MODEL-DRIVEN APPROACHES TO ANALYSING TIME- AND LOCATION-DEPENDENT ACCESS CONTROL SPECIFICATIONS

by

EMSAIEB MOSBAH GEEPALLA

A thesis submitted to
The University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Sciences
The University of Birmingham
October 2013

Abstract

Analysis of Access Control systems is an important task to ensure that unauthorised access to critical resources is protected. This thesis deals with a challenging problem related to the analysis of Access Control systems which depend on time and location against undesirable scenarios such as inconsistency. In particular, this thesis first provides formal algebraic notations for the Access Control specifications in the context of a Spatio-Temporal Role Based Access Control (STRBAC) model. This is followed by formulating the terms of inconsistency and semi-consistency in STRBAC specifications, which are accomplished with the help of the formal algebraic notations.

In order to analyse STRBAC specifications to detect inconsistencies and semi-consistencies, this thesis utilises Alloy and Timed Automata. A key challenge is how to automatically generate analysable formalisation such as Alloy and Timed Automata from the specifications. This thesis employs Model-Driven Architecture (MDA) technology to automate the transformation of the STRBAC model to Alloy as well as to Timed Automata and Timed Computation Tree Logic (TCTL). This is accomplished by defining one set of transformation rules for mapping STRBAC features to Alloy features and another set for mapping the features of the STRBAC model to Timed Automata and TCTL features. Details of how we implement model transformation in the SiTra transformation engine are also presented and described with the help of a case study. In addition, we present a comparative study between Alloy and Timed Automata from capability and performance points of view, following which we demonstrate that current Access Control models are not adequate for representing Physical Access Control (PAC) specifications and then discuss some of the limitations of the current models, which we highlight by conducting a case study involving the modelling of an Access Control mechanism used by British Telecom (BT). To overcome such limitations, we present an extension of the STRBAC model which considers the physical aspects of Access Control systems.

ACKNOWLEDGEMENTS

I would like to express my special gratitude to my advisor, Dr. Behzad Bordbar, for giving me the opportunity to be his Ph.D. student, and for his endless patience and constant encouragement whenever I was in doubt during my Ph.D. research. His invaluable guidance, critical feedback, and active involvement have made a significant contribution to the quality and presentation of my work. The experience to work with Dr. Behzad has been precious, and this experience will be the most useful and important for my future.

I also owe a special note of thank to my thesis group members Dr. Richard Dearden and Dr. Rami Bahsoon, for taking the time to give me constructive feedback on my work. Their constructive comments, enthusiastic help, diverse suggestions during the presentation of my preliminary results considerably enhance my final defence presentation. I would like to express special thanks to Professor Kozo Okano and Dr. Xiaofeng Du for fruitful research collaborations.

My thanks also to all faculty members in Computer Science department, for teaching me diverse computing knowledge and helping on my research capability growth. I thank the entire Computer Science Department staff for always smiling when they help me to complete paperwork. I also owe special gratitude to all friends at the University of Birmingham, for being my family and their continual encouragement and help.

Last, but not least, I would like to thank my parents, brothers and sisters for their continuous support, inspiration, encouragement, and for being patient with living abroad. I am especially indebted to my wife, for her unconditional love and support, who also makes my life better and complete, especially in the last two year. Her inimitable way of support is what keep me overcoming all the difficulties during my living in UK.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Introduction

Information is considered a highly valuable asset for most businesses, and in today's extremely competitive business environment, information assets need to be protected from unauthenticated access. The disclosure of highly sensitive information about an organisation's consumers, strategic plans or products to a competitor could lead to a huge financial losses, loss of reputation and legal liability [29]. Additionally, this information could provide competitors with the opportunity to leapfrog the organisation [2, 29] in terms of research and development advancement, as they would not need to incur the financial and time burdens involved in this key area. Furthermore, they would also have the opportunity to evolve counter-strategies to an organisation's plans before such strategies were even implemented. As such, the disclosure of critical information is almost impossible to recover from [2, 29]. Therefore, organisations have evolved various procedures and systems – often known as *information security* – aimed at protecting information assets from both internal and external threats [1]. The main security goal of information security is to ensure the confidentiality, integrity, availability and assurance of information assets [29, 61].

- *Confidentiality*: Privacy or the ability to control or restrict access so that only authorised users can access information resources [29, 61]. Several approaches can

be used to achieve this goal, such as Access Control.

- *Integrity*: Ensures that information is accurate and reliable and has not been subtly changed or tampered with by an unauthorized party [29, 61].

- *Availability*: The ability of authorised users to access information resources when they need or request it [29, 61].

- *Assurance*: Assurance is the degree of confidence in the security of the system with respect to predefined security goals [61].

To summarise, the objective of information security is to deny information resource access to unauthorised users whilst making it available to authorised users. This should be done in a manner that does not adversely affect business operations [29, 31], so companies invest heavily in this area. One of the technologies used to achieve this aim is *Access Control systems*.

An Access Control system is one of the key stages in computer security, as it provides the means of controlling which authority have access to which resources, as well as the nature of such access [4, 5]. By making information resources available to authorised users only, the mechanism ensures that only certain levels of data are available to certain levels of user. Many models have been developed to construct and manage Access Control systems that can be deployed by organisations to meet their information security needs, such as Mandatory Access Control (MAC) [7], Discretionary Access Control (DAC) [8, 9] and Role Based Access Control (RBAC) [3, 5, 10, 11].

Among these models, the RBAC model is receiving increasing attention as a generalised approach to Access Control [3, 5, 10, 11]. A study carried out by NIST [3] shows that in many organisations the Access Control decision is based on the user's role and responsibilities within the organisation, making the RBAC approach a perfect fit for expressing security requirements. Clark et al. [60] demonstrate that the more traditional MAC and DAC models do not sufficiently address the various security requirements of many organisations; however, the RBAC model can significantly simplify security admin-

istration. For instance, if a person moves to a new job within the organisation, then he/she can only be assigned to the new role and removed from the earlier role, while in the absence of an RBAC model, his/her old a privilege should be cancelled and a new privilege granted. In the RBAC model, roles can be structured into hierarchies to reflect organisational functional hierarchies. Role hierarchy means that a senior role in an organisation will inherit the permissions of a junior role, and it can considerably decrease explicit permission assignments to a role and therefore significantly reduce administration overheads. Another important feature of the RBAC model is that it allows for expressing a variety of separation of duty constraints, which are beneficial in many applications and help to reduce the risk of allowing a person conflicting roles or assigning a role that results in conflicting permissions. Moreover, the RBAC model is policy-neutral [3, 11]. More specifically, by configuring a role-based system appropriately, one can support various policies, including both MAC and DAC policies [5]. Such flexibility within the RBAC model is very important, as it can be adapted to support the Access Control requirements of enterprise-wide security administration and enforcement policies.

With increases in the growth of wireless networks, mobile devices and other technologies involved in the remote accessing of resources, we are moving towards an era where contextual information such as spatial and temporal information will be essential for Access Control [12, 56, 58]. For instance, a part-time PhD student in a university may be authorised to access the university's electronic library but only from the campus and during a specific period of time (i.e. January, February and March) every year. If a part-time PhD student is represented by a role, enforcing such rules requires that the student assumes the role in that time interval and on the university campus only. This role may be restricted further to only pre-specified days and hours during the three months. Traditional Access Control models, such as the RBAC model, cannot provide such Spatio-Temporal based Access Control mechanisms, so they need to be augmented in order to provide this function.

In order to support Spatio-Temporal based Access Control, Ray and Toahchoodee [12, 13] and other researchers [56, 57, 58] have proposed several Spatio-Temporal RBAC models as extensions of the standard RBAC. The Spatio-Temporal Role Based Access Control (STRBAC) model [12] is one such example and enhances the traditional RBAC model by incorporating time and location conditions with RBAC entities, relationships and constraints. In this model, users are assigned to roles based on time and location constraints, while permissions are also assigned to roles on this basis. In addition, the hierarchy of roles, the separation of duty and cardinality constraints are time and location dependent. Formalisation of the STRBAC model is presented in this thesis with the help of formal algebraic notations.

Incorporating the traditional RBAC model with both time and location information increases the complexity of Access Control models even further. As a consequence, this increases the possibility of contradictory statements in Access Control specification. Such statements are commonly known as inconsistencies. A formal definition of the term inconsistency in Access Control specifications is presented in this thesis. The existence of inconsistency can be caused by an error in the specification resulting in incorrect system implementation. On the other hand, it could be that system stakeholders are imposing conflicting demands. In both cases, however, it is crucial that these demands are resolved prior to the development of the system. Therefore, it is necessary to analyse the specifications of such systems prior to their implementation; however, due to the complexity and size of modern systems, discovering such inconsistencies is a formidable task and cannot be carried out manually.

Another most important issue with the STRBAC model that we have encountered is that, sometimes the STRBAC specification is consistent; however a minor change to the specification by choosing an unsuitable allocation of the users to roles makes the specification inconsistent. We refer to these as semi-consistency in the specification and a formal definition of semi-consistency is provided in this thesis [16]. A semi-consistency is a special case where the inconsistency can be avoided if the assignment of user to role is

controlled. If we can identify such scenarios, then we can adopt pre-emptive measures by providing certain constraints. This is particularly important, as these scenarios could pose dangerous security issues which could actually cause a huge loose to the organisations. It is therefore essential to perform an analysis of STRBAC models in order to identify any semi-consistencies in the specifications.

Currently, Alloy [20, 21, 22] and Timed Automata [23, 24] are used widely for modelling and analysing Access Control specifications because both are supported by automatic tools which are capable of checking a sufficient number of constraints to detect conflicts and inconsistencies. For example, Alloy is supported by the Alloy analyser tool, which is an automated constraint solver that transforms Alloy code into Boolean expressions, thus providing analysis through embedded SAT solvers, whereas Timed Automata is supported by the Uppaal model checker [25], which allows for the verification of properties that are expressed in Uppaal Requirement Specification Language. This language is a subset of Timed Computation Tree Logic (TCTL) where primitive expressions are location names, variables and clocks from the modelled system.

The main thesis of this research is to use Alloy and Timed Automata methods to specify and analyse Access Control specifications in the context of STRBAC and then to conduct a comparative study between the two methods from capability and performance points of view. The comparison is based on the case study of a SECURE bank system taken from [12].

Typically, the process of transformation between Access Control specifications and formal methods such as Alloy or Timed Automata has been done manually [15, 80]. This process, however, is prone to human error and is exceptionally time-consuming. There is also the problem of scoping as system grows in size, whereby the manual generation of formal models becomes extremely difficult, if not impossible. It is therefore essential to have an automated tool that will extract a formal model from the system's specifications for formal analysis. The Object Management Group's Model-Driven Architecture (MDA) methodology makes it possible to generate a model from another model [26]. Therefore,

in this thesis, we propose two MDA transformations to automate the transformation between the STRBAC specifications and Alloy and Timed Automata for the purpose of analysis. More precisely, the first set of transformation resulting in a tool, AC2Alloy [27], which automates the transformation between the STRBAC specifications and Alloy, thus allowing for powerful analysis to be carried out via the Alloy analyser tool, whereas the second set of transformation resulting in a tool called AC2Uppaal [28], which transforms the STRBAC specifications into Timed Automata and TCTL [88] statements, following which the produced Timed Automata network and TCTL statements are modelled and verified using the Uppaal model checker. The proposed methods can be used in the early phases of Access Control system development to ensure the consistency of the specifications. We evaluate our approaches by using the case study of a SECURE bank system taken from [12].

Although the STRBAC model is very useful for providing a high-level description of Access Control, especially in Cyber Access Control (CAC) systems, when time and location information is required to grant or deny access to resources, it does not provide a complete solution for all Access Control issues. This is because existing Access Control models, including the STRBAC model, are not adequate enough to represent the physical aspects of Access Control systems. One of the more serious limitations of existing Access Control models that we have come across when trying to model a Physical Access Control (PAC) mechanism used by British Telecom (BT) is the representation of locations. Current models deal with logical locations, which may not be suitable for PAC specifications. For instance, this thesis will argue that a logical location and a physical location are different, in particular when dealing with hierarchies. Therefore, extending current models in order to overcome such limitations is very important, because such an extension will assist system designers to create the correct Access Control specifications for supporting the physical aspect of Access Control. As a result, this thesis proposes a new extension to Spatio-Temporal Role Based Access Control for physical systems [30] to overcome such limitations. This involves introducing a graph capturing physical access to

geographical locations within building. This approach is described through a case study, which is provided by our industrial partner British Telecom (BT). An overview of the proposed approach is presented in the next section.

1.2 Overview of our Approach

Figure 1.1 depicts an outline of our approach to developing two model-driven solutions that could transform Access Control specifications in the context of STRBAC into formal languages for the purpose of analysis. The first approach transforms the STRBAC specifications into Alloy, while the second approach transforms them into Timed Automata and Timed Computation Tree Logic (TCTL) statements. In particular, in order to develop the first MDA transformation the STRBAC metamodel was retrieved from the specifications of the STRBAC. Moreover, a subset of Alloy metamodel was developed. To conduct model transformation, a set of transformation rules was defined, which map various elements of the source metamodel into the elements of the destination metamodel. Using this approach, every STRBAC model that is an instance of a STRBAC metamodel is automatically transformed into an Alloy model, which is an instance of Alloy metamodel. Similarly, in order to develop the second MDA transformation, Timed Automata and TCTL metamodels were developed. A number of transformation rules were defined to map the elements of the STRBAC metamodel to these Timed Automata and TCTL metamodels. Using this method, every STRBAC model that matches the STRBAC metamodel is automatically transformed into Timed Automata network and TCTL statements which are instances of Timed Automata and TCTL metamodels.

A detailed account of the contributions of this thesis is presented in the following section.

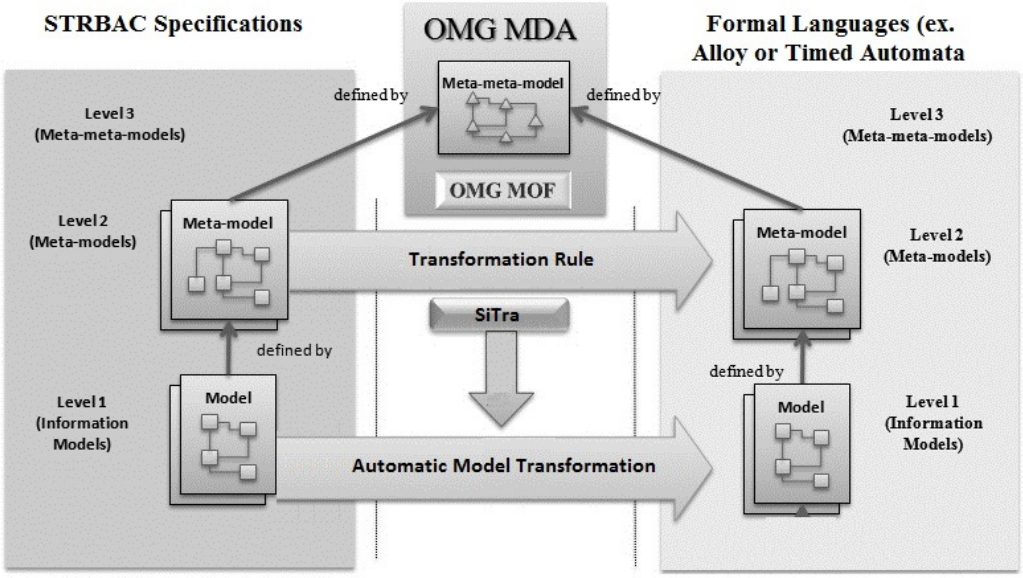

Figure 1.1: Overview of our Approach

1.3 Contributions of the Thesis

The contributions of this thesis are as follows:

- Formalisation of the STRBAC model. In particular:

  1. Formal algebraic notations for the STRBAC model to specify its components are provided.
  2. A formal definition of inconsistency in STRBAC specifications is provided, several examples of which are presented.
  3. The concept of semi-consistency in STRBAC specifications is introduced, and different scenarios that may cause this state are introduced.

- A model transformation framework (AC2Alloy) is developed and studied. In particular:

  1. A subset of the STRBAC metamodel, which is expressive enough to model basic components of the STRBAC model, such as Users, Roles, Permissions, Times and Locations and their interrelations (such as User Role Assignment), was identified.

2. The transformation rules from the STRBAC metamodel elements into the Alloy metamodel elements are defined.

3. The mapping rules from STRBAC to Alloy are implemented using the model driven architecture technique SiTra.

4. The transformation rules are implemented as an eclipse plug-in called AC2Alloy.

- The AC2Alloy model transformation is evaluated using a case study.

- A model transformation framework (AC2Uppaal) is developed. In particular:

  1. Transformation rules from the STRBAC metamodel elements to Timed Automata and TCTL metamodels elements are defined.

  2. Mapping rules from STRBAC to Timed Automata and TCTL quires are implemented using the model-driven architecture technique SiTra.

  3. Transformation rules are implemented as an eclipse plug-in called AC2Uppaal.

- The AC2Uppaal model transformation is evaluated using a case study.

- A comparison study between Alloy and Timed Automata from capability and performance points of view is presented.

- The current STRBAC model is extended to cover the physical aspect of Access Control systems.

- A case study provided by our industrial partner British Telecom (BT) is used to evaluate and demonstrate the feasibility of the presented approach.

1.4 Dissertation Structure

The rest of the dissertation is organised as follows.

- Chapter 2 introduces the reader to preliminary concepts, such as Access Control models, Alloy, Timed Automata and model driver architecture. Moreover, a review of the existing literature on the analysis of Access Control specifications is discussed.

- Chapter 3 presents formal algebraic notations for the Spatio-Temporal Role Based Access Control (STRBAC) model. In addition, it provides a formal definition of inconsistencies and semi-consistencies as well as several example that may cause these states in STRBAC specifications.

- In Chapter 4 we present the rules for transforming from STRBAC to Alloy and describe how the STRBAC model can be transformed into Alloy in order to enable automatic verification via the Alloy analyser.

- Chapter 5 describes how the transformation rules are implemented.

- Chapter 6 presents a AC2Uppaal model transformation which automates the transformation between STRBAC specifications and Timed Automata as well as TCTL. In particular, this chapter describes how the STRBAC model can be transformed into Timed Automata and TCTL statements to enable automatic verification via the Uppaal model checker. Moreover, it describes how to implement the transformation rules.

- Chapter 7 presents a comparison between Alloy and Timed Automata from capability and performance points of view.

- Chapter 8 discusses the limitations of the STRBAC model for physical Access Control systems by using a PAC case study provided by BT. In order to overcome the limitations of the STRBAC model this chapter presents our efforts to extend it to cover the physical aspect of Access Control. Finally, this chapter makes use of our AC2Alloy model transformation to analyse the specifications of physical Access Control.

- Chapter 9 concludes the dissertation and presents a discussion on proposals for future directions.

1.5 List of Publications

The following publications are a result of the research presented in this thesis:

- Emsaieb Geepalla, Behzad Bordbar, and Joel Last. Transformation of spatio-temporal role based access control specification to alloy. In Model and Data Engineering, pp. 67-78. Lecture Notes in Computer Science, Springer, 2012.

- Emsaieb Geepalla and Behzad Bordbar. On formalizing of Inconsistency and Semi-consistency in Spatio-Temporal Access Control. In the Seventh IEEE International Conference on Information Management (ICDIM), page 22-29. IEEE 2012.

- Emsaieb Geepalla, Behzad Bordbar, and Kozo Okano. Verification of Spatio-Temporal Role Based Access Control using Timed Automata." In Networked Embedded Systems for Every Application (NESEA), 2012 IEEE 3rd International Conference on, pp. 1-6. IEEE, 2012.

- Emsaieb Geepalla and Behzad Bordbar. An Automated Approach to Detect Inconsistency and Semi-consistency Spatio-Temporal Role Based Access Control Specification." Journal of Data Processing Volume 2, no. 3 (2012).

- Emsaieb Geepalla, Behzad Bordbar, Kozo Okano and Hamed Seyedali. AC2Uppaal: A Tool for automatic Generation of Timed Automata form Spatio-Temporal Role Based Access Control Specifications. To appear in Proceeding of World Congress of Internet Security (WorldCIS) 2013. IEEE, 2013.

- Emsaieb Geepalla, Behzad Bordbar, and Xiaofeng Du. Spatio-Temporal Role Based Access Control for Physical Access Control Systems. In Proceeding of the Fourth International Conference on Emerging Security Technologies (EST) 2013. IEEE, 2013.

CHAPTER 2

BACKGROUND MATERIAL AND RELATED WORK

In this chapter, we provide an overview of the relevant work categorised by the areas of our research. In particular, we present a background to Access Control models and the analysis of Access Control specifications using formal methods such as Alloy and Timed Automata. In addition, it provides a brief description of Alloy and Timed Automata as well as Model-Driven Architecture (MDA).

2.1 Access Control Model

With recent advances in mobile computing, wireless networks and other technologies involved in the remote accessing of resources have prompted an urgent need for the creation of an Access Control model which takes into consideration the location of the user and the time of access. Such information is essential for controlling various spatio-temporally sensitive applications in organisations which rely on the Access Control mechanism. Multiple Access Control models are investigated and discussed in order to understand the problem as well as to ensure that the Spatio-Temporal Role Based Access Control (STRBAC) model is the most appropriate choice for this thesis.

2.1.1 Discretionary Access Control (DAC)

The term 'Discretionary Access Control' (DAC) [8, 9, 32, 33] refers to access that allows users to alter the features of the object as well as to specify whether the object is accessible to other users. In simple terms, a DAC could be a password file or specify whether a particular file requires the knowledge and acceptance of the administrator while being accessed by other users. The DAC mechanism has a basic inherent weakness, though, in that it fails to recognise the fundamental differences between a human and computer program users. Access controls are determined by the owner of the file or other resources. The policy authorises the owner to choose which user has access to files, as well as what privileges can be accessed by the user. The merit of DAC is its capability to allow users access to data; however, the problem lies in its integrity. It is more fundamental to share the information rather than to protect it.

DAC is working in the centralised level and also in the distributed level. The Centralised levels are those when the administrator can access the user Data and other information. The changes required accessing the data through the department. The disadvantages of centralised level are in the large organisation because it is very overwhelming especially the administrators are outsourced. But in the distributed level it is good in the large organisation because it has distributed levels, those that allow a known person to access the data and other information service. It may be any single member for instance, manger or team leader. The main advantage of the distributed level is that the delay can be avoided when the administrator of account is away from the large area [33]. Following are the concepts of Discretionary Access Control.

- Files and Data Ownership

  In a system every object has an owner. The owner of the resources including files, data, system resources and devices determines the access policy. So we can say that an object without an owner is unprotected. This means that owner of the resources is the person who creates the resources.

- Access rights and permissions

  In this policy control an owner can assign access right and permissions to the individual user or group for specific resources.

2.1.2 Mandatory Access Control Policies (MAC)

The Mandatory Access Control (MAC) model was formalised by Bell and La-Padula [7]. In comparison to the DAC, an MAC attempts to regulate users as well as process resources pertaining to (higher level) security policy in any given organisation. The policy encompasses rules and specifications which determine the type of access that is allowed for each individual system. The system policy is highly relevant to MAC, in that it is similar to the firewall ruling related firewalls. In the MAC mechanism, a user has to be authorised in order to access an object. In the following, let us consider some examples in order to develop a clear understanding of MAC.

In a MAC model, the system is the policy determinant rather than the owner. It involves a multilevel system, which is capable of processing highly sensitive data, such as government and military information, which implies that the Access Control policy decision is far from the Access Control of individual objects. The central authority decides which information has to be accessed and by which system, as well as by whom. If a user is changed, it does not imply that system access will also change, as the main premise is based on the MAC labelling each object, i.e. each object is protected by the MAC. MAC security conditions imply that objects and applications are labelled by the system, and the label applies to the entire object and provides access protection. The function pertains to the protection of confidential and secret data; however, in the enterprise level it is very difficult to classify. Therefore, MAC is not suitable in Enterprise level [7].

Consider the example of a doctor, a patient, a secretary and a nurse. Confidential data may encompass the patient's name as well as an appointment time. The doctor's secretary may be authorised to update or change the appointment time, but it is beyond the authority of the nurse to do so. The nurse, however, is authorised to change any relevant

information that must remain confidential to the outside world, which could include minor information such as blood pressure or glucose levels as well as more important information relating to a disease or health disorder. The doctor is authorised to change the blood pressure or glucose levels as well as any other information that is relevant and confidential. However, it is not necessary for the doctor to have access to confidential information such as name and appointment times that pertain to the sensitive level. The example clearly defines variables as well as the functioning of information flow in the concept of data, and moreover information access. The hierarchical structure is given and implemented in the MAC mechanism, which implies that the access applies to accessing information that is secret and confidential. The advantage pertains to the data that is present in the security label, and therefore the entire object is protected, thus negating any chance of being accessed by unknown and/or unauthorised users [7]. The concepts behind MAC are discussed below.

- Sensitive labels: these labels are assigned to all subjects and objects in the MAC system. Subject sensitivity: determines the level of trust for the object so that, in order to access the object, the subject must possess a trust level that is requested or higher than requested [7].

- Data import and export: directs and monitors the flow of information from one system to another [7].

- Proper labelling: the entire functioning of MAC security is based on the proper labelling of the object, so as to ensure that confidential information is protected at all times. There are two methods for prompt labelling, which are discussed below:

    1. Rule-Based Access Control: This type of control defines specific constraints for accessing a requested object. All MAC-based systems implement a simple form of rule-based Access Control which determines whether access should be granted or denied via a subject- or object-sensitive label [3].

2. Lattice-Based Access Controls: A very complex Access Control decision which involves many objects and/or subjects [35].

2.1.3 Role Based Access Control Policies (RBAC)

Role Based Access Control (RBAC) [3, 5, 10, 11, 37] is another Access Control mechanism. It was introduced more recently than the MAC and DAC and is based on control models that cope with novel and altering resource hosts and users. RBAC involves simplifying several policy aspects as well as policy administration.

As per the National Institute of Standard and Technology, a project – the RBAC project – has been initiated with the purpose of designing a standardised Access Control model that will benefit the design of the security level and also be self-system dependent. The results of the implementation have been positive. The project was introduced by Dived Ferriolo and Rick Khun, who attempted to meet the requirements and scope of the RBAC solution in [3].

The key ideas behind the RBAC pertain to permissions related to the roles as well as users assigned to each role. The functioning simplifies the management of roles as well as users throughout the system. Roles pertain to undertaking several functions that influence alignment as well as the users that are assigned to each role related to the function. These roles are based on the qualifications and responsibilities of the user. The authority of the user encompasses several functions, as they are often moved from one role to another. Moreover, these roles and responsibilities represent a brand-new application and system requiring such permissions [5].

RBAC involves controlling the roles and responsibilities of individual users in an organisation. Roles are defined so as to analyse the goals and structure of an organisation. For example, in a medical organisation, there are nurses, doctors and patients, whereas in a bank there are managers, directors and accountants. These types of people undertake several roles and responsibilities whilst working for an organisation [11]. The approach, which involves protecting the system from unauthorised access, is newer than Manda-

tory Access Control (MAC) and Discretionary Access Control (DAC), both of which have already been discussed in the chapter.

The approach of RBAC is different from the Access Control Lists (ACLs) that are used in DAC systems, in which ACLs assign permissions to particular operations that relate to overall organisational functioning, contrary to low-level data objects. For instance, an ACL is authorised to grant or deny the access of a user to a particular file system, but it does not determine the ways in which that particular file can be modified [37].

For RBAC permissions, several roles and responsibilities have to be assigned in order to obtain access. A permission determines the operation that has to be carried out, as well as on what resource and to what level of Access Control. It is mandatory for a user to activate a set of roles and responsibilities pertaining to him as an individual. Each user has multiple sessions relating to their roles and responsibilities; however, each role may have one authorised user. RBACs support hierarchies very well and are good at defining an inherited relationship between roles and responsibilities pertaining to a user. To prevent any conflicts of interest within a business entity, RBACs allow for the separation of static and dynamic duties. An overview of RBAC components is depicted in Figure 2.1.

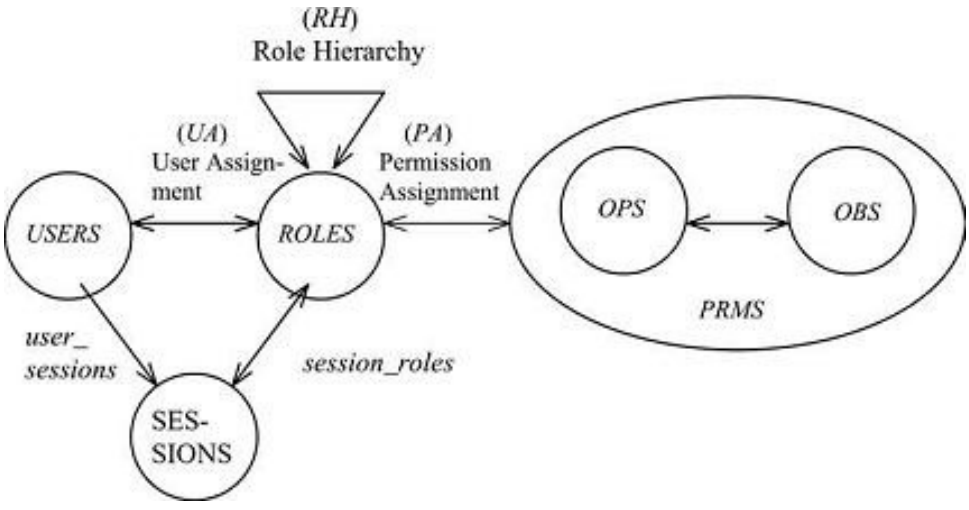

Figure 2.1: Role Based Access Control Model [3]

The RBAC approach is highly beneficial [3, 5, 6, 37, 55] and includes the following variables:

- *Group Objects*: The RBAC approach provides user classification as per the execution of activities. Similarly, it provides classification for objects, which are generally categorised by type (letters, manuals) or the respective area of their application (commercial, advertising letters). Furthermore, authorisation for access should be based on the classification of objects rather than on specific objects. For instance, a secretary could be given the role of reading and writing data, instead of authorising every single letter in the data. The approach leads to a better and more controlled authorisation regime. Furthermore, authorised access varies according to the object, without needing to specify authorisations pertaining to each object [37].

- *Security Management*: The RBAC model determines the authorisation of users by dividing the task into two further components – one that assigns roles and the other that assigns permissions to an object. This mechanism leads to greater simplification in security management. For example, when considering changing a user's responsibility level, most likely due to a promotion, the current role of the user could be substituted for new roles in addition to new responsibilities. Therefore, the changes would be concurrent with the authorised permission premise and control access to relevant and secret data. However, organisational hierarchies do not change frequently, and neither does assigning permissions to roles or management tasks [37].

- *Data Abstraction*: In spite of the provision of permissions for reading, writing or executing, as commanded by the operating system, RBAC can establish abstract permissions that authorise the credit and debit for a given object.

- *Policy Neutrality*: RBAC is policy-neutral, which means that it can be configured to model the specifications of other Access Controls, e.g. MAC policies, in which the system administrator maintains the access matrix or DAC policies and users create and update security policies for their devices [38, 39, 40, 41].

- *Supports Role Hierarchy*: Organisational structures involve lines of authority that

are referred to as a 'hierarchy'. The organisational structure encompasses the layout reflected in the features of RBAC [37]; role hierarchy pertains relations between the roles. The higher roles are at the top of the hierarchy, and therefore they are referred to as senior roles, whereas the lower roles are classed as junior roles. The major rationale behind RBAC relates to role management, in that senior roles may lead to junior role authorisation and other relevant functions – they both depend on the nature of the hierarchy as well as on each other. There is no need to assign permissions separately to users on the same hierarchical level.

- *Least Privilege*: Guarantees that a user should only be given the privileges required to complete the job [37]. Ensuring least privilege requires identifying what the user's job is, determining the minimum set of privileges required to perform that job, and restricting the user to a domain with those privileges and nothing more.

- *Supports Separation of Duties*: Separation of duties protects the organisation from fraudulent user activity [42] and relate to the invocation of mutually exclusive roles relating to a sensitive task [42]. Moreover, it makes fraud very difficult to achieve.

2.1.4 Context-Aware Role Based Access Control Model

With the colossal usage of wireless networking, as well as mobile and sensor devices, researchers have endeavoured to extend the RBAC to recognise the context information that supports ubiquitous computing applications.

The concept of environmental roles was introduced by Covington et al. [44], who introduced it in a generalised model of (GBRAC) to aid in Access Control relating to private information and resources in ubiquitous computing applications. Environmental roles in this model are entirely different from those specified in RBAC. However, there are similar properties such as the separation of duty, role hierarchy and role activation. In the Access Control framework, each element that has been assigned access permission has a set of environmental roles underneath. These roles are activated as per any changing

conditions specified in the environmental conditions. Furthermore, in a subsequent work [45], the authors also specify the Context-Aware Security Architecture (CASA), which is an implementation and application of the GRBAC model. Security services are provided through Access Control in the architecture, whereby policies are expressed through the execution of security management for roles. The environmental role activation pertains to managing environmental role activation and deactivation, as per the context-management variables.

As depicted by Chakraborthy et al. [46], another trust-based authorisation model is 'TrustBAC', which has its roots in the RBAC hierarchical framework. The model has a feature whereby the user can activate a permission and access data in relation to the level of trust he/she has been allocated for the system. The user's trust level is calculated for each individual user and is based on three factors: behaviour, knowledge and recommendations by other users. The model also discusses the concept of trust dominance, which is the same as inheritance hierarchy, although it does not account for the separation of duty. In a similar approach, Ya-Jun [47] proposed Trust-based Access Control (TBAC), based on the RBAC premise, in an extended form. It is useful for roles where users are unknown and access privileges depend upon the level of trust set by contextual information.

2.1.5 Temporal Aware Role Based Access Control

There are several other RBAC extensions, including the Temporal Role Based Access Control Model (TRBAC) proposed by Bertino et al. [48]. The extension encompasses a time dimension with all features that correspond to the RBAC framework. Bertino et al. introduced the concept of role enabling and role disabling for a user. The model determines the time at which a role can be enabled or disabled. This work was further extended by Joshi et al. [49, 50, 51, 52] in what is known as the Generalized Temporal Role Based Access Control Model (GTRBAC). The authors introduced two types of temporal hierarchies, namely the permission inheritance hierarchy, where seniors are authorised to

inherit the permission assigned to a junior role, and role activation, where a senior is authorised to assign a role to a junior. The author further proposes Separation of Duty (SoD), which involves two forms – weak and strong. The weak form considers that two roles can be assigned at the same time, whereas the strong form works as a non-temporal RBAC, in that two conflicting roles can be assigned to the same user at the same time.

2.1.6 Spatial Aware Role Based Access Control

Researchers have attempted to extend the standard RBAC in order to incorporate the spatial information [53, 54]. As proposed by Bertino, the GEO-RBAC incorporate the standard RBAC model with spatial information. The GEO-RBAC model supports spatial hierarchy of roles and it is based on the location of an individual user [53]. Therefore, a senior role is authorised with permissions that pertain to junior roles. However, this work does not discuss the impact of spatial information on the separation of duties.

As depicted by Ray et al. [54], the components of RBAC are influenced by the location of the user. The authors depict the model whilst using Z language specifications. Similarly, the paper does not address role hierarchy or the separation of duties and does not discuss the impact of location on time.

2.1.7 Spatio-Temporal Aware Role Based Access Control

Incorporating the standard RBAC model with spatial-temporal information has been addressed by several works [56, 57]. Tuyen Le Thi et al. [56] propose a Spatial Temporal Role Based Access Control (STRoBAC) model that can support both spatial and temporal aspects and other contextual conditions as well as Access Control based on the role of the subject. However, this study does not discuss the impact of time and location constraints on entities of the RBAC model, such as User Role Assignment and Permission Role Acquire. Another attempt on this subject was presented by Chen and Crampton in [57], where the authors outlined a graph-based representation of a Spatio-Temporal RBAC.

In this study, all of the elements of the RBAC model are represented by vertices, while assignment relationships and hierarchical constraints are represented by the edges of the graph. The authors categorised the model into three types: standard, weak and strong models. For the standard model, element v_1 can only be authorised to element v_n, if all vertices along the authorisation path satisfy time and location conditions. In the weak model, however, an element v_1 can be authorised to the element v_n, if both vertices satisfy time and location conditions. In the strong model, the element v_1 can be authorised to the element v_n, but only if all vertices, together with the edges along the authorisation path, satisfy time and location conditions. Even though the model presented in this study has well-defined semantics, it does not consider all the elements of the RBAC model, such as separation of duty constraints. In addition, it does not consider the time and location properties of the object before granting or denying access.

In [58], Chndrn et al. propose a new model which combines the main components of the GTRBAC model and the GEO-RBAC model. In the new model, roles are enabled for users if the users satisfy certain time conditions. In addition, a user can activate a role if the role is enabled to the same user and the user satisfies the location constraints related to role conviction. However, the proposed model still has some limitations. For example, the assignment between roles is not based on spatial or temporal conditions. There are many motivating scenarios which indicate that user-to-role assignment should be based on time and location constraints. Another example of these limitations is that the impacts of spatial-temporal information on the separation of duties and the role hierarchy is not addressed. Studying the impacts of time and location on role hierarchy and separation of duty can be useful for real-world applications.

Another approach for integrating the RBAC framework with both time and location was proposed by Samuel et al. in [59]. Samuel's Generalized Spatio-Temporal Role Based Access Control (GST-RBAC) model incorporates the existing GTRBAC model with location constraints. In this study, the authors show the capability of the new framework GST-RBAC to express some real-world constraints that are not possible in the

existing GTRBAC model. Nevertheless, the GST-RBAC model still has some limitations that should be overcome before being able to specify all the security requirements for real-world applications. One of the limitations of the GST-RBAC model is that assignments between roles and permissions are not based on time and location conditions. For instance, when a user activates a role, all of the permissions assigned to the role can be accessed. However, this might not be sufficient for real-world examples such as that described in the summary of Chndrn's work. Another example of limitations is that Samuel's model does not integrate time constraints into the separation of duty. Two conflicting roles are allowed to activate if the user is in a different location during the same time period when both roles are enabled. This might not be adequate for real-world applications; for instance, if a user can activate a Teaching Assistant role in a school office, then the same user should not be able to activate a Lab Operator role anywhere else during the same period of time.

In order to overcome the limitations of the previous models, Ray et al. [12, 13] proposed a Spatio-Temporal Role Based Access Control (STRBAC) model, which is an extension of the traditional RBAC. The STRBAC model incorporates all the features of the RBAC model, namely user role assignment, permission role acquire and role hierarchy, along with time and location constraints. The new framework also introduces new features such as separation of duty over roles, cardinality constraints over roles, cardinality constraints over permissions, role delegation and permission delegation. In addition, the authors discuss the impact of time and location constraints on all the newly introduced features. This model has been used successfully to express many real-world constraints, such as the Dengue Decision Support (DDS) system presented in [66]. Therefore, the STRBAC can be considered a best choice for model Access Control specifications, especially when time and location information are required. As a result, this thesis makes use of the STRBAC model. In the next section we will describe a simplified version of the STRBAC model for the purpose of this thesis. For more details about the STRBAC model we refer the reader to [12, 13, 14, 15, 64, 66].

2.1.7.1 Spatio-Temporal Role Based Access Control Model

The STRBAC model is an extension of the original RBAC model [3] that supports temporal and location constraints. An advanced Access Control framework was introduced in 2007 [12]. The STRBAC model is very useful for providing a high level of description regarding Access Control, especially when time and location information is required to grant or deny access to certain resources. Such information is essential for controlling various Spatio-Temporal sensitive applications which rely on the Access Control mechanism. For example, access to some resources in an organisation could be permissible but only at a specific time and at a specific location. STRBAC does not provide a complete solution for all Access Control issues such as the issues related the the Physical Access Control (PAC) systems which will be discussed in Chapter 8 of this thesis, but with its rich specification it has proven to be cost-effective by reducing the complexity inherent in the authorisation management of data when time information is required to grant or deny access to the data, especially for Cyber Access Control (CAC) systems.

This thesis uses a simplified version of the STRBAC model defined in [12]. For example, some elements, such as user role activation and delegation, are not considered in this thesis. It is however possible to extend our approach to include such elements. This simplified STRBAC model is defined in terms of seven model components:

- Core Spatio-Temporal Role Based Access Control (STRBAC).

  The Core STRBAC model describes the basic concepts for the STRBAC, as shown in Figure 2.2. These basic concepts are:

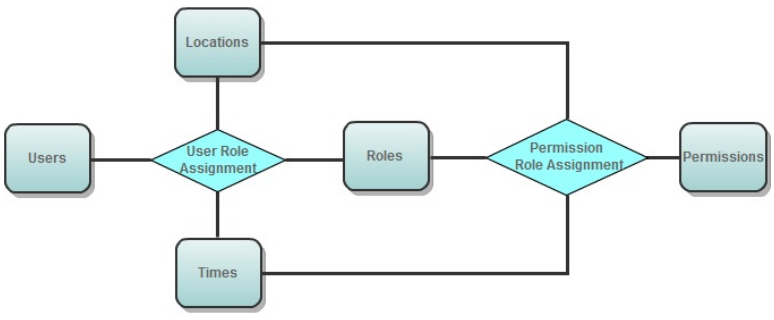

Figure 2.2: The Basic Concepts of the STRBAC Model

1. Users (U)

   A user is a human being or an autonomous agent.

2. Roles (R)

   A role is a job function within an organisation with some associated permissions to access protected resources.

3. Permissions (P)

   A permission is the ability to perform a corresponding task on protected resources.

4. Times (T)

   A time can be represented as a set of time instances on the time line. For example, a day can be divided into two categories – day time and night time.

5. Locations (L)

   A location is represented as a set of logical entities.

6. User Role Assignment (URA)

   In any organisation, roles are assigned to users based on time and location constraints. This process is known as 'user role assignment'. For example, user 'Mark' can access a lecturer's role during the daytime at a school of computer science. User role assignment is classed as a 'many-to-many relationship'. For example, a user can be assigned to one or more roles, and a role can have one or more users.

7. Permission Role Acquire (PRA)

   In organisations, roles are assigned to permissions based on time and location constraints. This is known as 'permission role acquire'. For example, the lecturer role can have to the permission modify the student mark during the daytime and at the location school of computer science. Permission role acquire is classed as a 'many-to-many relationship'. For example, a role can be assigned to one or more permissions, and a permission can have one or more roles

assigned to it.

- Role Hierarchy (RH)

  Normally, roles in an organisation have overlaps in permission assignments. For example, a lecturer can have the same permissions as students to read a course report, but he or she may also have additional permissions to add, delete or modify the report. Therefore, role hierarchies are introduced as a key aspect of hierarchical STRBAC models, in which a user who has a senior role can inherit a junior role and all the relevant permissions assigned to that particular junior role. Role hierarchies can be unrestricted, time-dependent, location-dependent or time- and location-dependent. An example of an unrestricted role hierarchy is shown in Figure 2.3.

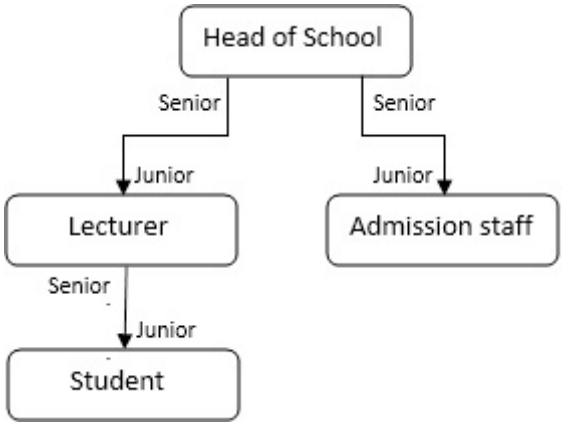

Figure 2.3: Role Hierarchy

- Location Hierarchy (LH)

  Usually, organisations are based across several locations, some of which may also contain inner locations. For example, a continent contains several countries and a country contains several cities, as illustrated in Figure 2.4. Therefore, location hierarchies are introduced as a key aspect of hierarchical STRBAC models, in which a user who has a role at the outer location should also have the same role at the inner location.

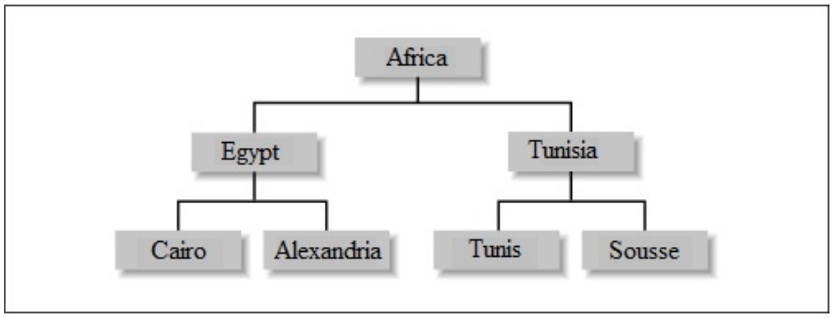

Figure 2.4: Location Hierarchy

- Separation of Duty between Roles (SoDR)

  The STRBAC model uses SoDR constraints to specify that two mutually exclusive roles cannot be simultaneously assigned to the same user at the same time and at the same location. For instance, the exclusive roles of 'lecturer' and 'admission staff' should not be assigned by the same user at the same time and at the same location.

- Separation of Duty between Permissions (SoDP)

  The STRBAC model uses SoDP constraints to specify that two mutually exclusive permissions cannot be simultaneously assigned to the same role at the same time and at the same location. For instance, the exclusive permissions modify student mark and process new student application should not be assigned by the same role at the same time and at the same location.

- Cardinality Constraint over Roles (CCR) This approach restricts the number of users who can have a role at the same time and at the same location. For instance, there should be only one user in the school of computer science that can have the role 'head of school' at the same time and at the same location.

- Cardinality Constraint over Permissions (CCP)

  This restricts the number of roles that can have a permission at the same time and at the same location. For example, there should be only one role that can have the permission 'modify head of school' profile at the same time and at the same location.

2.2 Analysis of the Access Control Model

There is a plethora of work using formal languages for modelling Access Control specifications so that they can be analysed. Some have used the Z language [17] for modelling RBAC [68] and LRBAC [69]. The authors described how the Z language can be used to represent the RBAC model and its features in a formal manner. However, the language itself lacks the tools to support the automatic analysis of the formalised model. Others have used an extension of the Unified Modelling Language (UML) [70], known as 'parametrised UML', to visualise the properties of RBAC constraints. The model describes how one can visualise the conflicts that may occur with RBAC constraints. However, it still lacks the ability to perform automatic model analysis. To this end, researchers [18, 19, 71, 72, 73, 74, 75] have investigated other approaches, such as Coloured Petri Nets (CPNs), for modelling Access Control specifications so that an automatic analysis can be performed. Jiang et al. [72] described how these CPNs can be used in verifying the security properties of the Bell LaPadula (BLP) model. Another study, presented by Laborde et al., proposed the use of CPNs for verifying the standard RBAC specification of network security mechanisms in [73]. In particular, the focus of this study was on verifying confidentiality, integrity and availability, as well as filtering rules.

Lu et al. [74] demonstrated how the Access Control specifications of work flows can be analysed using coloured Petri nets. In particular, they explained how to formalise control flow and authorisation rules, and how to conduct separation of duty constraints in a work flow in the presence of a role activation hierarchy. The authors first described how to model each part (i.e. control flow and separation of duty) in isolation. Subsequently, they proposed an approach for producing an integrated model which allows one to study interactions between the parts, such as an RBAC authorisation policy with separation of duty constraints. Reachability analysis was used to detect conflicts between the features. The size of the integrated model increased exponentially when new entities were added, so one of the major limitations of this approach is scalability. Another drawback of this work is that the model analysed does not support many features that are needed in work

flow applications, such as separation of duty between permissions.

Alloy [20], which uses SAT-solvers for analysis, has also been used successfully for the analysis of Access Control specifications [76, 77, 78, 79]. Alloy is a language used for modelling and specifying object-oriented systems, and it is supported by a tool called 'Alloy analyser'. The analyser is an automated constraint solver that transforms Alloy code into Boolean expressions, thus providing analysis via its embedded SAT solvers. Jackson et al. [76] demonstrated how to use Alloy to verify the internal consistency of an RBAC-96 schema. This study established that using Alloy has sufficient expressive power to prescribe implementation-independent specifications for Access Control systems. A similar approach was presented by Schaad et al. [77, 78], who modelled RBAC entities (e.g. users, roles), relations (user role assignment, role-permission assignment) and constraints such as role hierarchy and static separation of duties using Alloy. However, both of these studies did not consider environmental information such as time and location, which is needed in many applications today. Additionally, the transformation between RBAC and Alloy was carried out manually. For a small system, the creation of a model transformation between the RBAC model and Alloy could be managed manually; however, due to the complexity and size of modern systems, an automated transformation is required.

Another effort to verify the consistency of Access Control specification using Alloy was proposed by Samuel et al. [79]. This work illustrates how various features of the GST-RBAC model can be modelled and then analysed using Alloy. In a similar study, Ray et al. proposed using Alloy to model and analyse interactions between various features of the STRBAC model [14, 15]. The analysis process aimed to detect several types of inconsistencies that may occur due to interactions between the model's features. However, in both of these studies, transformations between Access Control specifications and Alloy were carried out manually. Although the transformation presented seems to be correct, in general, manual transformation is tedious and may cause errors, especially when the Access Control specification is very large.

Researchers such as [80, 81] have also advocated the use of Timed Automata mod-

elling and the analysis of Access Control policies. Samrat et al. [80, 81] demonstrated how to model and analyse the properties of a GTRBAC model using Timed Automata. The authors determined how to transform GTRBAC into Timed Automata models, following which a desirable set of liveness and safety security queries are constructed from the GTRBAC constraints. These queries are used later in the model verification process, which is done by using the Uppaal model checker [25, 82]. Although this study made it possible to perform an automated analysis of GTRBAC specifications, there were two major limitations. First, the impact of location on the Access Control specification is not discussed in this study, which is required in many applications today. The second limitation is that the Timed Automata network and the security queries were generated manually – manual transformation is tedious and may introduce unintended errors, especially when the Access Control specification is very large. Therefore, it is essential to automate the transformation between the Access Control specification and formal methods such as Timed Automata. As a result, this thesis presents a model transformation approach that automates the transformation between the STRBAC model and formal languages such as Alloy and Timed Automata for the purpose of analysis.

2.3 Alloy

Alloy is a language used for modelling and specifying object-oriented systems. It is based on first-order logic and was developed by the MIT Software Design Group in 1997 [20]. It is roughly a subset of the Z notation. Alloy is designed to bring a Z-style specification to the kind of automation offered by model checkers. Generally, an Alloy model consists of a set of modules, each of which consists of one or more paragraphs. The paragraphs of a module can be `signatures`, `relation declarations`, `facts`, `predicates`, or `check` commands. `Signatures` are used to define new sets of atoms. Constraints, such as `facts` and `predicates`, are used to specify constraints and expressions. A `fact` is a constraint that always holds and consists of an optional name, while a `predicate` is a constraint that

can be instantiated in different contexts and has a name. Commands such as `check` are an instruction to the Alloy analyser to perform an analysis. A `check` command helps to search for a counterexample showing that the model is inconsistent. Figure 2.5 presents the subset of an Alloy metamodel. For the purposes of this thesis, we have not considered all of the features of the Alloy language, which is explained fully by Jackson in [20].

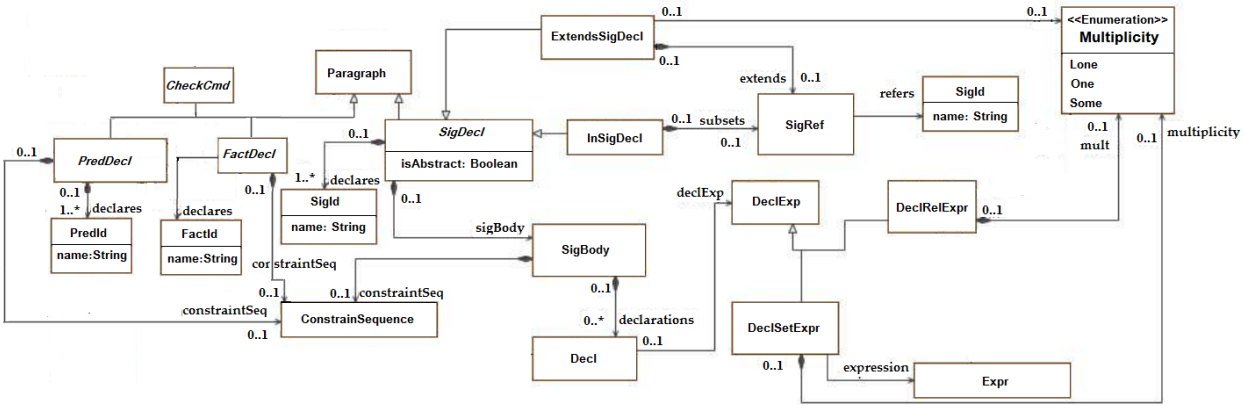

Figure 2.5: A Subset of an Alloy Metamodel [107]

Alloy is supported by an Alloy analyser [85], which supports the fully automated analysis of Alloy models. The analyser provides two main functionalities, namely simulations (`run command`) and assertions (`assertion and check command`). Simulation produces a random instance of the model which conforms to the specification. Assertions are constraints which the model needs to satisfy. The Alloy analyser works by translating Alloy formulas to Boolean expressions with the help of the KodKod [86] model finder. The Boolean expression is then automatically analysed using SAT solvers (i.e. SAT4J [84], ZChaff [83] and MiniSAT [87]), and the result of the analysis is then displayed to the user. The Alloy analyser is designed to perform finite scope checks, even on infinite models. Therefore, the user should specify a scope for the model elements to bound the domain. Without specifying a scope, Alloy will assume there are three atoms of everything unless specified otherwise. A scope is a positive integer number which limits the number of instances of each model element if the system is being analysed by the Sat-solver. If the analyser finds a counterexample within the scope, the assertion is not valid. However, if no counterexample is found, the assertion might be invalid in a larger scope. For more details on the

notion of scope, please refer to [[22], Sect. 5].

In the following section, we will describe a small Alloy model to demonstrate the most commonly used features of Alloy. The Alloy model is depicted in Figure 2.6. Line 1 of the Alloy model declares an `abstract signature` called *Person*. Marking a `signature` as an `abstract` means that it has no elements of its own that do not belong to its extensions. This `abstract signature` has two fields – the first field is called *children*, in line 2, and the second field is named *siblings*, in line 3. The *children* and *siblings* fields declare ordered pairs, who's first, and the second coordinates consist of instances of the `abstract signature` *Person*. The keyword *set* in the field's declaration means that there can be zero or any number of *Persons* related to *Person* through the relations *children* and *siblings*. Line 4 declares a signature named *Man*, which is a subset of the *Person* set due to *extend Person*, which is part of the signature declaration. The keyword *one*, before the `signature` declaration, specifies that the `signature` *Man* will be represented by a unique set. The `signature` declarations in line 5 and line 6 are similar to the `signatures` declaration already described. Line 8 declares a `function` called *parents*, which defines the parents' relation as an auxiliary one. Line 9 defines a `fact` constraint. The body of the `fact` constraint specifies that no person can be their own ancestor. An empty simulation command, which is defined in line 10, will produce a random instance of the model that conforms to the facts. The `for 3` part of the command specifies the scope for the elements of the system. More precisely, the Alloy analyser will attempt to produce an instance of the model, using up to three atoms for each of the signatures declared in the model. In this example the analyser will use a scope of three only for the Married signature, since the Person signature has been specified as abstract and the Man and Woman signatures are specified as singleton. Line 11 declares an `assertion` named *ChildrenArenotParents*, which specifies that no person has parents that are also children. Finally, in line 13, a `check` command is defined, which asks the Alloy analyser to confirm the `assertion` *ChildrenArenotParents* for a scope of three.

The execution of `check` (line 13 in Figure 2.6) shows that the Alloy analyser has found

```
1.    abstract sig Person {
2.       children: set Person,
3.       siblings: set Person}

4.    one sig Man extends Person {}

5.    one sig Woman extends Person {}

6.    sig Married in Person {
7.       spouse: one Married }

8.    fun parents [] : Person -> Person { ~children }

9.    fact {no p: Person | p in p.^parents}

10.   run {} 3

11.   assert ChildrenArenotParents {
12.       all p: Person | no p.children & p.parents }

13.   check ChildrenArenotParents for 3
```

Figure 2.6: A Sample of an Alloy model

no counterexample for the scope of three, which means that the `assertion` may be valid or the Alloy analyser might find a counterexample on a larger scope.

2.4 Timed Automata

Alur and Dill [24] introduced the idea of Timed Automata (TA), which is a directed graph containing a finite set of locations and a finite set of labelled edges extended with real-value variables that model logical clocks. In TA, time information is represented as a finite set of clocks which can be reset by state transitions. All the clocks progress synchronously when time elapses. An initial clock valuation maps each clock of a timed automaton to zero. A location or a state is associated with a clock constraint called the 'local invariant' of that location. Control can stay in a location but only if the clock valuation satisfies

the local invariant of that location. Local invariants are used to ensure the progress of the model [24], namely that a control cannot stay in a location forever. An edge in a timed automaton is called a transition, which is associated with a clock constraint, a subset of the clocks and a label (with a symbol). A clock constraint associated with a transition is called the 'guard' of that transition. A transition will be taken only if the guard is satisfied. Guards are used to restrict the behaviour of the automation, and each associated clock of a transition is reset to zero when the transition occurs. At any given instant, the value of a clock equals the time elapsed since the last time it was reset. While transitions are instantaneous, time can elapse in a location. Next we define Timed Automata formally.

Definition 1. Timed Automata (TA) are 6-tuples (S, Init, $\sum$, C, *Inv*, T), where S is a finite set of states, Init $\in$ S is the initial state, $\sum$ is a finite set of actions/labels, C is a finite set of clocks, *Inv* is a mapping that labels each state s with some clock constraints in $\Phi(C)$ and T is the transition relation. The transition $(s; a; \lambda; \delta; s')$ represents a transition from state s to state s' on event a.

Figure 2.7 depicts graphically a Timed Automata (TA) metamodel in the shape of a UML class diagram. The TA metamodel captures most of the concepts required for this thesis, such as state, transition and clock. To describe the concepts of Timed Automata, we will consider the following simple Timed Automata example, presented in Figure 2.8, with one clock c. The figure shows that s is the initial location and that it has no invariant constraint. This means that the system can be at this location for an arbitrary amount of time. When the system switches to another location *s'* or even a, the clock c resets to zero. While in location *s'*, the value of the clock c illustrates the time elapsed since the occurrence of the last transition. The transition from location *s'* to location s can be taken only if this value is greater than 1. The invariant $c \leqslant 2$ associated with location *s'* specifies the requirement that the system can stay in location *s'* for at most 2 units and a transition must occur before the invariant is violated. For more details about Timed Automata we refer the reader to [23, 24, 34, 36, 89, 90, 91].

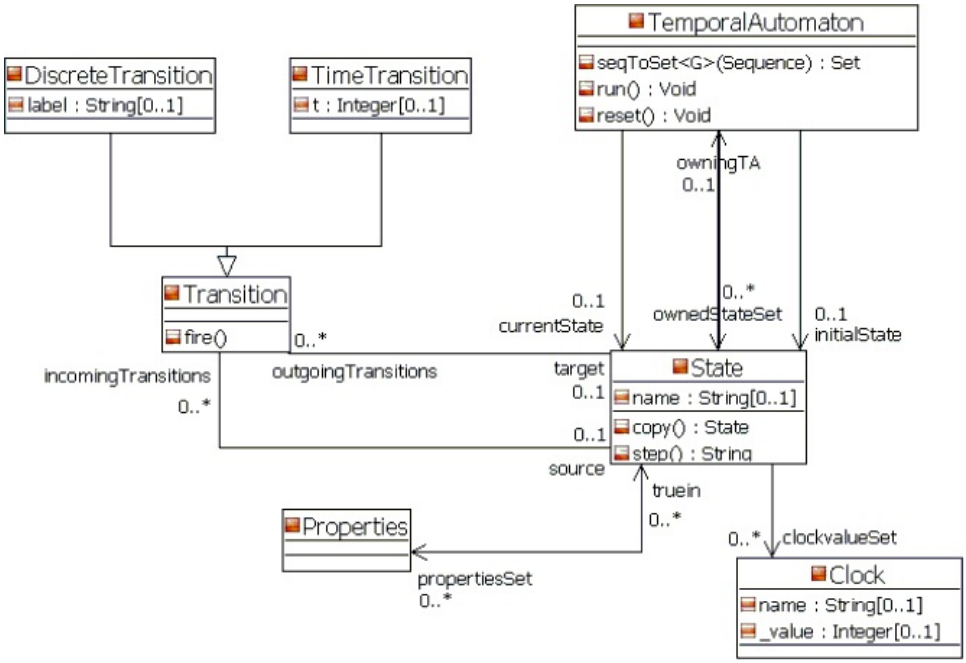

Figure 2.7: Timed Automata Metamodel [36]

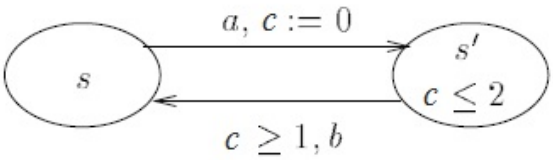

Figure 2.8: A sample Timed Automata [34]

2.4.1 Uppaal

Uppaal is a famous model checker created for extended Timed Automata by Yi-Wang et al. [25, 92]. It also supports model checking for conventional Timed Automata. The Uppaal model checker allows for the verification of specifications that are expressed in the Uppaal Requirement Specification language. This language is a subset of Timed Computation Tree Logic (TCTL), where primitive expressions are represented by location names, variables and clocks from the modelled system. In addition, Uppaal supports local and global integers and primitive operations on integers, such as addition, subtraction and multiplication with constants. Such expressions are also allowed on transition guards. A model of the system can be created from multiple Timed Automata which are synchronised via CCS-like synchronisation mechanisms [92]. In this thesis, we use the Uppaal model

checker to model and verify Timed Automata networks and TCTL statements.

2.5 Model-Driven Architecture (MDA)

The Model-Driven Architecture, abbreviated to MDA, is a standard initially introduced and developed by the OMG (Object Management Group), which has been an international, open membership, not-for-profit computer industry consortium since 1989 [93, 94]. It is one of the most important research initiatives in the model-driven engineering (MDE) area. Model-Driven Architecture (MDA) is a software design approach for modelling in software development. One of its main objectives is the idea of model transformation, which is central to the work presented in this thesis. In general, model transformation is considered a program that takes as its input a graph of objects and provides as an output another graph of objects. In the MDA context, a model transformation is defined by mapping the constructs of the metamodel of a source language into the constructs of the metamodel of a destination language, and then every model which is an instance of the source metamodel can be automatically transformed to an instance of the destination metamodel with the help of a model transformation framework [95, 96]. The metamodels of the source and target languages are specified using a common language known as the Meta Object Facility (MOF) [97], while models in the MDA are instances of metamodels.

Figure 2.9 depicts an MDA transformation approach. It shows that a typical model transformation framework requires the following inputs: source and target metamodels and a number of transformation rules. A set of transformation rules is used to define how various elements of the source metamodel are mapped to the elements of the destination metamodel. Then every model which conforms to the corresponding metamodel can be automatically transformed into a model that conforms to the target metamodel. For example, to map an Access Control specification in the context of the STRBAC model into formal languages such as Alloy or Timed Automata, a model transformation approach

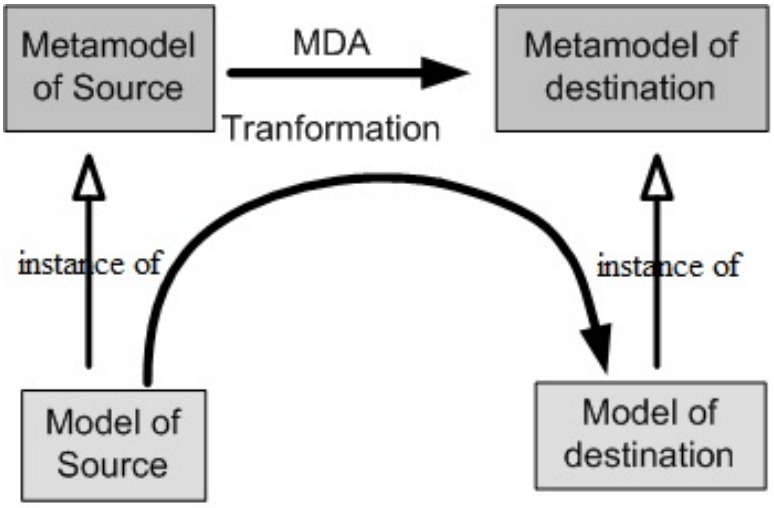

Figure 2.9: Overview of the MDA transformation Approach

that maps the elements of the STRBAC metamodel to the elements of the formal language metamodel is required.

Researchers have developed many industrial and academic case tools to support model transformations, such as Kermeta [98, 99], Arcstyler [103], OptimalJ [104], Xactium [105] and ATLAS [100, 101, 102]. Despite the fact that these tools allow for the specification and implementation of model transformations, which provide a rich set of functionalities, such tools are inherently complex. In particular, for researchers who are only interested in experimentation and the creation of prototypes, the steep learning curve is a significant hurdle. Therefore, in this thesis, we make use of Simple Transformer (SiTra) [108, 109], which provides a minimal framework for the execution of transformations to implement the transformation rules.

2.5.1 Simple Transformer (SiTra)

SiTra is a simple Java library developed by Akehurst et al. [108] to support a programming approach to writing transformations aiming to use Java for writing transformations, and it provides a solution for small-scale transformations. SiTra consists of two interfaces, as depicted in Figure 2.10 – rule interface, which user-defined mapping rules have to implement, and the transformer interface, which provides a framework for the methods

that carry out the transformation. The SiTra user needs only to define the transformation rules by implementing the rule interface, which consists of three methods: *check(), build()* and *setProperties()*. If the rule is applicable to the source element in question, the *check()* method of the rule implementation returns as true and the *build()* method is executed. The *build()* method generates the target model element. The *setProperties()* is used to set the attributes and links of the newly created target element. SiTra has been successfully applied to model transformation in various application domains [27, 28, 106, 110, 111, 112]. For further details on SiTra please refer to [108, 109].

```
interface Rule<S,T> {
   boolean check(S source);
   T build(S source, Transformer t);
   void setProperties(T target, S source, Transformer t);
 }
 interface Transformer {
   Object transform(Object source);
   List<Object> transformAll(List<Object> sourceObjects);
   <S,T> T transform(Class<Rule<S,T>> ruleType, S source);
   <S,T> List<T> transformAll(Class<Rule<S,T>> ruleType,
   List<S> source);
 }
```

Figure 2.10: SiTra Interfaces

Traceability is an important feature in model transformations, and as such it has received considerable attention [62, 63, 65, 67, 100] recently. Traceability is typically used during the model transformation process to keep a record of which element(s) of the source model(s) has been transformed to which element(s) of the destination model(s) and visa versa. In the context of model transformation, traceability can have many applications. For instance, if the source model has been changed slightly after executing the model transformation, traceability links can be used to update only the necessary elements of the destination model, without having to execute the whole transformation again. Tracing in SiTra consists of an instance *(ITrace)*, which holds a set of *TraceInstances (ts)*. Every

TraceInstances represents the mapping between the elements of the source and the target model, using a SiTra rule. For more details and a classification of traceability with SiTra, please refer to [67]. Figure 2.11 depicts the model for tracing M2M transformations.

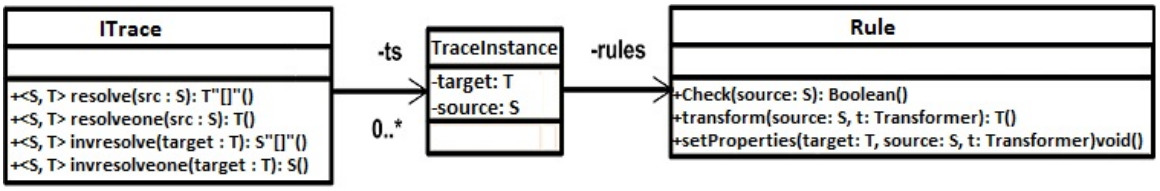

Figure 2.11: A Model of the Tracing Mechanism

2.6 Chapter Summary

This chapter provides a structured review of Access Control models and the verification of Access Control specification. Following this, it provides a brief introduction into Model Driven Architecture. Regarding the analysis of Access Control specifications, our research indicated that existing works have proposed several approaches such as Alloy and Timed Automata for modelling and analysing of Access Control specifications to detect inconsistencies. However, none of these works provide a method that automatically generates the analysable formalisation (i.e. Alloy and Timed Automata) from the Access Control specifications. In subsequent chapters, we will discuss our model transformation methods which fulfil this gap.

CHAPTER 3

FORMALISATION OF THE SPATIO-TEMPORAL ROLE BASED ACCESS CONTROL MODEL

As explained in Section 1.2. this work proposes MDA transformations to transform Access Control specifications in the context of a STRBAC model into Alloy and Timed Automata to perform an automated analysis. The aim of the analysis is to detect inconsistencies that may occur during interactions between Access Control specifications. Our approach is achieved as follows. Firstly, the formalisation of the STRBAC model is presented in Chapter 3. This is considered an important task, as it helps to formalise any inconsistencies in the STRBAC specifications. In addition, it helps to specify the STRBAC components that will be used to define the model transformations. Secondly, the first model transformation, which transforms the STRBAC into Alloy, is described in Chapter 4. Then, the implementation of the model transformation as an eclipse plug-in is discussed in Chapter 5. Fourthly, the second model transformation, which automates the transformation between the STRBAC and Timed Automata and its implementation as an eclipse plug-in, is discussed in Chapter 6. Fifthly, a comparison study between the presented Alloy and Timed Automata approaches is explained in Chapter 7. Finally, an extension of the STRBAC model, to cover the physical aspects of Access Control, is presented in Chapter 8.

In this chapter, we discuss the first step, which is the formalisation of the STRBAC model. This includes providing formal algebraic notations for the STRBAC framework.

These formal algebraic notations will be used later in this chapter to develop a formal definition of inconsistency in the STRBAC specifications [16, 113] as well as define different scenarios that might cause inconsistencies therein. Moreover, these formal algebraic notations will be used to introduce the concept of semi-consistency in the STRBAC specifications and define several examples in this case. Last but not least, the formal algebraic notations will be used in the next chapters to define our model transformations, in order to automate the transformation between the STRBAC and formal methods such as Alloy and Timed Automata, thus allowing for powerful analysis to take place to detect inconsistencies and semi-consistencies in the STRBAC specifications.

3.1 Formal Algebraic Notations of STRBAC

This section introduces formal algebraic notations for the STRBAC model to specify the components which will be used in this thesis.

Definition 2. Suppose that `U, R, P, T, L` are finite and non-empty sets of Users, Roles, Permissions, Times and Locations, respectively. In this case, an Access Control specification `A` can be seen as a subset of the Cartesian product `U` $\times$ `R` $\times$ `P` $\times$ `T` $\times$ `L`. We write `(u, r, p, t, l)` $\in$ `A`, meaning that the user `u` is assigned to the role `r` and has permission `p` at the location `l` and at that time `t`. The five tuples `(u, r, p, t, l)` are generated from the interactions between the Access Control rules, which are defined next. These rules specify permissible coordinates for the five tuples.

3.1.1 User Role Assignment (URA)

This is a relation that associates users with roles based on time and location, `URA` $\subseteq$ `U` $\times$ `R` $\times$ `T` $\times$ `L`. We write `URA`(u, r, t, l), meaning that a user u is assigned to a role r at time t and location l.

3.1.2 Permission Role Acquire (PRA)

This relation associates roles with permissions based on time and location $\texttt{PRA} \subseteq \texttt{R} \times \texttt{P} \times \texttt{T} \times \texttt{L}$. We write $\texttt{PRA}(r, p, t, l)$, meaning that a role r is assigned to a permission p at time t and location l.

3.1.3 Role Hierarchy (RH)

This is a transitive partial order on the set of roles, $\texttt{RH} \subseteq \texttt{R} \times \texttt{R} \times \texttt{T} \times \texttt{L}$. We write $r_i \succeq r_j$, meaning that the role r_i is senior to the role r_j at any time and at any location. This means that r_i inherits all the permissions of r_j, and if there is a user assigned to a senior role r_i then he/she can also be assigned to the junior role r_j. RH can be unrestricted, time-dependent, location-dependent or time- and location-dependent and written as $\succeq$, $\succeq_t$, $\succeq_l$ and $\succeq_{t,l}$, respectively.

- Unrestricted Role Hierarchy (URH): Let r_i and r_j be roles such that $r_i \succeq r_j$, i.e. the role r_i is a senior role to the role r_j at any time and at any location.

  $\forall r_i, r_j \in \texttt{R}$ $(r_i \succeq r_j)$, then

  $\forall u \in \texttt{U}$, $\forall t \in \texttt{T}$, $\forall l \in \texttt{L}$ $\texttt{URA}(u, r_i, t, l) \Longrightarrow \texttt{URA}(u, r_j, t, l) \wedge$

  $\forall p \in \texttt{P}$, $\forall t \in \texttt{T}$, $\forall l \in \texttt{L}$ $\texttt{PRA}(r_j, p, t, l) \Longrightarrow \texttt{PRA}(r_i, p, t, l)$

- Time-Dependent Role Hierarchy (TRH): Let r_i and r_j be roles such that $r_i \succeq_{t'} r_j$, i.e. the role r_i is a senior role to the role r_j only at time t' and at any location.

  $\forall r_i, r_j \in \texttt{R}$, $\forall t' \in \texttt{T}$ $(r_i \succeq_{t'} r_j)$, then

  $\forall u \in \texttt{U}$, $\forall l \in \texttt{L}$ $\texttt{URA}(u, r_i, t', l) \Longrightarrow \texttt{URA}(u, r_j, t', l) \wedge$

  $\forall p \in \texttt{P}$, $\forall l \in \texttt{L}$ $\texttt{PRA}(r_j, p, t', l) \Longrightarrow \texttt{PRA}(r_i, p, t', l)$

- Location-Dependent Role Hierarchy (LRH): Let r_i and r_j be roles such that $r_i \succeq_{l'} r_j$, i.e. the role r_i is a senior role to the role r_j only at the location l' and at any time.

  $\forall r_i, r_j \in \texttt{R}$, $\forall l' \in \texttt{L}$ $(r_i \succeq_{l'} r_j)$, then

$\forall u \in \texttt{U}, \forall t \in \texttt{T}\ \texttt{URA}(u, r_i, t, l') \implies \texttt{URA}(u, r_j, t, l') \wedge$

$\forall p \in \texttt{P}, \forall t \in \texttt{T}\ \texttt{PRA}(r_j, p, t, l') \implies \texttt{PRA}(r_i, p, t, l')$

- Time- and Location-Dependent Role Hierarchy (TLRH): Let r_i and r_j be roles such that $r_i \succeq_{t',l'} r_j$, i.e. the role r_i is a senior role to the role r_j only at the time t' and at location l'.

  $\forall r_i, r_j \in \texttt{R}, \forall t' \in \texttt{T}, \forall l' \in \texttt{L}\ (r_i \succeq_{t',l'} r_j)$, then

  $\forall u \in \texttt{U}\ \texttt{URA}(u, r_i, t', l') \implies \texttt{URA}(u, r_j, t', l') \wedge$

  $\forall p \in \texttt{P}\ \texttt{PRA}(r_j, p, t', l') \implies \texttt{PRA}(r_i, p, t', l')$

3.1.4 Location Hierarchy (LH)

This is a partial order on the set locations that specifies which location contains another location, $\texttt{LH} \subseteq \texttt{L} \times \texttt{L}$. We write $l_i \succeq l_j$, meaning that the location l_i is an outer location to the inner location l_j. This means that if there is a user who is assigned to the role r at the outer location l_i, then he/she can also be assigned to the role r at the inner location l_j, and if the role r has a permission p at the outer location l_i, then the role r should also has the same permission p at the inner location l_j.

$\forall l_i, l_j \in \texttt{L}\ (l_i \succeq l_j)$, then

$\forall u \in \texttt{U}, \forall r \in \texttt{R}, \forall t \in \texttt{T}\ \texttt{URA}(u, r, t, l_i) \implies \texttt{URA}(u, r, t, l_j) \wedge$

$\forall p \in \texttt{P}, \forall r \in \texttt{R}, \forall t \in \texttt{T}\ \texttt{PRA}(r, p, t, l_i) \implies \texttt{PRA}(r, p, t, l_j)$

3.1.5 Separation of Duty between Role (SoDR)

We write $\texttt{sodr}(r_i, r_j, t', l')$, meaning that the two exclusive roles r_i and r_j should not be assigned to the same user at time t' and location l'. SoDR can be unrestricted $\texttt{sodr}(r_i, r_j)$, time-dependent $\texttt{sodr}(r_i, r_j, t')$, location-dependent $\texttt{sodr}(r_i, r_j, l')$ or time- and location-dependent $\texttt{sodr}(r_i, r_j, t', l')$.

- Unrestricted Separation of Duty over Roles (SoDR): Let r_i and r_j be exclusive roles such that $\texttt{sodr}(r_i, r_j)$, i.e. the role r_i and the role r_j should not be assigned to the

same user at any location and at any time.

$\forall r_i, r_j \in$ R $\texttt{sodr}(r_i, r_j)$, then

$\forall t \in$ T, $\forall l \in$ L, $\neg\exists u \in$ U $\texttt{URA}(u, r_i, t, l) \wedge \texttt{URA}(u, r_j, t, l)$

- Time-Dependent Separation of Duty over Roles (TSoDR):

  Let r_i and r_j be exclusive roles such that $\texttt{sodr}(r_i, r_j, t')$, i.e. the role r_i and the role r_j should not be assigned to the same user at the time t' and at any location.

  $\forall r_i, r_j \in$ R, $\forall t' \in$ T $\texttt{sodr}(r_i, r_j, t')$, then

  $\forall l \in$ L, $\neg\exists u \in$ U $\texttt{URA}(u, r_i, t', l) \wedge \texttt{URA}(u, r_j, t', l)$

- Location-Dependent Separation of Duty over Roles (LSoDR):

  Let r_i and r_j be exclusive roles such that $\texttt{sodr}(r_i, r_j, l')$, i.e. the role r_i and the role r_j should not be assigned to the same user at the location l' and at any time.

  $\forall r_i, r_j \in$ R, $\forall l' \in$ L $\texttt{sodr}(r_i, r_j, l')$, then

  $\forall t \in$ T, $\neg\exists u \in$ U $\texttt{URA}(u, r_i, t, l') \wedge \texttt{URA}(u, r_j, t, l')$

- Time- and Location-Dependent Separation of Duty over Roles (TLSoDR):

  Let r_i and r_j be exclusive roles such that $\texttt{sodr}(r_i, r_j, t', l')$, i.e. the role r_i and the role r_j should not be assigned to the same user at the time t' and the location l'.

  $\forall r_i, r_j \in$ R, $\forall t' \in$ T, $\forall l' \in$ L $\texttt{sodr}(r_i, r_j, t', l')$, then

  $\neg\exists u \in$ U $\texttt{URA}(u, r_i, t', l') \wedge \texttt{URA}(u, r_j, t', l')$

3.1.6 Separation of Duty between Permissions (SoDP)

We write $\texttt{sodp}(p_i, p_j, t', l')$, meaning that the two exclusive permissions p_i and p_j should not be assigned to the same role at time t' and location l'. SoDP can be unrestricted $\texttt{sodp}(p_i, p_j)$, time-dependent $\texttt{sodp}(p_i, p_j, t')$, location-dependent $\texttt{sodp}(p_i, p_j, l')$ or time- and location-dependent $\texttt{sodp}(p_i, p_j, t', l')$.

- Unrestricted Separation of Duty between Permissions (USoDP):

  Let p_i and p_j be exclusive permissions such that $\texttt{sodp}(p_i, p_j)$, i.e. permission p_i and

permission p_j should not be assigned to the same role at any location and at any time.

$\forall p_i, p_j \in$ P $\texttt{sodp}(p_i, p_j)$, then

$\forall t \in$ T, $\forall l \in$ L, $\neg\exists r \in$ R $\texttt{PRA}(r, p_i, t, l) \wedge \texttt{PRA}(r, p_j, t, l)$

- Time-Dependent Separation of Duty between Permissions (TSoDP):

  Let p_i and p_j be exclusive permissions such that $\texttt{sodp}(p_i, p_j, t')$, i.e. permission p_i and permission p_j should not be assigned to the same role at the time t' and at any location. $\forall p_i, p_j \in$ P, $\forall t' \in$ T $\texttt{sodp}(p_i, p_j, t')$, then

  $\forall l \in$ L, $\neg\exists r \in$ R $\texttt{PRA}(r, p_i, t', l) \wedge \texttt{PRA}(r, p_j, t', l)$

- Location-Dependent Separation of Duty between Permissions (LSoDP):

  Let p_i and p_j be exclusive permissions such that $\texttt{sodp}(p_i, p_j, l')$, i.e. permission p_i and permission p_j should not be assigned to the same role at the location l' and at any time.

  $\forall p_i, p_j \in$ P, $\forall l' \in$ L $\texttt{sodp}(p_i, p_j, l')$, then

  $\forall t \in$ T, $\neg\exists r \in$ R $\texttt{PRA}(r, p_i, t, l') \wedge \texttt{PRA}(r, p_j, t, l')$

- Time- and Location-Dependent Separation of Duty between Permissions (STSoDP):

  Let p_i and p_j be exclusive permissions such that $\texttt{sodp}(p_i, p_j, t', l')$, i.e. permission p_i and permission p_j should not be assigned to the same role at the location l' and at the time t'.

  $\forall p_i, p_j \in$ P, $\forall t' \in$ T, $\forall l' \in$ L $\texttt{sodp}(p_i, p_j, t', l')$, then

  $\neg\exists r \in$ R $\texttt{PRA}(r, p_i, t', l') \wedge \texttt{PRA}(r, p_j, t', l')$

3.1.7 Cardinality Constraints over Roles (CCR)

We write $\texttt{ccr}(r_i, t', l', n)$, meaning that the role r_i has restrictions, so it should not be assigned to more than n users at time t' and location l'. CCR can be unrestricted $\texttt{ccr}(r_i, n)$, time-dependent $\texttt{ccr}(r_i, t', n)$, location-dependent $\texttt{ccr}(r_i, l', n)$ or time- and location-dependent $\texttt{ccr}(r_i, t', l', n)$.

- Unrestricted Cardinality Constraints over Roles (UCCR):

  Let the role r_i be a restricted role, such that $\texttt{ccr}(r_i, n)$, i.e. role r_i should not be assigned to more than n users at any time and any location. This should satisfy the following constraint:

  $\forall r_i \in \texttt{R}\ (r_i, n)$, then

  $\forall u \in \texttt{U}, \forall t \in \texttt{T}, \forall l \in \texttt{L}\ \#(\texttt{URA}(u, r_i, t, l)) \leq n$

- Time-Dependent Cardinality Constraints over Roles (TCCR):

  Let the role r_i be a restricted role, such that $\texttt{ccr}(r_i, t', n)$, i.e. role r_i should not be assigned to more than n users at time t' and at any location. This should satisfy the following constraint:

  $\forall r_i \in \texttt{R}, \forall t' \in \texttt{T}\ (r_i, t', n)$, then

  $\forall u \in \texttt{U}, \forall l \in \texttt{L}\ \#(\texttt{URA}(u, r_i, t', l)) \leq n$

- Location-Dependent Cardinality Constraints over Roles (LCCR):

  Let the role r_i be a restricted role, such that $\texttt{ccr}(r_i, l', n)$, i.e. role r_i should not be assigned to more than n users at location l' and at any time. This should satisfy the following constraint:

  $\forall r_i \in \texttt{R}, \forall l' \in \texttt{L}\ (r_i, l', n)$, then

  $\forall u \in \texttt{U}, \forall t \in \texttt{T}\ \#(\texttt{URA}(u, r_i, t, l')) \leq n$

- Time- and Location-Dependent Cardinality Constraints over Roles (TLCCR): Let the role r_i be a restricted role, such that $\texttt{ccr}(r_i, t', l', n)$, i.e. role r_i should not be assigned to more than n users at time t' and location l'. This should satisfy the following constraint.

  $\forall r_i \in \texttt{R}, \forall t' \in \texttt{T}, \forall l' \in \texttt{L}\ \texttt{ccr}(r_i, t', l', n)$, then

  $\forall u \in \texttt{U}\ \#(\texttt{URA}(u, r_i, t', l')) \leq n$

3.1.8 Cardinality Constraints over Permissions (CCP)

We write $\texttt{ccp}(p_i, t', l', m)$, meaning that permission p_i has restrictions, in which case it should not be assigned to more than m roles at time t' and location l'. CCP can be unrestricted $\texttt{ccp}(p_i, m)$, time-dependent $\texttt{ccp}(p_i, t', m)$, location-dependent $\texttt{ccp}(p_i, l', m)$ or time- and location-dependent $\texttt{ccp}(p_i, t', l', m)$.

- Unrestricted Cardinality Constraints over Permissions (UCCP):

  Let the permission p_i be a restricted permission, such that $\texttt{ccp}(p_i, m)$, i.e. permission p_i should not be assigned to more than m roles at any time and any location. This should satisfy the following constraint:

  $\forall p_i \in \texttt{P}\ \texttt{ccp}(p_i, m)$, then

  $\forall r \in \texttt{R}, \forall t \in \texttt{T}, \forall l \in \texttt{L}\ \#(\texttt{PRA}(r, p_i, t, l)) \leq m$

- Time-Dependent Cardinality Constraints over Permissions (TCCP):

  Let the permission p_i be a restricted permission, such that $\texttt{ccp}(p_i, t', m)$, i.e. permission p_i should not be assigned to more than m roles at time t' and any location. This should satisfy the following constraint:

  $\forall p_i \in \texttt{P}, \forall t' \in \texttt{T}\ (p_i, t', m)$, then

  $\forall r \in \texttt{R}, \forall l \in \texttt{L}\ \#(\texttt{PRA}(r, p_i, t', l)) \leq m$

- Location-Dependent Cardinality Constraints over Permissions (LCCP):

  Let the permission p_i be a restricted permission, such that $\texttt{ccp}(p_i, l', m)$, i.e. permission p_i should not be assigned to more than m roles at location l' and any time. This should satisfy the following constraint:

  $\forall p_i \in \texttt{P}, \forall l' \in \texttt{L}\ (p_i, l', m)$, then

  $\forall r \in \texttt{R}, \forall t \in \texttt{T}\ \#(\texttt{PRA}(r, p_i, t, l')) \leq m$

- Time- and Location-Dependent Cardinality Constraints over Permissions (STCCP):

  Let the permission p_i be a restricted permission, such that $\texttt{ccp}(p_i, t', l', m)$, i.e. permission p_i should not be assigned to more than m roles at time t' and location l'.

This should satisfy the following constraint:

$\forall p_i \in$ P, $\forall t' \in$ T, $\forall l' \in$ L ccp(p_i, t', l', m), then

$\forall r \in$ R #(PRA$(r, p_i, t', l')) \leq m$

3.2 Formalisation of Inconsistency and Semi-consistency in the STRBAC Specification

In this section we provide a formal definition of inconsistency in the STRBAC specification and then we present several examples of inconsistencies in the same setting. We also introduce the concept of semi-consistency in the STRBAC specification. A semi-consistency is a special case whereby the inconsistency can be avoided if the assignment of the user to a role is controlled [16, 113]. To the best of the authors' knowledge, we are the first to introduce this concept of semi-consistency. Finally, this section presents different scenarios that may cause semi-consistencies in the STRBAC specification.

3.2.1 Inconsistency in the STRBAC Specification

Various rules within the STRBAC model make the design of an Access Control system more simple by breaking down a complex specification into various rules, so that, for instance, User Role Assignment and Separation of Duty between Roles can be considered separately. However, multiple rules may result in inconsistency when they interact with each other. If this is the case, then there is no subset A of the Cartesian product U $\times$ R $\times$ P $\times$ T $\times$ L that satisfies all the STRBAC rules. As a result, an inconsistent specification means that it is not possible to find the tuples (u, r, p, t, l), so that the coordinates satisfy all rules (User Role Assignment, Permission Role Acquire, Role Hierarchy, Location Hierarchy, Separation of Duty constraints and Cardinality constraints).

Definition 3. Suppose that URA, PRA, RH, LH, SoDR, SoDP, CCR and CCP is a set of Access Control rules. An Access Control specification is termed inconsistent if there are no

non-empty sets $\mathtt{A} \subseteq \mathtt{U} \times \mathtt{R} \times \mathtt{P} \times \mathtt{T} \times \mathtt{L}$ such that $\mathtt{A}$ satisfies all Access Control rules.

In [114, 115], Chao Huang et al. present examples of inconsistencies that might occur during interactions between different RBAC rules. Following their examples, we also list here some examples of inconsistencies in the STRBAC specification that came across during our studies.

- Inconsistency between User Role Assignment (URA) and Separation of Duty over Roles (SoDR). For example, $\exists u \in \mathtt{U}$, $\exists t \in \mathtt{T}$, $\exists l \in \mathtt{L}$, $\exists r_i, r_j \in \mathtt{R}\ \mathtt{sodr}(r_i, r_j) \wedge \mathtt{URA}(u, r_i, t, l) \wedge \mathtt{URA}(u, r_j, t, l)$.

  Inconsistency is caused by interactions between the URA and the SoDR, because the user u accesses the two conflicting roles r_i and r_j, which is not permissible according to SoDR.

- Inconsistency between User Role Assignment (URA) and Cardinality Constraint over Roles (CCR). For example, $\exists u \in \mathtt{U}$, $\exists t \in \mathtt{T}$, $\exists l \in \mathtt{L}$, $\exists r_i \in \mathtt{R}\ \mathtt{ccr}(r_i, n) \wedge \#|\mathtt{URA}(u, r_i, t, l)| > n$.

  There is also inconsistency in the specification, which is caused by interactions between the URA and the CCR, because the number of User Role Assignments for the restricted role r_i is larger than n, which violates the cardinality constraint.

- Inconsistency between User Role Assignment (URA) and Role Hierarchy (RH). For example, $\exists r_i, r_j, r_k \in \mathtt{R}$, $\exists u \in \mathtt{U}$, $\exists t \in \mathtt{T}$, $\exists l \in \mathtt{L}\ (r_i \succeq r_j) \wedge (r_j \succeq r_k) \wedge (r_k \succeq r_i) \wedge (\mathtt{URA}(u, r_i, t, l) \vee \mathtt{URA}(u, r_j, t, l) \vee \mathtt{URA}(u, r_k, t, l))$.

  There is also inconsistency in the specification, caused by interactions between URA and RH, because the cycle of RH allows the user, who is assigned to the junior role, to get permissions for senior roles.

- Inconsistency between User Role Assignment (URA), Permission Role Acquire (PRA) and Separation of Duty between Permissions (SoDP). For example, $\exists u \in \mathtt{U}$, $\exists r \in \mathtt{R}$, $\exists t \in \mathtt{T}$, $\exists l \in \mathtt{L}$, $\exists p_i, p_j \in \mathtt{P}\ \mathtt{sodp}(p_i, p_j) \wedge \mathtt{URA}(u, r, t, l) \wedge \mathtt{PRA}(r, p_i, t, l) \wedge \mathtt{PRA}(r, p_j, t, l)$.

There is also inconsistency caused by interactions between the URA, PRA and the SoDP, because URA allows a user u to access role r, which is assigned to the two conflicting permissions p_i and p_j and is not permissible according to (SoDP).

- Inconsistency between User Role Assignment (URA), Role Hierarchy (RH) and Separation of Duty between Roles (SoDR). For example, $\exists u \in \mathtt{U}$, $\exists t \in \mathtt{T}$, $\exists l \in \mathtt{L}$, $\exists r_i, r_j \in \mathtt{R}$ $(r_i \succeq r_j) \wedge \mathtt{sodr}(r_i, r_j) \wedge \mathtt{URA}(u, r_i, t, l)$.

  There is also inconsistency between URA, RH and SoDR, because the user u can gain access to the two conflicting roles r_i based on URA and r_j based on URA and RH, which is not permissible according to SoDR.

- Inconsistency between User Role Assignment (URA), Role Hierarchy (RH), Permission Role Acquire (PRA) and Separation of Duty between Permissions (SoDP). For example, $\exists u \in \mathtt{U}$, $\exists r_i, r_j \in \mathtt{R}$, $\exists t \in \mathtt{T}$, $\exists l \in \mathtt{L}$, $\exists p_i, p_j \in \mathtt{P}$ $(r_i \succeq r_j) \wedge \mathtt{sodp}(p_i, p_j) \wedge \mathtt{URA}(u, r_i, t, l) \wedge \mathtt{PRA}(r_j, p_i, t, l) \wedge \mathtt{PRA}(r_j, p_j, t, l)$.

  There is also inconsistency between the URA, PRA, RH and the SoDP, because a user u can access role r_i based on URA, and, based on RH, the same user u can access role r_j, which is assigned to the two conflicting permissions p_i and p_j, which is not permissible according to SoDP.

The list above is not exhaustive, as we have only considered the unrestricted RH, SoDR, SoDP, CCR and CCP to define our chosen scenarios. As such, there could be many more complicated scenarios composed from the above scenarios when the time-dependent, location-dependent and time- and location-dependent constraints are considered.

3.2.2 Semi-consistency in STRBAC Specification

In this section we introduce the term 'semi-consistency' in the STRBAC specification. Semi-consistency is a weak form of consistency, in that the system is not inconsistent but it can be potentially inconsistent following a minor change to the specification [16].

Consider an Access Control specification that contains the following statements: a SoDP statement claims that the two permissions p_2 and p_2 should not be assigned by the same role at time t and location l, while a PRA statement claims that a role r can have the two permissions p_2 and p_2 at any time and at any location. Although these two statements are contradictory, there will be no inconsistency in the specification if no user is assigned to the role r at time t and location l. A minor change, for example assigning a user to the role r, will result in inconsistency. We refer to such scenarios as semi-consistency.

Definition 4. We call an Access Control specification S semi-consistent if i) is consistent (see Definition 3 in Section 3.1), ii). There are a finite number of users $u_1, u_2, ..., u_n \in$ U and roles $r_1, r_2, ..., r_n \in$ R, so that if a new specification S' consisting of S in addition to ura(u_1, r_1, t, l), ura(u_1, r_1, t, l), ..., ura(u_n, r_n, t, l), so that S' is inconsistent.

Remark:. It is possible changing PRA can cause inconsistency. For example, assume that the permissions p_1 and p_2 are conflicted permissions, which means the two permissions p_1 and p_2 should not be assigned to the same role at the same time and the same location. Changing the PRA by assigning a role r to the permissions p_1 and p_2 at the same time and the same location will cause inconsistency. We decided not to consider such changes in the specifications as a cause of semi-consistency as specified in Definition 3. This is because we argue that changes in PRA mean changes in the organisation's Access Control. This will happen much less frequently than adding users. We assume that when the PRA changes, we analyse everything from scratch.

In the case of semi-consistency we may try to either modify the specification to remove that semi-consistency or we may add logical constraint (Java assertions) to avoid an inappropriate user role assignment. In effect, the idea is to identify bad user role assignment which may cause inconsistencies, therefore preventing them from happening in the first place. Next we present some examples involving semi-consistencies.

- Semi-consistency because of an Interaction between Permission Role Acquire (PRA) and Separation of Duty between Permissions (SoDP). For example, $\exists r \in$ R, $\exists t \in$ T, $\exists l \in$ L, $\exists p_i, p_j \in$ P PRA$(r, p_i, t, l) \wedge$ PRA$(r, p_j, t, l) \wedge$ sodp(p_i, p_j).

The above specification is semi-consistent because a minor change, for example assigning a user to the role r, at the time t and the location l allows the user to have the two conflicting permissions p_1 and p_j, which are assigned to the role r at time t and location l. Such changes should be prevented.

- Semi-consistency because of Role Hierarchy (RH). For example, $\exists r_i, r_j, r_k \in \mathtt{R}$, $\exists t \in \mathtt{T}$, $\exists l \in \mathtt{L}$ $(r_i \succeq r_j) \wedge (r_j \succeq r_k) \wedge (r_k \succeq r_i)$.

  The above specification is semi-consistent because a minor change, for example assigning a user to one of the roles r_i, r_j or r_k, at time t and location l will result in inconsistency. Such changes should be prevented.

- Semi-consistency because of an Interaction between User Role Assignment (URA) and Cardinality Constraints over Roles (CCR). For example, $\exists u_1 \in \mathtt{U}$, $\exists r \in \mathtt{R}$, $\exists t \in \mathtt{T}$, $\exists l \in \mathtt{L}$ $\mathtt{ura}(u_1, r, t, l) \wedge \mathtt{ccr}(r, t, l, 2)$.

  The above specification is semi-consistent because a minor change, for example including $\mathtt{ura}(u_2, r, t, l)$ and $\mathtt{ura}(u_3, r, t, l)$, will result in inconsistency. Such changes should be prevented.

3.3 Chapter Summary

In this chapter, we introduced details about the formalisation of the STRBAC model. Session 3.1 explains the formal algebraic notations for all components of the STRBAC model. These formal algebraic notations are used in Section 3.2 to present a formal definition of inconsistency in the STRBAC specifications and to describe various examples of inconsistencies in the same instance. Next, the concept of semi-consistency and several examples of semi-consistencies in STRBAC specifications are discussed. Based on the above foundation and the background information introduced in Chapter 2, we are ready to define the model transformations that take place between formal representations, such as Alloy and Timed Automata, and the STRBAC specification.

CHAPTER 4

AC2ALLOY: TRANSFORMATION BETWEEN STRBAC AND ALLOY

As discussed in Chapter 2, using formal methods such as Alloy is always beneficial for modelling and analysing Access Control specifications. However, the transformation process between formal methods and Access Control specification is challenging, because the process of creating formal representation for analysis is manual and done by a human modeller. As a result, it is prone to human error and is time-consuming. There is also a problem of scoping as a system grows in size, whereby the manual generation of formal models becomes extremely difficult, if not totally unfeasible. As a result, this work utilises a Model-Driven Architecture approach to automate the transformation between the STRBAC model and Alloy.

In this chapter, a model transformation, AC2Alloy [27] is introduced. AC2Alloy provides a framework for STRBAC models to be transformed into Alloy, thus allowing for powerful analysis to take place using Alloy analyser utilising SAT-Solvers. To conduct the model transformation, a set of transformation rules have been defined. The transformation rules map various elements of the source metamodel, STRBAC into the elements of the destination metamodel, Alloy. For each rule an informal reasoning on why a particular STRBAC component maps to a particular Alloy concept is provided.

Before explaining our approach, a simplified example of The SECURE bank system taken from [29] will be introduced, for demonstration purposes.

4.1 A Running example: The SECURE bank System

In this section, a simplified version of the example given by Ray et al. [29] will be provided in order to demonstrate our approach. This example is based on a typical SECURE bank system consisting of several security policies. The application is used by various bank officers to perform several transactions (e.g. on customer loan accounts) at different locations and during specific periods of time. The SECURE bank system consists of the following: four users `Dave, Mark, Hanna` and `Sarah`; two locations `Office1` and `Office2`; four roles `Teller, Accountant, Accounting Manager` and `Loan Officer`; and four permissions `Read and Write Teller Files (RWTF), Read and Write Loan Officer Files (RWLF), Read and Write Accountant Files (RWAF)` and `Read and Write Accounting Manager Files (RWAMF)`. The security policies for the SECURE bank system are given below.

4.1.1 Security Policies

- **User Role Assignment**: Users are assigned to roles in the banking system as illustrated in Table 4.1.

Table 4.1: User Role Assignment Constraints

Users	**Roles**	**Times**	**Locations**
Mark	Accounting Manager	DayTime	Office1
Hanna	Accountant	DayTime	Office1
Sarah	Loan Officer	DayTime	Office2
Sarah	Teller	DayTime	Office2
Dave	Accounting Manager	NightTime	Office1

- **Permission Role Acquire**: Roles are assigned to permissions in the banking system as illustrated in Table 4.2.

- **Role Hierarchy**: Some roles in the SECURE bank system are related using restricted role hierarchy as follows: Role Hierarchy= {`rhtl(Accounting Manager, Accountant,`

Table 4.2: Permission Role Acquire

Roles	Permissions	Times	Locations
Teller	WRTF: Read and Write Teller Files	DayTime	Office2
Loan Officer	WRLF: Read and Write Loan Files	DayTime	Office2
Accountant	WRAF: Read and Write Accountant Files	DayTime	Office1
Accounting Manager	WRAMF: Read and Write Accounting Manager Files	NightTime	Office1

`DayTime, Office1)`}, which means the role `Accounting Manager` is a senior role to the `Accountant` role during the `DayTime` and at the location `Office1`.

- **Separation of Duty between Roles**: There are two separation of duty constraints between roles in the SECURE bank system. Separation of Duty Over Roles={`sodrtl(Teller, Loan Officer, DayTime, Office2), sodrtl(Accountant, Teller, DayTime, Office1)`}. This means that the role `Teller` and the role `Loan Officer` should not be assigned to the same user during the `DayTime` and at the location `Office2` and the role `Teller` and the role `Accountant` should not be assigned to the same user during the `DayTime` and at the location `Office1`.

- **Separation of Duty between Permissions**: There is one separation of duty between permissions in the SECURE bank system. Separation of Duty Over Permissions={`sodptl(RWTF, RWLF, DayTime, Office2)`}. This means that the permissions `Read and Write Teller Files (RWTF)` and `Read and Write Loan Files (RWLF)` should not be assigned to the same role during the time `DayTime` and at the location `Office2`.

- **Cardinality Constraints over Roles**: There is one restricted cardinality constraint over roles in the SECURE bank system. Cardinality Constraints over Roles= {`ccrtl(Accountant, DayTime, Office1, 1)`}. This means that the maximum number of users that should be assigned to the `Accountant` role during the `DayTime` and at the location `Office1` is `One`.

- **Cardinality Constraints over Permissions**: There is one restricted cardinality constraint over permissions in the SECURE bank system. Cardinality Constraints over Permission={`ccptl(Read and Write Accountant Files (RWAF), DayTime, Office1, 1)`}. This means that the maximum number of roles that should be assigned to permission `RWAF` during the `DayTime` and at `Office1` is `One`.

4.2 AC2Alloy – Model Transformation

AC2Alloy is an automated approach that makes use of MDA techniques to transform STRBAC models into Alloy. The model transformation process is hereby described in three stages:

- Metamodels for the source and target models.
- Transformation rules to map the elements of the STRBAC and Alloy.
- Implementation of the transformation rules (which will be discussed in Chapter 5).

In order to use an MDA methodology to automate the transformation between STRBAC and Alloy, metamodels for the source model, STRBAC model and the target model, an Alloy model needs to be constructed to specify the elements of STRBAC that will be mapped to Alloy. The Alloy metamodel is presented in Section 2.3. In this work, we do not consider the complete features of the Alloy language, but instead depict only those features that are used in our transformation. The metamodel of the STRBAC, on the other hand, is declared in the XML Schema Definition (XSD) [116]. XSD is a document structure and type definition specification, which provides the syntax specification and constraints for both structure and content. Figure 4.1 depicts the core elements of the STRBAC metamodel specified in XSD. A more elaborate version of the STRBAC metamodel is presented in Appendix A.

Figure 4.1 illustrates that there are 13 elements in the STRBAC metamodel that should be mapped into Alloy features. Next we shall describe our transformation rules for

```
<xs:element name="STRBAC">
  <xs:complexType>
  <xs:sequence>
     <xs:element name="Users" type="userType" />
     <xs:element name="Roles" type="roleType" />
     <xs:element name="Permissions" type="permissionType" />
     <xs:element name="Times" type="timeType" />
     <xs:element name="Locations" type="locationType" />
     <xs:element name="UserRoleAssignment" type="URAType" />
     <xs:element name="PermissionRoleAquire" type="PRAType" />
     <xs:element name="RoleHierarchy" type="RHType" />
     <xs:element name="LocationHierarchy" type="LHType" />
     <xs:element name="SoDR" type="SODRType" />
     <xs:element name="SoDP" type="SODPType" />
     <xs:element name="CardinalityRole" type="CRType" />
     <xs:element name="CardinalityPermission" type="CPType" />
  </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure 4.1: Elements of the STRBAC Metamodel

mapping the elements of the STRBAC metamodel to the elements of the Alloy metamodel.

4.3 Transformation Rules

This section describes the second stage of the model transformation process where any STRBAC models that conform to the STRBAC metamodel transform into Alloy. This requires us to define a set of 13 transformation rules to map the elements of the STRBAC into Alloy features – one for each element. These transformation rules are written in Java. Figure 4.2 depicts the correspondence between the elements in the STRBAC and Alloy. Next we describe these rules to demonstrate challenging aspects of the transformation.

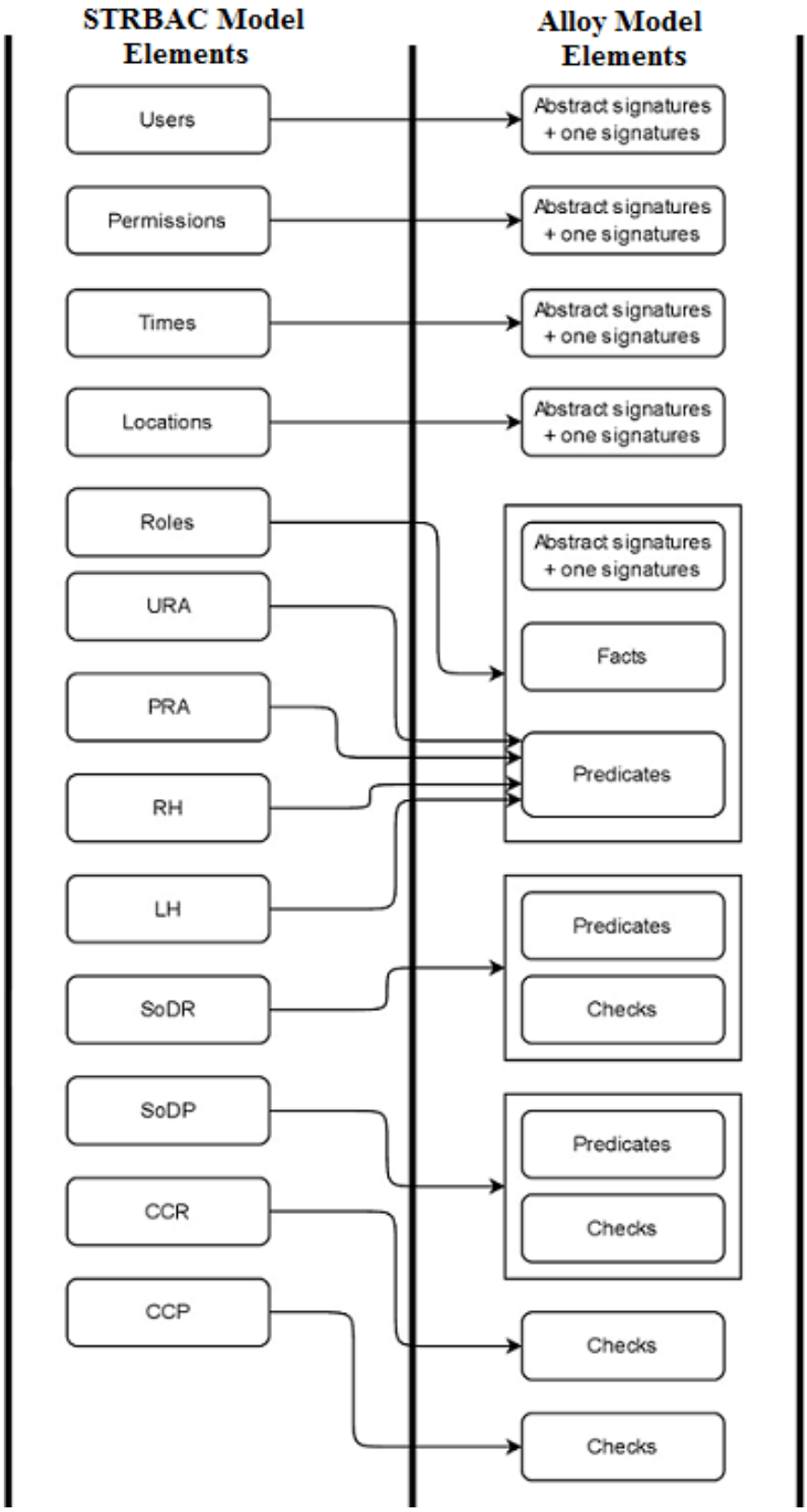

Figure 4.2: Overview on the Transformation Rules

4.3.1 Rule 1: Transformation of Users to Alloy

As previously established, 'Users' in the STRBAC model represents a finite set of users. As depicted in Figure 4.2, users are transformed into an equivalent Alloy code which consists of an `abstract signature` and one or more `one extends signatures`. More precisely, the set of users i.e. Users=$\{u_1, u_2, ..., u_m\}$, will be mapped into an `abstract signature` and every element in the set of users i.e. u_1 will be mapped into `one extends signature`, as depicted in Figure 4.3.

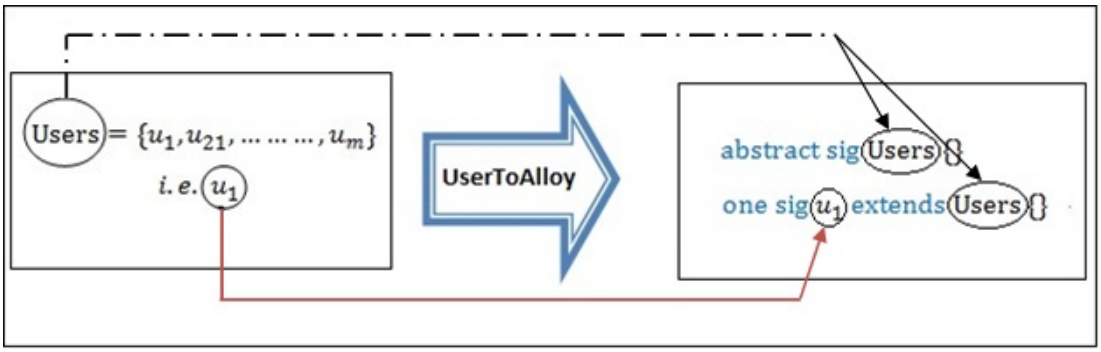

Figure 4.3: Transformation of Users to Alloy

For example, the set of users in the SECURE bank system, as described in Section 4.1, consists of the following users `(Dave, Sarah, Hanna, Mark)` and will be transformed into the following Alloy code.

```
abstract sig Users {}
one sig Dave extends Users {}
one sig Sarah extends Users {}
one sig Hanna extends Users {}
one sig Mark extends Users{}
```

Figure 4.4: Alloy code for the set of users in the SECURE bank system

This means that the `abstract signature` Users has no elements except those belong to its extension `(Dave, Sarah, Hanna, Mark)`. These elements are unique atoms because they are specified as singletons. Figure 4.5 presents the transformation rule that maps users to Alloy, which is written in Java.

```
import JAXB2.STRBAC;
import JAXB2.UserType;
import sitra.Rule;
import sitra.Transformer;
public class Users2Alloy implements Rule{
public boolean check(Object source) {
  if(source instanceof STRBAC.Users){ return true; }
  else{ return false; } }
public String build(Object source, Transformer t) {
  try{ STRBAC.Users users = (STRBAC.Users)source;
  String userAlloy = "abstract sig User{}\n";
  abstract user sig String innerAlloy = "";
int size = users.getUser().size();
int count = 0;
if(users.getUser().size() > 0){
for(UserType user : users.getUser()){
if(size > 1){ innerAlloy += user.getUserId() + ", "; count ++;
if(count == 7){ innerAlloy+= "\n"; count  = 0; } }
else{ innerAlloy += user.getUserId(); }
size --; }
userAlloy += "one sig " + innerAlloy + " extends User{}\n"; }
else { userAlloy += " "; }
return userAlloy; }
catch(Exception e){ return null; } }
public void setProperties(Object target, Object source,
  Transformer t) { } }
```

Figure 4.5: Code for the Users2Alloy transformation rule

Similarly, the sets of permissions, times and locations are mapped to `abstract signatures` and their elements are mapped to `one extend signatures` in Alloy.

4.3.2 Rule 2: Transforming Roles to Alloy

'Roles' in the STRBAC model is defined as a finite set of roles. As depicted in Figure 4.2, roles are mapped into Alloy code which consists of `abstract signature, one extends signatures, facts` and `predicates`. More precisely, the set of roles, i.e. Roles=$\{r_1, r_2, ..., r_n\}$, will be mapped into an `abstract signature` and every element in the set of roles, i.e. r_1, will be mapped into `one extends signature, fact` and `predicate`, as depicted

in Figure 4.6. Inside the `abstract signature`, four Alloy relations, namely time, location, permissions and users, are defined. These relations are created to represent the relationships between roles, users, permissions, times and locations. Such relationships are expressed by User Role Assignment `(URA)`, Permission Role Acquire `(PRA)`, Role Hierarchy `(RH)` and Location Hierarchy `(LH)` with the help of `predicates` which are generated for every role.

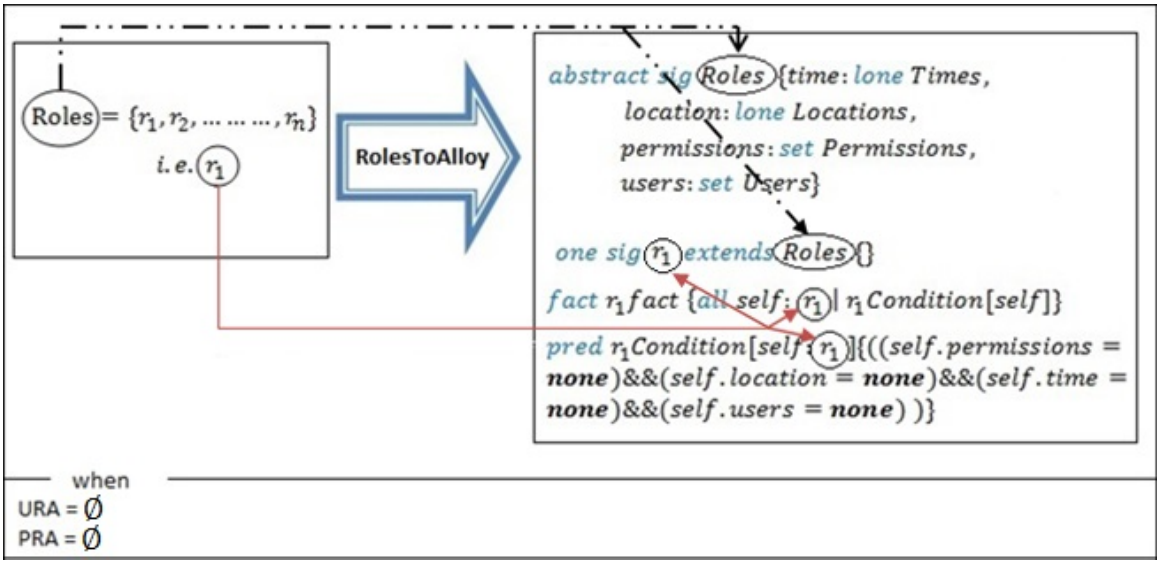

Figure 4.6: Transformation of Roles to Alloy

The multiplicity keyword `lone`, which means zero or one, is used to control the relation between roles and times as well as between roles and locations, while `set`, which means zero or more, is used to control the relation between roles and users as well as between roles and permissions. The four relations inside the `abstract signature`, which represent the assignments between Roles, Users, Permissions, Times and Locations are empty at this stage because the User Role Assignment `(URA)` and the Permission Role Acquire `(PRA)` have not been transformed yet. Therefore, the constant `none`, which represents an empty relation, is used as depicted in Figure 4.6 to specify that all the relations between the role r_1 and other elements are empty.

For example, the set of roles in the SECURE bank system as described in Section 4.1 consists of the users `(Teller, Accounting Manager, Loan Officer, Accountant)` which will be

transformed into the following Alloy code.

```
abstract sig Roles{
  time: lone Times,
  location: lone Locations,
  permissions: set Permissions,
  users: set Users}

one sig AcMan extends Roles{}
fact AcMan_fact{all self: AcMan | AcManCondition[self]}
pred AcManCondition[self: AcMan]{((self.permissions= none)&&
  (self.location= noen)&&(self.time= none)&&(self.users= none))}

one sig Teller extends Roles{}
fact Teller_fact{all self: Teller | TellerCondition[self]}
pred TellerCondition[self: Teller]{((self.permissions= none)&&
   (self.location= noen)&&(self.time= none)&&(self.users= none))}

one sig Accountant extends Roles{}
fact Accountant_fact{all self: Accountant | AccountantCondition[self]}
pred AccountantCondition[self: Accountant]{((self.permissions= none)&&
  (self.location= noen)&&(self.time= none)&&(self.users= none))}

one sig LoanOfficer extends Roles{}
fact LoanOfficer_fact{all self: LoanOfficer | LoanOfficerCondition[self]}
pred LoanOfficerCondition[self: LoanOfficer]{((self.permissions= none)&&
  (self.location= noen)&&(self.time= none)&&(self.users= none))}
```

Figure 4.7: Alloy code for the set of roles in the SECURE bank system

The transformation rule for roles has been defined in this way to constrain the number of Alloy `signatures` in our model and thus to enable larger Access Control models to be created and analysed. This is because it has been found that the tuples produced from `signatures` decrease the scalability of the approach.

4.3.3 Rule 3: Transformation of User Role Assignment (URA)

The User Role Assignment (URA) is a relation that defines how roles are assigned to users based on time and location conditions. As described in the previous section, the

`predicates` created with the Alloy code for the roles are used to represent relationships between users, roles, permissions, times and locations, such as User Role Assignment `(URA)`. Therefore, the execution of this transformation rule will update the `predicates` within the Alloy code for the roles by adding new assignment information, as depicted in Figure 4.2. For example, the transformation of the following User Role Assignment URA=$\{(u_1, r_1, t_1, l_1), ..., (u_m, r_n, t_k, l_o)\}$ will inject the predicates into the Alloy code for roles with new assignment information (i.e. (u_1, r_1, t_1, l_1) will inject the predicate created for the role r_1 with new assignment information), as depicted in Figure 4.8. There are three possible cases for the execution of this rule:

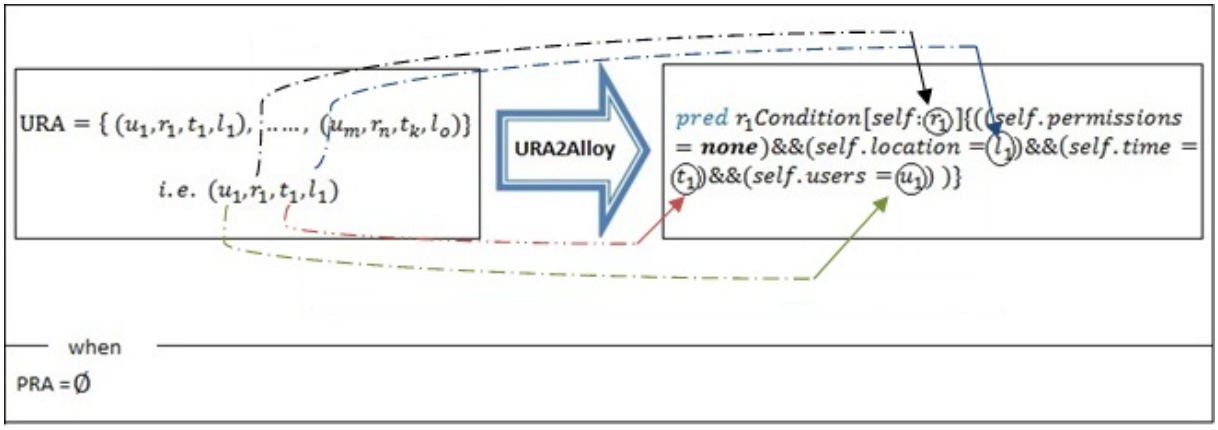

Figure 4.8: Transformation of User Role Assignment

Case 1: If a role is only assigned to one user at the same time and the same location. Figure 4.8 illustrates this case when role r_1 is only assigned to user u_1 at time t_1 and location l_1.

Case 2: If a role is assigned to more than one user at the same time and the same location. Figure 4.9 illustrates this case when r_1 is assigned to users u_1, u_2, ..., u_m at time t_1 and location l_1.

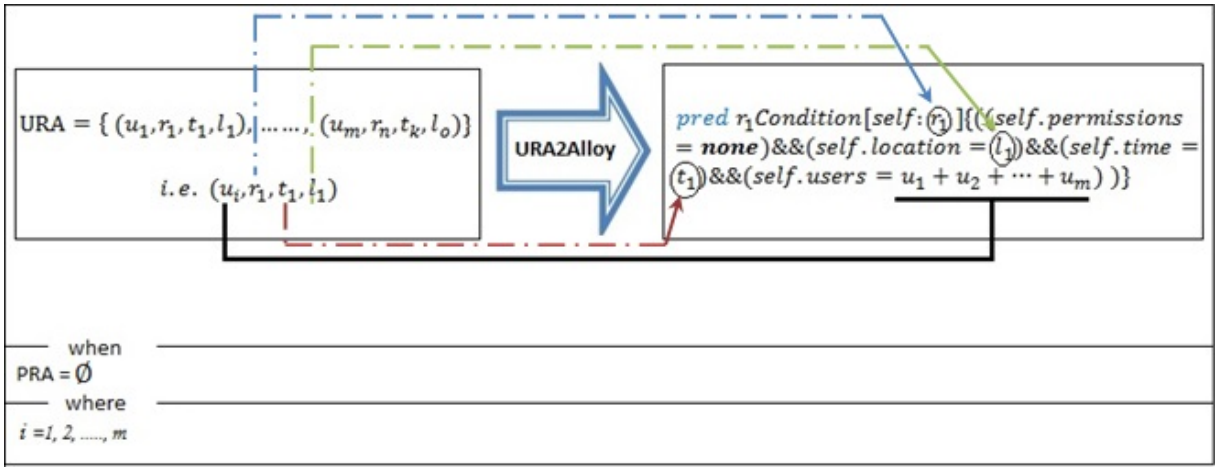

Figure 4.9: Transformation of User Role Assignment – Case 2

Case 3: If a role is assigned to one or more users but at a different time or different locations. Figure 4.10 illustrates this case when r_1 is assigned to users u_1, u_2, ..., u_m at time t_1 and location l_1 and assigned to users u_1, u_2, ..., u_n at time t_2 and location l_1.

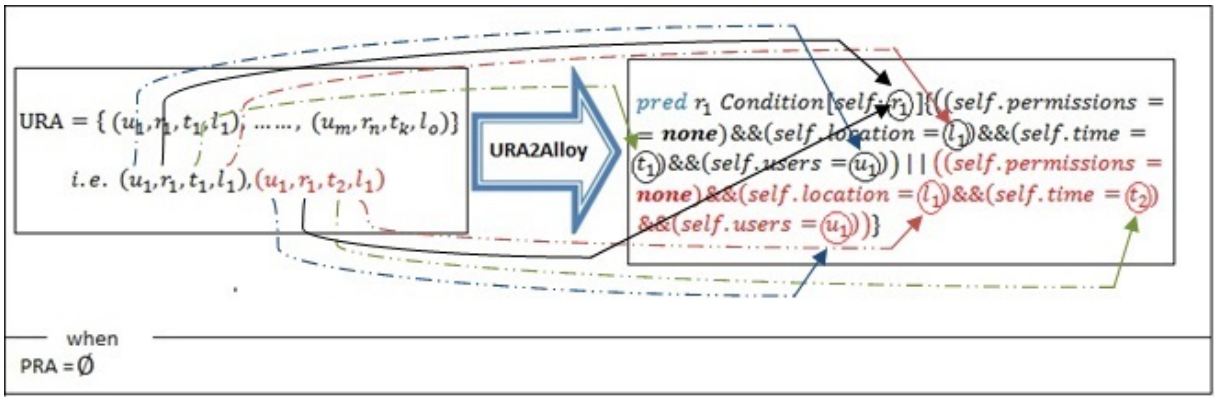

Figure 4.10: Transformation of User Role Assignment - Case 3

In the three cases above, we consider that the Permission Role Acquire has not be transformed yet, which means there will be no permission assigned to the role r_1. Therefore, the constant `none` is still used to represent the relation between roles and permissions, as depicted in Figure 4.8, Figure 4.9 and Figure 4.10. However, if the Permission Role Acquire is already transformed (i.e. permission p_1 is assigned to role r_1 at time t_1 and at location l_1), the transformation of User Role Assignment can be done in the same way as described in case 1, case 2 and case 3, although the constant `none` will be replaced with permission p_1.

4.3.4 Rule 4: Transformation of Permissions Role Acquire (PRA)

The Permission Role Acquire (PRA) is a relation that defines how roles are assigned to permissions based on time and location conditions. As described in Section 4.3.2, the predicates created with the Alloy code for the roles are used to represent the relationships between roles, permissions, times and locations, such as Permission Role Acquire (PRA). Therefore, the execution of this transformation rule will update the predicates within the Alloy code for the roles by adding new assignment information, as depicted in Figure 4.2. For example, The Permission Role Acquire PRA=$\{pra(r_1, p_1, t_1, l_1), ..., pra(r_n, p_x, t_k, l_o)\}$ will inject the predicates into the Alloy code for roles with new assignment information (i.e. $pra(r_1, p_1, t_1, l_1)$ will inject the predicate created for the role r_1 with new assignment information), as depicted in Figure 4.11. There are three possible cases for the execution of this rule:

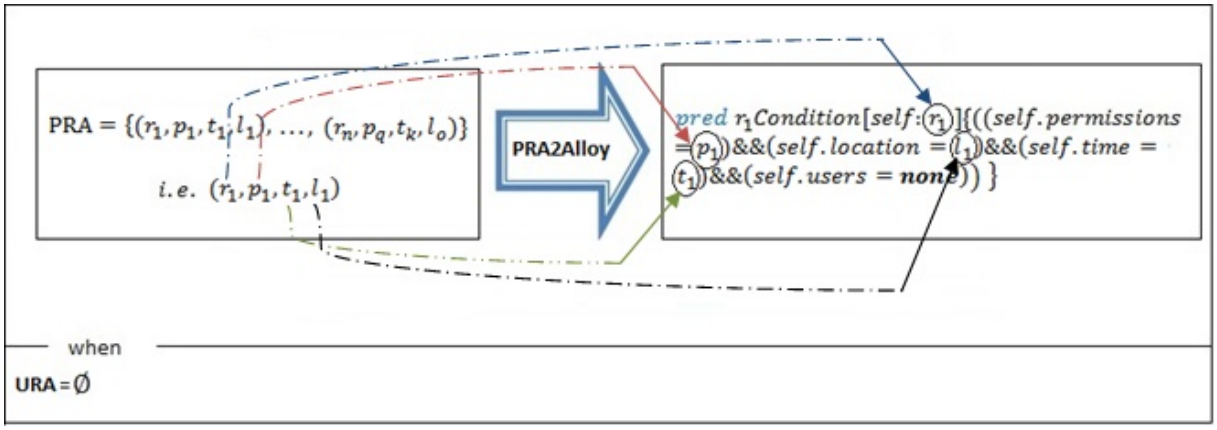

Figure 4.11: Transformation of Permission Role Acquire

Case 1: If a role is only assigned to one permission at the same time and the same location. Figure 4.11 illustrates this case when role r_1 is only assigned to permission p_1 at time t_1 and location l_1.

Case 2: If a role is assigned to more than one permission at the same time and the same location. For example, if role r_1 is assigned to permissions p_1, p_2, ..., p_x at time

t_1 and location l_1. This case is similar to Case 2 in Section 4.3.3 and can be described in the same way.

Case 3: If a role is assigned to one or more permissions but at different times or different locations. For example, if role r_1 is assigned to permissions p_1, p_2, ..., p_q at time t_1 and location l_1 and assigned to permissions p_1, p_2, ..., p_q at time t_2 and location l_1. This case is similar to Case 3 in Section 4.3.3 and can be described in the same way.

In the three cases above we consider that the User Role Assignment has not been transformed yet, which means there will be no user assigned to role r_1. Therefore, the constant `none` is still used to represent the relation between roles and users, as depicted in Figure 4.11. However, if the User Role Assignment is already transformed (i.e. user u_1 is assigned to role r_1 at time t_1 and at location l_1), the transformation of the Permission Role Acquire can be done in the same way as described in case 1, case 2 and case 3, although the constant `none` will be replaced with user u_1.

4.3.5 Rule 5: Transformation of Role Hierarchy (RH)

Role hierarchy, as described in Section 3.1.3, specifies which roles are senior to other roles. This means that a user who has a senior role can also have a junior role and its permissions and the senior role can inherit all of the permissions assigned to the junior role. Therefore, the execution of this transformation rule will update the `predicates` within Alloy code generated for the senior and junior roles with new assignment information. More specifically, the predicates generated for the senior role will be injected with a new permission into the role assignment information, while the predicate generated for the junior roles will be injected with a new user into the role assignment information. For example, The Role Hierarchy RH=$\{rh(r_1 \succeq r_2), ...,\}$ will inject the predicates within Alloy code for Roles with new assignment information (i.e. $rh(r_1 \succeq r_2)$) will inject the predicates generated for the roles r_1 and r_1 with new assignment information. There are

two possible cases for the execution of this transformation rule.

Case 1 If a role is senior to another role/s and the junior role/s is not assigned to any permission. There are two scenarios for this case. The first scenario occurs when no user is assigned to the senior role. In this scenario, the predicates generated for the senior and junior roles will not be updated. The second scenario involves a senior role being assigned to one user or more. To describe this scenario, let us consider that role r_i is senior to role r_j and senior role r_i is assigned to user u_1 at time t_1 and at location l_1. Figure 4.12 and Figure 4.13 depict the predicates generated for the senior role r_i and the junior role r_j following transformation rules 2, 3 and 4.

$$pred\ r_i Condition[self : r_i]\{((self.permissions = \textbf{none})\&\&(self.location = l_1)\&\&(self.time = t_1)\&\&(self.users = u_1))\}$$

Figure 4.12: `Predicate` for role r_i

$$pred\ r_j Condition[self : r_j]\{((self.permissions = \textbf{none})\&\&(self.location = \textbf{none})\&\&(self.time = \textbf{none})\&\&(self.users = \textbf{none}))\}$$

Figure 4.13: `Predicate` for role r_j

The execution of this transformation rule will only inject the predicate generated for the junior role with new assignment information, as illustrated in Figure 4.14, while the predicate for the senior role will not be updated. This happens because of the junior role r_j is not assigned to any permission, then the senior r_i will not inherit any permission.

$$pred\ r_j Condition[self : r_j]\{((self.permissions = \textbf{none})\&\&(self.location = l_1)\&\&(self.time = t_1)\&\&(self.users = u_1))\}$$

Figure 4.14: Updated `predicate` for junior role r_j

Case 2 If a role is senior to another role/s and the junior role/s is assigned to one permission or more. There are also two scenarios for this case. The first scenario occurs when no user is assigned to the senior role. To describe this scenario, let us consider that role r_i is senior to role r_j and junior role r_j is assigned to permission p_1 at time t_1 and

at location l_1. Figure 4.15 and Figure 4.16 depict the `predicates` generated for the senior role r_i and the junior role r_j following transformation rules 2, 3 and 4.

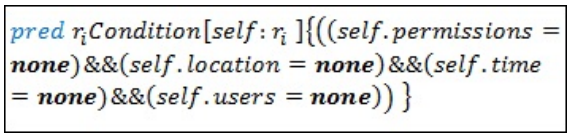

Figure 4.15: `Predicate` for role r_i

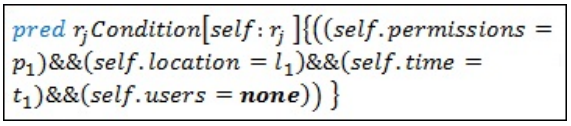

Figure 4.16: `Predicate` for role r_j

The execution of this transformation rule will inject only the `predicate` generated for the senior role with new assignment information, as illustrated in Figure 4.17, while the `predicate` for the junior role will not be updated. This happens because the senior role is not assigned to a user. Therefore, there will be no user to inherit the junior role and its permissions.

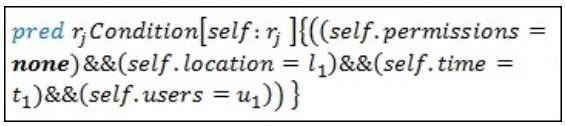

Figure 4.17: Updated `predicate` for senior role r_i

The second scenario occurs when one user or more is assigned to the senior role. To describe this scenario, let us consider that role r_i is senior to role r_j, junior role r_j is assigned to permission p_1 at time t_1 and at location l_1 and senior role r_i is assigned to user u_1 at time t_1 and at location l_1. Figure 4.18 and Figure 4.19 depict the `predicates` generated for the senior role r_i and the junior role r_j using transformation rules 2, 3 and 4.

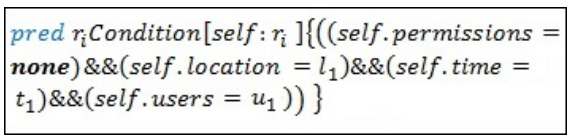

Figure 4.18: Updated `Predicate` for role r_i

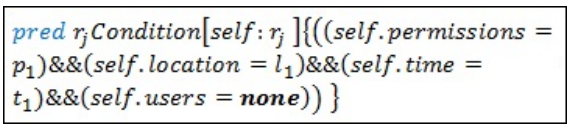

Figure 4.19: Updated `Predicate` for role r_j

The execution of this transformation rule will inject the `predicates` generated for the senior r_i and the junior roles r_j with new assignment information, as illustrated in Figure 4.20 and Figure 4.21.

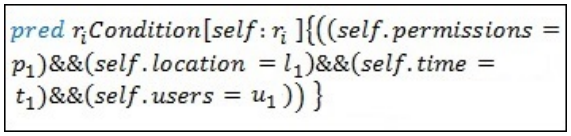

Figure 4.20: Updated `Predicate` for role r_i

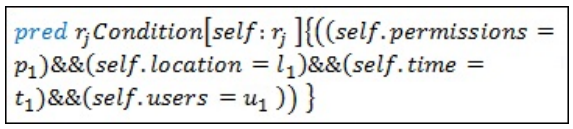

Figure 4.21: Updated `Predicate` for role r_j

4.3.6 Rule 6: Transformation of the Location Hierarchy to Alloy (LH)

The transformation of the Location Hierarchy is similar to the transformation for Role Hierarchy. This is because the Location Hierarchy between two locations, as described in Section 3.1.4, means that if there is an assignment between a user and a role at the outer location and at any point in time, then the same user can have the same role at the inner location at the same point in time. Additionally, if there is an assignment between a role and a permission at the outer location and at any point in time, then the same role can have the same permission at the inner location and at the same point in time. As a result, the transformation of the Location Hierarchy will update the `predicates` within the Alloy code for roles with new assignment information, as depicted in Figure 4.2. In particular, the execution of the transformation rule will inject the `predicates` generated for roles that have a user to role assignment or a role to permission assignment at the outer location with new assignment information. There are two possible cases for the execution of this rule.

Case 1 If a location is an outer location in relation to another location/s and there is a role/s that is assigned to a user/s or a permission/s at the outer location and at any point in time. To describe this case, let us consider that location l_i is an outer location in relation to location l_j and role r_i is assigned to user u_i at the outer location and at time t_i. Figure 4.22 depicts the `predicate` generated for the role r_i by following transformation rules 2, 3, and 4. The execution of this transformation rule will inject the `predicate` presented in Figure 4.22 with new assignment information, as depicted in Figure 4.23.

```
pred r_iCondition[self: r_i ]{((self.permissions =
none)&&(self.location = l_i)&&(self.time =
t_i)&&(self.users = u_i)) }
```

Figure 4.22: `Predicate` for the role r_i

```
pred r_iCondition[self: r_i ]{((self.permissions =
none)&&(self.location = l_i)&&(self.time =
t_i)&&(self.users = u_i)) || ((self.permissions =
none)&&(self.location = l_j)&&(self.time =
t_i)&&(self.users = u_i))}
```

Figure 4.23: Updated `predicate` for the role r_i

Case 2 If a location is an outer location in relation to another location/s and there is no assignment between roles and users or roles and permissions at the outer location. In this case, the execution of this rule will have no effect on the Alloy code.

4.3.7 Rule 7: Transformation of Separation of Duty between Roles to Alloy (SoDR)

The Separation of Duty between Roles (SoDR), as previously established, is a set of constraints over roles which specifies the exclusive set of roles. As depicted in Figure 4.2, Separation of Duty between Roles is mapped into a `predicate` and `checks` in Alloy. In particular, the Separation of Duty over Roles set SoDR=$\{sodr(r_1, r_2, t_1, l_1), ...\}$ is mapped into a `predicate` in Alloy, and every element in the set of SoDR (i.e. $sodr(r_1, r_2, t_1, l_1)$) is mapped into a `check` in Alloy, as depicted in Figure 4.24. These `checks` can be used later to verify whether the SoDR constraints hold or do not hold. Here, we did not specify the scope, since the Users, Roles, Permissions, Times and Locations signatures are specified as abstract and their elements are specified as singleton.

```
pred SoDR[r_i, r_j: Roles, l: Locations, t: Times]
{all u: Users | ((u in r_i.users) &&(t in r_i.time)&&
(l in r_i.location) => ((u not in r_j.users) &&
(t in r_j.time)&&(l in r_j.location))

check {SoDR[r_1, r_2, t, l]}
```

Figure 4.24: Alloy code generated for SoDR

4.3.8 Rule 8: Transformation of Separation of Duty between Permission to Alloy (SoDP)

The Separation of Duty between Permission (SoDP), as previously established, is a set of constraints over permissions which specifies the exclusive set of permissions. The transformation of Separation of Duty between Permission is similar to the transformation of Separation of Duty between Roles. In particular, the set of Separation of Duty between Permissions SoDP=$\{sodp(p_1, p_2, t_1, l_1), ...\}$ is mapped into `predicates` in Alloy, and every element in the set SoDP (i.e. $sodp(p_1, p_2, t_1, l_1)$) is mapped into a `check` in Alloy.

4.3.9 Rule 9: Transformation of Cardinality Constraint over Roles (CCR)

As described in Section 3.1.7, the Cardinality Constraint over Roles (CCR) is a set of constraints over roles which specifies the maximum number of users that can be assigned to certain roles. As depicted in Figure 4.2, the set of Cardinality Constraints over Roles (CCR) is mapped into `checks` in Alloy. More specifically, every element in the set of Cardinality Constraints over Roles CCR=$\{ccr(r_1, t_1, l_1, m), ...\}$, i.e. $ccr(r_1, t_1, l_1, m)$, is mapped into a `check` in Alloy, as depicted in Figure 4.25. These `checks` are used to verify whether the CCR hold or do not hold. Here, once again, the scope has not been specified because the Users, Roles, Permissions, Times and Locations signatures are specified as abstract and their elements are specified as singleton.

```
check {((l1 in r1.location)&&(t1 in r1.time) =>
(# r1.users < m))}
```

Figure 4.25: Alloy code generated for CCR

4.3.10 Rule 10: Transformation of Cardinality Constraints over Permissions (CCP)

The CCP is a set of constraints which specifies the maximum number of roles that can be assigned to certain permissions. The transformation of CCP is similar to the transformation of CCR. As depicted in Figure 4.2, the set of Cardinality Constraints over Permissions (CCP) is mapped into checks in Alloy. More specifically, every element in the set of Cardinality Constraints over Permissions CCP=$\{ccp(p_1, t_1, l_1, n), ...\}$ i.e $ccp(p_1, t_1, l_1, n)$ is mapped into a check in Alloy, as depicted in Figure 4.2. These checks are used to verify whether the CCP hold or do not hold.

4.4 Chapter Summary

Chapter 4 presented the model transformation AC2Alloy, which transforms the Access Control specifications in the context of the STRBAC model into Alloy for the purpose of analysis. In particular, this chapter first introduced the running example SECURE Bank system, which is used to demonstrate our approach, and then described the set of transformation rules that map the elements of the STRBAC model presented in Section 3.1 into Alloy. The model transformations presented in this chapter were implemented in order to automate the creation of Alloy from the STRBAC models. In the following chapter, we shall describe the implementation of our approach, which is developed as an eclipse plug-in called AC2Alloy.

CHAPTER 5

AC2ALLOY: IMPLEMENTATION OF A TRANSFORMATION FRAMEWORK

This chapter will introduce the implementation of the transformation rules presented in the previous chapter. The approach has been implemented as a plug-in called AC2Alloy. In order to demonstrate the feasibility of our approach, we make use of the running example presented in Section 4.1.

5.1 AC2Alloy Architecture

Figure 5.1 depicts an overview of the approach as described in the previous chapters. In particular, a set of transformation rules has been defined to conduct the model transformation. The transformation rules map various elements of the STRBAC metamodel into instances of the Alloy metamodel. These rules are then executed via the Simple Transformer (SiTra) transformation engine. Finally, the model transformation is implemented as an eclipse plug-in application called AC2Alloy. If a STRBAC model is provided as an input to AC2Alloy, an Alloy model is automatically generated as an output, following which the produced Alloy model can be analysed using the Alloy analyser. The rest of this section describes the implementation of AC2Alloy in more detail.

Figure 5.2 shows that AC2Alloy makes use of several distinct technologies. First of all, the tool transforms the STRBAC specification, which is described in Section 3.1, to an

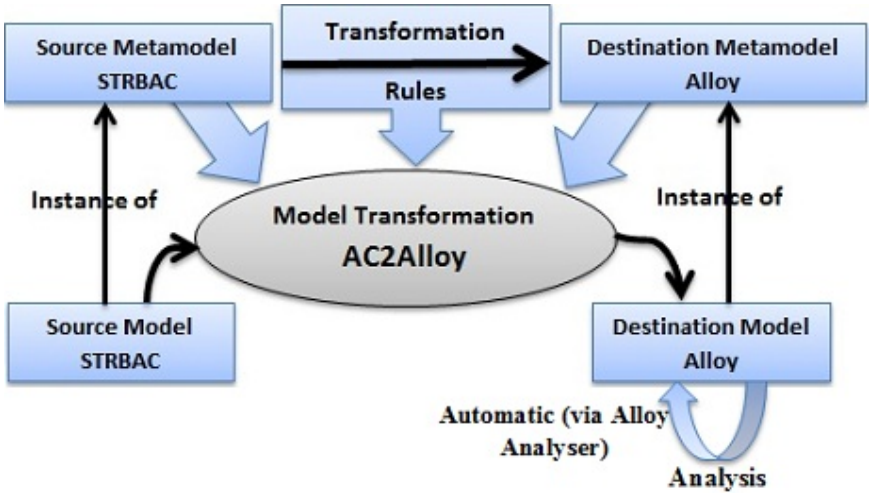

Figure 5.1: Overview of the Approach

XML representation at the Eclipse GUI level. Secondly, it makes use of Java architecture for XML Binding (JAXB) [117], which is an open source library for providing a fast and convenient way to bind between XML schemas and Java representations, thus making it easy for Java developers to incorporate XML data and processing functions in Java applications. The JAXB library is used to take the XML [118] representation of the STRBAC specification and to produce Java objects for the STRBAC specification. It then makes use of the Simple Transformer (SiTra) to transform the Java objects generated for the STRBAC to Alloy. Finally, the analysis of the generated Alloy model can be carried out using the Alloy analyser.

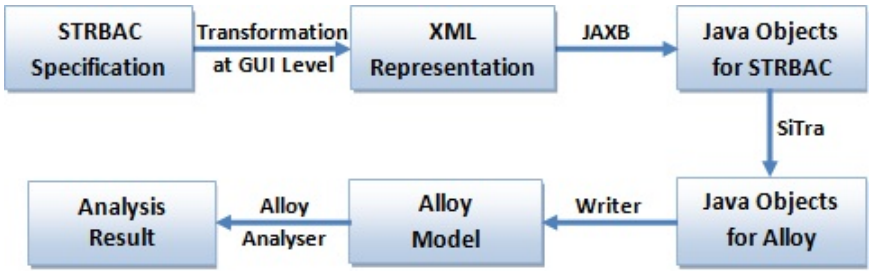

Figure 5.2: Technologies used during the development of AC2Alloy

5.1.1 XML for representing the STRBAC Model

Extensible Markup Language (XML) [118] is used to represent the STRBAC specification, as described in the previous section. It is the technology of choice because it has the following capabilities:

- Heterogeneity (Each record can hold different fields). This is necessary, as types such as User Role Assignment will need to hold different fields such as User, Role, Time and Location.
- Extensibility (New types of data can be added when needed). This is necessary, as an Access Control model is likely to have many users, roles, times, locations and permissions added.
- Flexibility (Fields can vary in size and configuration). This is necessary, as an Access Control modelling one organisation will very likely be different to another organisation.
- Can use XSD files to ensure that correctly conforming XML is used.
- It is used and understood worldwide.
- There are many useful programming libraries that use XML files for various operations that can be taken advantage of.

Figure 5.3 shows an instance of the XML file that represents the STRBAC specification in the running example presented in Section 4.1.

5.1.2 Parsing XML Data into Java Objects

The parsing process, also known as syntax analysis, refers to analysing an input sequence (read from a file or a keyboard, for example) in order to determine its grammatical structure with respect to a given formal grammar. In this case, parsing is required to transform the XML data generated from the STRBAC model, which an instance of the XSD data

```
<users>
<user user_id= "Dave "/>
<user user_id= "Hanna "/>
<user user_id= "Mark "/>
<user user_id= "Sarah "/>
</users>
```

Figure 5.3: XML for the set of Users

that represents the STRBAC metamodel into Java objects that could be manipulated by SiTra. Parsing is performed using Java Architecture for XML Binding (JAXB) [117], a technology that provides tooling to enable users to convert XML documents to and from Java objects.

5.1.3 SiTra for Executing the Transformation Rules

Following on from the process of parsing the XML data into Java objects, the actual model transformation process takes place. This process is conducted using the Simple Transformer (SiTra), which is a Java library that provides a lightweight framework for performing transformations. SiTra has become the technology of choice for executing transformation rules because it is easy to use and is able to conduct complex transformations. As described in Chapter 2, SiTra consists of two interfaces, the rule interface and the transformer interface. The rule interface should be implemented for each transformation rule written, while the transformer interface provides the framework of methods that carry out the transformation. The rule interface consists of three methods: *check()*, *build()* and *setProperties()*. If the rule is applicable for the source element in question, the *check()* method of the rule implementation returns true and the *build()* method is executed. The build() method generates the target model element. The *setProperties()* is used to set the attributes and links for the newly created target element.

5.2 AC2Alloy: An Eclipse Plug-in

The model transformation framework described in the previous chapters was implemented as an eclipse plug-in called AC2Alloy. Figure 5.4 depicts a screen shot of the AC2Alloy tool and shows that there are three panels. The left panel allows the user to add STRBAC elements or to modify them accordingly. The middle panel produces an XML representation and readable English descriptions for the specification of the uploaded STRBAC and also allows users to upload XML files. The third panel presents the automatic creation of an Alloy model and also allows the user to modify the model, if needed, before the analysis is applied.

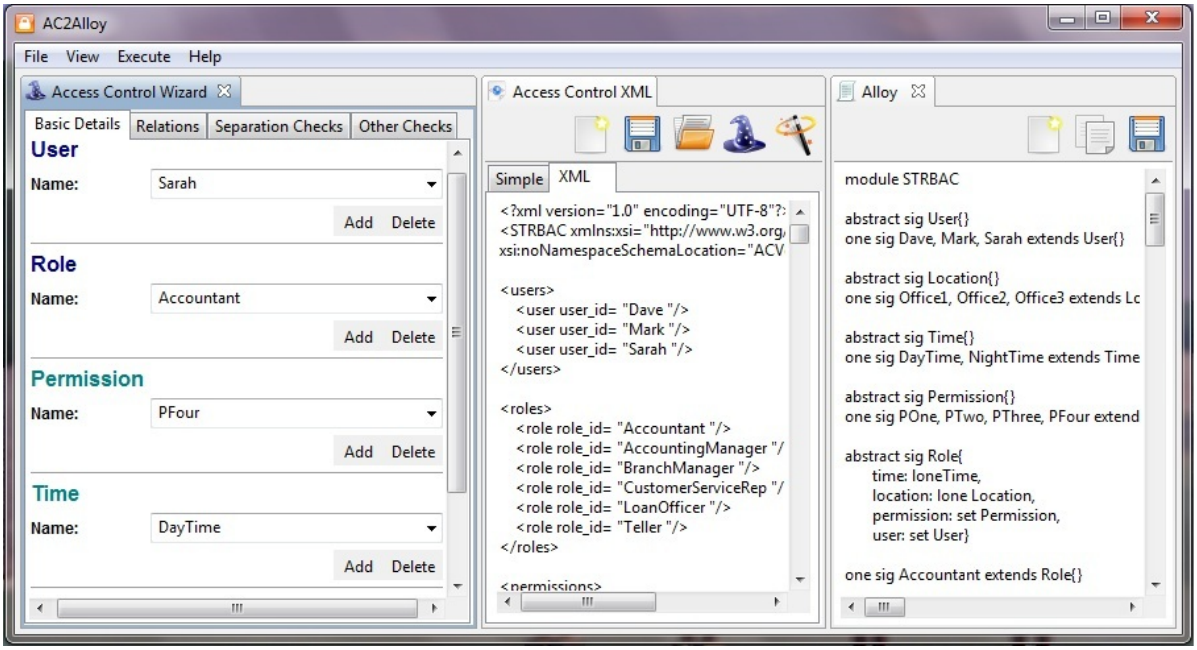

Figure 5.4: Screen Shot of AC2Alloy

5.2.1 Consistency Statements that can be analysed using AC2Alloy

The AC2Alloy tool automatically generates several Alloy checks that can be used to analyse the produced Alloy model. The following list highlights some examples of the Alloy checks that will be generated using AC2Alloy for the purpose of analysis.

- Separation of Duty over Roles checks: these checks can be used to identify in-

consistencies that might be caused due to interactions between Separation of Duty between Roles, User Role Assignment, Role Hierarchy and Location Hierarchy rules.

- Separation of Duty over Permissions checks: these checks can be used to identify inconsistencies that might be caused due to interactions between Separation of Duty between permissions, User Role Assignment, Permission Role Acquire, Location Hierarchy and Role Hierarchy rules.

- Cardinality Constraints checks: these checks can be used to identify inconsistencies that might occur due to interactions between Cardinality Constraints, User Role Assignment, Permission Role Acquire, Location Hierarchy and Role Hierarchy rules.

- User Role Assignment checks: these checks can be used to identify if a user *(u)* is assigned to role *(r)* at time *(t)* and location *(l)*, but he/she does not have any permission at that point in time or at that location.

- Permission Role Acquire checks: these checks can be used to identify if a role *(r)* is assigned to a permission *(p)* at time *(t)* and location *(l)*, but the role *(r)* is not assigned to any user at that point in time or at that location.

5.3 AC2Alloy in Practice

By interfacing with the Alloy analyser, the AC2Alloy implementation is able to analyse the Access Control specification in the context of the STRBAC model. In this section, the running example presented in Section 4.1 is used to demonstrate the feasibility of our approach. In addition, a performance study is carried out in order to evaluate our approach.

5.3.1 Generating Alloy Model from the Running Example

In order to use AC2Alloy to auto-generate Alloy from the STRBAC specification in the context of the running example of the SECURE bank system, the following steps will be carried out. Firstly, the user or AC2Alloy should provide an AC2Alloy description of the STRBAC specification of the SECURE bank system. Secondly, the STRBAC specification should be transformed to an XML representation, which can then be saved and transformed into an Alloy model, thus allowing for powerful analysis to be conducted using the Alloy analyser.

Figure 5.5 depicts part of the Alloy code generated using AC2Alloy from the STRBAC specification in the running example. The complete Alloy code for the running example is presented in Appendix B.

```
abstract sig Users {}
one sig Dave, Sarah, Hanna, Mark extends Users {}

abstract sig Permissions {}
one sig RWTF, RWLF, RWAF, RWAMF extends Permissions {}

abstract sig Times {}
one sig DayTime, NightTime extends Times {}

abstract sig Locations {}
one sig office1, office2 extends Locations {}

abstract sig Role {time: lone Times,
        location: lone Locations,
        permissions: set Permissions,
        users: set Users }

one sig Teller extends Role {}
fact TellerFactfall self:Teller {TellerCondition[self]}
pred TellerCondition[self:Teller]{((self.permissions= RWTF)&&
       (self.location= Office2)&&(self.time= DayTime)&&
       (self.users= Sarah))}
```

Figure 5.5: Part of the Alloy code for the SECURE bank system

5.3.2 Model Analysis

Once the STRBAC specification in the context of the SECURE bank system has been transformed into Alloy, we need to formally ensure that there are no inconsistencies or semi-consistencies in the specification. We carry out this formal analysis using an Alloy analyser. The generated Alloy model using AC2Alloy contains several Alloy checks to detect inconsistencies and semi-consistencies in the specification. For example, the Separation of Duty constraint between the roles `Teller` and `Loan Officer` in the SECURE bank specification, as presented in Section 4.1.1, is mapped into the following Alloy formula.

```
pred SODR[r1, r2: Roles, l: Locations, t: Times]{
     all u: Users |((u in r1.users) &&(t in r1.time)&&
     (l in r1.location))=>((u not in r2.users)&&
     (t in r2.time)&&(l in r2.location)) }

SoDR1 :check {
 SODR[Taller, LoanOfficer, DayTime, Office2]}
```

Figure 5.6: Alloy formula for the SoD between `Teller` and `Loan Officer`

The execution of the Alloy `check` presented in Figure 5.6 shows that the Alloy analyser picked up a counterexample, as depicted in Figure 5.7, which shows that the specification is inconsistent because the two conflicting roles `Teller` and `Loan Officer` are assigned to the same user `Sarah` during time `DayTime` and at the location `Office2`, due to the direct user-to-role assignment. Therefore, in order to avoid this situation, the system designer can remove either the assignment between the user `Sarah` and the role `Loan Officer` or the role `Teller` during time `DayTime` at location `Office2`. The analysis was performed without specifying the scope of the model, because the Users, Roles, Times and Locations `signatures` are specified as abstract and `Teller, LoanOfficer, DayTime` and `Office2 signatures` are specified as singleton.

Another similar example is that the Separation of Duty constraint between the roles `Accountant` and `Teller` at the time `DayTime` and at location `Office1` will be transformed into

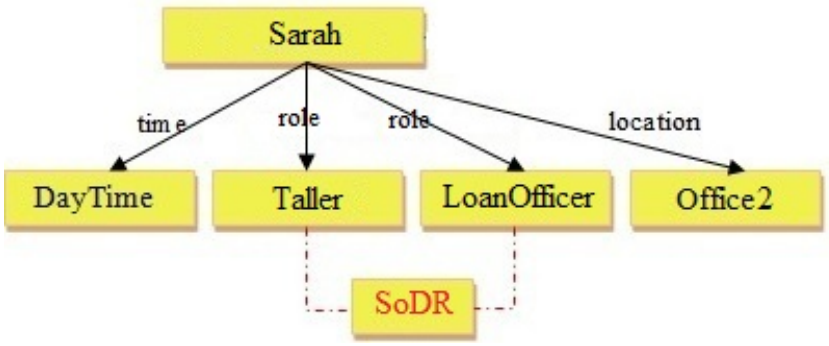

Figure 5.7: Counterexample for SoDR1 check: Inconsistency Detection

a `check` in Alloy as depicted in Figure 5.8.

```
SoDR2 :check {
 SODR[Accountant, Taller, DayTime, Office1]}
```

Figure 5.8: Alloy formula for the SoD between `Accountant` and `Teller`

The execution of the above Alloy `check` shows that Alloy Analyser did not find a counterexample as illustrated in Figure 5.9. This means that the Separation of Duty constraint between the roles `Accountant` and `Teller` at the time `DayTime` and at location `Office1` satisfies all the other rules of the SECURE Bank specification presented in Section 4.1.1.

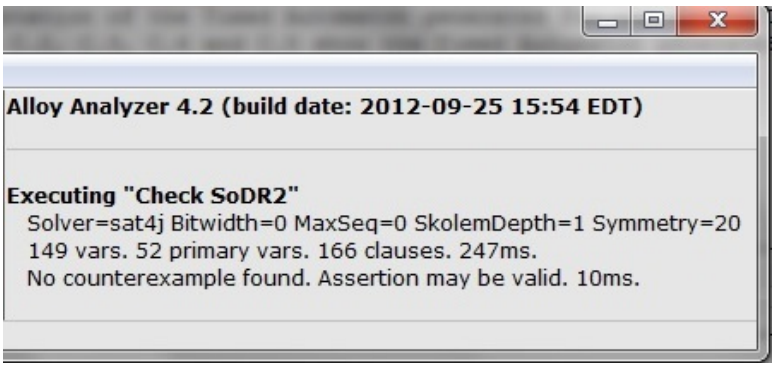

Figure 5.9: The result of the Execution of SoDR2 `check`

Another example of the Alloy `checks`, which will be generated using AC2Alloy, in order to perform the analysis on the SECURE bank system, is that the cardinality constraint

over the Accountant role during time DayTime and at location Office1 is mapped into the following Alloy formula. The following constraint means that the Accountant role can be assigned to only one user during time DayTime and at location Office1.

```
CC1 :check {((Office1 in Accountant.location)&&
      (DayTime in Accountant.time)=>(#Accountant.user <2))}
```

Figure 5.10: Alloy Formula for the Cardinality Constraint over the Accountant role

The execution of the above Alloy checks shows that the Alloy analyser picked up a counterexample, as depicted in Figure 5.11, which shows that the specification is inconsistent because more than one user (Mark and Hanna) is assigned to the Accountant role, which is not permissible according to the Cardinality constraint over the Accountant role. The Accountant role is assigned to the user Hanna because of the direct assignment using User Role Assignment, while it is assigned to the user Mark because Mark is assigned to the role Accounting Manager during time DayTime and at location Office1, which is senior to the junior Accountant role. Therefore, in order to avoid this situation, the system designer should modify the specification by removing either the assignment between the user Hanna and the Accountant role during time DayTime and at location Office1 or the hierarchy between the roles Accounting Manager and Accountant during time DayTime and at location Office1. Here also, the analysis was performed without specifying the scope of the model because the Users, Roles, Times and Locations signatures are specified as abstract and Accountant, DayTime and Office1 signatures are specified as singleton. The

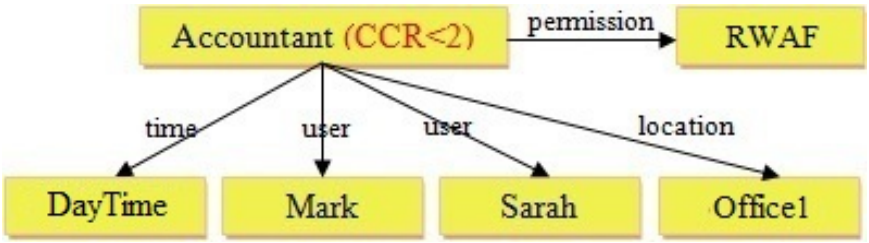

Figure 5.11: Counterexample for CC1 check: Inconsistency Detection

5.4 Chapter Summary

In this Chapter we presented the implementation of the model transformation AC2Alloy. The description of AC2Alloy architecture and the technologies used in the development of AC2Alloy are discussed in Section 5.1. We followed this by providing a description of the eclipse plug-in AC2Alloy. We also described several types of consistency statements that could be analysed using AC2Alloy. In section 5.3 we described the usage of AC2Alloy for modelling and analysing Access Control specifications to ensure the consistency of the specifications.

CHAPTER 6

AC2UPPAAL: TRANSFORMATION BETWEEN THE STRBAC MODEL, TIMED AUTOMATA AND TCTL

In this chapter we describe a new model transformation framework called AC2Uppaal which transforms the Access Control specification in the context of the STRBAC model into Timed Automata and Timed Computation Tree Logic (TCTL) statements. The Timed Automata network and TCTL statements can then be modelled and verified using the Uppaal model checker to detect conflicts in the STRBAC specification. To conduct the model transformation, this chapter describes a set of transformation rules that have been defined to map the elements of the STRBAC, Timed Automata and TCTL. This is followed by an explanation about the implementation and performance of the model transformation with the help of the running example presented in Section 4.1.

6.1 AC2Uppaal: Model Transformation

AC2Uppaal is a tool that makes use of Model-Driven Architecture (MDA) techniques for integrating the STRBAC, Timed Automata and Timed Computation Tree Logic (TCTL) into a single tool. It enables the user to create STRBAC specifications and transforms them into XML representation. The XML representation of the STRBAC model is then automatically transformed into Timed Automata and TCTL statements. The produced

Timed Automata and TCTL statements will be modelled and verified using the Uppaal model checker to detect conflicts in the STRBAC specification. The model transformation process is hereby described in the following three stages:

- Metamodels for the source and the target models.
- Transformation rules to map the elements of the STRBAC and Alloy.
- Implementation of the transformation rules.

As described in Section 4.2, conducting a model transformation requires metamodels for the source and the target to be constructed to specify the elements of the source model that will be mapped to the elements of the target model. The source's metamodel, STRBAC, has already been defined in the previous chapters. The metamodels of the target, Timed Automata and TCTL on the other hand are declared in Java. In Section 2.4 we presented a Timed Automata metamodel which included most of the elements that will be used in our mapping.

6.2 Transformation Rules

In this section, we introduce the transformation rules that we have defined to map the elements of the STRBAC to Timed Automata and TCTL. The transformation of the STRBAC to Timed Automata and TCTL is divided into two phases. The first phase transforms some of the STRBAC's features, such as times and users, into Timed Automata, while the second phase generates TCTL statements from STRBAC features such as Permission Role Acquire. With the help of the running example discussed in Section 4.1, we describe how various features of the STRBAC model are mapped to Timed Automata and TCTL statements.

6.2.1 Phase 1: Mapping STRBAC features to Timed Automata

The process of transforming some features of the STRBAC model to Timed Automata (TA) involves two kinds of Timed Automata. The first describes the cycle of times defined in the set of times, while the second describes the set of users and the user role assignment information.

6.2.1.1 Producing a Timed Automaton that captures STRBAC times and their evolution

The set of times, which is one of the basic components of the STRBAC model, is mapped to a Timed Automaton (TA) called 'Times'. For example, if a given set of Times i.e. Times=$\{t_1, t_2, ...t_m\}$ is mapped, then the execution of the transformation rule will produce a Timed Automaton called 'Times', as described in Figure 6.1.

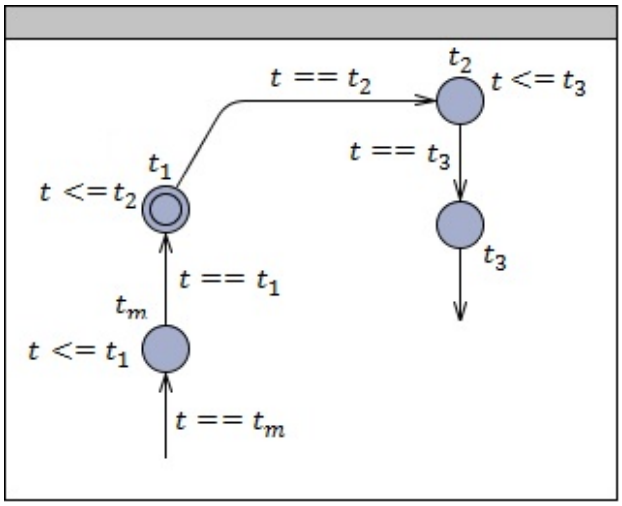

Figure 6.1: Timed Automaton for Times

For example, in the SECURE bank system the set of times is divided into two periods `DayTime` – from 9:00 to 16:00 – and `NightTime` – from 16:00 to 9:00. These data are transformed into a Timed Automaton called 'Times', as depicted in Figure 6.2.

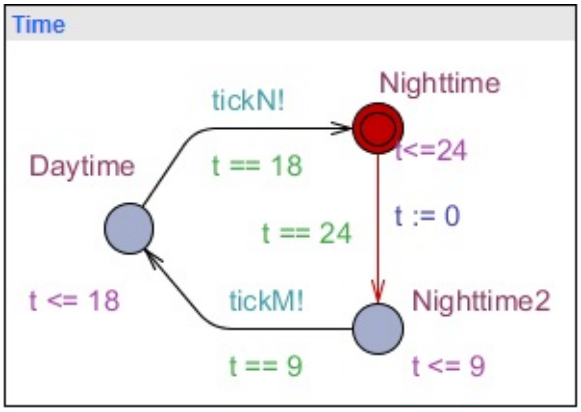

Figure 6.2: Timed Automaton for Times in the SECURE bank specification

6.2.2 Transforming the STRBAC Users and User Role Assignment information to a Timed Automaton

The process of transforming the Set of Users and User Role Assignment information into a Timed Automaton involve the following two steps. Step 1: for every user in the Users set we create a Timed Automaton (TA). For example, every user in the set of Users=$\{u_1, u_2, ..., u_n\}$ (i.e. u_1) will be transformed into a Timed Automata called u_1, as depicted in Figure 6.3. Each TA must represent the activities of the user in the locations at each point in time. The worst scenario will be if we have m times and k locations, in which case we will have $m \times k$ Timed Automata locations, which is the other term used for the states. At each location (i.e. l_j) time can pass, as a result the Timed Automata locations of the form $(t_1, l_j), (t_2, l_j), ..., (t_m, l_j)$ follow each other as time progress. The edges that connect one Timed Automata location (state) to another are called transitions, as illustrated in Figure 6.3. Each Timed Automata location has a role switch which corresponds to an element of User Role Assignment. If a Timed Automata location has a role switch {label=true}, then each edge coming into the Timed Automata location and going out from the Timed Automata location without the same label has an updated expression label=true, while if a Timed Automata location has a role switch {label=false}, then each edge coming into the Timed Automata location and going out from the Timed Automata location without the same label has an updated expression label=false. For example, if user u_1 is assigned to role r at time t_j and location l_k, which

represent the Timed Automata location (t_j, l_k), this means that the Timed Automata location (t_j, l_k) has a role switch {r=true}, while if user u is not assigned to role r at the Timed Automata location (t_m, l_k), then state (t_m, l_k) has a role switch r=false, as depicted in Figure 6.3.

Step 2: the aim of this step is to enhance the generated Timed Automaton from the first step to capture the new User Role Assignment information which will be generated because of the Role Hierarchy. As described in Section 3.1.3, Role Hierarchy means that a user who has a senior role can inherit junior roles and all permissions assigned to those roles, and the senior role can inherit all the permissions assigned to the junior roles, too. This means that new User Role Assignment and Permission Role Acquire functions will be added to the specification. Therefore, the new User Role Assignment information will be used to update the Timed Automaton generated for users, while the new Permission Role Acquire information will be transformed into TCTL in the second phase of the transformation. For example, if role r is senior to role r' and role r is assigned to user u_1 at state (t_j, l_k), then the same user u_1 will have junior role r' at the same state, as depicted in Figure 6.3.

For example, a Timed Automaton will be generated for every user in the set of Users in the SECURE bank system presented in Section 4.1. The Timed Automaton generated for every user captures all the User Role Assignment information related to that user. For instance, a Timed Automaton will be generated for user `Mark`, as depicted in Figure 6.4. This Timed Automata captures the User Role Assignment information (i.e. `(Mark, Accounting Manager, DayTime, Office1)`) presented in Table 4.1, which is related to the user `Mark`.

The Timed Automata generated for every user also captures User Role Assignment information, which is generated because of Role Hierarchy. For example, the following Role Hierarchy in the SECURE bank specification `(Accounting Manager, Accountant, DayTime, Office1)` means that user `Mark`, who has the senior role `Accounting Manager` during the `DayTime` and at `Office1`, based on Table 4.1, can inherit the junior `Accountant` role. The

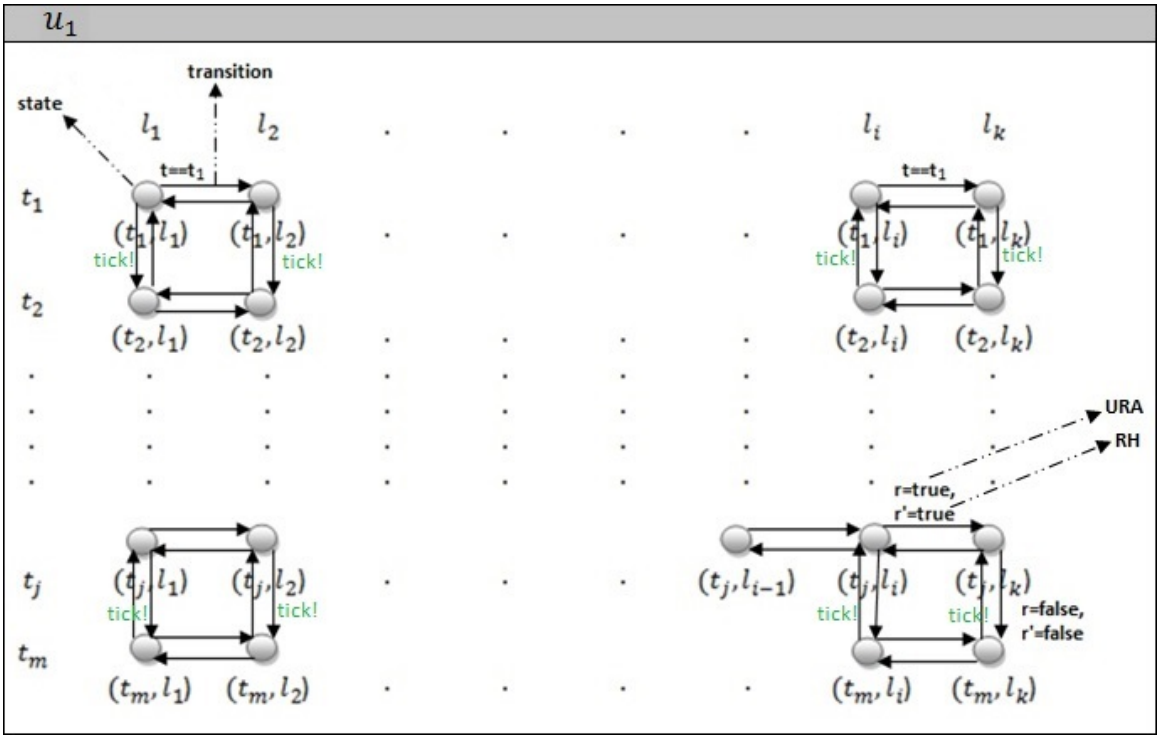

Figure 6.3: Timed Automata generated for the user u_1

new user-to-role assignment information `(Mark, Accountant, DayTime, Office1)` will be expressed using a Timed Automaton. More precisely, the Timed Automaton for user `Mark` depicted in Figure 6.4 will be updated with new assignment information, as illustrated in Figure 6.5.

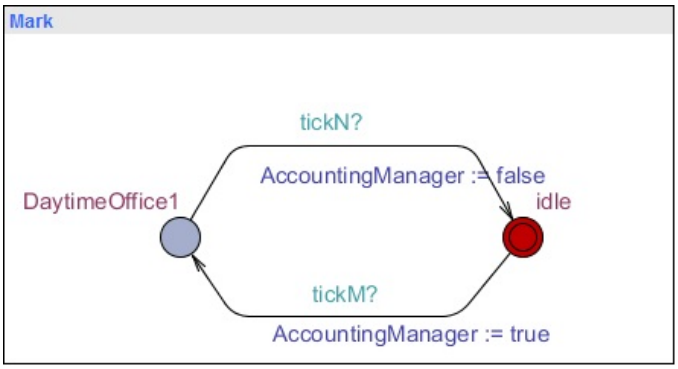

Figure 6.4: Timed Automaton for Mark

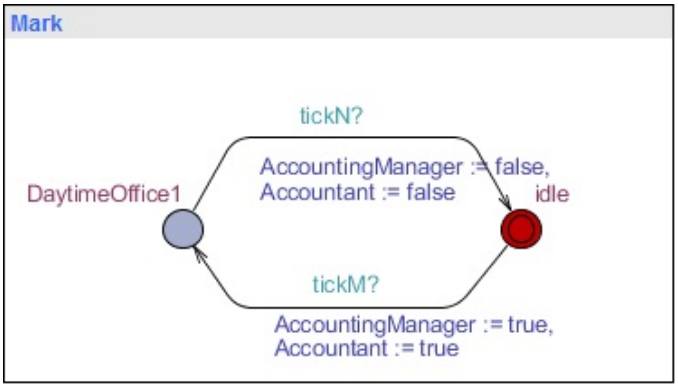

Figure 6.5: Updated Timed Automaton for Mark

6.2.3 Phase 2: Mapping STRBAC features into TCTL

Some features of the STRBAC model, such as Permission Role Acquire and Separation of Duty between Permissions, will be expressed using TCTL, and then they will be used as sub-expressions of queries for Timed Automata.

6.2.3.1 Transforming Permission Role Acquire to TCTL Statements

The Permission Role Acquire (PRA) set will be mapped to TCTL statements. More specifically, every element in the PRA set will be transformed into TCTL statements. For example, every element in PRA=$\{pra(r_1, p_1, t_1, l_1), ..., ura(r_o, p_x, t_m, l_k)\}$ will be transformed into a TCTL statement, i.e. $pra(r_1, p_1, t_1, l_1)$ will be transformed into a TCTL statement as illustrated in Figure 6.6.

$p_1(u) \coloneqq u.t_1\_l_1 \ \& \ u.r_1$
$p_1 \coloneqq \$\backslash bigvee\_\{u \backslash in \ Users\}\$p_1\$(u)\$$

Figure 6.6: TCTL statement for Permission Role Acquire

This means any user u in the set of Users who has role r_1 at time t_1 and at location l_1 can have permission p_1 at the same time t_1 and at the same location l_1.

For example, the first element of Permissions Role Acquire presented in Table 4.2 `(Teller, Read Write Teller File (RWTF), DayTime, Office2)` will be expressed using TCTL as depicted in Figure 6.7.

```
RWTF(u):=u.DayTime_Office2 & u.Teller
RWTF:=$\bigvee_{u \in Users}$ RWTF$(u)$
```

Figure 6.7: TCTL Statements for the Permissions Role Acquire

This means any user u in the set of Users who is a `Teller` can have permission `Read and Write Teller File`, where Users is a finite set, so that we can have a finite expression of `RWTF`. In the SECURE bank system we have a finite set of Users consisting of the following four users: `Dave, Sarah, Mark` and `Hanna`. This will result in a finite set of `RWTF` expressions as follows:

```
RWTF:=Dave.Daytime_Office2 & Dave.Teller
   || Sarah.DayTime_Office2 & Sarah.Teller
   || Hanna.DayTime_Office2 & Mark.Teller
   || Mark.DayTime_Office2 & Hanna.Teller
```

Figure 6.8: An Example of a TCTL Statement for the Permission Role Acquire in the SECURE Bank System

The other elements of the Permission Role Acquire presented in Table 4.2 can be transformed into TCTL statements in the same way.

6.2.3.2 Transforming Role Hierarchy to TCTL Statements

As described in Section 6.2.2, new User Role Assignment and Permission Role Acquire will be added to the specification due to the effect of Role Hierarchy. The new User Role Assignment information will update the Timed Automata generated for users, as illustrated in Section 6.2.2, while the new Permission Role Acquire information will be expressed using TCTL. The new Permission Role Acquire can be transformed into TCTL, as described in Section 6.2.3.1.

For example, in the SECURE bank system the following Role Hierarchy `(Accounting Manager, Accountant, DayTime, Office1)` means that the senior role `Accounting Manager` can

inherit all the permissions assigned to the junior role `Accountant`, such as `RWAF` during the `DayTime` and at `Office1`. The new Permission Role Acquire information `(Accounting Manager, RWAF, DayTime, Office1)` will be represented using TCTL, as shown in Figure 6.9.

```
RWAF2(u):= u.DayTime_Office1 & u.AccountingManager
RWAF2:= $\bigvee_{u \in Users}$ RWAF2$(u)$
```

Figure 6.9: TCTL Statement generated due to the effect of Role Hierarchy

6.2.3.3 Transforming Separation of Duty between Roles into TCTL Statements

The Separation of Duty between Roles can also be expressed using TCTL. For example, every element in the Separation of Duty between Roles set SoDR=$\{sodr(r_1, r_2, t_1, l_1), ...\}$ will be transformed into a TCTL statement, i.e. $sodr(r_1, r_2, t_1, l_1)$ will be transformed into the following TCTL statement:

$$SoDR := And_{u \in Users}(u.t_1\_l_1 \ implies \ NOT \ (u.r_1 \ \& \ u.r_2))$$

Figure 6.10: TCTL Statement for the Separation of Duty between Roles

The TCTL statement presented in Figure 6.10 means that there no user should be in the set of Users with role r_1 and role r_2 at time t_1 and at location l_1.

For example, in the SECURE bank system the following Separation of Duty between Roles `(Teller, Loan Officer, DayTime, Office2)` will be expressed using TCTL, as illustrated in Figure 6.11.

This means that no user should be in the set of Users that has the two conflicting roles `Teller` and `Loan Officer` during `DayTime` and at `Office2`.

Another similar example is that the Separation of Duty constraint between the roles `(Accountant, Teller, DayTime, Office1)` will be transformed into TCTL statement as il-

```
SoDR1:=And_{u \in Users}(u.DayTime_Office2
      implies NOT (u.Teller & u.LoanOfficer))
```

Figure 6.11: TCTL Statement for the SoD between `Teller` and `LoanOfficer`

lustrated in Figure 6.12.

```
SoDR2:=And_{u \in Users}(u.DayTime_Office1
      implies NOT (u.Accountant & u.Teller))
```

Figure 6.12: TCTL Statement for the SoD between `Accountant` and `Teller`

The TCTL statement depicted in Figure 6.12 means that there should be no user in the set of Users that can have the two conflicting roles `Accountant` and `Teller` during `DayTime` and at `Office1`.

6.3 Implementation of the Model transformation

In this section we describe the process of implementing the transformation rules presented in Section 6.2 as an eclipse plug-in called AC2Uppaal.

6.3.1 AC2Uppaal Architecture

Our work is motivated by the need to bridge the gap between the STRBAC model and the formal Timed Automata method, in order to allow for powerful analysis to be carried out. Figure 6.13 depicts an overview of our approach to automate the transformation between the STRBAC, Timed Automata and TCTL based on Model-Driven Architecture (MDA). More precisely, the STRBAC metamodel is mapped into the Timed Automata and TCTL metamodels using a number of transformation rules which are then executed using SiTra and implemented as an eclipse plug-in called AC2Uppaal. If an instance of the STRBAC

metamodel is provided as an input, an equivalent Timed Automata network and TCTL statements are generated. The produced Timed Automata and the TCTL statement can be modelled and analysed using the Uppaal model checker.

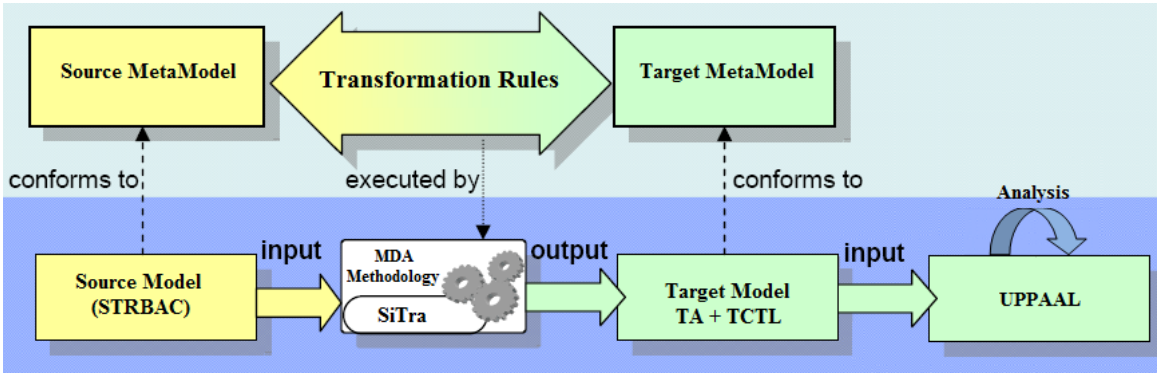

Figure 6.13: Outline of the Approach

The process of generating Timed Automata and TCTL statements from STRBAC models was achieved in the following manner:

- Formalisation of the STRBAC specification. This step has been discussed in Section 3.1.
- Defining a set of transformation rules to map the elements of the STRBAC, Timed Automata and TCTL.
- Representing the STRBAC specification using XML. This step has been described in Section 5.1.1.
- Parsing the XML data into Java objects. This has been discussed in Section 5.1.2.
- Using the Simple Transformer (SiTra) to execute the transformation rules presented in Section 6.2.
- Implementing the model transformation as an eclipse plug-in, so that if a STRBAC model is provided as an input, an equivalent Timed Automata and TCTL will be produced.

6.3.2 AC2Uppaal: An Eclipse Plug-in

The model transformation framework described in the previous sections was implemented as an eclipse plug-in called AC2Uppaal. Figure 6.14 depicts a screen shot of the tool and shows that there are three panels. The left panel allows the user to add STRBAC elements or to modify them. The middle panel produces an XML representation and readable English descriptions for the specification of the uploaded STRBAC and also allows users to upload XML files. The third panel presents the automatic creation of Timed Automata and TCTL and also allows the user to modify both of them, if needed, before the analysis is applied using the Uppaal model checker.

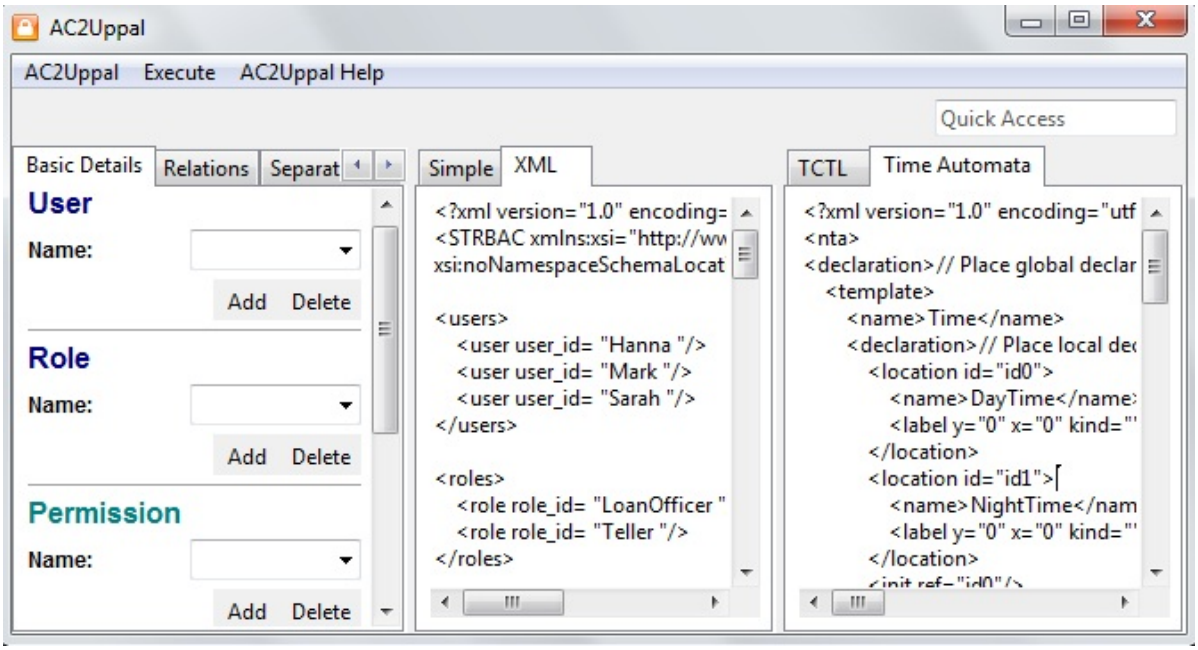

Figure 6.14: Screen Shot of AC2Uppaal

6.3.3 AC2Uppaal in Practice

Interfacing with the Uppaal model checker, the method presented in this chapter is able to analyse the STRBAC specification. Using case studies is one of the most appropriate techniques for evaluating software engineering frameworks and tools, so the running example presented in Section 4.1 is used to demonstrate the feasibility of our approach.

6.3.3.1 Model Analysis

In the previous sections, we illustrated how various elements of the STRBAC model are transformed into Timed Automata and TCTL statements, and the graphical representation of the Timed Automaton as well as the TCTL statements are presented in Appendix C. However, this work would be incomplete without demonstrating how we can analyse the produced Timed Automata models to detect inconsistencies. To ensure the consistency of the SECURE bank system the AC2Uppaal tool will automatically generate several security queries using TCTL. For example, the tool generates the TCTL query shown in Figure 6.15, which is built from the TCTL statements which are generated using the transformation rules presented in Sections 6.2.3.1, 6.2.3.2 and 6.2.3.3.

```
A[] ((RWTF & RWLF & RWAF & RWBMF &
      RWAMF & RWAF2) implies  SoDR1)
```

Figure 6.15: An Example of the TCTL Queries Generated to Analyse the Specification

The TCTL query in Figure 6.15 means that if all the Permission Role Acquire and the Role Hierarchy statements such as `RWTF, RWLF, RWAF, RWBMF, RWAMF` and `RWAF2`, which are generated using the transformation rules presented in Section 6.2.3.1 and Section 6.2.3.2, hold, then the `SoDR1` statement which is generated using the transformation rule presented in Section 6.2.3.3 should also hold for any time and location configuration. This security query and the produced Timed Automata network were implemented using the Uppaal model checker to identify conflicts in the SECURE bank specification. The execution of the Uppaal verifier found that the property was not satisfied, as depicted in Figure 6.16. This means that the SECURE bank system is inconsistent, because the Uppaal model checker has found that the two conflicting roles `Teller` and `Loan Officer` have been assigned by the same user `Sarah` during time `DayTime` and at location `Office2`, which is not permissible according to the `SoDR`. In order to avoid such a situation, the designer should modify the specification by removing either the assignment between the user `Sarah` and

the role `Teller` during time `DayTime` at location `Office2` or the same user `Sarah` and role `Loan Officer` during time `DayTime` at location `Office2`.

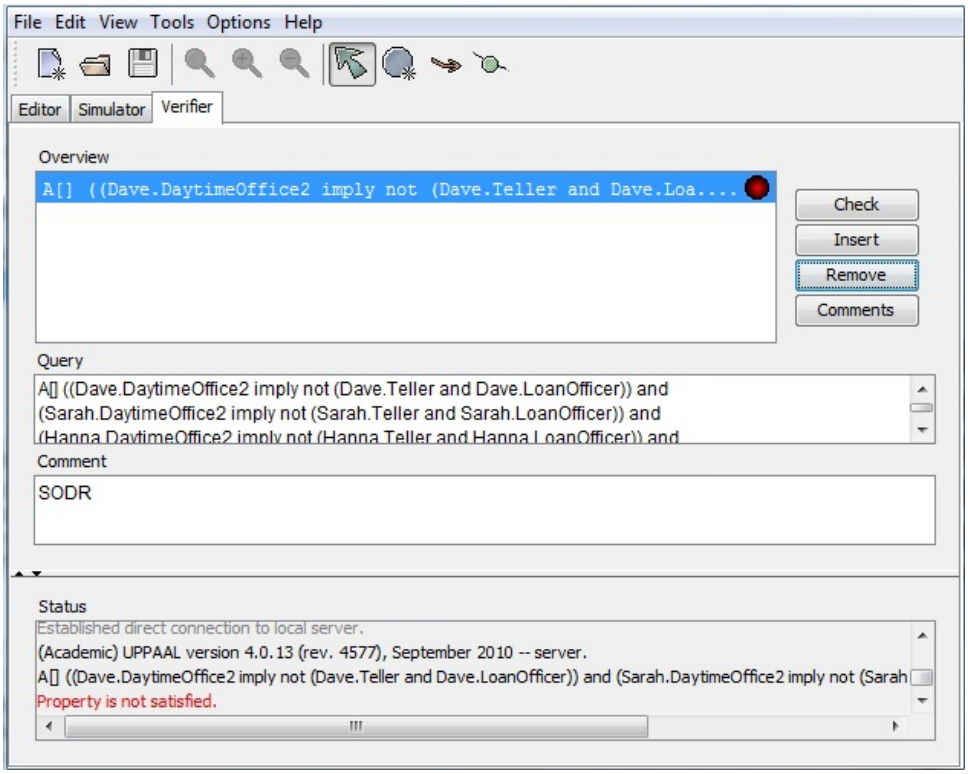

Figure 6.16: UPPAAL Analysis Result

Another example of the TCTL queries that will be generated by the tool AC2Uppaal is illustrated in Figure 6.17. This TCTL query means that if all the Permission Role Acquire and the Role Hierarchy statements such as `RWTF`, `RWLF`, `RWAF`, `RWBMF`, `RWAMF` and `RWAF2`, which are generated using the transformation rules presented in Section 6.2.3.1 and Section 6.2.3.2, hold, then the `SoDR2` statement which is generated using the transformation rule presented in Section 6.2.3.3 should also hold at time `DayTime` and location `Office1`.

```
A[] ((RWTF & RWLF & RWAF & RWBMF &
      RWAMF & RWAF2) implies  SoDR2)
```

Figure 6.17: An Example of the TCTL Queries Generated to Analyse the Specification

The execution of the model checker Uppaal found that the property was satisfied, as depicted in Figure 6.18. This means that the `SoDR2` satisfies all the rules of the SECURE

Bank system. More precisely, this means that the model checker Uppaal did not find any user that has the two conflicted roles, `Accountant` and `Teller`, at the time `DayTime` and at the location `Office1`.

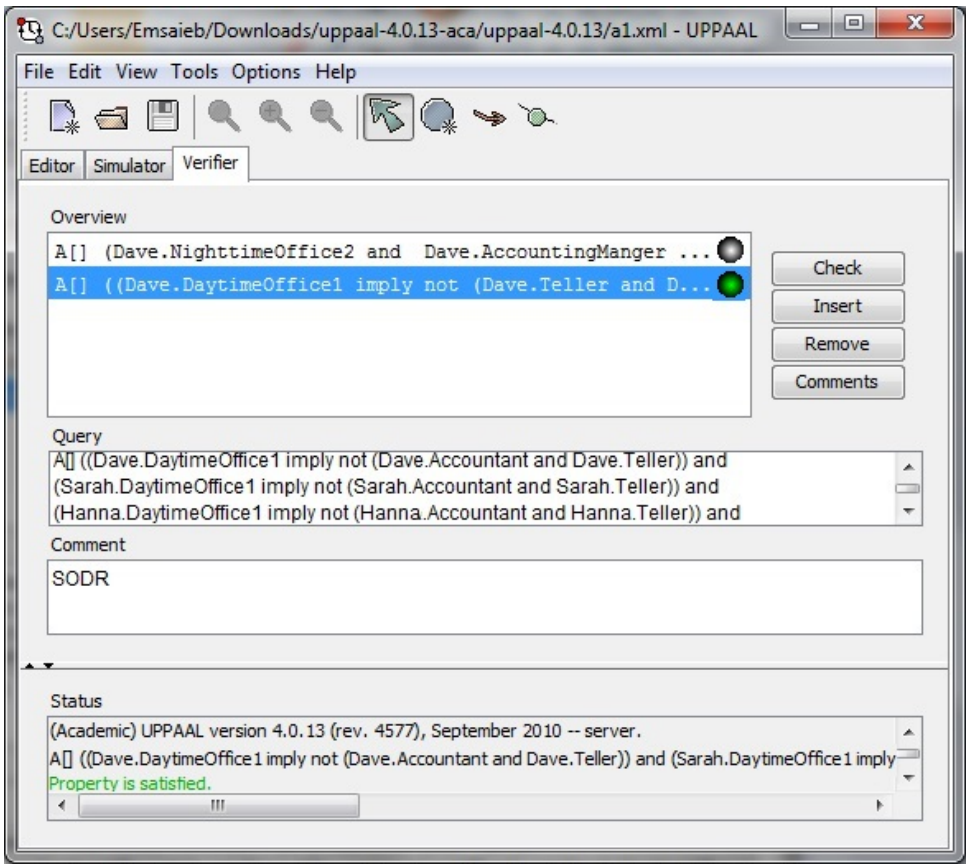

Figure 6.18: UPPAAL Analysis Result

6.4 Chapter Summary

Chapter 6 presented the model transformation AC2Uppaal, which transforms the Access Control Specifications in the context of the STRBAC model into Timed Automata and TCTL statements for the purpose of analysis. Section 6.1 provided an introduction of the model transformation AC2Uppaal. The transformation rules that map the elements of the STRBAC model into Timed Automata and TCTL statements are discussed in Section 6.2. Section 6.3 described the implementation of the model transformation AC2Uppaal

and discussed the usage of AC2Uppaal for modelling and analysing Access Control specifications to ensure the consistency of the specifications.

CHAPTER 7

COMPARISON OF ALLOY AND TIMED AUTOMATA FOR ACCESS CONTROL SYSTEMS

In this Chapter we perform a comparative study of formal Alloy and Timed Automata specifications from capability and performance points of view. In particular, this Chapter presents a comparison between Alloy and Timed Automata for modelling and the analysis of Access Control specifications in the context of Spatio-Temporal Role Based Access Control, in order to evaluate the integration methods presented in Chapter 5 and Chapter 6. The comparison study is based on the SECURE bank system case study, which is presented in Section 4.1. In order to compare the performance of Alloy against Timed Automata, the Access Control specification of the SECURE bank system must be translated into Alloy and Timed Automata. To achieve this goal, we make use of the two model transformations AC2Alloy and AC2Uppaal presented in Chapter 5 and Chapter 6, respectively.

7.1 Capability

A capability study is carried out to determine whether our approaches are capable of modelling and performing an analysis of the specifications for the Access Control in the context of the STRBAC model. In particular, the study aims at answering the following two questions:

- Is the modelling language capable of modelling the specifications for the Access Control model?
- Is the method capable of checking a sufficient number of constraints for the Access Control specification?

7.1.1 Capability of modelling STRBAC specifications

To answer the first question, Table 7.1 presents the STRBAC specifications that we were able to model using our AC2Alloy and AC2Uppaal approaches. In particular, it shows that not all the STRBAC features (i.e. Location Hierarchy and Cardinality Constraints) that we modelled in Alloy using AC2Alloy were modelled in Timed Automata and TCTL using AC2Uppaal. It is however possible to model them in Timed Automata. For example, the Location Hierarchy can be transformed into Timed Automata and TCTL in a similar way to the transformation of Role Hierarchy presented in Section 6.2.2. This is possible because new User Role Assignment information and Permission Role Acquire will be generated due to the effect of Location Hierarchy and the Role Hierarchy, as described in Section 3.1.3 and Section 3.1.4. Extending our method AC2Uppaal to include elements such as Location Hierarchy and Cardinality Constraints remains for future research.

Table 7.1: Modelling the STRBAC model using Alloy and Timed Automata

STRBAC Specification	**Alloy**	**Timed Automata+TCTL**
User Role Assignment	√	√
Permission Role Acquire	√	√
Role Hierarchy	√	√
Location Hierarchy	√	×
Separation of Duty between Roles	√	√
Separation of Duty between Permissions	√	√
Cardinality Constraints over Roles	√	×
Cardinality Constraints over Permissions	√	×

7.1.2 Capability of Checking Constraints of STRBAC specifications

Evaluating the capability of Alloy and Timed Automata to check STRBAC specification constraints is an important task. Therefore, we shall next discuss some of the advantages and disadvantages of using Alloy and Timed Automata for checking STRBAC specification constraints. Firstly, Alloy is supported by an automated tool called Alloy Analyser. Alloy Analyser is an automated constraint solver that transforms the Alloy code into Boolean expressions, providing the analysis by its embedded SAT solvers. This kind of constraint solver works by finding models that form counterexamples to assertions made by the user. Alloy is capable of checking most of the constraints presented in Section 3.2; however, it has some limitations with respect to the types of verification it can perform. One of the major limitation of Alloy is that, analysing and understanding the behaviour of systems using Alloy is non-trivial. For example currently, it is not possible to use Alloy to check whether an activated role will eventually be deactivated, which is because models in Alloy are static. As a further example, Alloy models capture the entities of a system, their relationships and constraints within the system. An Alloy analyser defines an instance of a system where the constraints are satisfied. More specifically, Alloy does not have any built-in notion of state [20]. Therefore, modelling of scenarios involving a number of states and evolution is not trivial. However, recent advances in dynamic Alloy might allow such modelling [124], but this is an area for future research.

Another major limitation of Alloy is scope. An Alloy user needs to specify the scope prior to performing an analysis. When the user executes the commands that he/she wants to check, the Alloy Analyser will search for counterexamples. If Alloy Analyser returns a counterexample is found, this means the model is inconsistent, while if the analyser returns no counterexample is found, this means either a counterexample may still exist in a larger scope and the model is not consistent or there is no counterexample and the model is consistent [21].

Timed Automata is a well-established model for representing real-time systems because

it is adequate enough to represent systems and is supported by temporal logic, which is a popular formalism for expressing system properties. Timed Automata and temporal logic can be analysed automatically with several well-known and mature model checkers, such as Uppaal. In addition, Timed Automata have a notion of state and evolution from one state to another. In Timed Automata the reachability problem is decidable [24]. Reachability helps to decide whether a certain state in a system is reachable from a given initial state. Therefore, Timed Automata can be a better choice for modelling and analysing liveness queries. In [80, 81], Mandol et al. show how to verify liveness queries using Timed Automata. For example, this work illustrates that it is possible to check whether or not a deactivated role will be activated eventually. Moreover, Timed Automata is also capable of checking most of the inconsistencies presented in Section 3.2. As a result, it is clear that Timed Automata has greater advantages from the capability point of view.

7.2 Performance

In this section we evaluate the performance between Alloy and Timed Automata in terms of time required to complete an analysis. In order to evaluate the performance of Alloy for an analysis of STRBAC specifications against Timed Automata, we only take the STRBAC elements that can be handled by the AC2Alloy and AC2Uppaal tools, as depicted in Table 7.1. In this case, AC2Alloy and AC2Uppaal are used to transform the Access Control specification of the SECURE bank system into Alloy and Timed Automata, respectively. This is followed by using the Alloy analyser to analyse the generated Alloy model and the Uppaal model checker to verify the Timed Automata. In the meantime, the time required to complete the analysis using the Alloy analyser and Uppaal has been computed. This process is repeated five times. To be more accurate, the test is repeated five times for every method. Every time the number of one of the STRBAC elements (Users, Roles, User Role Assignment, Permission Role Acquire and Role Hierarchy) is

increased gradually, while the other elements are kept fixed. The results are depicted as line charts in Figures 7.1, 7.2, 7.3, 7.4 and 7.5, which represents the time required to complete the analysis (in milliseconds) of all STRBAC elements.

The machine used in this test had the following configuration: HP Laptop running Windows 7 on an AMD Athlon(tm)x2 dual-core ql-64 2.10 GHZ with 3GB RAM. Time spent was calculated in milliseconds (ms).

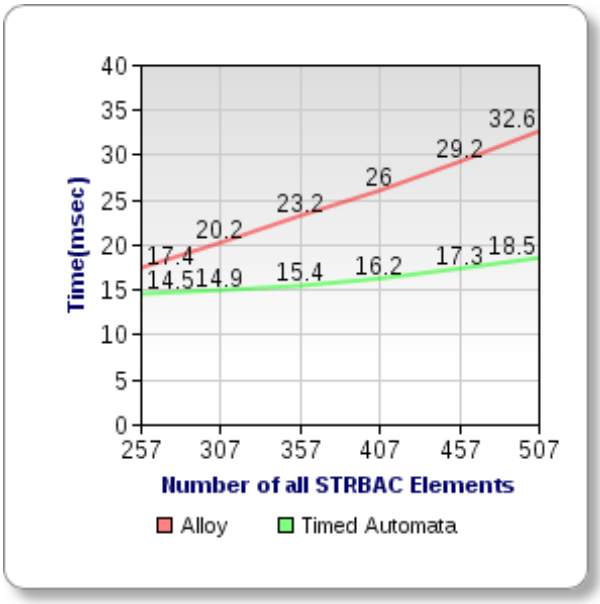

Figure 7.1: Increasing the number of Users

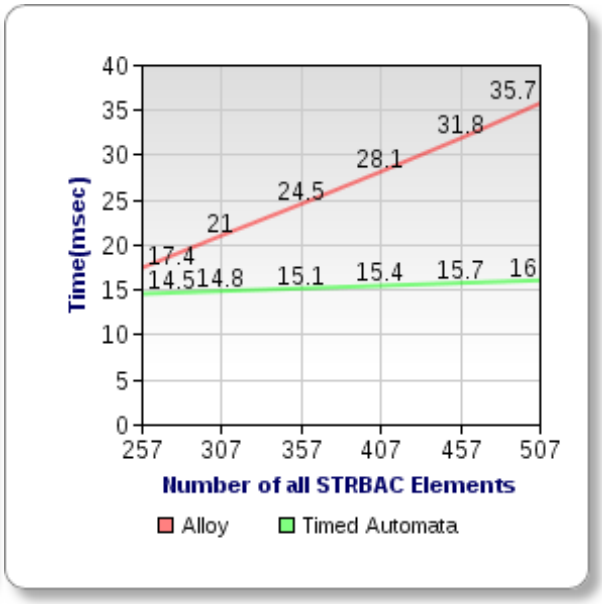

Figure 7.2: Increasing the number of Roles

A general observation from the line charts depicted in Figures 7.1, 7.2, 7.3 and 7.4 is that increasing the elements in the STRBAC model increases the time required to complete the analysis and has different effects on the time required to complete the analysis. For example, on one hand, it is noteworthy that increasing the number of roles increases the time required to complete analysis using Alloy more than any other element in the STRBAC model. We speculate the reason for this is because every role is transformed into a `signature`, `fact` and `predicate` in Alloy based on our transformation rules. This means the number of variables and clauses which will be created by the SAT-solver for every role is larger than the number of variables and clauses which will be created by the SAT-solver for any other elements such as user or permission. As a consequence, when the number of roles is increased by N number, the time spent on analysis will be longer than

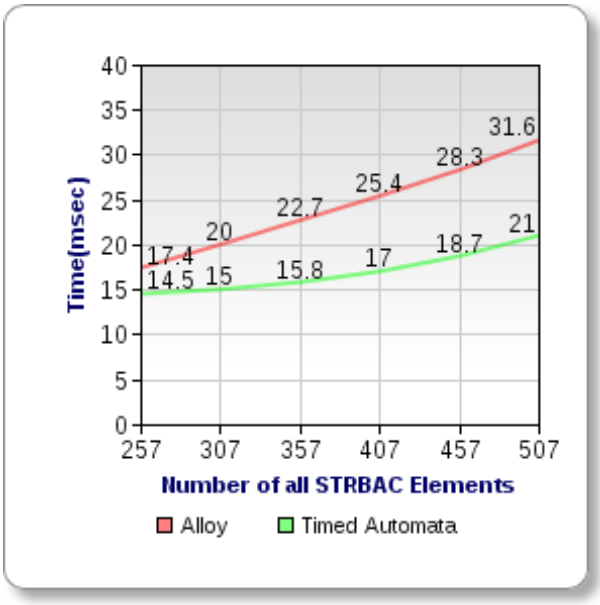

Figure 7.3: Increasing the number of URA

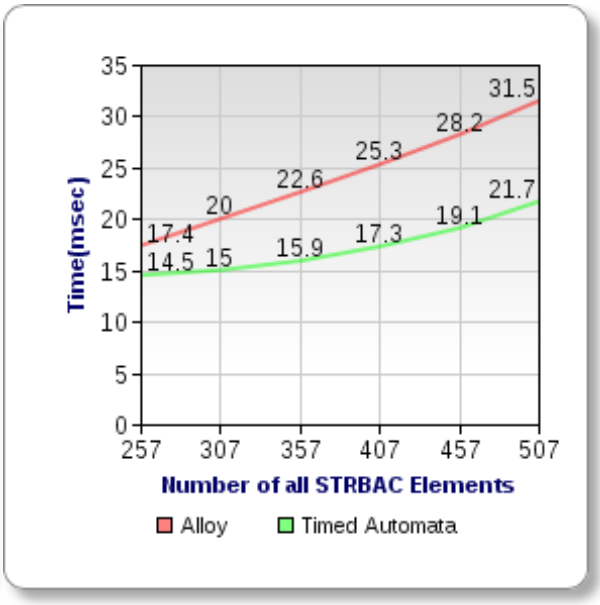

Figure 7.4: Increasing the number of PRA

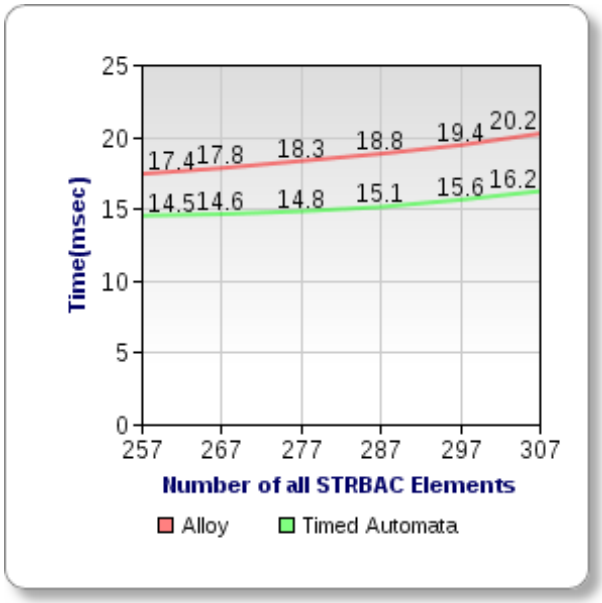

Figure 7.5: Increasing the number of RH

the time spent when any other elements of the STRBAC increase by the same number N. This is because the Alloy analyser is SAT-solver-based and SAT-solving time may vary enormously depending on factors such as clause ordering, the number of variables and the average length of clauses [119].

On the other hand, it is noticeable from the line charts that increasing the number of Role Hierarchy increases the time spent completing the analysis using Timed Automata

more than any other element in the STRBAC model. We speculate the reason for this is because the transformation of every Role Hierarchy will add new information to the Timed Automata and the TCTL because a new User Role Assignment (URA) and Permission Role Acquire will be generated and added to the specification.

In conclusion, as expected, the time required to complete verification increases gradually as more elements are added to the STRBAC specification. However, Timed Automata seems to perform better than Alloy in this sense, which shows that Timed Automata is a slightly better performer, as it completes the analysis of the five scenarios faster than Alloy, especially when the size of the specification is very large. For more information relating to the numerical values of the performances of Alloy and Timed Automata, please refer to Appendix D and Appendix E.

7.3 Chapter Summary

This Chapter presented a comparative study between Alloy and Timed Automata from the capability and performance point of view. The comparison study aimed to evaluate the methods Alloy and Timed Automata for modelling and analysing of Access Control specifications in the context of STRBAC model. This study has thrown up many questions in need of further investigation. Firstly, further experimental investigations are needed to establish complexity of checking inconsistency in our analysis.

It would be also interesting to know about the behaviour of the graphs when we move to larger examples with 1000 or 10000 elements. This is because the graphs are not giving a clear picture. Therefore, future research should concentrate on the investigation of using stronger machines to run larger Access Control examples in orders of magnitude larger.

CHAPTER 8

EXTENDING THE SPATIO-TEMPORAL ROLE BASED ACCESS CONTROL MODEL FOR PHYSICAL SYSTEMS

Our preliminary investigation revealed the following. Researchers have proposed several Spatio-Temporal Access Control models, such as Spatio-Temporal Role Based Access Control (STRBAC), to support policy designers in designing proper Access Control specifications, especially the Cyber Access Control (CAC) specification. Although these models offer many benefits for implementing CAC systems, they are not adequate enough to represent Physical Access Control (PAC) specifications and still have some limitations that should be overcome prior to modelling the physical aspect of Access Control. One of the major drawbacks to these models is the representation of locations, as current models deal with logical location, which may not be suitable for representing the physical aspect of Access Control. This is a critical point, because logical location and physical location are different, especially when dealing with hierarchy. We came across such limitations when trying to model a PAC mechanism used in a leading telecommunications company (British Telecom).

Physical Access Control (PAC) is as important as Cyber Access Control (CAC) because physical access to resources such as computers, servers and network cables may allow for bypassing the CAC mechanism. Therefore, it is important to develop a new model that can support the physical aspect of Access Control, in order to reduce the

complexity of security management. As a result of our study of different Access Control models, we decided to extend the existing models to overcome their limitations, in order to be capable of modelling Physical Access Control (PAC) specifications. Our model extends that proposed by Ray and Toahchoode [12] and consists of various rules that can be employed to support various system requirements. These rules may interact with each other in subtle ways and result in inconsistencies, which must be detected before the implementation of the system. Therefore, we make use of the method presented in Chapter 5, AC2Alloy, to ensure the consistency of the specification. In order to demonstrate the feasibility of our approach, we make use of a case study provided by our industrial partner British Telecom (BT).

The rest of this chapter is divided as follows. In Section 8.1 we provide an overview of Physical Access Control (PAC) systems. In Section 8.2 a case study provided by our industrial partner British Telecom (BT) is introduced. Section 8.3 addresses the limitations of the current STRBAC model. Section 8.4 presents our efforts to develop the new STRBAC-PS framework, and in Section 8.5 we make use of the method presented in Chapter 5, AC2Alloy, to transform the specification of the case study into Alloy language, after which an Alloy analyser is used to analyse the specification.

8.1 Physical Access Control (PAC) System

In physical security, the term 'Access Control' refers to the practice of restricting entrance to a property, a building or a room to authorised persons. Physical Access Control (PAC) can be achieved by a human (a guard, bouncer or receptionist) through mechanical means such as locks and keys, or through technological means such as physical Access Control systems (e.g. a card reader). Figure 8.1 illustrates an example of PAC systems. In a large organisation such as BT, which deals with physical and cyber infrastructures, managing Physical Access Control (PAC) policies is a complex task. The complexity is mainly attributable to the following five aspects:

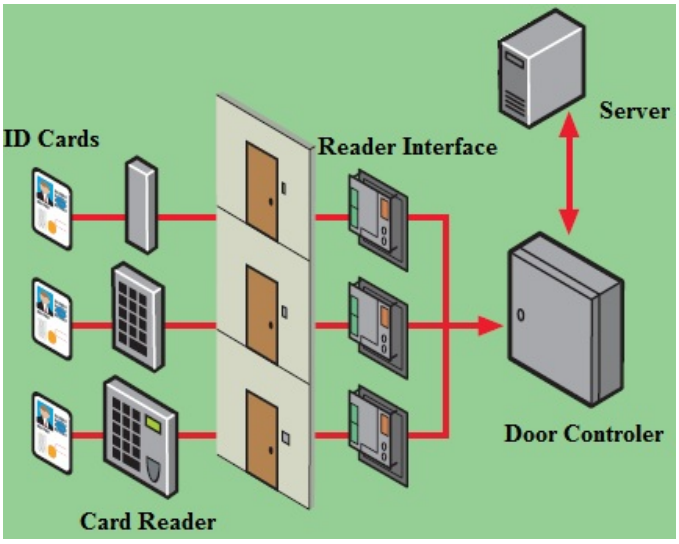

Figure 8.1: Physical Access Control System

- A large number of buildings/zones with distributed geo-locations.
- Buildings with different risk levels which require different levels of Access Control.
- A mixture of users who have access to buildings based on a mix of roles such as permanent employees, contractors, third parties and outsourcing workforce.
- Time constraints for accessing the building/zones. For example, certain zones can be accessed during the day, while at the night they are locked.
- Physical access can invalidate cyber access. For example, if a machine is protected by a firewall but people can physically access it, then they can access the hard disk and copy it. This can be done by, for example, mounting the hard drive via an OS which is on a memory stick.

Considering the above factors, a typical PAC system for a large organisation should be able to assess a combination of risk levels regarding buildings/zones, personal roles, time and location. As a result, such models can be very complex.

An example of PAC is as follows:

- A user swipes an ID card to submit his or her user profile to the system to verify the user's information. The user profile contains user information such as the roles

that are assigned to him or her.

- The card reader reveals which building or zone the user is accessing, and therefore it submits the user profile and building profile to the system. The building/zone profile contains the building/zone's risk level information for Access Control.

- Both profiles (user profile and building profile) are processed by the Physical Access Control (PAC) system. The process involves a rule engine that has knowledge of the physical access policies. It compares the two profiles and makes an access decision.

- Three possible results (access, cannot access or access without approval) can be returned by the Access Control process. If it returns "access without approval", then the system will look for registered approvals in the user profile for the required zone, in order to decide whether access should be granted or denied. If the rule engine returns "cannot access", then Access Control is denied.

During the process of granting access, time and location are also important factors. For example, if a user tries to access a building/zone outside his/her working hours or in a different geographical location from his/her normal working region, access may not be granted. For example, if a user who is an Essex-based cabling engineer tries to gain access to a cable chamber in Edinburgh, access may be denied, as it is out of his/her working region. However, if there are pre-registered approvals or special events, such as a meeting, emergency repair or disaster recovery, then access can still be granted.

8.2 Case Study: Physical Access Control System

In this section we introduce a simplified version of a Physical Access Control (PAC) example used in a large telecommunications company to demonstrate our approach.

8.2.1 Physical System Security Policies

8.2.1.1 Entity

In organisations such as telecommunications companies, users are mainly categorised based on their job type, for example `Company Employees`, `Technical Employees`, `Clerical Employees`, and `Cabling Engineer`. These organisations are also based in several regions (i.e. `Birmingham region`, `Manchester Region`), every one of which consists of several buildings such as *x*, *y*, *z*, which contain several zones or rooms such as server rooms and common rooms. Each physical Access Control point (e.g a card reader) can be considered a zone or a room entrance. In this thesis, we assume that the organisation is based in one region (`Birmingham Region`) and it only works out of one building (*x*) which contains three zones: the Low Risk Zone, Medium Risk Zone and High Risk Zone, as depicted in Figure 8.2. The organisation also employs thousands of users. *Dave*, *Mark*, *Tom*, *Sarah*,

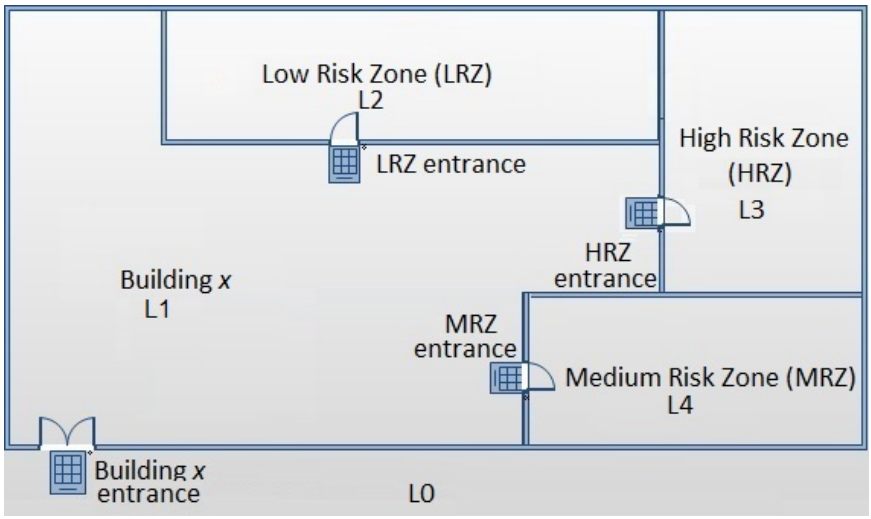

Figure 8.2: Location

Amy and *Hannah* represent a small list of users within the organisation chosen to illustrate our approach. Examples of the permissions available to users are listed in Table 8.1. Each user from the above list could be assigned to roles in the organisation based on the time and location constraints illustrated in Table 8.2, and each role can be assigned to the permissions based on the time and location constraints summarised in Table 8.3.

Table 8.1: List of Permissions

	Permission
P1	Access Building x
P2	Access Low-level zone (i.e. common room)
P3	Access Medium-level zone (i.e. data centre)
P4	Access High-level zone (i.e. server room)
P5	Access Street Cabinets

Table 8.2: Users to Roles Assignment Constraints

Users	**Roles**	**Times**	**Locations**
Dave	Cabling Engineer	DayTime	L5: street cabinets Birmingham Region L2: Low Risk Zone Birmingham Region
Sarah	Cabling Engineer	DayTime	L5: street cabinets Birmingham Region L2: Low Risk Zone Birmingham Region
Tom	Cabling Engineer	DayTime	L5: street cabinets Birmingham Region L2: Low Risk Zone Birmingham Region
Amy	Technical Engineer	DayTime	L3: High Risk Zone Birmingham Region
Mark	Clerical Employee	DayTime	L4: Medium Risk Zone Birmingham Region
Hannah	Company Employee	DayTime	L2: Low Risk Zone Birmingham Region

8.2.1.2 Role Hierarchy

The hierarchy of roles in the telecommunications company is depicted in Figure 8.3. The figure shows that the `Cable Engineer` is a senior role in relation to a `Technical Engineer`, while the `Technical Engineer` is senior to a `Company Employee`. In the same vein, a `Clerical Employee` holds seniority over a `Company Employee`.

8.2.1.3 Separation of Duty between Roles

The telecommunications company requires that the same user should not be a `Clerical Employee` and a `Cable Engineer` at the same time and at the same location.

Table 8.3: Permissions to Roles Assignment Constraints

Role	**Permission**	**Time**	**Location**
company employee	P2	DayTime	L2
cabling engineer	P5 P2	DayTime	L5 L2
technical engineer	P4	DayTime	L3
clerical employee	P3	DayTime	L4

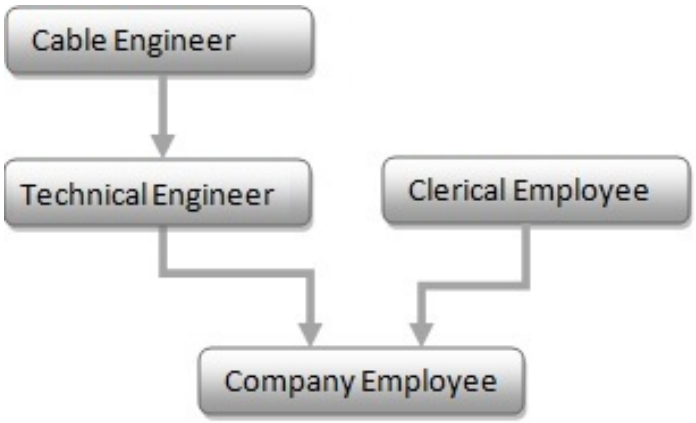

Figure 8.3: Role Hierarchy

8.2.1.4 Cardinality Constraint over Roles

For health and safety reasons, the telecommunications company requires that no more than *2* users should have the role `Cable Engineer` at location `L1` (`street cabinet`) at the same time.

8.3 Limitations of the STRBAC model

In this section we discuss some of the limitations of the current Spatio-Temporal Role Based Access Control (STRBAC) model that we came across when modelling the case study presented in Section 8.2 [30].

8.3.1 Logical Location vs. Physical Location

Locations in CAC systems are specified as a set of logical entities. This model is not sufficient for capturing access to resources. For example, because of doors and locks we need to create a model which represents how access to a location requires credentials to access another location.

8.3.2 Differences between Hierarchy of Location in the Cyber and Physical Systems

In general, Location Hierarchy is a partial order on a set of locations that specifies which location is inside another location (e.g. l_i is inside l_j). In cyber systems this means that if a user u is assigned a role r and its permission p at time t and at the outer location l_j, then the same user u can have role r and permission p at the inner location l_i. This may not be true in PAC, where having permission to access to the outer location does not necessarily mean the same user will be granted access to the inner locations. For example, in Figure 8.1, if a user u has permission to access location `L1`, this does not necessarily mean the same user u will have permission to access any of the inner locations `L2, L3` and `L4`. As a result, hierarchy of location in PAC is not the same as the hierarchy in Cyber Access Control.

8.4 The Proposed Model: Spatio-Temporal Role Based Access Control for Physical Systems (STRBAC-PS)

In this section we describe a new Spatio-Temporal Role Based Access Control for Physical Systems (STRBAC-PS) framework which integrates different components of the RBAC model with location and time in order to describe PAC specifications. Similar to other Spatio-Temporal RBAC models, the STRBAC-PS model has basic sets of entities: Users `(U)`, Roles `(R)`, Permissions `(P)` and Times `(T)`. We also introduced Location Graph `(LG)`

instead of Locations, which was represented as a finite set. We shall now formally describe these entities, but before that we present the location models for PAC.

8.4.1 Extended Location Models for Physical Access Control (PAC)

In the previous section we explained that the cyber model of location needs to be extended to capture sufficient information for specifying PAC systems. In order to formalise the physical location we introduced the Location Graph (LG) concept, which is a graphical model that emphasises both the locality and connectivity of PAC systems.

Definition 5. Location Graph LG=(l_0, L, E), where L is a finite set of nodes and $E \subseteq L \times L$ is a set of directed edges connecting the nodes of L. Nodes in L, which are sometimes called *locations*, consist of (possibly) multiple rooms, corridors, etc, so that if a person can access one of them, then he/she can access all of them. L has a unique node called an *outside* location, represented as $l_0 \in L$, which represents the entire world outside the premises. Edges E represent links which are marked by permissions. We define the allocated permissions function as $m : E \to P \cup \{\epsilon\}$, where $m(e)$ is the permission marking edge e. $e = (l, l')$ shows the permissions required to go from l to l'. If $m(e) = \epsilon$, then a person does not need any permission to go from l to l'. Sometimes we simply write the permissions on the edge to represent the allocated permission. In this case we can write $l \xrightarrow{p} l'$ to say that, in order to go from l to l', permission p is required.

For example, the physical location depicted in Figure 8.2 can be represented using the Location Graph, as illustrated in Figure 8.4 which shows that a user requires a set of permissions in order to be able to move from one node to another. Moreover, it shows that a user needs to have permission (p_1) in order to move from node l_0, which represents the *outside*, to node *L1*, which represents Building *x*.

This location model has sufficient information to deal with the two problems addressed in Section 8.3.

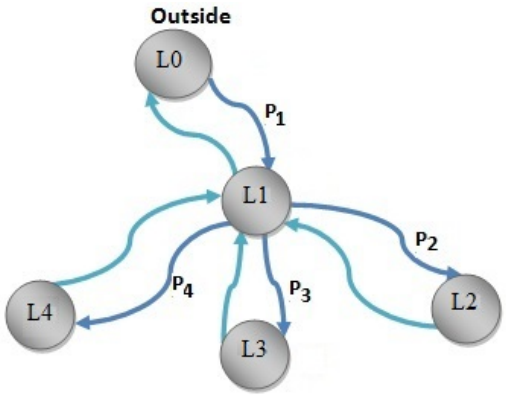

Figure 8.4: Location Graph

Notation: Assuming that $LG = (l_0, L, E)$ is the Location Graph as defined in Definition 5, we write $\delta = l_1 \xrightarrow{p_1} l_2 \xrightarrow{p_2} l_3.....l_{k-1} \xrightarrow{p_{k-1}} l_k$ for a path of edges connecting locations, in which p_i represents permissions required to go from location l_i to l_{i+1}, where $i = 1, 2, 3, ..., k-1$.

8.4.2 Times (T)

Time is a set of continuous time intervals on a time line. In our STRBAC-PS model, it is represented as recurring intervals which are constructed from primitive recurring intervals through set operations. Suppose that access to common rooms in an organisation is only permissible on weekdays from Monday to Thursday and during the working hours between 9:00am and 6:00pm. In this case, we need an access policy that can specify this recurring interval. We represent these intervals in a manner similar to Bertino et al. [48]. Primitive recurring intervals contain the following four calenders according to their period: Daily, Weekly, Monthly and Yearly. Each calender may be divided further according to the atomic interval of the time used: minute, day, week or month. An example of primitive recurring intervals is 9:00am to 6:00pm, Monday to Friday, during November in 2012. In this thesis we use the notation T=$\{t_1, t_2, ..., t_n\}$ to mean that Times (T) is a finite set of time intervals (i.e. t_1, t_2,....., t_n), where n is an integer value.

8.4.3 Users (U)

In Physical Access Control (PAC) systems, Users (U) can be a set of human beings. We write U=$\{u_1, u_2, ..., u_m\}$ to mean that Users (U) is a finite set of users (i.e. $u_1, u_2, ..., u_m$), where m is an integer value. Every user should have an ID card which in turn holds information on the user profile such as the user's ID and role(s). The user's location changes in line with time, and whenever a user tries to access a new zone the card reader will reveal the location of the user. The relation that relates users to roles is described in the next section.

8.4.4 Roles (R)

The user's roles in Physical Access Control (PAC) systems are mainly categorised based on the user's job type (i.e. *common employees and visitors*). In our model, we formally write R=$\{r_1, r_2, ..., r_k\}$, meaning that Roles is a finite set of roles (i.e. $r_1, r_2, ..., r_k$), where k is an integer value.

In PAC systems, roles are associated with other entities using two relations, namely User Role Assignment and Permission Role Acquire. We will now describe User Role Assignment, while Permission Role Acquire will be described later.

8.4.5 User Role Assignment (URA)

This relation associates users with roles based on time and location constraints. For example, a user can be assigned to a cable engineer role in Birmingham, but only at a specific time and in a specific location, URA $\subseteq$ U $\times$ R $\times$ T $\times$ L. It is therefore a many-to-many relation, which means that a user can be assigned to one role or more and the same role can be assigned to one user or more. We write ura(u, r, t, l), meaning that the user u is assigned to the role r during the intervals time t and at the node l.

8.4.6 Permissions (P)

Permissions are the ability to access secured rooms, zones or buildings. In our STRBAC-PS model, we formally denote Permissions (P) as P=$\{p_1, p_2, ..., p_o\}$ to mean that Permissions is a finite set of permissions (i.e. $p_1, p_2, ..., p_o$), where o is an integer value.

Permissions are assigned to users through roles, and the relation that associates these roles with permissions based on time and location conditions is called Permission Role Acquire, which we describe in the next subsection.

8.4.7 Permission Role Acquire (PRA)

Permission Role Acquire (PRA) is a relation that associates permissions with roles based on time and location conditions. For example, the role cable engineer can have the permission to access server room in Birmingham region only at specific time, $\texttt{PRA} \subseteq \texttt{R} \times \texttt{P} \times \texttt{T} \times \texttt{L}$. It is many-to-many relation. This means a role can have one permission or more and the same permission can be assigned to one role or more. We formally write $\texttt{pra}(r, p, t, l)$ to mean that the role r is assigned to the permission p to access the node l during the intervals in time t.

8.4.7.1 Impact of the Location Graph on Permission Role Acquire (PRA):

In order to gain physical access to an inner location in a building, for example a room, a person needs to have access to a set of doors and corridors which allow him/her to go through them and end up in the inner location. There might be multiple paths to the inner location, but he/she needs to be able to go through at least one of them to reach to the inner location. To formalise this process, consider a Location Graph $LG = (l_0, L, E)$ as defined in Definition 5. Assume that a role r has permissions p to access a location l at time t. In this case there exists at least one path of the form $l_0 \xrightarrow{p_0} l_1 \xrightarrow{p_1} l_2.....l_k \xrightarrow{p_k} l$, such that role r has all permissions $p_0, p_1, p_2, ...p_k$. In other words, $\forall i\ pra(r, p_i, t, l_i)$, where $0 \leq i \leq k-1$.

8.4.8 Role Hierarchy (RH)

The Role Hierarchy concept is also an important factor for Physical Access Control (PAC). Normally, roles in an organisation have overlaps in permission assignments. For example, a company employee can have the same permissions as a visitor to access to coffee rooms, and has additional permissions to access common rooms. Therefore, role hierarchies are introduced as a key aspect of hierarchy in our STRBAC-PS model, in which a user who has a senior role can inherit all the permissions assigned to a junior role. We write $rh(r_i \succeq r_j$, meaning that the role r_i is senior to the role r_j. This means that a user who has the senior role r_i can inherit the junior role r_j and the senior role r_i can inherit all the permissions assigned to the junior role r_j. We formally write this as follows:

$$rh(r_i \succeq r_j) \wedge ura(u, r_i, t, l) \Longrightarrow ura(u, r_j, t, l) \wedge pra(r_j, p, t, l) \Longrightarrow pra(r_i, p, t, l)$$

8.4.9 Separation of Duty between Roles (SoDR)

Many organisations, when they design their Physical Access Control (PAC) system, require that some roles should not be assigned to the same users at the same time and the same node. For example, an organisation such as a telecommunications company may require that the same user should not be a *Clerical Employee* and a *Cable Engineer* at the same time and in the same node. We write $sodr(r_i, r_j, t, l)$. This means that the same users should not have the two roles r_i and r_j at time t and node l. We formally write this as follows:

$$sodr(r_i, r_j, t, l) \Longrightarrow ura(u, r_i, t, l) \wedge \neg ura(u, r_j, t, l) \vee ura(u, r_j, t, l) \wedge \neg ura(u, r_i, t, l)$$

8.4.10 Cardinality Constraint over Roles (CCoR)

Another important factor in Physical Access Control (PAC) is Cardinality Constraint over Roles (CCoR). CCoR is a set of constraints that restrict the number of users who can

have a role at a specific node and during a specific time. For example, an organisation may require that the role *Cable Engineer* should not be assigned to more than n users at the node, which represents the location *street cabinet* at any time. We write $ccor(r_i, t, l, n)$, meaning that the role r_i has restrictions, so that it should not be assigned to more than n users at time t and node l. We formally denote this as follows:

$$ccor(r_i, t, l, n) \Longrightarrow \#(ura(u, r_i, t, l)) \leq n$$

8.5 Analysing the running example using AC2Alloy

The most frequent question posed when modelling Access Control is whether or not the specification is consistent with the functional requirements and is compliant with security requirements. To answer such a question, we make use of AC2Alloy to generate an Alloy model from the STRBAC-PS specification, and then we use an Alloy analyser to verify the generated Alloy model.

8.5.1 Transformation of the Running example into Alloy

As described in Section 5.3.1, when we enter the the Access Control specification into the AC2Alloy tool, an XML representation of the Access Control specification is automatically generated and then the XML representation is automatically transformed into Alloy. The basic components of the STRBAC-PS model, such as sets of Users, Permissions, Times and Nodes, will be transformed to into Alloy `signatures` as illustrated in Figure 8.5.

The set of Roles will be transformed into `signatures, facts` and `predicates`. These `facts` and `predicates` are used to represent the relationships between roles, users, permissions, times and locations. Such relationships are expressed by User Role Assignment `(URA)` and Permission Role Acquire `(PRA)`. For example, AC2Alloy will transform the role *Cable Engineer* and its User Role Assignment into `abstract signature, one signature, fact` and `predicate` in Alloy, as shown in Figure 8.6.

```
abstract sig Users{}
one sig Dave, Amy, Tom, Mark, Hannah, extends Users{}

abstract sig Nodes{}
one sig L1, L2, L3, L4, L5 extends Nodes{}

abstract sig Times{}
one sig DayTime, NightTime extends Times{}

abstract sig Permissions{}
one sig P1, P2, P3, P4, P5 extends Permissions{}
```

Figure 8.5: Transformation of Users, Permissions, Times and Nodes

```
abstract sig Roles {time: lone Times,
   node: lone Nodes,
   users: set Users,
   permissions: set Permissions}
one sig CableEng extends Roles{}

fact CableEng_fact{all self: CableEng | CableEngCondition[self]}

pred CableEngCondition[self: CableEng]{((self.permissions= none)
   &&(self.node= L2)&&(self.time= DayTime)&&(self.users=
   Dave+Sarah+Tome ))||((self.permissions= none)&&(self.node=
   L5)&&(self.time= DayTime)&&(self.users= Dave+Sarah+Tome))}
```

Figure 8.6: Example of a Roles Transformation

The Permission Role Acquire injects the predicate into the Alloy code for roles with new assignment information. For example, the transformation of the Permission Role Assignment of the role *Cable Engineer* will inject the `predicate CableEngCondition` into the Alloy code for the role *Cable Engineer* with new assignment information, as depicted in Figure 8.7.

Role Hierarchy also injects `predicates` into the Alloy code for roles with new assignment information. For example, the effect of the transformation of the hierarchy between the roles *Clerical Employees* and *Company Employees* will be transformed into new as-

```
pred CableEngCondition[self: CableEng]{((self.permissions= P5)
   &&(self.node= L5)&&(self.time= DayTime)&&(self.users=
   Dave+Sarah+Tome ))|||((self.permissions= P2)&&(self.node=
   L2)&&(self.time= DayTime)&&(self.users= Dave+Sarah+Tome))}
```

Figure 8.7: Example of a Permission Role Acquire Transformation

signment information which will be injected into the `predicates` within the Alloy code for the roles *Clerical Employees*. Moreover, the senior role *Clerical Employees* can inherit all of the permissions assigned to the junior role *Company Employees*, in which case the `predicate ClerEmpCondition` within the Alloy code for the role *Clerical Employees* will be injected with new assignment information, as shown in the Alloy code depicted in Figure 8.8.

```
pred ClerEmpCondition[self: ClerEmp]{((self.permissions= P3)&&
   (self.node= L4)&&(self.time= DayTime)&&(self.users= Mark))||
   ((self.permissions= P2)&&(self.node= L2)&&(self.time= DayTime)
   &&(self.users= Mark ))}
```

Figure 8.8: Example of Role Hierarchy Transformation

8.5.2 Model Analysis

In the previous section we demonstrated the transformation between the Access Control model and the corresponding Alloy. However, this work would be incomplete without demonstrating how we can verify the produced Alloy model to detect inconsistencies. In this section we provide a brief description of the automatic analysis task that was carried out via the Alloy analyser, but beforehand we shall introduce several common inconsistencies that occur in the Physical Access Control (PAC) policy.

1. A user has access to an inner zone but does not have access to the outer zone.

2. A user has given default access to zones that are outside of his/her work region.

3. A user has two conflicting roles that are constrained by Separation of Duty between Roles at the same time and the same location.

4. A user has the right to access two conflicting permissions that are constrained by Separation of Duty between Permissions at the same time and the same location.

5. A role is assigned by a number of users that exceed the maximum number of users that should be assigned to that role at the same time and the same location.

To ensure that the PAC system is consistent, several Alloy `checks` have been produced via AC2Alloy. For example, an Alloy `check` will be generated with every Cardinality Constraint, so that the `check` can be used to verify whether or not the constraint will hold. An instance of this scenario is that the cardinality constraint over the role *Cable Engineer*, which is presented in Section 8.2.1.4, will be transformed into an Alloy `check`, as illustrated in Figure 8.9.

```
CC1 :check{((L5 in CableEng.node)&&
  (DayTime in CableEng.time)=>(#CableEng.user <3))}
```

Figure 8.9: Example of a Cardinality Constraint Transformation

The execution of the Alloy `check` presented in Figure 8.9 shows that the Alloy analyser picked up a counterexample, as depicted in Figure 8.10. This means that the policy is inconsistent because there are more than two users (Tome, Dave and Sarah) are assigned to the role *Cabling Engineer* at node `L5`, which represents the *(street cabinet)* during the *DayTime*, which is not permissible according to the Cardinality constraint presented in Section 8.2.1.4.

Another example of the Alloy `checks` that will be generated using AC2Alloy, in order to perform analysis on PAC specifications, is that the Separation of Duty constraint between

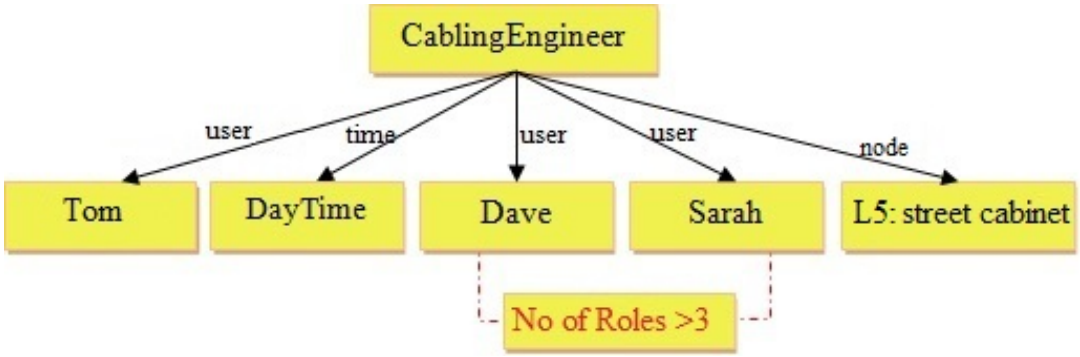

Figure 8.10: Counterexample for Cardinality `check CC1`

the roles `Clerical Employee` and `Cable Engineer` at the time `DayTime` and at node `L4` will be transformed into an Alloy `check` as depicted in Figure 8.11.

```
SoDR1 :check {
 SODR[ClerEmp, CableEng, DayTime, L4]}
```

Figure 8.11: Alloy formula for the SoD between `Clerical Employee` and `Cable Engineer`

The execution of the above Alloy `check` shows that Alloy Analyser did not find a counterexample as illustrated in Figure 8.12. This means that the Separation of Duty constraint between the roles `Clerical Employee` and `Cable Engineer` at the time `DayTime` and at the node `L4` satisfies all the other rules of the PAC specification presented in Section 8.2.1.

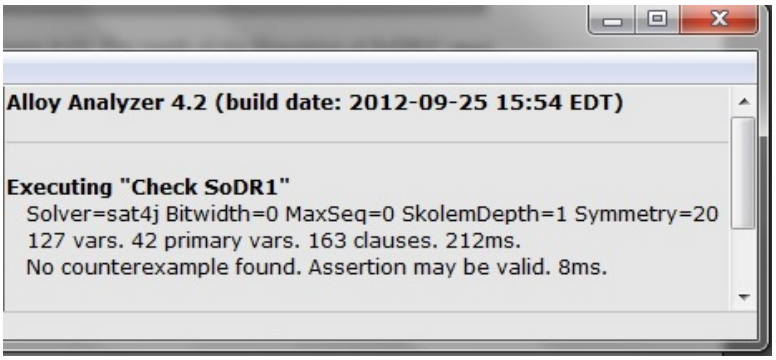

Figure 8.12: The result of the Execution of SoDR1 `check`

8.6 Chapter Summary

Chapter 8 provided an extension of the existing STRBAC model in a manner such that it can be used to specify Physical Access Control (PAC) policies. Section 8.1 provided some background information of Physical Access Control. Following this, we provided a case study that is used to demonstrate our approach. The limitation of the existing STRBAC model is discussed in Section 8.3. In Section 8.4 we presented our extension of the STRBAC model to support the physical aspect of Access Control. Section 8.5, described how AC2Alloy can be used to analyse the Physical Access Control specifications.

CHAPTER 9

CONCLUSIONS AND FUTURE WORK

This chapter concludes the work presented in this thesis. In Section 9.1, a summary of the contributions made in this thesis is presented. Section 9.2 presents a discussion on future work that could be undertaken to expand upon and improve this research.

9.1 Summary of Contributions

One of the main contribution of this thesis is that it presents formal algebraic notations for the Spatio-Temporal Role Based Access Control (STRBAC) model, which are used to formulate the concepts of inconsistency and semi-consistency in STRBAC specifications. Another major contribution of our research is the development of two model transformation methods to transform Access Control specifications in the context of Spatio-Temporal Role Based Access Control (STRBAC) into formal languages via Model-Driven Architecture techniques for the purpose of analysing these specifications. In particular, this thesis describes two model-driven approaches – AC2Alloy and AC2Uppaal. The first model transformation, AC2Alloy, was developed to transform the STRBAC model into Alloy, thus allowing for powerful analysis to take place using an Alloy analyser utilising SAT-Solvers, while the second model transformation, AC2Uppaal, was developed to transform the STRBAC model into Timed Automata and TCTL statements, following which the Uppaal model checker was used to model and verify the Timed Automata and the TCTL

statements to identify inconsistencies in STRBAC specifications. The feasibility of this approach was demonstrated through a case study. We believe that the methodology that we have used in this thesis to automate the transformation between STRBAC model and formal representation such as Alloy and Timed Automata can be generalised and applied in the domain of model transformations between other Access Control models and formal methods. Moreover, the thesis presents a comparison study between Alloy and Timed Automata from capability and performance points of view. Finally, we extend the current STRBAC model to cover the physical aspect of Access Control. This is described with the help of a case study offered by our industrial partner British Telecom (BT).

In Chapter 2 of the thesis, an introduction to Access Control is presented, in particular Access Control models. This is followed by a review of existing work on the analysis of Access Control specifications. Next, an introduction to Alloy, including various features of the language, is presented. This is followed by an introduction to Timed Automata, TCTL and the Uppaal model checker. Finally, a preliminary examination of MDA, with a focus on a particular MDA model transformation framework, SiTra, is presented.

Formal algebraic notations for the STRBAC model are introduced in Chapter 3 and are then used to provide a formal definition of inconsistencies in STRBAC specifications and several scenarios that may cause inconsistencies in said specifications. This chapter also uses these formal algebraic notations to introduce the concept of semi-consistency in STRBAC specifications, and it gives several examples that may cause this scenario.

Chapter 4 presents an application for automatic transformations between the STRBAC and Alloy by defining a transformation model. Moreover, Chapter 4 introduces a running example that is used later to describe the model transformation. The model transformation, named AC2Alloy, is described in two stages – metamodels for the source and target models and transformation rules to map the elements of the source and target metamodels. To conduct the model transformation, a set of transformation rules are defined in the second stage. These transformation rules are described with the help of the running example.

Chapter 5 describes the implementation of the transformation rules presented in Chapter 4. The model transformation is implemented as an eclipse plug-in called AC2Alloy. This chapter starts by outlining the architecture of the AC2Alloy tool, followed by descriptions of the technologies used in developing the model transformation, AC2Alloy. This is followed by a description of the eclipse plug-in AC2Alloy, which is a tool used for integrating STRBAC and Alloy into a single tool. It enables the user to create STRBAC specifications and transform them into XML representations which are then transformed into an Alloy model. The chapter then introduces several Alloy checks that can be generated using AC2Alloy to conduct an analysis of the specifications via an Alloy analyser. Finally, this chapter illustrates how the specifications of the running example can be transformed into Alloy using AC2Alloy, and then it describes how the generated model can be analysed using the Alloy analyser, in order to detect inconsistencies.

Chapter 6 describes a new model transformation named AC2Uppaal, which transforms Access Control specifications in the context of the Spatio-Temporal Role Based Access Control into Timed Automata network and TCTL statements. This chapter starts by introducing the model transformation. The transformation rules that map the elements of the source metamodel, STRBAC and the target metamodels, Timed Automata and TCTL are described next with the help of the running example of the SECURE bank system presented in Chapter 4. These transformation rules are then implemented as an eclipse application called AC2Uppaal. Next, in order to demonstrate the visibility of our approach, this chapter discusses how the running example can be transformed in Timed Automata and TCTL and then performs the analysis using the Uppaal model checker to detect inconsistencies.

Chapter 7 presents a comparison study between Alloy and the Uppaal tool from capability and performance points of view is presented. The comparison study is based on a case study.

The extension of the STRBAC model for physical Access Control systems is presented in Chapter 8. In this chapter, a case study provided by BT is presented, followed by an

overview of the limitations of the current STRBAC model. Subsequently, an extension of the STRBAC model, which supports the physical aspect of Access Control systems, is presented. Finally, the method presented in Chapter 5 – AC2Alloy – is used to analyse the physical Access Control specifications.

As an overall reflection, this thesis contributes two model-driven approaches – AC2Alloy and AC2Uppal – that automate the transformation between the STRBAC model and Alloy and STRBAC and Timed Automata and TCTL, respectively. These model transformations then contribute towards the analysis of STRBAC specifications to identify any erroneous design flaws. Next, a comparative study between the two Alloy and Timed Automata methods is presented. Finally, the STRBAC model is extended to cover the physical Access Control specifications. The entire research is described using various examples.

9.2 Future Work

Following the advances made in this thesis, a number of directions for future research have arisen. Some of these extensions would help to overcome some of the limitations of this research, whilst others would provide additional capabilities.

The STRBAC model used in this research, as presented in Chapter 3, is a subset of the STRBAC model derived from [12, 13]. Therefore, there are some elements that were not considered in this research, such as Delegation. Delegation in the STRBAC model is classified into two categories, namely Role Delegation and Permission Delegation [13]. Role Delegation means that the delegator delegates a role to the delegatee at any time and at any location where the delegator can have that role, while Permission Delegation means that the delegatee can invoke the delegator's permissions at any time and at any place where the delegator can invoke these permissions. This means new User Role Assignment and Permission Role Acquire will be added as a result of the presence of Role Delegation and Permission Delegation, respectively. Consequently, the transformation of

Role Delegation and Permission Delegation can done in a similar way to that of Role Hierarchy, as presented in Section 4.3.5 and Section 6.2.4. In addition, in [13], Ray and Toahchoodee show how delegation can be transformed manually into Alloy. Therefore, our approach can be extended directly to include elements such as delegation.

Plans are also being drawn up to enhance the existing STRBAC model for cyber-physical Access Control systems [120, 121]. At present, the STRBAC model does not support these systems' specifications. A Cyber-Physical System (CPS) is the integration of computing, communication and storage capabilities with monitoring and controlling entities in the physical world [120]. An architecture for such a system is depicted in Figure 9.1 [123]. The emergence of these systems has had a revolutionary effect on medical devices and systems, advanced automotive systems, process control, assisted living, traffic control and safety, distributed robotics, energy conservation, instrumentation, critical infrastructure control, manufacturing, defence systems and smart structures [122]. Security measures are crucial for CPS applications because entities within the systems interact not only with each other, but also with the physical environment; thus, any security issues must be addressed before CPS applications are deployed on a major basis.

Our initial research indicates that Access Control for Cyber-Physical Systems (CPSs) depends on several factors, for instance the trustworthiness of entities and the environmental context. Entity trustworthiness plays a crucial role in Access Control for cyber-physical systems because CPS does not have well-defined security parameters, and interactions between entities may be unidentified in advance. Moreover, because many entities in CPS belong to the physical world, there is a need to include the impacts of physical security into Access Control decisions in the cyber world. Although this thesis has presented an extension of the STRBAC models to also cover the physical aspect of Access Control, interactions between the two Access Control systems – cyber and physical – have not been discussed. The overarching idea between the two types of Access Control is the concept of trust, as the type of interaction an entity performs with another often depends on the trust relationship between the two systems.

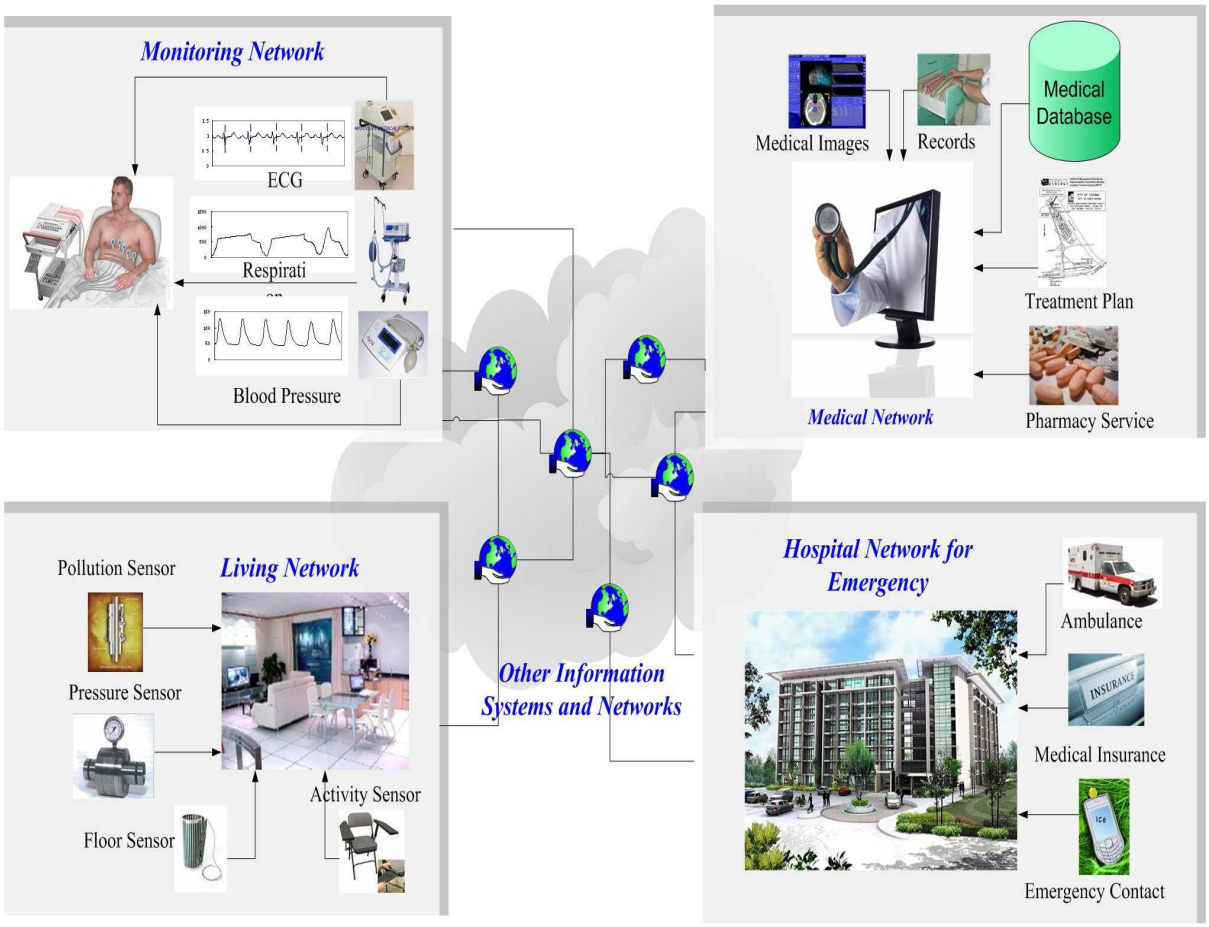

Figure 9.1: Architecture of a Cyber-Physical System [123]

In traditional Access Control models, the concept of trust is implicit [43], which means that only authorised users are totally trusted and acquire all the associated permissions, while unauthorised users are totally untrusted and acquire no permissions whatsoever [43]. Treating trust as a binary concept, determining whether or not a user can be trusted completely may not be suitable for cyber-physical systems, as complete trust may not be possible every time because an entity may only have incomplete knowledge of its counterpart. Entities will not interact with untrusted counterparts, which will often result in the unavailability of systems and services. Therefore, one area of research would be to develop an appropriate non-binary trust framework that would be suitable for cyber-physical systems. The new framework must conform to the concept of different degrees of trust and identify how to quantify and measure the trust value for different devices and users in these systems. Additionally, the model should define how trust evolves in a

dynamic setting.

APPENDIX A

SPATIO-TEMPORAL ROLE BASED ACCESS CONTROL (STRBAC) METAMODEL

This section presents the XSD representation of the STRBAC metamodel that we have created based on the formalisation of the STRBAC model presented in Chapter 3 to conduct the model transformations AC2Alloy and AC2Uppaal presented in Chapter 4, Chapter 5 and Chapter 6 of this thesis.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="STRBAC">
<xs:complexType>
<xs:sequence>
<xs:element name="users">
<xs:complexType>
<xs:sequence>
<xs:element name="user" type="userType" maxOccurs="unbounded" />
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="roles">
<xs:complexType>
```

```
<xs:sequence>
<xs:element name="role" type="roleType" maxOccurs="unbounded" />
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="permissions">
<xs:complexType>
<xs:sequence>
<xs:element name="permission" type="permissionType"
maxOccurs="unbounded" />
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="times">
<xs:complexType>
<xs:sequence>
<xs:element name="time" type="timeType" maxOccurs="unbounded" />
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="locations">
<xs:complexType>
<xs:sequence>
<xs:element name="location" type="locationType"
maxOccurs="unbounded" />
</xs:sequence>
</xs:complexType>
</xs:element>
```

```
<xs:element name="userRoleAssignments">
<xs:complexType>
<xs:sequence>
<xs:element name="userRoleAssignment" type="URAType"
maxOccurs="unbounded" minOccurs="0" />
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="permissionRoleAquires">
<xs:complexType>
<xs:sequence>
<xs:element name="permissionRoleAquire" type="PRAType"
maxOccurs="unbounded" minOccurs="0" />
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="roleHierarchies">
<xs:complexType>
<xs:sequence>
<xs:element name="roleHierarchy" type="roleHierarchyType"
maxOccurs="unbounded" minOccurs="0" />
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="locationHierarchies">
<xs:complexType>
<xs:sequence>
<xs:element name="locationHierarchy" type="locationHierarchyType"
```

```
maxOccurs="unbounded" minOccurs="0" />
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="SODRS">
<xs:complexType>
<xs:sequence>
<xs:element name="SODR" type="SODRType" maxOccurs="unbounded"
minOccurs="0" />
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="SODPS">
<xs:complexType>
<xs:sequence>
<xs:element name="SODP" type="SODPType" maxOccurs="unbounded"
minOccurs="0" />
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="SODRPS">
<xs:complexType>
<xs:sequence>
<xs:element name="SODRP" type="SODRPType" maxOccurs="unbounded"
minOccurs="0" />
</xs:sequence>
</xs:complexType>
</xs:element>
```

```
<xs:element name="Cardinalities">
<xs:complexType>
<xs:sequence>
<xs:element name="Cardinality" type="cardType"
maxOccurs="unbounded" minOccurs="0" />
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="RoleUnassignments">
<xs:complexType>
<xs:sequence>
<xs:element name="RoleUnassigned" type="roleUnassignmentType"
maxOccurs="unbounded" minOccurs="0" />
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="PermissionUnassignments">
<xs:complexType>
<xs:sequence>
<xs:element name="PermissionUnassigned" type="permissionUnassignmentType"
maxOccurs="unbounded" minOccurs="0" />
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:complexType name="userType">
```

```
<xs:attribute name="user_id" type="xs:ID" use="required">
</xs:attribute>
</xs:complexType>
<xs:complexType name="roleType">
<xs:attribute name="role_id" type="xs:ID" use="required">
</xs:attribute>
</xs:complexType>
<xs:complexType name="permissionType">
<xs:attribute name="permission_id" type="xs:ID" use="required">
</xs:attribute>
</xs:complexType>
<xs:complexType name="timeType">
<xs:attribute name="time_id" type="xs:ID" use="required">
</xs:attribute>
</xs:complexType>
<xs:complexType name="locationType">
<xs:attribute name="location_id" type="xs:ID" use="required">
</xs:attribute>
</xs:complexType>
<xs:complexType name="URAType">
<xs:sequence>
<xs:element name="user" type="xs:IDREF" />
<xs:element name="role" type="xs:IDREF" />
<xs:element name="time" type="xs:IDREF" />
<xs:element name="location" type="xs:IDREF" />
</xs:sequence>
<xs:attribute name="URAType_id" type="xs:ID" use="required">
</xs:attribute>
```

```
</xs:complexType>
<xs:complexType name="PRAType">
<xs:sequence>
<xs:element name="role" type="xs:IDREF" />
<xs:element name="permission" type="xs:IDREF" />
<xs:element name="time" type="xs:IDREF" />
<xs:element name="location" type="xs:IDREF" />
</xs:sequence>
<xs:attribute name="PRAType_id" type="xs:ID" use="required">
</xs:attribute>
</xs:complexType>
<xs:complexType name="roleHierarchyType">
<xs:sequence>
<xs:element name="seniorRole" type="xs:IDREF" />
<xs:element name="juniorRole" type="xs:IDREF" />
<xs:element name="time" type="xs:IDREF" />
<xs:element name="location" type="xs:IDREF" />
</xs:sequence>
<xs:attribute name="roleHierarchy_id" type="xs:ID" use="required">
</xs:attribute>
</xs:complexType>
<xs:complexType name="SODRType">
<xs:sequence>
<xs:element name="role" type="xs:IDREF" />
<xs:element name="role2" type="xs:IDREF" />
<xs:element name="time" type="xs:IDREF" />
<xs:element name="location" type="xs:IDREF" />
</xs:sequence>
```

```
<xs:attribute name="SODR_id" type="xs:ID" use="required">
</xs:attribute>
</xs:complexType>
<xs:complexType name="SODPType">
<xs:sequence>
<xs:element name="permission" type="xs:IDREF" />
<xs:element name="permission2" type="xs:IDREF" />
<xs:element name="time" type="xs:IDREF" />
<xs:element name="location" type="xs:IDREF" />
</xs:sequence>
<xs:attribute name="SODP_id" type="xs:ID" use="required">
</xs:attribute>
</xs:complexType>
<xs:complexType name="SODRPType">
<xs:sequence>
<xs:element name="role" type="xs:IDREF" />
<xs:element name="permission" type="xs:IDREF" />
<xs:element name="time" type="xs:IDREF" />
<xs:element name="location" type="xs:IDREF" />
</xs:sequence>
<xs:attribute name="SODRP_id" type="xs:ID" use="required">
</xs:attribute>
</xs:complexType>
<xs:complexType name="locationHierarchyType">
<xs:sequence>
<xs:element name="location" type="xs:IDREF" />
<xs:element name="location2" type="xs:IDREF" />
</xs:sequence>
```

```
<xs:attribute name="LH_id" type="xs:ID" use="required">
</xs:attribute>
</xs:complexType>
<xs:complexType name="cardType">
<xs:sequence>
<xs:element name="role" type="xs:IDREF" />
<xs:element name="time" type="xs:IDREF" />
<xs:element name="location" type="xs:IDREF" />
<xs:element name="cardinality" type="xs:int" />
</xs:sequence>
<xs:attribute name="card_id" type="xs:ID" use="required">
</xs:attribute>
</xs:complexType>
<xs:complexType name="roleUnassignmentType">
<xs:sequence>
<xs:element name="role" type="xs:IDREF" />
<xs:element name="time" type="xs:IDREF" />
<xs:element name="location" type="xs:IDREF" />
</xs:sequence>
<xs:attribute name="roleUnassign_id" type="xs:ID" use=
"required"></xs:attribute>
</xs:complexType>
<xs:complexType name="permissionUnassignmentType">
<xs:sequence>
<xs:element name="role" type="xs:IDREF" />
<xs:element name="time" type="xs:IDREF" />
<xs:element name="location" type="xs:IDREF" />
</xs:sequence>
```

```
<xs:attribute name="permissionUnassign_id" type="xs:ID"
use="required"></xs:attribute>
</xs:complexType>
</xs:schema>
```

APPENDIX B

ALLOY MODEL OF THE SECURE BANK SYSTEM EXAMPLE IN SECTION 4.1

This section presents the complete Alloy textual code automatically generated from the STRBAC specifications relating to the SECURE bank system presented in Section 4.1, using the AC2Alloy model transformation presented in Chapter 5. Some comments have been manually inserted to illustrate which part of the model came from which part of the STRBAC specifications.

```
// Module declaration
module STRBAC

// Corresponds to the Element Users
abstract sig Users{}
one sig Dave, Hanna, Mark, Sarah extends Users{}

// Corresponds to the Element Locations
abstract sig Location{}
one sig Office1, Office2 extends Locations{}

// Corresponds to the Element Times
abstract sig Times{}
```

```
one sig DayTime, NightTime extends Times{}

// Corresponds to the Element Permissions
abstract sig Permissions{}
one sig RWACF, RWACMF, RWLF, RWTF extends Permissions{}

// Corresponds to the Element Roles
abstract sig Roles{
  time: lone Times,
  location: lone Locations,
  permissions: set Permissions,
  users: set Users}

// Corresponds to the role Accounting Manager and URA,
// PRA and RH related to it.
one sig AcountingManager extends Roles{}

fact AcountingManager_fact{all self: AcountingManager |
  AcountingManagerCondition[self]}

pred AcountingManagerCondition[self: AcountingManager]{
  ((self.permissions = RWACF)&&(self.location = Office1)&&
  (self.time = DayTime)&&(self.users = Mark))||((self.permissions =RWACMF)
  &&(self.location =Office1)&&(self.time =NightTime)&&(self.users =Dave))}

// Corresponds to the role Loan Officer and URA, PRA
// and RH related to it.
one sig LoanOfficer extends Roles{}
```

```
fact LoanOfficer_fact{all self: LoanOfficer | LoanOfficerCondition[self]}

pred LoanOfficerCondition[self: LoanOfficer]{
  ((self.permissions = RWLF)&&(self.location = Office2)&&
  (self.time = DayTime)&&(self.users =Sarah))}

// Corresponds to the role Accountant and URA, PRA
// and RH related to it.
one sig Accountant extends Roles{}

fact Accountant_fact{all self: Accountant | AccountantCondition[self]}

pred AccountantCondition[self: Accountant]{
  ((self.permissions = RWACF)&&(self.location = Office1)&&(self.time =
  DayTime)&&(self.users = Hanna))}

// Corresponds to the role Teller and URA, PRA and
// RH related to it.
one sig Taller extends Roles{}

fact Taller_fact{all self: Taller | TallerCondition[self]}

pred TallerCondition[self: Taller]{((self.permissions = RWTF)
  &&(self.location = Office2) && (self.time = DayTime)&&
  (self.users = Sarah))}

//SOD between Roles (Loan Officer and Teller)
```

```
pred SODR[r1, r2 :Roles, l: Locations, t: Times]{
  all u: Users | ((u in r1.users) &&(t in r1.time)&&(l in r1.location)
  &&(t in r2.time)&&(l in r2.location)&&(u != no_users))=>
  ((u not in r2.users)&&(t in r2.time)&&(l in r2.location)) }

SoDR1 :check {SODR[Taller, LoanOfficer, Office2, DayTime]}

//SOD between Permissions (RWTF and RWLF)
pred SODP1[p1, p2 :Permissions,  l: Locations, t: Times]{
  all r1: Roles |((p1 in r1.permissions) &&(t in r1.time)&&
  (l in r1.location))=>((p2 not in r1.permissions)&&(t in r1.time)
  &&(l in r1.location))}

pred SODP2[p1, p2 :Permissions,  l: Locations, t: Time]{
  all r1, r2 : Roles, u: Users | ((p1 in r1.permissions)
  &&(t in r1.time)&&(l in r1.location)&&(u in r1.users)&&
  (t in r2.time)&&(l in r2.location)&&(u in r2.users)=>
  ((p2 not in r2.permissions)&&(t in r2.time)&&
  (l in r2.location)&&(u in r2.users))) }

SoDP1 :check {SODP1[ RWTF, RWLF, Office2, DayTime]}


// Cardinality Constraint over Role
CCP :check {
  ((Office2 in AcountingManager.location)&&(DayTime in
  AcountingManager.time)=>(#AcountingManager.users < 2))}
```

```
// Cardinality Constraint over Permissions
CCP :check {
((Office2 in AcountingManager.location)&&(DayTime in
  AcountingManager.time)=>(#AcountingManager.permissions < 2))}
```

APPENDIX C

TIMED AUTOMATA AND TCTL STATEMENTS FOR THE SECURE BANK SYSTEM EXAMPLE IN SECTION 4.1

C.1 Timed Automata

This section presents the complete graphical representations of the Timed Automata which were automatically generated from the Spatio-Temporal Role Based Access Control (STRBAC) specifications relating to the SECURE bank system presented in Section 4.1, using the AC2Uppaal model transformation presented in Chapter 6. Figure C.1 shows the graphical representation of the Timed Automaton generated for the set of Times in the SECURE bank system example and Figures C.2, C.3, C.4 and C.5 show the Timed Automaton generated for the users `Dave, Sarah, Hanna` and `Mark` respectively.

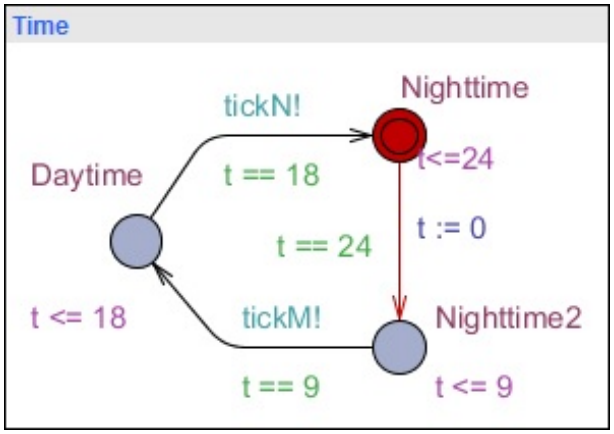

Figure C.1: Timed Automaton generated for the set of Times

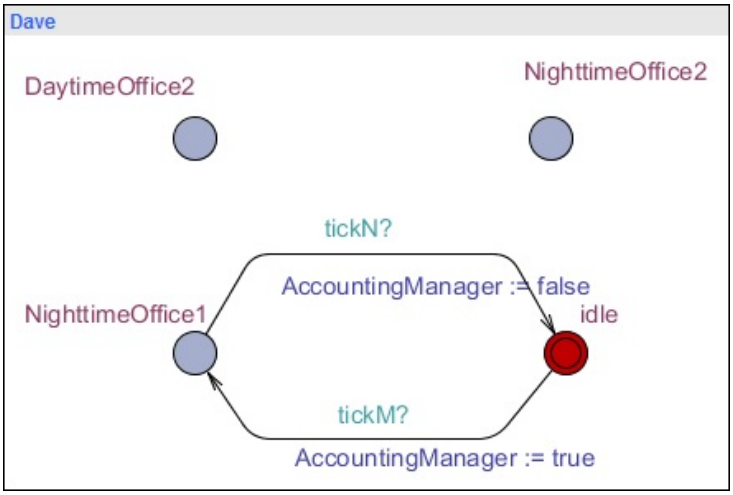

Figure C.2: Timed Automaton generated for the user Dave

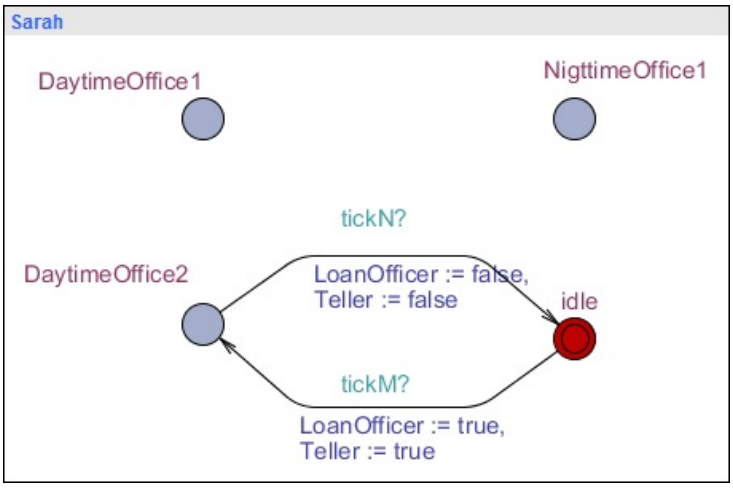

Figure C.3: Timed Automaton generated for the user Sarah

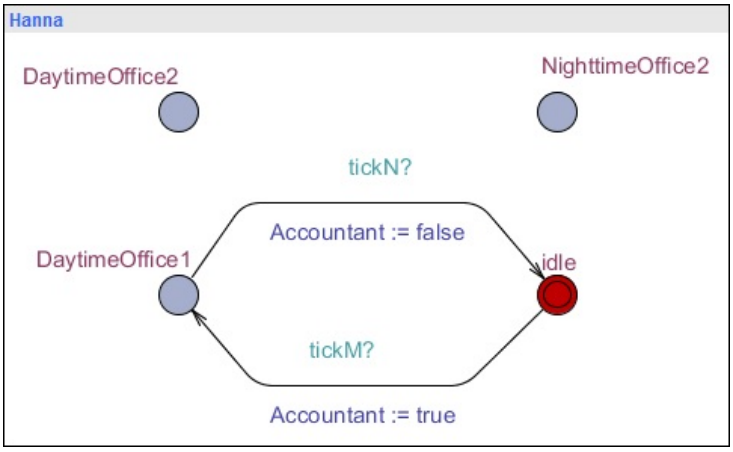

Figure C.4: Timed Automaton generated for the user Hanna

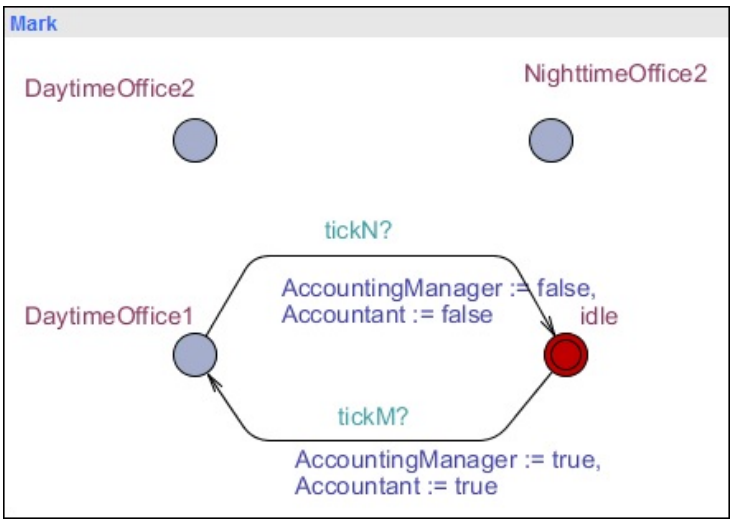

Figure C.5: Timed Automaton generated for the user Mark

C.2 TCTL Statements

This section presents the complete TCTL statements which were automatically generated from the STRBAC specifications pertaining to the SECURE bank system presented in Section 4.1, using the AC2Uppaal model transformation presented in Chapter 6. Some comments have been manually inserted to illustrate which part of the model came from which part of the STRBAC specifications.

```
/* TCTL Statement for the following Permission Role Acquire:
```

```
   (Accountant, RWAMF, Daytime, Office2) */
A[] (Sarah.DaytimeOffice2 and Sarah.Teller or\
     Dave.DaytimeOffice2 and Dave.Teller or\
     Hanna.DaytimeOffice2 and Hanna.Teller or\
     Mark.DaytimeOffice2 and Mark.Teller )

/* TCTL Statement for the following Permission Role Acquire:
(Loan Officer, RWLF, Daytime, Office2) */
A[] (Sarah.DaytimeOffice2 and Dave.LoanOfficer or\
    Dave.DaytimeOffice2 and Dave.LoanOfficer or\
    Hanna.DaytimeOffice2 and John.LoanOfficer or\
    Mark.DaytimeOffice2 and Mark.LoanOfficer )

/* TCTL Statement for the following Permission Role Acquire:
(Accountant, RWAMF, Daytime, Office1) */
A[] (Sarah.DaytimeOffice1 and Sarah.AccountingManager or\
    Dave.DaytimeOfficeQ and Dave.AccountingManager or\
    Hanna.DaytimeOffice1 and Hanna.AccountingManager or\
    Mark.DaytimeOffice1 and Mark.AccountingManager )

/* TCTL Statement for the following Permission Role Acquire:
(Accounting Manager, RWAMF, Nighttime, Office1) */
A[] (Sarah.NighttimeOffice1 and Sarah.AccountingManager or\
    Dave.NighttimeOffice1 and Dave.AccountingManager or\
    Hanna.NighttimeOffice1 and Hanna.AccountingManager or\
    Mark.NighttimeOffice1 and Mark.AccountingManager )

/* TCTL Statement for the following Permission Role Acquire:
```

```
(Accountant, RWAF, Daytime, Office1) */
A[] (Sarah.DaytimeOffice1 and Sarah.Accountant or\
    Dave.DaytimeOffice1 and Dave.Accountant or\
    Hanna.DaytimeOffice1 and Hanna.Accountant or\
    Mark.DaytimeOffice1 and Mark.Accountant )

/* TCTL Statement for the following Role Hierarchy:
(Accounting Manager, Accountant, Daytime, Office1) */
A[] (Sarah.DaytimeOffice1 and Sarah.Accountant or\
    Dave.DaytimeOffice1 and Dave.Accountant or\
    Hanna.DaytimeOffice1 and Hanna.Accountant or\
    Mark.DaytimeOffice1 and Mark.Accountant )

/* TCTL Statement for the following Separation of Duty constraint:
(Teller, Loan Officer, Daytime, Office2) */
A[]((Sarah.DaytimeOffice2 imply not (Sarah.Teller and Sarah.LoanOfficer))
 and\ (Dave.DaytimeOffice2 imply not (Dave.Teller and Dave.LoanOfficer))
 and\ (Hanna.DaytimeOffice2 imply not (Hanna.Teller and Hanna.LoanOfficer))
 and\ (Mark.DaytimeOffice2 imply not (Mark.Teller and Mark.LoanOfficer)))
```

APPENDIX D

ALLOY EXPERIMENTAL RESULTS

The following five tables present the results of using Alloy for analysing the Access Control specifications of the SECURE bank system. Every table shows the time required to complete the analysis of the Access Control specification when one of the following elements of the STRBAC is increased while the other elements remain fixed.

Table D.1: Alloy Evaluation Result – Increasing the number of Users

Scenario	User	Role	Permission	Time	Location	URA	PRA	RH	SoD	No of all Elements	Speed of Analysis
Scenario 1	50	50	50	2	2	50	50	2	1	257	17.4ms
Scenario 2	100	50	50	2	2	50	50	2	1	307	20.2ms
Scenario 3	150	50	50	2	2	50	50	2	1	357	23.2ms
Scenario 4	200	50	50	2	2	50	50	2	1	407	26ms
Scenario 5	250	50	50	2	2	50	50	2	1	457	29.2ms
Scenario 6	300	50	50	2	2	50	50	2	1	507	32.6ms

Table D.2: Alloy Evaluation Result – Increasing the number of Roles

Scenario	User	Role	Permission	Time	Location	URA	PRA	RH	SoD	No of all Elements	Speed of Analysis
Scenario 1	50	50	50	2	2	50	50	2	1	257	17.4ms
Scenario 2	50	100	50	2	2	50	50	2	1	307	21ms
Scenario 3	50	150	50	2	2	50	50	2	1	357	24.5ms
Scenario 4	50	200	50	2	2	50	50	2	1	407	28.1ms
Scenario 5	50	250	50	2	2	50	50	2	1	457	31.8ms
Scenario 6	50	300	50	2	2	50	50	2	1	507	35.7ms

Table D.3: Alloy Evaluation Result – Increasing the number of URA

Scenario	User	Role	Permission	Time	Location	URA	PRA	RH	SoD	No of all Elements	Speed of Analysis
Scenario 1	50	50	50	2	2	50	50	2	1	257	17.4ms
Scenario 2	50	50	50	2	2	100	50	2	1	307	20ms
Scenario 3	50	50	50	2	2	150	50	2	1	357	22.7ms
Scenario 4	50	50	50	2	2	200	50	2	1	407	25.4ms
Scenario 5	50	50	50	2	2	250	50	2	1	457	28.3ms
Scenario 6	50	50	50	2	2	300	50	2	1	507	31.6ms

Table D.4: Alloy Evaluation Result – Increasing the number of PRA

Scenario	User	Role	Permission	Time	Location	URA	PRA	RH	SoD	No of all Elements	Speed of Analysis
Scenario 1	50	50	50	2	2	50	50	2	1	257	17.4ms
Scenario 2	50	50	50	2	2	50	100	2	1	307	20ms
Scenario 3	50	50	50	2	2	50	150	2	1	357	22.6ms
Scenario 4	50	50	50	2	2	50	200	2	1	407	25.3ms
Scenario 5	50	50	50	2	2	50	250	2	1	457	28.2ms
Scenario 6	50	50	50	2	2	50	300	2	1	507	31.5ms

Table D.5: Alloy Evaluation Result – Increasing the number of RH

Scenario	User	Role	Permission	Time	Location	URA	PRA	RH	SoD	No of all Elements	Speed of Analysis
Scenario 1	50	50	50	2	2	50	50	2	1	257	17.4ms
Scenario 2	50	50	50	2	2	50	50	12	1	267	17.8ms
Scenario 3	50	50	50	2	2	50	50	22	1	277	18.3ms
Scenario 4	50	50	50	2	2	50	50	32	1	287	18.8ms
Scenario 5	50	50	50	2	2	50	50	42	1	297	19.4ms
Scenario 6	50	50	50	2	2	50	50	52	1	307	20.2ms

APPENDIX E

TIMED AUTOMATA EXPERIMENTAL RESULTS

The following five tables present the results of using Timed Automata for analysing the Access Control specifications of the SECURE bank system. Every table shows the time required to complete the analysis of the Access Control specification when one of the following elements of the STRBAC is increased while the other elements remain fixed.

Table E.1: Timed Automata Evaluation Result – Increasing the number of Users

Scenario	User	Role	Permission	Time	Location	URA	PRA	RH	SoD	No of all Elements	Speed of Analysis
Scenario 1	50	50	50	2	2	50	50	2	1	257	14.5ms
Scenario 2	100	50	50	2	2	50	50	2	1	307	14.9ms
Scenario 3	150	50	50	2	2	50	50	2	1	357	15.4ms
Scenario 4	200	50	50	2	2	50	50	2	1	407	16.2ms
Scenario 5	250	50	50	2	2	50	50	2	1	457	17.3ms
Scenario 6	300	50	50	2	2	50	50	2	1	507	18.5ms

Table E.2: Timed Automata Evaluation Result – Increasing the number of Roles

Scenario	User	Role	Permission	Time	Location	URA	PRA	RH	SoD	No of all Elements	Speed of Analysis
Scenario 1	50	50	50	2	2	50	50	2	1	257	14.5ms
Scenario 2	50	100	50	2	2	50	50	2	1	307	14.8ms
Scenario 3	50	150	50	2	2	50	50	2	1	357	15.1ms
Scenario 4	50	200	50	2	2	50	50	2	1	407	15.4ms
Scenario 5	50	250	50	2	2	50	50	2	1	457	15.7ms
Scenario 6	50	300	50	2	2	50	50	2	1	507	16ms

Table E.3: Timed Automata Evaluation Result – Increasing the number of URA

Scenario	User	Role	Permission	Time	Location	URA	PRA	RH	SoD	No of all Elements	Speed of Analysis
Scenario 1	50	50	50	2	2	50	50	2	1	257	14.5ms
Scenario 2	50	50	50	2	2	100	50	2	1	307	15ms
Scenario 3	50	50	50	2	2	150	50	2	1	357	15.8ms
Scenario 4	50	50	50	2	2	200	50	2	1	407	17ms
Scenario 5	50	50	50	2	2	250	50	2	1	457	18.7ms
Scenario 6	50	50	50	2	2	300	50	2	1	507	21ms

Table E.4: Timed Automata Evaluation Result – Increasing the number of PRA

Scenario	User	Role	Permission	Time	Location	URA	PRA	RH	SoD	No of all Elements	Speed of Analysis
Scenario 1	50	50	50	2	2	50	50	2	1	257	14.5ms
Scenario 2	50	50	50	2	2	50	100	2	1	307	15ms
Scenario 3	50	50	50	2	2	50	150	2	1	357	15.9ms
Scenario 4	50	50	50	2	2	50	200	2	1	407	17.3ms
Scenario 5	50	50	50	2	2	50	250	2	1	457	19.1ms
Scenario 6	50	50	50	2	2	50	300	2	1	507	21.7ms

Table E.5: Timed Automata Evaluation Result – Increasing the number of RH

Scenario	User	Role	Permission	Time	Location	URA	PRA	RH	SoD	No of all Elements	Speed of Analysis
Scenario 1	50	50	50	2	2	50	50	2	1	257	14.5ms
Scenario 2	50	50	50	2	2	50	50	12	1	267	14.6ms
Scenario 3	50	50	50	2	2	50	50	22	1	277	14.8ms
Scenario 4	50	50	50	2	2	50	50	32	1	287	15.1ms
Scenario 5	50	50	50	2	2	50	50	42	1	297	15.6ms
Scenario 6	50	50	50	2	2	50	50	52	1	307	16.2ms

LIST OF REFERENCES

[1] X. Su, D. Bolzoni, P. van Eck, R. Wieringa A Business Goal-Driven Approach for Understanding and Specifying Information Security Requirements, 2006

[2] J.A Schweitzer, Protecting Business Information: a manager's guide, Butterworth-Heinemann, Boston, 1996.

[3] D.F. Ferraiolo, R. Sandhu, S. Gavrila, D.R. Kuhn and R. Chandramouli, Proposed NIST Standard for Role-Based Access Control, ACM Trans. Information and System Security, vol. 4, no. 3, pp. 224-274, Aug. 2001.

[4] D. E. Denning, Information Warfare and Security, Addison-Wesley, Reading, MA, 2000.

[5] D.F Ferraiolo, D.R. Kuhn, R. Chandramouli, Role Based Access Control, Artech House, MA, 2003

[6] R. Anderson, Security Engineering A guide to building dependable distributed systems, Wiley, NY, 2001

[7] D. Bell and L. J. LaPadula. Secure computer systems: unified exposition and Multics interpretation. Technical Report MTR-2997, MITRE Corporation, July 1975.

[8] Michael V. Hayden. National Information Systems Security Glossary. National Security Telecommunications and Information Systems Security Committee, Sep 2000.

[9] Ravi S. Sandhu and Pierangela Samarati. Access Control: Principles and practice. IEEE Communications Magazine, 1994.

[10] Ravi Sandhu, Edward Coyne, Hal Feinstein, and Charles Youman. Role-based access control models. IEEE Computer, 29(2): pp. 38-47, Feb. 1996.

[11] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role Based Access Control models. IEEE Computer, 29(2): pp. 38-47, February 1996.

[12] Indrakshi Ray and Manachai Toahchoodee. A Spatio-Temporal Role Based Access Control Model. In Proceedings of the 21st Annual IFIP WG 11.3 Working Conference on Data and Applications Security, pp. 211-226, Redondo Beach, CA, July 2007.

[13] Indrakshi Ray and Manachai Toahchoodee. A Spatio-Temporal Access Control Model Supporting Delegation for Pervasive Computing Applications. In Proceedings of the 5th International Conference on Trust, Privacy and Security in Digital Business, pp. 48-58, Turin, Italy, September 2008.

[14] Manachai Toahchoodee and Indrakshi Ray. On the Formal Analysis of a Spatio-Temporal Role Based Access Control Model. In Proceedings of the 22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security, pp. 17-32, London, U.K., July 2008.

[15] Manachai Toahchoodee and Indrakshi Ray. Using Alloy to Analyze a Spatio-Temporal Access Control Model Supporting Delegation. IET Information Security, 3(3):75-113, September 2009.

[16] Emsaieb Geepalla and Behzad Bordbar. On formalizing of Inconsistency and Semi-consistency in Spatio-Temporal Access Control. In the Seventh IEEE International Conference on Information Management (ICDIM), pp. 22-29. IEEE 2012.

[17] B.Potter, J.Sinclair, and D.Till. An Introduction to Formal Specification and Z. Prentice-Hall, New York, NY, 1991.

[18] Kurt Jensen. Coloured Petri nets: basic concepts, analysis methods and practical use. Springer-Verlag New York, Inc., New York, NY, USA, 1997.

[19] Kurt Jensen and Lars M. Kristensen. Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Springer, 2009.

[20] Daniel Jackson. Alloy: a lightweight object modelling notation. In ACM Transactions on Software Engineering and Methodology (TOSEM), pp. 256-290. ACM Press, Apr 2002.

[21] Daniel Jackson. Micromodels of Software: Lightweight Modelling and Analysis with Alloy. Software Design Group, MIT Lab for Computer Science, Feb 2002. This document and the tool can be obtained from http://alloy.mit.edu/.

[22] Daniel Jackson. Software Abstractions: Logic, Language, and Analysis. The MIT Press, London, England, 2006.

[23] J. Bengtsson and W .Yi. Timed automata: Semantics, algorithms and tools. In Lecture Notes on Concurrency and Petri Nets, volume 3098, pp. 87-124, 2004.

[24] R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking for real-time systems. In Proc. of the 5th Annual Symposium on Logic in Computer Science, pp. 414-425. IEEE, 1990.

[25] G. Behrmann, A. David, and K. G. Larsen. A Tutorial on Uppaal 4.0. Department of computer science, Aalborg university, 2006.

[26] Anneke Kleppe, Jos Warmer, and Wim Bast. MDA Explained: The Model-Driven ArchitecturePractice and Promise. The Addison-Wesley Object Technology Series. Addison-Wesley, 2003.

[27] Emsaieb Geepalla, Behzad Bordbar, and Joel Last. Transformation of spatio-temporal role based access control specification to alloy. In Model and Data Engineering, pp. 67-78. Lecture Notes in Computer Science, Springer, 2012.

[28] Emsaieb Geepalla, Behzad Bordbar, Kozo Okano and Seyedhamed Seyedali. AC2Uppaal: A Tool for automatic Generation of Timed Automata form Spatio-Temporal Role Based Access Control Specifications. To appear in Proceeding of World Congress of Internet Security (WorldCIS) 2013. IEEE, 2013.

[29] F.M. Kugblenu and M. Asim, Separation of Duty in Role Based Access Control System: A Case Study, MS Thesis ,Thesis no: MCS-2006:16, January 2007.

[30] Emsaieb Geepalla, Behzad Bordbar, and Xiaofeng Du. Spatio-Temporal Role Based Access Control for Physical Access Control Systems. To appear in the Fourth International Conference on Emerging Security Technologies (EST) 2013. IEEE, 2013.

[31] Sabrina De Capitani di Vimercati, Stefano Paraboschi, and Pierangela Samarati. Access Control: principles and solutions. Software Practice and Experience, 33: pp. 397-421, 2003.

[32] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in Operating Systems. Communications of the ACM, 19(8): pp. 461-471, August 1976.

[33] National Computer Security Center. A Guide to Understanding Discretionary Access Control in Trusted Systems, September 1987.

[34] Alur R. Timed automata. In Proceedings of the 11th international conference on computer-aided verification (CAV), July 1999. Lecture notes in computer science, vol 1633. Springer, Berlin Heidelberg New York, pp. 8-22.

[35] Ravi S. Sandhu. Lattice-Based Access Control Models. Computer, 26(11): pp. 9-19, November 1993.

[36] Sbastien Saudrais, Olivier Barais, Laurence Duchien and Nol Plouzeau. From formal specifications to QoS monitors. Journal of Object Technology, Special Issue on Advances in Quality of Service Management, 6(11), pp. 7-24 - 2007.

[37] James B. D. Joshi and Elisa Bertino. Fine-grained role-based delegation in presence of the hybrid role hierarchy. In Proceedings of the 11th ACM Symposium on Access Control Models and Technologies, pp. 81-90, Lake Tahoe, California, USA, June 2006.

[38] Matunda Nyanchama and Sylvia Osborn. Modeling Mandatory Access Control in Role- Based Security Systems. In Proceedings of the 9th annual IFIP TC11 WG11.3 working conference on Database security IX : status and prospects, pp. 129-144, Rennselaerville, NY, USA, August 1995.

[39] Sylvia Osborn. Mandatory Access Control and Role Based Access Control Revisited. In Proceedings of the 2nd ACM workshop on Role Based Access Control, pp. 31-40, Fairfax, VA, USA, November 1997.

[40] Sylvia Osborn, Ravi Sandhu, and Qamar Munawer. Configuring Role Based Access Control to enforce mandatory and discretionary Access Control policies. ACM Transactions on Information and System Security, 3(2): pp. 85-106, May 2000.

[41] Ravi Sandhu and QamarMunawer. How to do Discretionary Access Control using Roles. In Proceedings of the 3rd ACM Workshop on Role Based Access Control, pp. 47-54, Fairfax, VA, USA, October 1998.

[42] Richard Simon and Mary Ellen Zurko. Separation of Duty in Role-based Environments. In Proceedings of the 10th Computer Security Foundations Workshop, pp. 183-194, Rockport, MA, USA, June 1997.

[43] Indrakshi Ray and Indrajit Ray. Challenges for Cyber-Physical Systems. In Proceedings of the 1st NSF Workshop on Cyber-Physical Systems, Newark, NJ, July 2009.

[44] Michael J. Covington, Wende Long, Srividhya Srinivasan, Anind Dey, Mustaque Ahamad, and Gregory Abowd. Securing Context-Aware Applications Using Environment Roles. In Proceedings of the 6th ACM Symposium on Access Control Models and Technologies, pp. 10-20, Chantilly, VA, USA, May 2001.

[45] Michael J. Covington, Prahlad Fogla, Zhiyuan Zhan, and Mustaque Ahamad. A Context-Aware Security Architecture for Emerging Applications. In Proceedings of the Annual Computer Security Applications Conference , pp. 249-260, Las Vegas, NV, USA, December 2002.

[46] Sudip Chakraborty and Indrajit Ray. TrustBAC: integrating trust relationships into the RBAC model for Access Control in open systems. In Proceedings of the 11th ACM symposium on Access Control models and technologies, pp. 49-58, Lake Tahoe, California, USA, 2006. ACM.

[47] Guo Ya-Jun, Hong Fan, Zhang Qing-Guo, and Li Rong. An Access Control Model for Ubiquitous Computing Application. In Proceedings of the 2nd International Conference on Mobile Technology, Applications and Systems, pp. 1-6, Guangzhou, China, November 2005.

[48] Elisa Bertino, Piero Andrea Bonatti, and Elena Ferrari. TRBAC: a temporal Role Based Access Control model. In Proceedings of the 5th ACM Workshop on Role Based Access Control, pp. 21-30, Berlin, Germany, July 2000.

[49] James B. D. Joshi. A Generalized Temporal Role Based Access Control Model for Developing Secure Systems. PhD thesis, Purdue University, August 2003.

[50] James B. D. Joshi and Elisa Bertino. Fine-grained role-based delegation in presence of the hybrid role hierarchy. In Proceedings of the 11th ACM Symposium on Access Control Models and Technologies, pp. 81-90, Lake Tahoe, California, USA, June 2006.

[51] James B. D. Joshi, Basit Shafiq, Arif Ghafoor, and Elisa Bertino. Dependencies and separation of duty constraints in GTRBAC. In Proceedings of the 8th ACM Symposium on Access Control Models and Technologies, pp. 51-64, New York, NY, USA, 2003.

[52] James B.D. Joshi, Elisa Bertino, Usman Latif, and Arif Ghafoor. A Generalized Temporal Role Based Access Control Model. IEEE Transactions on Knowledge and Data Engineering, 17(1): pp. 4-23, January 2005.

[53] Elisa Bertino, Barbara Catania, Maria Luisa Damiani, and Paolo Perlasca. GEO-RBAC: a spatially aware RBAC. In Proceedings of the 10th ACM Symposium on Access Control Models and Technologies, pp. 29-37, Stockholm, Sweden, June 2005.

[54] Indrakshi Ray, Mahendra Kumar, and Lijun Yu. LRBAC: A Location-Aware Role Based Access Control Model. In Proceedings of the 2nd International Conference on Information Systems Security, pp. 147-161, Kolkata, India, December 2006.

[55] M. P. Gallagher, A.C. OConnor, and B. Kropp. The economic impact of Role Based Access Control. In Planning report 02-1, NIST,, March 2002.

[56] Thi, K.T.L., Dang, T.K., Kuonen, P., Drissi, H.C. STRoBAC-Spatial Temporal Role Based Access Control. In Proceedings of the 4th International Conference on Computational Collective Intelligence Technologies and Applications (ICCCI), Part II, LNAI 7654, Springer-Verlag Heidelberg, pp. 201-211, 28-30 November 2012, Ho Chi Minh city, Vietnam.

[57] Liang Chen and Jason Crampton. On Spatio-Temporal Constraints and Inheritance in Role Based Access Control. In Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security, pp. 205-216, Tokyo, Japan, March 2008.

[58] SuroopMohan Chandran and James B. D. Joshi. LoT-RBAC: A Location and Time-Based RBAC Model. In Proceedings of the 6th International Conference on Web Information Systems Engineering, pp. 361-375, New York, NY, USA, November 2005.

[59] Arjmand Samuel, Arif Ghafoor, and Elisa Bertino. A Framework for Specification and Verification of Generalized Spatio-Temporal Role Based Access Control Model. Technical report, Purdue University, February 2007. CERIAS TR 2007-08.

[60] D. D. Clark, D. R. Wilson, A Comparison of Commercial and Military Computer Security Policies, IEEE Symposium on Security and Privacy, pp. 184-194, 1987.

[61] J. B. D. Joshi, W. G. Aref, A. Ghafoor and E. H. Spafford. Security Models for Web-based Applications. Communications of the ACM, 44, 2 (Feb. 2001), pp. 38-72.

[62] Dimitrios S. Kolovos, Richard F. Paige, and Fiona A.C. Polack. Eclipse Development Tools for Epsilon. In Eclipse Modeling Symposium, Eclipse Summit Europe 2006, Esslingen, Germany, 2006.

[63] J. R. Falleri, M. Huchard, and C. Nebut. Towards a traceability framework for model transformations in Kermeta. In ECMDA-TW'06: ECMDA Traceability Workshop, pp. 31-40, Sintef ICT, Norway, 2006.

[64] Indrakshi Ray: Using Graph Theory to Represent a Spatio-Temporal Role Based Access Control Model. IJNGC 1(2) (2010).

[65] Richard F. Paige, Gran K. Olsen, Dimitrios S. Kolovos, Steffen Zschaler, and Christopher Power. Building model-driven engineering traceability classifications. In 4th ECMDA Traceability Workshop, 2008.

[66] Manachai Toahchoodee, Indrakshi Ray, Kyriakos Anastasakis, Geri Georg, Behzad Bordbar: Ensuring Spatio-Temporal Access Control for real-world applications. SACMAT 2009, pp. 13-22

[67] S. Shah, K. Anastasakis, and B. Bordbar. Using traceability for reverse instance transformations with SiTra. In Design and Architectures for Signal and Image Processing (DASIP 2008). Special Session on Formal Models, Transformations and Architectures for Re liable Embedded System Design, Bruxelles, Belgium, 2008.

[68] Chunyang Yuan, Yeping He, Jianbo He, and Zhouyi Zhou. A Verifiable Formal Specification for RBAC Model with Constraints of Separation of Duty. In Proceedings of the 2nd SKLOIS Conference on Information Security and Cryptology, pp. 196-210, Beijing, China, November 2006.

[69] Indrakshi Ray, Mahendra Kumar, and Lijun Yu. LRBAC: A Location-Aware Role Based Access Control Model. In Proceedings of the 2nd International Conference on Information Systems Security, pp. 147-161, Kolkata, India, December 2006.

[70] Indrakshi Ray, Na Li, Robert France, and Dae-Kyoo Kim. Using UML to Visualize Role Based Access Control Constraints. In Proceedings of the 9th ACM symposium on Access Control Models and Technologies, pp. 115-124, Yorktown Heights, NY, USA, June 2004.

[71] Vijayalakshmi Atluri and Wei-Kuang Huang. A petri net based safety analysis of workflow authorization models. Journal of Computer Security, 8(2,3): pp. 209-240, August 2000.

[72] Yixin Jiang, Chuang Lin, Hao Yin, and Zhangxi Tan. Security analysis of mandatory Access Control model. In Proceedings of IEEE International Conference on Systems, Man and Cybernetics, volume 6, pp. 5013-5018 vol.6, October 2004.

[73] Romain Laborde, Bassem Nasser, Frederic Grasset, Francois Barrere, and Abdelmalek Benzekri. A Formal Approach for the Evaluation of Network Security Mechanisms Based on RBAC Policies. Electronic Notes in Theoretical Computer Science, 121: pp. 117-142, February 2005.

[74] Yahui Lu, Li Zhang, and Jiaguang Sun. Using colored Petri nets to model and analyze workflow with separation of duty constraints. The International Journal of Advanced Manufacturing Technology, 40(1,2): pp. 179-192, January 2009.

[75] Jens Linneberg Rasmussen and Mejar Singh. Designing a Security System by Means of Coloured Petri Nets. In Proceedings of the 17th International Conference on Application and Theory of Petri Nets, pp. 400-419, London, UK, June 1996. Springer-Verlag.

[76] John Zao, Hoetech Wee, Jonathan Chu, and Daniel Jackson. RBAC Schema Verification Using Lightweight Formal Model and Constraint Analysis. At http://alloy.mit.edu/publications.php, 2002.

[77] Andreas Schaad and Jonathan Moffett. A lightweight approach to specification and analysis of Role Based Access Control extensions. In SACMAT'02, Monterey, California, USA, Jun 2002.

[78] Andreas Schaad, Jonathan Moffett, and Jeremy Jacob. The Role Based Access Control system of a European bank: a case study and discussion. In SACMAT'01, Chantilly, Virginia, USA, May 2001. ACM Press.

[79] Arjmand Samuel, Arif Ghafoor, and Elisa Bertino. A Framework for Specification and Verification of Generalized Spatio-Temporal Role Based Access Control Model. Technical report, Purdue University, February 2007. CERIAS TR 2007-08.

[80] Samrat Mondal, Shamik Sural, and Vijayalakshmi Atluri. Towards Formal Security Analysis of GTRBAC using Timed Automata. In Proceedings of the 14th ACM Symposium on Access Control Models and Technologies, pp. 33-42, Stresa, Italy, June 2009.

[81] Samrat Mondal and Shamik Sural, Security Analysis of Temporal RBAC Using Timed Automata, in Proceedings of the 4th International Conference on Information Assurance and Security (IAS), 2008, pp 37-40.

[82] Gerd Behrmann, Alexandre David, and Kim Guldstrand Larsen. A Tutorial on Uppaal. In 4th International School on FormalMethods for the Design of Computer, Communication and Software Systems, pp. 200-236, Bertinoro, Italy, September 2004.

[83] Y. S. Mahajan, Z. Fu, and S. Malik, Zchaff2004: An efficient sat solver. Lecture Notes in Computer Science: Theory and Applications of Satisability Testing, 7th International Conference, Invited Paper, vol. 3542, pp. 360-375, 2005.

[84] SAT4J: A satisfiability library for Java [online]. Available at: URL: http://www.sat4j.org/. [cited Feb 2013].

[85] Alloy Analyzer, Available from: URL: http://alloy.mit.edu/alloy/. [cited Dec 2012].

[86] Emina Torlak and Daniel Jackson. Kodkod: A relational model finder. In Orna Grumberg and Michael Huth, editors, TACAS, volume 4424 of Lecture Notes in Computer Science, pp. 632-647. Springer, 2007.

[87] Niklas En and Niklas Srensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, SAT, volume 2919 of Lecture Notes in Computer Science, pp. 502-518. Springer, 2003.

[88] Boucheneb, H., Gardey, G., Roux, O.H.: TCTL Model Checking of Time Petri Nets. Journal of Logic and Computation 19(6), pp 1509-1540 (2009)

[89] Behrmann G, Bouyer P, Fleury E, Larsen KG. Static guard analysis in Timed Automata versification. Proceedings of the 9th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'03). Springer; 2003; pp. 254-277. Vol. 2619 of Lecture Notes in Computer Science.

[90] Behrmann G, Bouyer P, Larsen KG, Pelnek R. Lower and upper bounds in zone based abstractions of Timed Automata. Proceedings of the 10th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'04). Springer; 2004; pp. 312-326. Vol. 2988 of Lecture Notes in Computer Science.

[91] Md Tawhid Bin Waez, Juergen Dingel, and Karen Rudie. Timed automata for the development of real-time systems. Technical report, Queen's University, Ontario, Canada, 2011. URL http:// research.cs.queensu.ca/TechReports/Reports/2011-579.pdf.

[92] Gerd Behrmann, Alexandre David, and Kim Guldstrand Larsen. A Tutorial on Uppaal. In 4th International School on Formal Methods for the Design of Computer, Communication and Software Systems, pp. 200-236, Bertinoro, Italy, September 2004.

[93] Anneke Kleppe, Jos Warmer, and Wim Bast. MDA Explained: The Model-Driven ArchitecturePractice and Promise. The Addison-Wesley Object Technology Series. Addison-Wesley, 2003.

[94] David S. Frankel. Model-Driven Architecture: Applying MDA to Enterprise Computing. Wiley Publishing, Indianapolis, Indiana, 2003.

[95] Soon-Kyeong Kim, Damian Burger, and David Carrington. An MDA Approach Towards Integrating Formal and Informal Modeling Languages. In John Fitzgerald, Ian J. Hayes, and Andrzej Tarlecki, editors, FM 2005: Formal Methods: International Symposium of Formal Methods Europe, volume 3582 of Lecture Notes in Computer Science, pp. 448-464. Springer, 2005.

[96] F. Fleurey, J. Steel, and B. Baudry. Validation in Model-Driven Engineering: Testing Model Transformations. In First International Workshop on Model, Design and Validation, pp. 29-40, 2004.

[97] MOF. Meta Object Facility (MOF) 2.0 Core Specification, Object Management Group, available at http://www.omg.org. [cited Mar 2013].

[98] J. R. Falleri, M. Huchard, and C. Nebut. Towards a traceability framework for model transformations in Kermeta. In ECMDA-TW'06: ECMDA Traceability Workshop, pp 31-40, Sintef ICT, Norway, 2006.

[99] Kermeta, Triskell Metamodelling Kernel, Available from: http://www.kermeta.org/ 2013. [cited Feb 2013].

[100] Frdric Jouault. Loosely Coupled Traceability for ATL. In Proceedings of the European Conference on Model-Driven Architecture (ECMDA) workshop on traceability, pp. 29-37, Nuremberg, Germany, 2005.

[101] Frdric Jouault and Ivan Kurtev. On the Architectural Alignment of ATL and QVT. In SAC'06: Proceedings of the 2006 ACM symposium on Applied computing, pp. 1188-1195, Dijon, France, 2006. ACM.

[102] Frdric Jouault and Ivan Kurtev. Transforming Models with ATL. In Satellite Events at the MoDELS 2005 Conference, volume 3844 of Lecture Notes in Computer Science, pp. 128-138. Springer, 2006.

[103] ArcStyler 4.0, Available at: http://www.arcstyler.com/. [cited Jan 2013].

[104] OptimalJ. Computare software corporation, Available at http://technology.amis.nl/2006/10/22/optimalj-tutorial-for-dummies/. [cited Jan 2013].

[105] The Xactium XMF Mosaic. Available at: http://www.modelbased.net/www.xactium.com/. [cited Jan 2013].

[106] Kyriakos Anastasakis, Behzad Bordbar, Geri Georg, and Indrakshi Ray. UML2Alloy: A Challenging Model Transformation. In ACM/IEEE 10th International Conference on Model-Driven Engineering Languages and Systems (MoDELS 2007), 2007.

[107] Behzad Bordbar and Kyriakos Anastasakis. UML2Alloy: A tool for lightweight modelling of Discrete Event Systems. In Nuno Guimares and Pedro Isaas, editors, IADIS International Conference in Applied Computing 2005, volume 1, pp. 209-216, Algarve, Portugal, 2005. IADIS Press.

[108] David H. Akehurst, Behzad Bordbar, M. J. Evans, W. Gareth J. Howells, Klaus D. Mcdonald-maier. SiTra: Simple Transformations in Java, ACM/IEE 9th ICMELS, LNCS, 4199, pp. 351-364.

[109] SiTra: Simple Transformer, Available at: http://www.cs.bham.ac.uk/∼ bxb/Sitra/index.html. [cited Oct 2012].

[110] Steve K. Wood, David H. Akehurst, Oleg Uzenkov, W. G. J. Howells, and Klaus D. McDonald-Maier. A model-driven development approach to mapping uml state diagrams to synthesizable vhdl. IEEE Trans. Comput., 57: pp. 1357-1371, 2008.

[111] Behzad Bordbar, Gareth Howells, Michael Evans, and Athanasios Staikopoulos. Model transformation from owl-s to bpel via sitra. In Proceedings of the 3rd European conference on Model driven architecture-foundations and applications, ECMDA-FA'07, pp. 43-58, Berlin, Heidelberg, 2007. Springer-Verlag.

[112] Mohammed Alodib, Behzad Bordbar, and Basim Majeed. A model driven approach to the design and implementing of fault tolerant service oriented architectures. In 3rd

International Conference on Digital Information Management (ICDIM), pp. 464-469, 2008.

[113] Emsaieb Geepalla and Behzad Bordbar. An Automated Approach to Detect Inconsistency and Semi-consistency Spatio-Temporal Role Based Access Control Specification. Journal of Data Processing Volume 2, no. 3 (2012).

[114] Chao Huang, Jianling Sun, Xinyu Wang, Yuanjie Si: Minimal role mining method for Web service composition. Journal of Zhejiang University - Science C 11(5): pp. 328-339 (2010).

[115] Xinyu Wang, Jianling Sun, Xiaohu Yang, Chao Huang, Di Wu: Security Violation Detection for RBAC Based Interoperation in Distributed Environment. IEICE Transactions 91-D(5): pp. 1447-1456 (2008).

[116] XSD, XML Schema. Available at: http://www.w3schools.com/schema/default.asp. [cited Dec 2012].

[117] Java Architecture for XML Binding. Available from: http://jaxb.java.net/ tutorial/. [cited Dec 2012].

[118] XML: EXtensible Markup Language, Available at: http://www.w3schools.com/ xml/xmlwhatis.asp. [cited Dec 2012].

[119] N. D'Ippolito, Marcelo Frias, Juan Pablo Galeotti, Esteban Lanzarotti, Sergio Mera. Alloy+HotCore: A Fast Approximation to Unsat Core, 2nd International Conference on Abstract State Machines, Alloy, B and Z, Volume 5977, pp. 160-173 - 2010

[120] E. A. Lee. Cyber Physical Systems: design challenges. 11th IEEE Symposium on Object Oriented Real Time Distributed Computing (ISORC). 2008: pp. 363-369.

[121] T. Abdelzaher. Research Challenges in Distributed Cyber-Physical Systems[C]. In Proc of IEEE/IFIP International Conference on Embedded and Ubiquitous Computing. December 2008.

[122] V. Muppavarapu and S. M. Chung, Role Based Access Control for cyber-physical systems using shibboleth, in Proceedings of DHS Workshop on Future Directions in Cyber-Physical Systems Security, pp. 57-60, 2009.

[123] Architecture of cyber-physical information network. Available at: http://rt.cs. virginia.edu/ stankovic/cps.html. [cited Mar 2013].

[124] Marcelo F. Frias , Juan P. Galeotti , Carlos G. Lpez Pombo , Nazareno M. Aguirre, DynAlloy: upgrading alloy with actions, Proceedings of the 27th international conference on Software engineering, pp. 442-451, May 15-21, 2005, St. Louis, MO, USA