textit{PERSON} = \textit{person?} \\
\textit{Person}' = \textit{Person} \setminus \{\textit{person?}\} \cup \{\theta \textit{PERSON}'\} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{SubstitutePersonInRels} \\
\Delta \textit{MeetingOwnerRel} \\
\Delta \textit{MeetingParticipantsRel} \\
\Xi \textit{MeetingExt} \\
\Xi \textit{MeetingExt} \\
\Delta \textit{PERSON} \\
\hline
\textit{meetingsOfOwner}' = (\{x : \textit{Person} \setminus \{\theta \textit{PERSON}\} \bullet x \mapsto x\} \\
\qquad \cup \{\theta \textit{PERSON}' \mapsto \theta \textit{PERSON}\}) \fatsemi \textit{meetingsOfOwner} \\
\textit{meetingsOfParticipant}' = (\{x : \textit{Person} \setminus \{\theta \textit{PERSON}\} \bullet x \mapsto x\} \\
\qquad \cup \{\theta \textit{PERSON}' \mapsto \theta \textit{PERSON}\}) \fatsemi \textit{meetingsOfParticipant} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{PERSONChangeName} \\
\Delta \textit{PERSON} \\
\textit{newname?} : \textit{STRING} \\
\hline
\textit{name}' = \textit{newname?} \\
\hline
\end{array}
$$

$$\textit{PersonChangeName} == (\textit{ChangePerson} \wedge \textit{PERSONChangeName}) \setminus (\textit{name}, \textit{name}')$$

$$
\begin{aligned}
\textit{PersonChangeNameandRels} == (&\textit{ChangePerson} \wedge \textit{PERSONChangeName} \\
&\wedge \textit{SubstitutePersonInRels}) \setminus (\textit{name}, \textit{name}')
\end{aligned}
$$

PersonAddPerson

$\Delta PersonExt$
$person? : PERSON$

$Person' = Person \cup \{person?\}$

PersonRemovePerson

$\Delta PersonExt$
$person? : PERSON$

$Person' = Person \setminus \{person?\}$

PersonRemovePersonAndLinks

$\Delta PersonExt$
$\Delta MeetingOwnerRel$
$\Delta MeetingParticipantsRel$
$\Xi MeetingExt$
$person? : PERSON$

$Person' = Person \setminus \{person?\}$
$meetingsOfParticipant' = \ \mathbf{if}\ person? \in \mathrm{dom}\, meetingsOfParticipant$
$\quad \mathbf{then}\ (meetingsOfParticipant \setminus \{person?$
$\quad \mapsto meetingsOfParticipant(person?)\})\ \mathbf{else}\ meetingsOfParticipant$
$meetingsOfOwner' = \ \mathbf{if}\ person? \in \mathrm{dom}\, meetingsOfOwner\ \mathbf{then}$
$\quad meetingsOfOwner \setminus \{person? \mapsto meetingsOfOwner(person?)\}$
$\quad \mathbf{else}\ meetingsOfOwner$

personLinkowner

$\Xi PersonExt;\ \Xi MeetingExt$
$\Delta MeetingOwnerRel$
$person? : PERSON$
$meeting? : MEETING$

$owner' = owner \oplus \{meeting? \mapsto person?\}$

personLinkparticipants

$\Xi PersonExt;\ \Xi MeetingExt$
$\Delta MeetingParticipantsRel$
$person? : PERSON$
$meeting? : MEETING$

$(person? \in \mathrm{dom}\, meetingsOfParticipant) \Rightarrow$
$\quad (meetingsOfParticipant' = meetingsOfParticipant \oplus$
$\quad \{person? \mapsto (meetingsOfParticipant(person?) \cup \{meeting?\})\})$
$(person? \notin \mathrm{dom}\, meetingsOfParticipant) \Rightarrow$
$\quad (meetingsOfParticipant' = meetingsOfParticipant \oplus$
$\quad \{person? \mapsto \{meeting?\}\})$

InitGlobalView

$MeetingExt'$
$PersonExt'$
$MeetingOwnerRel'$
$MeetingParticipantsRel'$

$Meeting' = \varnothing$
$Person' = \varnothing$
$owner' = \varnothing$
$meetingsOfOwner' = \varnothing$
$participants' = \varnothing$
$meetingsOfParticipant' = \varnothing$

CheckGlobalInvariant

$\Xi GlobalView$

C.3 Secure Operations of Meeting Scheduler Example

SecurePersonAddPerson

$SecureOperation$
$PersonAddPerson$

$atm_action? = AddPerson1$
$resource? = Persons$

$$SecurePersonAddPerson2 == SecurePersonAddPerson \setminus (userid?, user?, abs_action?, atm_action?, resource?, permission?, role?)$$

SecurePersonRemovePerson

$SecureOperation$
$PersonRemovePersonAndLinks$

$atm_action? = RemovePerson1$
$resource? = Persons$

$$SecurePersonRemovePerson2 == SecurePersonRemovePerson \setminus (userid?, user?, abs_action?, atm_action?, resource?, permission?, role?)$$

SecuremeetingcreateMeeting

$SecureOperation$
$meetingcreateMeeting$

$atm_action? = CreateMeeting1$
$resource? = Meetings$
$role? = SystemUser \Rightarrow (user? = (owner'(meeting?)).name)$

$SecuremeetingcreateMeeting2 == SecuremeetingcreateMeeting \setminus (userid?,$
$user?, abs_action?, atm_action?, resource?, permission?, role?)$

$SecureMEETINGChangeDuration$

$SecureOperation$
$MeetingChangeDurationandRels$

$atm_action? = ChangeDuration1$
$resource? = Meetings$
$user? = (owner(meeting?)).name$

$SecuremeetingChangeDuration2 == SecureMEETINGChangeDuration \setminus (userid?,$
$user?, abs_action?, atm_action?, resource?, permission?, role?)$

$SecureMEETINGChangeStart$

$SecureOperation$
$MeetingChangeStartandRels$

$atm_action? = ChangeStart1$
$resource? = Meetings$
$user? = (owner(meeting?)).name$

$SecuremeetingChangeStart2 == SecureMEETINGChangeStart \setminus (userid?,$
$user?, abs_action?, atm_action?, resource?, permission?, role?)$

$SecurepersonLinkowner$

$SecureOperation$
$personLinkowner$

$atm_action? = Linkowner1$
$resource? = Persons$

$SecurepersonLinkowner2 == SecurepersonLinkowner \setminus (userid?,$
$user?, abs_action?, atm_action?, resource?, permission?, role?)$

$SecureMEETINGChangeDuration2 == SecureMEETINGChangeDuration \setminus$
$(userid?, user?, abs_action?, atm_action?, resource?, permission?, role?)$

$SecurepersonLinkparticipants$

$SecureOperation$
$personLinkparticipants$
$\Xi MeetingOwnerRel$

$atm_action? = Linkparticipants1$
$resource? = Persons$
$user? = (owner(meeting?)).name$

$SecurepersonLinkparticipants2 == SecurepersonLinkparticipants \setminus$
$(userid?, user?, abs_action?, atm_action?, resource?, permission?, role?)$

$Securemeetingnotify$

$SecureOperation$
$meetingnotify$

$atm_action? = Notify1$
$resource? = Meetings$

$SecuremeetingLinkmeetingsOfOwner$

$SecureOperation$
$meetingLinkmeetingsOfOwner$

$atm_action? = LinkmeetingsOfOwner1$
$resource? = Meetings$
$user? = (owner(meeting?)).name$

$SecuremeetingLinkmeetingsOfOwner2 == SecuremeetingLinkmeetingsOfOwner$
$\setminus(userid?, user?, abs_action?, atm_action?, resource?, permission?, role?)$

$SecuremeetingLinkmeetingsOfParticipant$

$SecureOperation$
$meetingLinkmeetingsOfParticipant$
$\Xi MeetingOwnerRel$

$atm_action? = LinkmeetingsOfParticipant1$
$resource? = Meetings$
$user? = (owner(meeting?)).name$

$SecuremeetingLinkmeetingsOfParticipant2 == SecuremeetingLinkmeetingsOfParticipant$
$\setminus(userid?, user?, abs_action?, atm_action?, resource?, permission?, role?)$

$Securemeetingnotify2 == Securemeetingnotify \setminus (userid?, user?,$
$abs_action?, atm_action?, resource?, permission?, role?)$

$Securemeetingcancel$

$SecureOperation$
$meetingcancel$

$atm_action? = Cancel1$
$resource? = Meetings$
$meeting? \in Meeting$
$role?! = Supervisor \Rightarrow (user? = (owner(meeting?)).name)$

$$Securemeetingcancel2 == Securemeetingcancel \backslash (userid?, user?, abs_action?, atm_action?, resource?, permission?, role?)$$

SecureTransaction

$$SecurePersonAddPerson$$
$$SecuremeetingcreateMeeting$$
$$SecurepersonLinkparticipants$$

TransactionRoleNeededToPlayAction

$$DisplayRole$$
$$AddSessionRole$$
$$Securemeetingcancel2$$

$$RoleNeededForMeetingCancel == (NewSession \backslash (role?)) \fatsemi (Securemeetingcancel2)$$

$$UserAndRoleNeededForMeetingCancel == (NewSession \backslash (user?, role?)) \fatsemi (Securemeetingcancel2)$$

$$UserNeededForMeetingCancel == (NewSession \backslash (user?)) \fatsemi (Securemeetingcancel2)$$

$$RoleNeededForMeetingChangeStart == (NewSession \backslash (role?)) \fatsemi (SecuremeetingChangeStart2)$$

$$TransactionRoleNeededToPlayAction2 == (AddSessionRole \backslash (role?, user?)) \fatsemi (Securemeetingcancel2)$$

$$SecureTransaction2 == (SecurePersonAddPerson[role2?/role?, action2?/abs_action?, action3?/atm_action, permission2?/permission?, resource2?/resource?]) \fatsemi (SecuremeetingcreateMeeting[role3?/role?, action4?/abs_action?, action5?/atm_action?, permission3?/permission?, resource3?/resource?]) \fatsemi SecurepersonLinkparticipants$$

$$SecureTransaction3 == (SecurePersonAddPerson[role2?/role?, action2?/abs_action?, action3?/atm_action, permission2?/permission?, resource2?/resource?]) \fatsemi (SecuremeetingcreateMeeting[role3?/role?, action4?/abs_action?, action5?/atm_action?, permission3?/permission?, resource3?/resource?])$$

C.4 RoZ Data

MEETING

$start : DATE$
$duration : TIME$

MeetingExt

$Meeting : \mathbb{F}\, MEETING$

PERSON

$name : STRING$

PersonExt

$Person : \mathbb{F}\, PERSON$

MeetingOwnerRel

$PersonExt;\ MeetingExt$
$owner : MEETING \nrightarrow PERSON$
$meetingsOfOwner : PERSON \nrightarrow \mathbb{F}\, MEETING$

$\mathrm{dom}\, owner = Meeting$
$\mathrm{ran}\, owner \subseteq Person$
$meetingsOfOwner = \{person : \mathrm{ran}\, owner \bullet person \mapsto$
$\quad \{meeting : \mathrm{dom}\, owner \mid owner(meeting) = person \bullet meeting\}\}$
$owner = \bigcup\{person : \mathrm{dom}\, meetingsOfOwner \bullet$
$\quad \{meeting : meetingsOfOwner(person) \bullet meeting \mapsto person\}\}$

MeetingParticipantsRel1

$PersonExt;\ MeetingExt$
$participants : MEETING \nrightarrow \mathbb{F}\, PERSON$
$meetingsOfParticipant : PERSON \nrightarrow \mathbb{F}\, MEETING$

$\mathrm{dom}\, participants = Meeting$
$\bigcup(\mathrm{ran}\, participants) \subseteq Person$
$participants = \{meeting : \bigcup(\mathrm{ran}\, meetingsOfParticipant)$
$\quad \bullet meeting \mapsto \{person : \mathrm{dom}\, meetingsOfParticipant$
$\quad\quad \mid meeting \in meetingsOfParticipant(person) \bullet person\}\}$
$meetingsOfParticipant = \{meeting : \bigcup(\mathrm{ran}\, participants) \bullet meeting \mapsto$
$\quad \{person : \mathrm{dom}\, participants \mid meeting$
$\quad\quad \in participants(person) \bullet person\}\}$

MeetingParticipantsRel

$MeetingParticipantsRel1$

$true$

GlobalView

$MeetingOwnerRel$

$MeetingParticipantsRel$

$\forall\, m : Meeting \bullet owner(m) \in participants(m)$

Bibliography

[Abdallah & Khayat 2006] A. E. Abdallah and E. J. Khayat. *Formal Z Specifications of Several Flat Role-Based Access Control Models.* In Proceedings of the 30th Annual IEEE/NASA Software Engineering Workshop (SEW'06), pages 282–292, 2006. (Cited on pages 6 and 62.)

[Abou El Kalam *et al.* 2003] A. Abou El Kalam, R. El Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel and G. Trouessin. *Organization Based Access Control.* In 4th IEEE International Workshop on Policies for Distributed Systems and Networks (Policy'03), June 2003. (Cited on page 47.)

[Abrial 1996] J.R. Abrial. The B-Book. Cambridge Univ. Press, 1996. (Cited on pages 2 and 90.)

[Ahn & Hu 2007] G. Ahn and H. Hu. *Towards Realizing a Formal RBAC Model in Real Systems.* In Proceedings of the 12th ACM symposium on Access control models and technologies (SACMAT'07). ACM Press, 2007. (Cited on pages 6, 54, 57, 59, 60 and 61.)

[Ahn & Shin 2001] Gail-Joon Ahn and Michael E. Shin. *Role-Based Authorization Constraints Specification Using Object Constraint Language.* In WETICE, pages 157–162, 2001. (Cited on page 59.)

[Aït-Sadoune & Ameur 2008] Idir Aït-Sadoune and Yamine Aït Ameur. *Animating Event B Models by Formal Data Models.* In ISoLA, pages 37–55, 2008. (Cited on page 103.)

[Alghathbar 2007] Khaled Alghathbar. *Validating the enforcement of access control policies and separation of duty principle in requirement engineering.* Information & Software Technology, vol. 49, no. 2, pages 142–157, 2007. (Cited on pages 57 and 59.)

[Anastasakis *et al.* 2010] Kyriakos Anastasakis, Behzad Bordbar, Geri Georg and Indrakshi Ray. *On challenges of model transformation from UML to Alloy.* Software and System Modeling, vol. 9, no. 1, pages 69–86, 2010. (Cited on page 62.)

[Andersson 2001] R. Andersson. *Security Engineering - A Guide to Building Dependable Distribution Systems.* John Wiley & Sons, Inc.„ 2001. (Cited on page 39.)

[Autrel *et al.* 2008] Fabien Autrel, Frédéric Cuppens, Nora Cuppens-Boulahia and Céline Coma-Brebel. *MotOrBAC 2: a security policy tool.* In SARSSI'08 : 3e conf. Sécurité des Architectures Réseaux et des Systèmes d'Information, 2008. (Cited on page 90.)

[Barden *et al.* 1996] Rosalind Barden, Susan Stepney and David Cooper. Z in Practice. BCS Practitioner Series. Prentice-Hall, ISBN 0-13-124934-7, 1996. (Cited on page 103.)

[Barjaktarovic 1998] M. Barjaktarovic. The state-of-the-art in formal methods. AFOSR Summer Research Technical Report for Rome Research Site, Formal Methods Framework-Monthly Status Report, F30602-99-C-0166, WetStone Technologies, 1998. (Cited on page 1.)

[Barker 2000] D. Barker. *Requirements Modeling Technology, A Vision for Better, Faster and Cheaper Systems.* In Proceedings of the VHDL International Users Forum Fall Workshop, pages 3–7, 2000. (Cited on pages 1 and 3.)

[Basin *et al.* 2006a] D. Basin, J. Doser and T. Lodderstedt. *Model Driven Security: From UML Models to Access Control Infrastructures.* Proceedings of the ACM Transactions on Software Engineering and Methodology (TOSEM'06), vol. 15, no. 1, pages 39–91, 2006. (Cited on page 54.)

[Basin *et al.* 2006b] David A. Basin, Jürgen Doser and Torsten Lodderstedt. *Model driven security: From UML models to access control infrastructures.* ACM Trans. Softw. Eng. Methodol., vol. 15, no. 1, pages 39–91, 2006. (Cited on page 57.)

[Basin *et al.* 2009] David A. Basin, Manuel Clavel, Jürgen Doser and Marina Egea. *Automated analysis of security-design models.* Information & Software Technology, vol. 51, no. 5, pages 815–831, 2009. (Cited on pages 1, 5, 6, 22, 49, 50, 51, 54, 56, 57, 58, 79, 88, 89, 96, 100 and 116.)

[Bell & LaPadula 1975] D. Bell and L. LaPadula. *Secure Computer System: Unified Exposition and Multics Interpretation.* 1975. (Cited on pages 2, 36 and 37.)

[Bishop 2003] M. Bishop. *Computer Security, Art and Science.* Addison Wesley, 2003. (Cited on pages 35, 47 and 48.)

[Boehm 1984] Barry W. Boehm. *Verifying and Validating Software Requirements and Design Specifications.* IEEE Software, vol. 1, no. 1, pages 75–88, 1984. (Cited on pages 1 and 20.)

[Boswell 1995] A. Boswell. *Specification and Validation of a Security Policy Model.* Proceedings of the IEEE Transactions on Software Engineering, (TSE'06), vol. 21, no. 2, pages 63–68, 1995. (Cited on page 62.)

[Bowen 1996] Jonathan P. Bowen. *Formal Specification and Documentation using Z: A Case Study Approach.* 1996. (Cited on page 11.)

[Bowen 2003] J. Bowen. *Formal Specification and Documentation using Z: A Case Study Approach.* Thomson Publishing, 2003. (Cited on pages 12 and 17.)

[Ç. Cirit & Buzluca 2009] Ç. Cirit and Feza Buzluca. *A UML profile for role-based access control.* In SIN, pages 83–92, 2009. (Cited on pages 57 and 58.)

[Clark & Wilson 1987] D. D. Clark and D. R. Wilson. *A Comparison of Commercial and Military Computer Security Policies.* In IEEE Symp. on Security and Privacy, 1987. (Cited on page 86.)

[Clarke & Wing 1996] Edmund M. Clarke and Jeannette M. Wing. *Formal Methods: State of the Art and Future Directions.* ACM Computing Surveys, vol. 28, pages 626–643, 1996. (Cited on page 1.)

[Davies & Woodcock 1996] J. Davies and J. Woodcock. Using Z: Specification, Refinement, and Proof. Prentice Hall, ISBN 0-13-948472-8, 1996. (Cited on page 11.)

[DOD 5200.28-STD 1985] DOD 5200.28-STD. *Trusted Computer System Evaluation Criteria.* Rapport technique, United States Department of Defense, 1985. (Cited on page 36.)

[Drouineaud *et al.* 2004] M. Drouineaud, M. Bortin, P. Torrini and K. Sohr. *A First Step Towards Formal Verification of Security Policy Properties for RBAC.* In Proceedings of the International Conference on Quality Software (QSIC-2004), pages 60–67. IEEE, 2004. (Cited on page 63.)

[Dupuy *et al.* 2000] S. Dupuy, Y. Ledru and M. Chabre-Peccoud. *An Overview of RoZ: A Tool for Integrating UML and Z Specifications.* In Proc. 12th Conf. on Advanced information Systems Engineering (CAiSE'2000), pages 417–430. LNCS, Vol. 1789, 2000. (Cited on pages 2, 20, 27, 32 and 93.)

[Ferraiolo *et al.* 1999] D. F. Ferraiolo, J. F. Barkley and D. R. Kuhn. *A Role-Based Access Control Model and Reference Implementation within a Corporate Intranet.* ACM Trans. Inf. Syst. Secur., vol. 2, no. 1, pages 34–64, 1999. (Cited on page 47.)

[Ferraiolo *et al.* 2001] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn and Ramaswamy Chandramouli. *Proposed NIST Standard for Role-Based Access Control*, 2001. (Cited on pages 1, 2, 6, 36, 40, 42, 43, 44, 45, 46, 48, 54 and 90.)

[Ferraiolo *et al.* 2007] D. A. Ferraiolo, D. R. Kuhn and R. Chandramouli. *Role-Based Access Control (2nd edition).* Artech House Publishers, 2007. (Cited on page 39.)

[Gallaher *et al.* 1996] M. P. Gallaher, A.C. OŠConnor and B. Kropp. The economic impact of role-based access control. National Institute of Standards & Technology, Planning Report 02-1, 1996. (Cited on page 2.)

[Gogolla *et al.* 2007] M. Gogolla, F. Büttner and M. Richters. *USE: A UML-based Specification Environment for Validating UML and OCL.* Science of Computer Programming, vol. 69, pages 27–34, 2007. (Cited on pages 4 and 87.)

[Hall 1994] A. Hall. *Specifying and Interpreting Class Hierarchies in Z.* In Proceedings of the Z User Workshop, (Cambridge'94), ed. J. P. Bowen and J. A. Hall, pages 120–138. Springer, 1994. (Cited on page 61.)

[Hu & Ahn 2008] Hongxin Hu and Gail-Joon Ahn. *Enabling verification and conformance testing for access control model.* In SACMAT, pages 195–204, 2008. (Cited on pages 6, 54 and 60.)

[IBM] IBM. *Rational Rose.* (Cited on page 21.)

[IEEE 1982] IEEE. *Glossary of Software Engineering Terminology*, September 23,1982. (Cited on page 1.)

[ISO 2002] ISO. *Information technology – Z formal specification notation – Syntax, type system and semantics*, 2002. (Cited on page 90.)

[Jackson 2002] D. Jackson. *Alloy: A Lightweight Object Modelling Notation.* ACM Trans. Softw. Eng. Methodol., vol. 11, no. 2, pages 256–290, 2002. (Cited on pages 54 and 59.)

[Jackson 2006] Daniel Jackson. Software abstractions: logic, language and analysis. MIT Press, 2006. (Cited on pages 2 and 90.)

[Jia 1995] Xiaoping Jia. *An Approach to Animating Z Specifications.* In In Proc. 19th Annual IEEE International Computer Software and Applications Conference (COMPSACŠ95, pages 108–113, 1995. (Cited on page 103.)

[Jia 2002] Xiaoping Jia. *ZTC: Z Type Checker.* 2002. (Cited on pages 11 and 12.)

[Jürjens 2005] Jan Jürjens. Secure systems development with uml. Springer Academic Publishers, ISBN: 978-3-540-00701-2, 2005. (Cited on pages 4, 49, 59 and 86.)

[Keyes 2002] J. Keyes. Software Engineering Handbook. Auerbach Publications, ISBN 978-0849314797, 2002. (Cited on page 1.)

[Kim *et al.* 2004] D. Kim, I. Ray, R. B. France and N. Li. *Modeling Role-based Access Control Using Parameterized UML Models.* In Proceedings of the Fundamental Approaches to Software Engineering (FASE'04), pages 180–193. LNCS 2984, Springer, 2004. (Cited on pages 54, 57 and 59.)

[Latham 1983] Donald C. Latham. *Department of Defense Trusted Computer System Evaluation Criteria*, 1983. (Cited on page 2.)

[Ledru *et al.* 2011] Y. Ledru, N. Qamar, A. Idani, Jean-luc and M. Labiadh. *Validation of Security Policies by the Animation of Z Specifications.* In SACMAT 2011, 16th ACM Symp. on Access Control Models and Technologies. ACM, 2011. (Cited on page 85.)

[Ledru 1998] Yves Ledru. *Identifying Pre-Conditions with the Z/EVES Theorem Prover.* In ASE, pages 32–, 1998. (Cited on pages 22 and 27.)

[Ledru 2006] Y. Ledru. *Using Jaza to Animate RoZ Specifications of UML Class Diagrams.* In Proceedings of the 30th Annual IEEE/NASA Software Engineering Workshop (SEW-30 2006), pages 253–262. IEEE Computer Society, 2006. (Cited on page 32.)

[Leuschel & Butler 2008] Michael Leuschel and Michael J. Butler. *ProB: an automated analysis toolset for the B method.* Software Tools for Technology Transfer, vol. 10, no. 2, pages 185–203, 2008. (Cited on page 117.)

[Li & Tripunitara 2006] N. Li and M. V. Tripunitara. *Security analysis in role-based access control.* ACM Trans. Inf. Syst. Secur., vol. 9, no. 4, pages 391–420, 2006. (Cited on page 47.)

[Lilius & Paltor 1999] Johan Lilius and Ivan Paltor. *vUML: A Tool for Verifying UML Models.* In ASE, pages 255–258, 1999. (Cited on pages 3 and 4.)

[Martin 2002] Robert Cecil Martin. Uml for java programmers. Prentice Hall, Englewood Cliffs, New Jersey 07632, 2002. (Cited on pages 2 and 3.)

[Morimoto *et al.* 2007] S. Morimoto, S. Shigematsu, Y. Goto and J. Cheng. *Formal verification of security specifications with common criteria.* In Proceedings of the 22nd Annual ACM Symposium on Applied Computing (SAC'07), pages 1506–1512, 2007. (Cited on page 62.)

[Oh *et al.* 2006] S. Oh, R. S. Sandhu and X. Zhang. *An effective role administration model using organization structure.* ACM Trans. Inf. Syst. Secur., vol. 9, no. 2, pages 113–137, 2006. (Cited on page 46.)

[Osborn *et al.* 2000] S. L. Osborn, R. S. Sandhu and Q. Munawer. *Configuring role-based access control to enforce mandatory and discretionary access control policies.* ACM Trans. Inf. Syst. Secur., vol. 3, no. 2, pages 85–106, 2000. (Cited on page 46.)

[Paul 2009] Mano Paul. *The Ten Best Practices for Secure Software Development.* White paper, December 2009. Available online (8 pages). (Cited on page 3.)

[Power *et al.* 2010] D. Power, M. Slaymaker and A. Simpson. *On the modelling and analysis of Amazon Web Services access policies.* In Proceedings of Abstract State Machines, Alloy, B and Z (ABZ 2010), page 394. Springer-Verlag LNCS, volume 5977, 2010. (Cited on pages 61 and 63.)

[Qamar *et al.* 2011a] Nafees Qamar, Yves Ledru and Akram Idani. *Evaluating RBAC Supported Techniques and their Validation and Verification.* In ARES, pages 734–739, 2011. (Cited on page 53.)

[Qamar *et al.* 2011b] Nafees Qamar, Yves Ledru and Akram Idani. *Validation of Security-Design Models Using Z.* In ICFEM, pages 259–274, 2011. (Cited on page 68.)

[Ray *et al.* 2004] I. Ray, N. Li and R. France. *Using UML to visualize role-based access-control constraints.* In Proceedings of the 9th ACM symposium on Access control models and technologies (SACMAT'04), pages 115–124. ACM Press, 2004. (Cited on pages 57 and 59.)

[Richters & Gogolla 2000] Mark Richters and Martin Gogolla. *Validating UML Models and OCL Constraints.* In UML, pages 265–277, 2000. (Cited on page 3.)

[Saaltink 1997] Mark Saaltink. *The Z/EVES System.* In ZUMŠ97: Z Formal Specification Notation, pages 72–85. Springer-Verlag, 1997. (Cited on pages 11, 20 and 22.)

[Sandhu *et al.* 1999] R. S. Sandhu, V. Bhamidipati and Q. Munawer. *The ARBAC97 Model for Role-Based Administration of Roles.* ACM Trans. Inf. Syst. Secur., vol. 2, no. 1, pages 105–135, 1999. (Cited on page 46.)

[Schaad & Moffett 2002] A. Schaad and J. D. Moffett. *A Lightweight Approach to Specification and Analysis of Role-based Access Control Extensions.* In In Proc. of 7th SACMAT, pages 13–22. ACM Press, 2002. (Cited on pages 6, 54, 60 and 61.)

[Schneier 2000] B. Schneier. *Secrets and Lies.* John Wiley & Sons, Inc.„ 2000. (Cited on page 48.)

[Shin & Ahn 2000] Michael E. Shin and Gail-Joon Ahn. *UML-based Representation of Role-Based Access Control.* In WETICE, pages 195–200, 2000. (Cited on page 58.)

[Sohr *et al.* 2005] Karsten Sohr, Gail-Joon Ahn, Martin Gogolla and Lars Migge. *Specification and Validation of Authorisation Constraints Using UML and OCL.* In Sabrina De Capitani di Vimercati, Paul F. Syverson and Dieter Gollmann, editeurs, Computer Security - ESORICS 2005, 10th European Symposium on Research in Computer Security, Milan, Italy, September 12-14, 2005, Proceedings, volume 3679 of *Lecture Notes in Computer Science*, pages 64–79. Springer, 2005. (Cited on pages 6, 49, 57, 58 and 63.)

[Sohr *et al.* 2008] Karsten Sohr, Michael Drouineaud, Gail-Joon Ahn and Martin Gogolla. *Analyzing and Managing Role-Based Access Control Policies.* IEEE

Trans. Knowl. Data Eng., vol. 20, no. 7, pages 924–939, 2008. (Cited on pages 6, 86, 87, 88, 89, 90 and 116.)

[Spivey 1992] J. M. Spivey. *The Z Notation: A reference manual (2nd edition ed.).* Prentice Hall International (UK) Ltd., 1992. (Cited on pages 2, 11, 12, 30, 54 and 59.)

[Strembeck & Neumann 2004] M. Strembeck and G. Neumann. *An integrated approach to engineer and enforce context constraints in RBAC environments.* ACM Trans. Inf. Syst. Secur., vol. 7, no. 3, pages 392–427, 2004. (Cited on page 47.)

[Talhi *et al.* 2009] Chamseddine Talhi, Djedjiga Mouheb, Vitor Lima, Mourad Debbabi, Lingyu Wang and Makan Pourzandi. *Usability of Security Specification Approaches for UML Design: A Survey.* Journal of Object Technology, vol. 8, no. 6, pages 102–122, 2009. (Cited on page 63.)

[Toahchoodee *et al.* 2009] Manachai Toahchoodee, Indrakshi Ray, Kyriakos Anastasakis, Geri Georg and Behzad Bordbar. *Ensuring spatio-temporal access control for real-world applications.* In SACMAT, pages 13–22, 2009. (Cited on pages 3, 60, 61 and 90.)

[Utting 2005] M. Utting. *JAZA: Just Another Z Animator.* 2005. (Cited on pages 20 and 31.)

[Vaziri & Jackson 2000] Mandana Vaziri and Daniel Jackson. *Some Shortcomings of OCL, the Object Constraint Language of UML.* In TOOLS (34), pages 555–562, 2000. (Cited on page 62.)

[Warmer & Kleppe 1998] Jos B. Warmer and Anneke G. Kleppe. The object constraint language: Precise modeling with UML. Addison-Wesley, October 1998. (Cited on page 2.)

[Woodcock *et al.* 2009] J. Woodcock, P. G. Larsen, J. Bicarregui and J. S. Fitzgeraldt. *Formal methods: Practice and experience.* ACM Computing Surveys, vol. 41, no. 4, pages 39–91, 2009. (Cited on pages 4 and 5.)

[Wordsworth 1992] J. Wordsworth. Software development with z : a practical approach to formal methods. Addison-Wesley (1992), 1992. (Cited on page 72.)

[Yu *et al.* 2008] Lijun Yu, Robert B. France and Indrakshi Ray. *Scenario-Based Static Analysis of UML Class Models.* In MoDELS, pages 234–248, 2008. (Cited on pages 54, 57, 58 and 63.)

[Yu *et al.* 2009] L. Yu, R. France, I. Ray and S. Ghosh. *A Rigorous Approach to Uncovering Security Policy Violations in UML Designs.* In Proceedings of the International Conference on Engineering Complex Computer Systems (ICECCS'09). IEEE, 2009. (Cited on pages 54, 57, 58 and 63.)

[Yuan *et al.* 2006] Chunyang Yuan, Yeping He, Jianbo He and Zhouyi Zhou. *A Verifiable Formal Specification for RBAC Model with Constraints of Separation of Duty.* In Inscrypt, pages 196–210, 2006. (Cited on pages 6 and 62.)

[Zao *et al.* 2003] J. Zao, H. Wee, J. Chu and D. Jackson. *RBAC Schema Verification Using Lightweight Formal Model and Constraint Analysis.* 2003. (Cited on pages 6, 54, 60 and 61.)