DISSERTATION

AN ACCESS CONTROL FRAMEWORK FOR MOBILE APPLICATIONS

Submitted by

Ramadan Abdunabi

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Spring 2013

ABSTRACT

AN ACCESS CONTROL FRAMEWORK FOR MOBILE APPLICATIONS

With the advent of wireless and mobile devices, many new applications are being developed that make use of the spatio-temporal information of a user in order to provide better functionality. Such applications also necessitate sophisticated authorization models where access to a resource depends on the credentials of the user and also on the location and time of access. Consequently, traditional access control models, such as, Role-Based Access Control (RBAC), has been augmented to provide spatio-temporal access control. However, the velocity of technological development imposes sophisticated constraints that might not be possible to support with earlier works. In this dissertation, we provide an access control framework that allows one to specify, verify, and enforce spatio-temporal policies of mobile applications.

Our specification of spatio-temporal access control improves the expressiveness upon earlier works by providing features that are useful for mobile applications. Thus, an application using our model can specify different types of spatio-temporal constraints. It defines a number of novel concepts that allow ease of integration of access control policies with applications and make policy models more amenable to analysis. Our access control models are presented using both theoretical and practical methods.

Our models have numerous features that may interact to produce conflicts. Towards this end, we also develop automated analysis approaches for conflict detection and correction at model and application levels. These approaches rigorously check policy models and provide feedback when some properties do not hold. For strict temporal behaviour, our analysis can be used to perform a quantitative verification of the temporal properties while considering mobility. We also provide a number of techniques to reduce the state-space explosion problem that is inherent in model checkers.

Furthermore, we introduce a policy enforcement mechanism illustrates the practical viability of our models and discusses potential challenges with possible solutions. Specifically, we propose an event-based architecture for enforcing spatio-temporal access control and demonstrate its feasibility by developing a prototype. We also provide a number of protocols for granting and revoking access and formally analyze these protocols in order to provide assurance that our proposed architecture is indeed secure.

ACKNOWLEDGEMENTS

I would like to express my special gratitude to my advisor, Professor Indrakshi Ray, for giving me the opportunity to be her Ph.D. student, and for her endless patience and constant encouragement whenever I was in doubt during my Ph.D. research. Her 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 Professor Indrakshi has been precious, and this experience will be the most useful and important for my prospective faculty position at University of Benghazi, Libya. I am fortunate to be her Ph.D. advisee and this definitely enables me to advise, inspire, and provide such valuable guidances to my students. I also owe a special note of thank to my advisory committee members, Professors Robert France, Indrajit Ray, and Daniel Turk, for taking the time to give me constructive feedback on my work.

I would like to express special thanks to Professor Daniel Turk for the guidelines of a concise presentation of my work. His constructive comments, enthusiastic help, diverse suggestions during the presentation of my preliminary results considerably enhance my final defense presentation.

I am grateful to my advisory committee member, Professor Indrajit Ray, for his endless help during my study and for educating me all the computer security principles and practices, especially in the area of the access control model, which constructs the backbone knowledge of this dissertation. His constructive comments during the presentation of my partial work in the Information Security Group meetings were really thoughtful and significantly improve my Ph.D. significance and contributions; and his invaluable comments after the presentation of my preliminary results guide me for better preparation of the final presentation. I am also thankful to Professor Indrajit and Indrakshi for granting me a Graduate Teaching Assistantship, which was extremely helpful to my family and I in alleviating the financial burden of my studies and living expenses. Their insightful advice and inspiring vision have contributed to my personal life, publications, and completing this dissertation.

I owe a great debt to Professor Robert France and his Ph.D. advisee students, Mustafa Al Lail and Wuliang Sun, for their contribution in the work and inspiring discussions of my research, and generally for those being great office mates. In particular, I would like to thank Professor Robert

France for the many stimulating conversions we have had, and for the time he spent reading and commenting on my work towards the completion of the dissertation. 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 Libyan Student Association, for being my family and their continual encouragement and help.

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

DEDICATION

This dissertation is dedicated to my parents,

to my wife Ilham,

and to all of those who

helped me to complete it.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

The proliferation of wireless networks and mobile devices technologies spreads the development of mobile applications. Mobile applications or "mobile apps" are the phrases that are used to define the network applications that run on nomadic devices. They are composed of software programs that provide various services for mobile users. A mobile application typically has two components, one that runs on a user's mobile device and communicates over a wireless data transmission network with another component that executes in an application server.

On one hand, mobile applications allow the end-users to access Internet anytime and anywhere. That is, users on the move can access information stored on shared devices over wireless networks. The usage of such applications becomes ideal for many day-to-day services. For example, they provide us with news, entertain us, connect us with family and friends, allow us to arrange tasks, keep us informed about traffic, and support location awareness services.

On the other hand, mobile applications are not without serious security issues. Mobile applications introduce issues for both end-users and developers. Protecting access to applications data remains a security issue. A number of security policies have recently emerged to control access to shared data. These policies typically have new authorization requirements where environmental conditions, such as location and time, are used together with the credentials of the user to determine access. For example, lost or stolen mobile devices due to a holder's carelessness or theft may allow an assailant to access sensitive services or information from undesired locations and time. As such, a proper spatio-temporal policy is needed to protect access to sensitive resources in such incidents. With novel mobile applications' requirements, such policies become complex to specify, analyze, and enforce. The research work described in this dissertation focuses on providing solutions for secure access in mobile applications.

This chapter is organized as follows: Section 1.1 elaborates the need of spatio-temporal access control; Section 1.2 presents the motivations of the research; Section 1.3 introduces a number of tasks needed to fulfil the research objectives; Section 1.4 discusses the significance and contributions of this dissertation in comparison to the related studies; and Section 1.5 presents the structure of this Ph.D. dissertation.

1.1 Problem Motivation

With growth of mobile device technologies, many new applications have been integrated into our daily lives. Mobile devices use Global Positioning System (GPS) [1] to alert drivers for exceeding the speed limit in school zones [2]. In healthcare systems, mobile devices allow e-health applications to be available on request by medical staff or patients [3]. For example, an emergency management and response application (iFall) is an alert system for both detecting and notifying personnel of a patient's fall. Such applications primarily help people suffering from chronic diseases [4].

There is no doubt that the new technology improves the deployment of various application domains including e-commerce, electronic government, healthcare, and power-control systems. Enterprises are aware of the great capabilities of the ubiquitous devices, but there are also concerns about access control issues due to their mobility. The greatest concern is that a mobile computer might fall on hands of malicious users, especially, out of the work environment. Therefore, such applications create the requirements that access control depends on the location and time of access.

An example will help illustrate this point. Consider a real-world example of a spatio-temporal policy for the telemedicine application iMediK [5]. The iMedik is a mobile application accessible by handheld devices that are integrated with Global Positioning System (GPS) which identifies its physical location. With help of the mobile devices, doctors can access their patient information on the move. The security policy requires that doctors can use handheld devices to view complete Patient Medical Record (PMR) information in the clinic during day-time, whereas the same doctors can view only partial PMR information outside the clinic during night-time. Such policy is needed to protect patient sensitive information in case of lost or stolen devices.

Spatiality and temporality are also needed for controlling access to sensitive services. For example, when a mobile user is currently out-of-home and trying to terminate the home motion detector system from a mobile device after midnight, then such service request should be denied by an access control policy. A circumvention of spatiality and temporality constraints may cause system malfunction and, in some critical systems, it implies loss of human life and assets. Consider the following military scenario: a malicious user might use a hand-held device from an unclassified location to penetrate a missile launcher system and fire a missile causing excessive damage and death. Therefore, access to such critical military systems should only be allowed from high-secure locations.

Besides the safety requirements, there are other important considerations for employing spatio-temporal access control. Spatio-temporal access control can greatly guarantee the enforcement of law and legislation. A full-time student can be granted a campus license to use software packages, access digital libraries, or watch movies only inside the campus and during the semester. Furthermore, in distributed work environments, users need to prove they are performing certain jobs within certain locations and time. A spatio-temporal proof covers the case when an individual has to do a certain job on-site and at a certain time. For example, an on-site repair mechanic should be able to prove that he was repairing a machine at customer site during working hours.

Additionally, job functions in mobile environment are often times subject to spatio-temporal pre-requisite, post-requisite and triggers. For example, once a lab director receives an order for analyzing a soil sample in different labs, two lab workers' are notified to work in two different lab rooms to analyze different splits of that sample. Periodicity and spatiality together are important aspects in capturing the mobility behavior of emerging applications. For example, a nurse has a commitment of working nights every Monday and Wednesday from 8 pm to 8 am at the main building of Poudre Valley Hospital (PVH) in Fort Collins, Colorado and every Tuesday and Thursday night at another branch of the hospital in the same city during the calendar year 2013.

1.2 Problem Definition

In the pertinent literature, there is a significant body of work that has introduced approaches for spatio-temporal access control. However, the aforementioned security issues introduce novel spatio-temporal requirements that might not be possible to address with the existing approaches.

In one of our recent RBAC studies [6], we have emphasised for the need for access control extensions to support mobile RBAC systems. In this work, we found that there are quite a few newer types of applications that impose authorization requirements which are not satisfied by many of the proposed RBAC extensions. We outlined a new authorization model to fill this gap and conclude that there is still need of continued research in this area. In another work [7] on analyzing RBAC, we examined a number of analysis approaches and discussed their suitability for RBAC policy verification. We concluded that one common problem is that most automated approaches do not scale well for analyzing many RBAC systems. As such, future work in this area should investigate techniques for reducing the state-space size and also approaches for reducing the verification times. We also highlighted some properties that are unanalysed following those approaches. In this dissertation, we will also show that there is a little work on enforcing RBAC models in real-world applications.

This research focuses on addressing three primary problems of the spatio-temporal access control in mobile environment: (1) specification, (2) verification, and (3) enforcement of mobile access control policies. In this section, we elaborate on the difficulties of using current approaches to address the above noted spatio-temporal requirements, which provide the motivation for our research. We also explain some related problems and describe how existing work provides limited solutions to address some of them. We now present these issues in their order of importance with regard to our contributions.

1.2.1 Policy Specification:

Traditional access control models, such as Role-Based Access Control (RBAC), do not take into account spatio-temporal information while performing access control. Therefore, researchers have addressed this need by extending RBAC that allows it to do access control based on the contextual information associated with users and RBAC entities. In one of our previous works [8], we proposed a trust based RBAC model for pervasive computing systems. This trust model addresses the problem of unknown user in access control. It provides access based on the trustworthiness of users and the trust ranges associated with RBAC entities such as roles and permissions. The trust level of a user is computed on some role context based on three factors: properties, experience, and recommendations information for that user. However, this model cannot allow RBAC to do location and time based access control for mobile applications.

To the best of our knowledge, the most known and detailed spatio-temporal RBAC extensions are in [9, 10, 11, 12, 13]. Most of the work on spatio-temporal RBAC associate two entities, namely, location and time with users, roles, and permissions. The location and time associated with a user give the current time and his present location. The location and time associated with a role designate when and where the role can be activated. The location and time associated with a permission signify when and where a permission can be invoked. In addition, researchers have also suggested how spatio-temporal constraints can be associated with role hierarchy (RH) and separation of duties (SoD) relations.

The current spatio-temporal RBAC models lack one or more of the following requirements. Most of the work on spatio-temporal RBAC do not consider the requirements of periodicity of mobile roles as well as the pre-requisite, post-requisite, and trigger constraints. These models are also not easily configurable to support a multi-dimension policy that has different domain requirements. A typical example of such policy is the policy that allows access based on strong, spatiality, or temporality conditions in addition to spatial-temporal requirements. For each domain requirement, a new model or predicate is defined to enforce a certain type of a requirement in a policy. As such, we argue that these models define a large number of predicates and use many models in order to specify such polices, henceforth makes it difficult to use and check access requests.

Furthermore, some of these models define new notions (e.g., eliminating constraints or trusted entities), which are not consistent with the standard RBAC semantics and introduce ambiguities and conflicts [13]. In these models, the relationship between predicates and their authorization semantics are unclear. Such predicates only evaluate a single level in role hierarchies , do not consider cycling in role hierarchies, or ignore enabling conditions on some intermediate roles and permissions along multiple hierarchy paths. For example, the spatio-temporal permission usage hierarchy predicate gives raise of the following problems: the predicate is defined recursively, and as such, there is no base case replacing role names will create a cycle; the intermediate roles between a senior and a junior role in the hierarchy are ignored; and it might produce conflicts due to the inconsistent enabling of roles in invalid location or interval points. Such problems might allow unauthorized access in some undesirable spatio-temporal points.

The location and time information are represented as an external data structure for determining access apart from the model structure. The relationship between such information and RBAC components in the model structure is often unclear. In other words, these models define user, role, permission entities and entity relationships as their core components; but they lack the definition of an entity that contains spatio-temporal information in the model and also its association with the model entities. Some models also lack the definition of the object entity. Moreover, treating location and time as separate entities often times creates additional complexity in specifications and analysis, and inadequacy for checking access requests. Consider the following two examples to make the idea clear.

The first example considers the implication of checking location and time as separate entities while performing spatio-temporal access control. While performing access control using existing models, the $\mathit{RoleEnableLoc}(r)$ and $\mathit{RoleEnableDur}(r)$ functions separately determine the sets of locations and time intervals in which role r is enabled. However, elements of these locations and intervals sets are not related with each other in the spatio-temporal policy. Consequently, role r might be activated by users in invalid spatio-temporal regions. Consider the role of a software engineer in a software corporation so that role members can work from the set of locations (e.g., $\{$*programming office, testing office, home*$\}$) and during the set of time intervals (e.g.,$\{$*daytime:* $[8a.m, 5p.m]$, *night-time:* $[5p.m.and12a.m.]\}$). Now suppose the policy enforces that once a user is appointed to test programs, that user can only work from the *testing office* and during the *daytime*. However, the isolation of locations and time entities might improperly allow a user to access from unacceptable locations and durations. In a case where a user attempts access from the *testing office* during the *night-time*, the user will be granted access because the *testing office* and *night-time* elements are respectively in the locations set as well as the interval set associated with the role of software engineer. Though this environmental information in this combination (e.g., (*testing office*, *night-time*)), should not authorize the user to test programs.

The second example illustrates the number of entities that need to be managed and the problem of creating new entities when spatio-temporal information constraints associated with them are changed. Suppose a *doctor* role can be activated at locations $\{hospital, clinic\}$ from 8:00 a.m. to 5:00 p.m. This means that the *doctor* can activate his/her role either in the *hospital* or *clinic* anytime from 8:00 a.m. to 5:00 p.m. Suppose the medical board decides to change the spatio-temporal constraints such that the *doctor* can only activate his/her role in the *hospital* from 8:00 a.m. to 1:00 p.m. and can only activate his/her role in the *clinic* from 12:00 p.m. to 5:00 p.m. In order to specify such a constraint, we would have to split the doctor role into two roles, namely, *hospital doctor* and *clinic doctor* and associate the respective location and temporal constraints with each of them.

Thus, a simple change to the spatio-temporal constraint requires the creation of new roles and changes all the relationships that are associated with the original role. Such a change is non-trivial. Treating location and time as distinct entities also causes a significant increase in the number of entities to be managed as location and time that are associated with every entity and relation in RBAC. This not only reduces ease of understanding for the security administrators but also makes automated verification more challenging due to state-space explosion.

1.2.2 Policy Verification:

Spatio-temporal RBAC models have numerous features that may interact to produce inconsistencies and conflicts. A potential flaw in a policy because such problems or incompleteness in authorization constraints can cause security breaches. Therefore, a number of verification approaches have been developed which utilize software tools for performing automated analysis. Earlier works that use the de facto software modeling language UML [14] to specify access control requirements have typically resorted to the use of other formalisms for automated analysis. Such an approach typically involves a transformation process where the UML is converted to another specification language, such as Alloy [15], for the purpose of analysis. The results of the analysis depend on the correctness of the transformation procedure.

Researchers have proposed analysis approaches for verifying spatio-temporal RBAC policies using automated tools. Examples include Alloy [16, 17] and Colored Petri Nets [18, 19]. Most often it is non-trivial to specify strict temporal constraints following these approaches. Many of these approaches perform qualitative analysis of temporal behaviors. With these techniques, time is represented in an approximate sense where the temporal properties change over continuing time. Approaches that perform qualitative temporal analysis abstract away from quantitative time analysis and they can only retain the sequencing of events modeled as a sequence of states. Such time representation might not be suitable for modeling and analyzing the behavior of hard real-time systems whose correct functioning is proportional to the dense-time delays (for instance, time-out).

UML and Alloy analysis approaches cannot specify temporal liveness properties indicating something will happens in the future. For example, an active role will be deactivated later at some point of time. With Colored Petri Net (CPN) approaches, the time is implicitly represented using string data type. Furthermore, some techniques express the behavior of real-time systems as discrete time behaviors using many integer valued variables. In such techniques, continuous time is approximated to some fixed quantum. However, events do not always happen at integer-valued times. This, in turn, limits the accuracy of real-time verification.

The notion of complex real-time in spatio-temporal policies necessitates the use of formalization that supports the quantitative analysis of temporal properties and easy-to-specify temporal properties. We need to consider temporal properties that not only refer to the order in which certain events take place (before and after), but also the properties that take place or change over time at exact real-time units. Thus, the state of the practice is to develop an analysis approach that explicitly models the behavior of the mobile users, simply, to specify temporal and spatial requirements using appropriate logics, and then use some tools to automatically check whether the model satisfies the requirements or not. In general, the model checking approaches suffer from the problem of state-space explosion. Introducing optimization techniques to improve the analysis performance is also missing in the current RBAC analysis approaches.

1.2.3 Policy Enforcement:

Generally speaking, the enforcement of RBAC policies has received insufficient attention. The development of novel applications and RBAC models leads to a number of interesting questions about the policy enforcement. The policy enforcement mechanism is the mean that helps to analyze requirements, provide solutions, and demonstrate the applicability of access control models in real-world applications. In particular, it helps to illustrate how to integrate the access control component into a typical application architecture and answers whether a system requires major changes in order to implement a policy model. Sandhu et al. [20] have provided a clear distinction between policy, enforcement, implementation (PEI) models to fill the gap between policy models and real implementation. That is, access control models traditionally define policies from high-level and abstract perspectives, and enforcement mechanisms describe a useful implementation architecture. We should be able to bridge the gap from abstract polices to implementation.

However, only few works that assessed the difficulties and cost of implementing RBAC policies in practical applications. Most of the existing work on spatio-temporal RBAC have mainly focused on the development and analysis of policy models. We believe that the enforcement of these models, especially in mobile applications, introduces a number of interested implementation challenges that have not been addressed yet. For example, verifying integrity of the current user location and access time adds a significant difficulty to the access control enforcement, especially while the user on-the-move. Our main goal in this dissertation is to examine the above noted issues, to provide a framework that allows one to specify, verify, and enforce spatio-temporal access control.

1.2.4 Security Requirements

In this Ph.D. dissertation, we propose an access control framework to address the aforementioned open issues in the mobile application security. Our main research focus is balancing the security of spatio-temporal applications without introducing additional complexity. Here, we state a number of security requirements for a spatio-temporal authorization model for specifying mobile application policies.

1. The access control model should support the key requirement for spatio-temporal applications of providing the right data or services to the right person, in the appropriate location and at the right time.

2. It can support the requirements of pre-requisite, post-requisite, and triggers.

3. The access control model is supposed to be flexibly realized at the implementation level and also be understood by security administrators at the design level; it should preserve a small number of entities that need to be managed in a system.

4. Policies specified by our model should be amenable to analyze in order to ensure data security.

5. It should be possible to incorporate the policy model in many mobile applications.

1.3 Research Goals and Tasks

In the following, we discuss the main three research tasks that we perform to address the access control requirements listed in the previous section.

1.3.1 Task1: Access Control Models

The first task in this dissertation focuses on developing spatio-temporal authorization models that should address the above listed issues. We base our models on RBAC due to its popularity in many commercial sectors. RBAC is policy-neutral and it simplifies security management [21]. The proposed models tend to provide the followings: concise and clear semantics, a simple syntax of spatio-temporal requirements, easy configurable to address different requirements, realize the promise of RBAC, mange a small number of entities, use minimal number of predicates and functions, define constraints on all RBAC entities and relations, have a well-defined languages to specify properties that must be checked, feasible to validate authorizations conflicts, concisely make access control decisions, and viable to apply in practical applications. We divide this task into three subtasks.

First, we formalize the novel concept of spatio-temporal zone. The zone abstracts the location and time into one entity. With this concept, we are able to simplify policy management and policy analysis. In the mobile environment, policies could change also very dynamically, such as adding new entities and associating spatio-temporal constraints. Thus a well-designed model should be able to handle dynamic policy changes in a clean and efficient way. In our models, we show that the design of the zone structure fully focuses on such requirements and handles the problem well.

Furthermore, when considering location and time as additional supporting factor, the real-time permission validation, on a pre-request level, should be done on a secure and efficient manner. In this research, the zone concept is designed to be associated with not only roles, but also permissions and objects. Such design supports the real-time permission validation. The efficiency of such validation, especially on a pre-request level with multiple requests from a moving user, is important to investigate.

In the second subtask, we proposed a model which we refer to as the Generalized Spatio-Temporal Role-Based Access Control (GSTRBAC) that is formalized using Unified Modeling Language (UML) [14] and Object Constraint Language (OCL) [22]. The GSTRBAC model defines the syntax, semantics, and pragmatics of spatio-temporal constraints in a UML/OCL class model. The semantics of the GSTRBAC model is visualized in UML class diagram model, and the class diagram components and OCL syntax expresses the spatio-temporal constraints. We use association classes between model entities to specify model relationships rather than traditional binary associations. The association classes reduce the complexity of policy specification and validation to a great extent. It streamlines the OCL expression definition in presence of spatio-temporal conditions and explicitly reflects the fact that GSTRBAC relationships are spatio-temporal dependent. This mechanism is not followed by existing UML/OCL specification of RBAC.

A number of reasons motivated our choice. First, UML is a general-propose language that has been considered as the de facto standard in modeling software. Thus, applications are likely to be specified in UML. This will make it easier for us to integrate the access control policies with many applications. Second, UML has a set of graphical notations for specifying static as well as dynamic aspects of software systems. The graphical diagrams of the UML model make it easy to understand and use. Third, UML has supporting tools [23] that can be used for automated analysis. Fourth, UML can be used in all the phases of the software development process. Thus, it will be easy to check whether access control implementation satisfies a policy if both are specified using the same language.

In the occasion of defining the UML/OCL model, we define a spatio-temporal zone class in order to express spatio-temporal constraints. These constraints are functional predicates that have to be evaluated for each access decision to some roles and permissions. They are formally expressed in first order logic; consequently, we represent them using OCL pre- and post-conditions as well as invariants. In particular, the operations in an application using our model might be restricted via OCL invariants that must be evaluated to true in order to allow successful completion of that operations.

In completion of this subtask, we also propose an expressive model extending the GSTRBAC model to consider more important features such as spatio-temporal periodicity, pre-requisites, post-requisites and triggers. Furthermore, this model formalizes different kind of zones into the semantic of RBAC. Henceforth, the second model we refer to as the consolidated spatio-temporal RBAC model because a multi-dimensions policy can be specified without the need of a major retuning of the model. The formal semantics of this model is expressed using predicate logic. An application using our model must satisfy the predicates in order to behave correctly.

1.3.2 Task2: Policy Verification Approaches

When it comes to the first task of introducing access control solutions, we compose two subtasks to develop verification approaches for policies specified in our models. In the first subtask, we utilize the UML-based Specification Environment (USE) tool [23] for validating GSTRBAC policies. The USE tool provide an interactive environment facilitating the validation of properties in UML models specified in the form of OCL invariants, preconditions, and post-conditions against some test scenarios. It supports the manual and automated generation of snapshot instances. The verification is carried out by an embedded constraint solver.

However, validating the entire policy object model in USE tool is not feasible. That is, entities that are not related to a property in question degrade the accuracy of the validation process. In this subtask, we also propose an algorithm to generate a sub-object model based on properties that are being checked in a policy. In this way, a property is investigated under a certain set of significant entities, which are the only ones that have an impact on that property.

For the second subtask, we introduce a timed-automata based verification with supporting tools to perform automated checking of strict real-time properties while mobility is still considered. Timed-automata [24] provides a framework to model the behavior of real-time systems in annotated state transition graphs that have timed transitions labeled with piece-wise real-valued clocks. A number of motives support the choice of using timed-automata language for this purpose.

First, spatio-temporal RBAC policies can be viewed as a timed state transition system. Temporal constraints are expressed by real-timed clocks that precisely capture the elapsed time between events since the last reset of the clocks. Spatial constraints are specified by using shared integer variables and control states. Second, timed-automata has been successfully applied in many case studies [25, 26, 27, 28] for verifying complex real-time systems that relay on strict timing constraints, including timing delays, periodicity, bounded response time, and execution time. Third, a number of interactive software tools, including COSPAN [29], KRONOS [30], and UPPAAL [31] are available for modeling, specifying, and verifying the correctness of timed-automaton models. Most of these tools incorporate many additional features for improving performance. With several such existing choices, we decided to use model checker UPPAAL for the following aspects.

UPPAAL supports model checking of branching time requirements and allows checking for safety temporal properties. It extends timed-automata with additional features that help for expressing spatio-temporal behaviour. It supports the modeling of urgent responses or events via urgent and committed locations, or urgent channels. An example of the urgent actions in spatio-temporal policy is that a role should be instantly deactivated when a user leaves a room. Furthermore, these features allow the specification and verification of atomic actions (transactions) that enforce a number of transitions to be taken simultaneously (e.g., enabling a role triggers the enabling of a number of roles). UPPAAL also defines symbols to express bounded liveness properties. The bounded liveness properties refer to properties that are not only guaranteed to eventually happen, but they take place after certain delays. An example of the bounded liveness property is that once a role is activated, it should be eventually deactivated after some real-time instances, in a certain location. Such application requirements must fulfil the strict temporal conditions. The UPPAAL verifier checks properties expressed in Timed Computational Tree Logic (TCTL) [32, 33]. TCTL has many rules and symbols that allow us to specify a variety of temporal properties.

However, the state-space explosion in the model checkers becomes a problem when we are verifying properties changing over continuous time. Consequently, in this subtask, we propose some techniques to confine this problem. We employ some optimization techniques supported by UPPAAL for improving the analysis performance, and carefully compose timed-automata with a minimum number of temporal conditions. We also introduce a technique that reduces the state-space size based on a property being verified. This technique excludes the instantiation of non-dependent timed-automata processes during the verification of timed-automata, and as such, improves the analysis performance.

1.3.3 Task3: Policy Enforcement Mechanisms

The third task in this research is to develop a software architecture that defines a framework for enforcing our spatio-temporal models. The architecture is necessary to identify the implications in terms of space and computation overhead and configuring a system to implement a policy model. The proposed architecture separates the security policy from the point of use, thereby making it possible to be integrated in many applications. In the development of the architecture model, we address the following subtasks.

First, identify the desirable design characteristics that our architecture should support. For example, the architecture model should be as general as possible, efficient, and secure. We exclude the operating cost of involving third parties for providing location and time proof.

Second, we study the device's capability and storage requirements based on the responsibilities of the architecture components implementing those devices. Such analysis is important in identifying where each architecture component should be installed.

Third, we develop a number of protocols for our architecture model which should securely control the communication exchanges for accessing resources under various circumstances of mobile users. These protocols should be fully automated. We also need to provide a security analysis to ensure that these protocols are secure from common threats that are most likely to have an impact on our design. Such analysis is necessary to identify countermeasures that can be employed to prevent attackers from exploiting some common security vulnerabilities.

In the last subtask, we carry out an experimental evaluation of a prototype implementing our spatio-temporal model in a mobile application. Thus, data observed in the empirical study provides an assurance about the significance of our architecture design.

1.4 Contributions and Significance

There is a substantial body of work on the specification and verification of spatio-temporal RBAC polices. As we described in the previous section, augmenting RBAC with location and time, limits the user access, henceforth protects the use of critical resources from undesirable spatio-temporal points. In this Ph.D. dissertation, we investigate a number of related problems in the existing works: the perceived limitations in the spatio-temporal specifications, complexity and inconsistency caused by the large number of entities and predicates in the models, lack of adequate conflict detection and correction among model constraints and reporting feedback when properties do not hold, the verification of complex temporal properties while considering mobility, the analysis performance and state-space explosion problem in model checking, and a very limited works that assess the difficulties of implementing these policy models in practical mobile applications.

This research presents models that addresses a new and challenging problem raised by today's internet environment with widely-used mobile devices. The novelty of this research is clear in three aspects that are needed to be addressed in the design of such models and this work has dealt with them well. Unlike existing works in this area, we strive to provide flexibility and precise authorization semantics in our specification model, cost effective and rigorous verification approaches for uncovering model conflicts, and a practical and complete enforcement mechanism with precise measure of efficiencies and generalities. The follows describe the contributions of this Ph.D. dissertation in the order of their importance, and briefly explain their significance compared with the limitations in the existing works (see Section 1.2).

Existing works on RBAC are insufficient to capture the entirety of the listed security issues in mobile applications. Current works authorize access based on the present of environmental conditions related to a user, but do not provide mechanisms for *persistent* spatio-temporal control after resources are accessed. They also do not consider the important requirements of the spatio-temporal pre-requisites, post-requisites, and triggers. Neither of the presented mobile application requirements are handled by previous RBAC models in a fully satisfactory manner. Therefore, the current research endeavors to extend the earlier RBAC approaches along different dimensions to fulfil those requirements.

First, we propose an extended spatio-temporal RBAC model that is flexible and consolidated to support the aforementioned mobile application requirements. The models systematically span the spectrum from existing spatio-temporal RBAC models at one end to a quite expressive supporting complex security objectives at the other. In addition to precisely specify a broad range of spatio-temporal access requirements, we provide a mechanism to guarantee *persistent* access control after approving an access, which is an essential aspect of many applications. Our model supports the novel feature that we called *termination triggers*; it enforces a system to revoke a user access at the moment his current environmental information violates policy constraints. None of the existing work on spatio-temporal RBAC have attempted to provide control after access is authorized. Strictly speaking, our models also preserve the promise of RBAC which is flexible, easy-to-customize, and reduces the authorizations management overhead.

The RBAC models should have a limited complexity when adding new coordinates for location and time information. Compared to other works, this work introduce the novel spatio-temporal zone to model such additional information. Such design achieves the required complexity and functionality at the same time. The main advantages of this concept is that it allows us to abstract location and time into a single model entity. This, in turn, reduces the number of entities that must be managed and also prevents creation of new roles or permissions when spatio-temporal constraints associated with them change. We show the simplicity of our models in specifying spatio-temporal constraints that compares favourably with previous RBAC attempts. Compared with existing works, this feature makes our models easy managed, concise, and have precise authorizations.

The spatio-temporal zone is a new logical RBAC entity that encapsulates the particularities of location and time to achieve the granular formalisms. It tackles the problems of isolating time and location entities and streamlines the access control checking. The spatio-temporal zone is defined as a new component of our RBAC model that is linked to model entities and contains contextual information for restricting access. In other words, it is like an authorization token injected into every model entity to enforce spatio-temporal conditions.

We also define a number of relations on STZones and introduce the zone minimality concept that make our models flexibly and efficiently express many spatio-temporal access scenarios, which other models cannot provide such features. These features also ensure that conflicts do not arise among the spatio-temporal constraints specification. Furthermore, such relations fill the gap in the authorization inconsistencies that some of existing models are suffering from. We also defined the notion of a universal STZone (e.g., <anywhere, anytime>) to enforce strong constraints. These features in our model allow users to specify spatio-temporal constraints in many scenarios with a small number of predicates and easy understood semantic and syntax, making the model consistent to check access requests.

Existing models add spatio-temporal constraints to the RBAC entities and relationships using many functions. This approach makes it harder to capture the number, types, and the relationship between the various spatio-temporal constraints. On the other hand, we consider a spatio-temporal zone as an entity of RBAC along with the other defined entities. This allows a more uniform treatment, the zone pertinent to the application are enumerated and their relationships can be easily evaluated. The zone can also handle different states of a role and records various events that are typical for a system. For instance, due to the mobile nature of users, a role might be blocked from execution once a user moves out from a valid zone. This feature also allows the definition of many types of constraints among roles based on the occurrence of certain events in different zones. The zone concept can also consider the definition of the periodic behavior while allowing the users' mobility.

Access control models proposed so far typically support a single security domain without changing the formal semantic of the model or splitting their model into multiple subtypes to support certain requirements. Conversely, we define different kinds of zones to hide perceived complexity in the existing works. These zones allow a policy designer to flexibly specify multi-dimension policy requirements in one model. All kinds of zones have the same format and are treated in the model predicates in the same manner, but their contents vary for different requirements. For example, a temporal zone class ignores the location constraints to express temporal requirements. Thus, a simple change in the zones' contents allows us to use the same predicate to express different domain requirements.

The spatio-temporal zone introduced here is referred to as STZone which defines where and when an entity is available. The STZone entity is formally a pair of location and time, (i.e., STZone = <location,time>), that are of interest to the RBAC entities in the spatio-temporal domain. The set of STZone elements is referred to as STZones which defines where and when an entity is available. The STZones set is easily managed by the security designer, by changing the zone content, we are able to avoid the definition of new relations and entities. In the previous example, the *doctor* role is initially associated with the following set of STZones: { <*hospital*, [8:00 a.m. – 5:00 p.m.]>, <*clinic*, [8:00 a.m. – 5 p.m.]> }. When the medical board decides to change the policy, this can be achieved by simply changing the STZones set associated with the *doctor* role as follows: { <*hospital*, [8:00 a.m. – 1:00 p.m.]>, <*clinic*, [12:00 p.m. – 5 p.m.]> }. Therefore, abstracting location and time into a single STZone reduces the number of entities in the model making it easier to administrate and verify policies.

These contributions are presented in the following sections: Section 3.1: The definition of the spatio-temporal zone concept, Section 3.2: The UML/OCL based access control model that is suitable to handle environment changes in mobile applications; it describes the syntax and semantics of this model, and Section 4.1: The extended access control model which adds more important features to the previous one.

Second, have developed a spatio-temporal RBAC models with various features, it is natural to validate the model constraints that may interact to produce inconsistencies and conflicts. For conflict detection and correction, we develop an automated analysis approaches illustrating how our policy models are checked at model and application levels. Our analysis approaches fill the gaps in earlier works that relay on model transformation or cannot be used to check critical temporal properties.

Our analysis approaches are automated, use appropriate logic for specifying properties, provide feedback when some property does not hold, rigorously check various types of conflicts between model constraints, and also can be used for verifying complex temporal properties. Unlike existing analysis approaches, our approaches can be concisely used to model and verify a number of important properties such as bounded liveness, atomic actions, urgent actions, pre-requisite, post-requisite, triggers, and granular features interactions. These approaches are also effective in uncovering subtle interactions between model features.

In addition to the model verification, we define some optimization techniques and algorithms for improving the verification precision and performance. We also provide some effective techniques to condense the state-space explosion problem in model checking. Such techniques are important to scale the verification for large and complex access control policies. We show how our approaches can be adapted to some real-world mobile applications. We also provide some results that demonstrate the analysis performance due to the decrease in the number of model states that need to be explored. Most of the existing analysis approaches of RBAC do not scale well and lack the verification of the complex temporal properties, and do not provide solutions to the performance and state-space explosion problem.

A detailed discussion of these contributions can be found in these sections: Section 3.3: The UML/OCL analysis approach to uncover errors in the policy model, Section 3.3.2: The sub-object model generation technique to improve the UML/OCL analysis precision, Section 4.5: The timed-automata based analysis approach that is suitable for checking complex temporal properties while considering mobility, and Section 4.5.4: Techniques to improve the analysis performance and alleviate the state-space problem in model checking of complex temporal properties.

Third, once a spatio-temporal policy is formally specified and analyzed, it is fundamentally required to perform a syntactic analysis about the practical viability of the model to provide the required level of access control. However, most of previous works do not address the challenges raised by enforcing spatio-temporal RBAC policies in practical applications. In this dissertation, we provide an enforcement mechanism that details on practical strategies for using our GSTR-BAC model in mobile applications. We introduce a platform-independent architecture model for designing applications enforcing a spatio-temporal policy.

The model separates a security policy from the point of use, and thus makes it possible to be integrated in many applications. Additionally, we specify a number of architecture characteristics such as model generality (e.g., centralized and distributed systems) and efficiency (e.g., using small number of passes and operations). The proposed architecture consists of modules that own or request access to application resources. Access is granted or denied by an authorization module in accordance to the spatio-temporal constraints. In the form of making access decision, the current STZone of a user must be presented before access is authorized. In the form of maintaining a valid access to the resources, the condition related to the future is that the access must be revoked at the event the user migrates to an arbitrary invalid STZone while exercising the authorized resources. We implement reference monitor components in the user mobile device to enforce *termination* triggers conditions.

In the development of our architecture model, we address the following requirements. We develop a number of event-based protocols that securely control and maintain access to resources. We develop a threat model to identify possible attacks on our architecture model as well as to assess the applied countermeasures to tackle these attacks. We provide a formal analysis approach to uncover vulnerabilities in the protocol design. We resort to the use of model finder Alloy that rigorously checks whether some attackers can break our protocol. The analysis approach demonstrates the soundness of our proposed authentication protocols in the context of some well-known attack methods, and it can be followed as a general analysis approach for RBAC authorization protocols.

We also describe an experimental evaluation of a proof-of-concept prototype implementing our policy model in a mobile application. The experiment results show that the overhead imposed by each node computations in our enforcement design is minimal. Additionally, these results enable us to state that our design does not have bottlenecks and performs access control with high success rates. As such, implementing our architecture for a real spatio-temporal RBAC application is certainly practicable.

We present the details of these contributions in these sections: Section 5.1: The architecture model for enforcing our model in the mobile environment, Section 5.2: The communication protocols that demonstrate how access can be granted and revoked in the context of our model, Section 5.3: The threat model identifying common security attacks on our design and to assess the proposed countermeasures, Section 5.4: The formal analysis approach rigorously analyzing the vulnerabilities in the protocol design, and Section 5.5: The proof-of-concept prototype implementing our architecture in a real-world mobile application.

1.5 Dissertation Structure

This Ph.D. dissertation is organized as follows: Chapter 2 reviews and compares research studies related to our current research. Chapter 3 introduces the concept of spatio-temporal zones and describes the formalization and analysis of the proposed access control model using UML/OCL. We present the specification and verification of the extended model using first predicate logic in Chapter 4. Chapter 5 discusses the proposed implementation architecture model and describes an experimental evaluation of a prototype. Chapter 6 concludes the dissertation with pointers to future work directions.

Chapter 2

Related Work

This chapter provides an overview of the relevant work to our research areas. In Section 2.1, we review some traditional access control models. The extensions of Role-Based Access Control (RBAC) for integrating spatio and temporal information in controlling access are discussed in Section 2.2. We review analysis approaches for Role-based Access Control models (RBAC) in Section 2.3. Section 2.4 presents existing works on enforcing RBAC polices.

2.1 Access Control Policy Models

Access control has always been a fundamental security technique in systems in which multiple users share access to common resources. Many organizations including companies, hospitals, governments, and universities implement appropriate access control mechanisms to protect their information from improper access. Access control is the process of expressing security policies that determine whether a subject (e.g., process, computer, or human user) is allowed to perform an operation (e.g., read, write, execute, delete, and search) on an object (e.g., a tuple in a database, a table, a file, or a service). Nevertheless, administering users' privileges in a system is one of the most challenging tasks in access control.

In the last three decades, several access control models have been proposed, such as, Discretionary [34] and Mandatory [35] access control models (DAC and MAC), and Role Based Access Control model (RBAC)[36] models. Among several existing access control models, DAC and MAC are widely implemented models in the information security industry, Trusted Computer System Evaluation Criteria (TCSEC) [37]. Latter, RBAC has emerged as an alternative access control mechanism to DAC and MAC, because it reduces the complexity of security management and is a policy-neutral; RBAC can be easily configured to support DAC and MAC. These models arguably draw different effective paradigms for specifying a wide variety of security policies and authorization management. The following sections provide an overview of these models.

2.1.1 Discretionary Access Control Model

Discretionary access control (DAC) was proposed in early 1970s by Lampson [34]. Lampson defined the notions of subjects, objects, and the access control matrix. The subject-object distinction is the basis to control access. DAC is discretionary in that it allows subjects to propagate permissions to others to access their objects. Subjects initiate actions or operations on objects. These actions are permitted or denied based on the authorizations specified in a system. DAC policies control access to objects based on subjects' identities and permissions. That is, when a system receives an access request, the authorization mechanism checks the subject's identity and the granted permissions on the requested object.

Authorizations are usually expressed in terms of access rights or access modes. The access matrix is a conceptual model which specifies the rights that each subject possesses for each object. It also provides a useful framework for describing resource protection in operating systems. For instance, an access control matrix A, with subjects that are represented by the rows and objects that are represented by the columns, is used to determine the status of the protection. That is, $A[s, o]$ represents the access rights that subject s has over object o.

The access matrix can be implemented in three ways depending on the implementation details. Every matrix can be read either by rows, columns, or tables. When a matrix is read by rows, it is interpreted as a capability list. It determines what is permitted for each user. Such a matrix is widely implemented in distributed systems. The access matrix is interpreted as an access control list (ACL) when it is read by columns. It defines which permissions are granted to each object. This method is widely used in centralized systems. As tables, the access matrix is interpreted as access control triples that are represented in a table. This implementation is widely adopted in common database systems. Each row of this table specifies one access right of a subject to an object. Thus, the table contains three columns (subject, access mode, and object).

DAC, however, has some limitations. A complete users' control on object access permissions introduces some issues. DAC policies are susceptible to a Trojan Horse in which the content of files is maliciously copied from one file to another. Furthermore, the verifications of DAC policies are complicated due to unrestricted ownership of objects' permissions. The lack of constraints on the propagation of rights and copying information expose underlying policies to serious safety issues. Furthermore, with a vast number of subjects and objects, it is really complicated to employ DAC to manage access rights.

2.1.2 Mandatory Access Control Model

Mandatory Access Control (MAC) is commonly used in multi-level secure systems, where information to which users are granted access is not owned by the users. Thus, MAC prevents Trojan Horse from improper writing to files. In mandatory policies, objects are classified and the access is controlled based on the users' clearances and the objects' classifications. In this regard, the most confidential informations are given higher security levels. Usually, users are allowed to access information with security levels up to and including their clearance levels.

Dorothy Denning [35] was the first to introduce the notion of the Lattice Based Access Control (LBAC) model for formalizing information flow policies in MAC. LBAC enforces a secure information flow based on a lattice structure resulting from the security classes and the semantics of organization's information hierarchy. The goal of the control flow model is guaranteeing the confidentiality and integrity in a computer system. The information flow model is viewed as a tuple $<N, P, SC, \rightarrow, \bigoplus>$ where N is a set of objects (i.e., a user may be considered an object), P is a set of processes that are responsible for information flow, SC is a finite set of security classes (also called security levels), $\rightarrow$ is a binary flow relation defined on SC (i.e., $\rightarrow \subseteq SC \times SC$), and $\bigoplus$ is a binary operator called combining or joining operator on SC (i.e., $\bigoplus : SC \times SC \Longrightarrow SC$).

The set of security classes/levels are defined as a totally ordered set, i.e., $\{$ *Top-Secret (TS), Secret (S), Confidential (C), Unclassified (U)* $\}$, with ordering relation $TS \succeq S \succeq C \succeq U$. The order relation determines the dominance relation between two security levels. That is, we say that the security level L_i dominates the security level L_j (denoted by $L_i \geq L_j$) if L_i precedes L_j in the ordering of the security levels. In this relation, L_i is referred to as the dominating security level and L_j is the dominated one. In a lattice-based policy, the relation $L_i \geq L_j$ is defined only if the information can flow from L_j to L_i. We say that L_i and L_j are *incomparable* if $L_i \ngeq L_j$ and $L_j \ngeq L_i$. Labels $\leftthreetimes(s)$ and $\leftthreetimes(o)$ denote respectively the security levels of subject s and object o.

MAC rules were formalized by Bell-LaPadula into a mathematical model suitable for ensuring the information confidentiality [38]. The Bell-LaPadula (BLP) model defines two authorization rules that must be satisfied for guaranteeing a system confidentiality. The *BLP Simple-Security* property is mostly referred to as the "no read up" rule states that subject *s* is permitted a read access to an object *o* only if the security label of subject *s* dominates the security label of object *o*, i.e., $\lambda(s) \geq \lambda(o)$. The *BLP *-Property* property is known as the "no write down" rule and declares that subject *s* can write to object *o* only if the security class of the object *o* dominates the security class of the subject *s*, i.e., $\lambda(s) \leq \lambda(o)$. The **-Property* property, however, can cause integrity breach for information at the dominating level.

Biba model [39] has been developed as a counterpart model of the BLP confidentiality model for data integrity. It enforces integrity of data through reversing the reading and writing properties of BLP model. The Biba integrity model governs information access based on two security properties. The *Biba Simple-Integrity* property, which is referred to as the "No read down" permits subject *s* for a read access mode to object *o* only if the object *o* security level dominates the subject *s* level, i.e., $\lambda(s) \leq \lambda(o)$. The *Biba *-Property* property also known as the "No write up" allows subject *s* to write object *o* only if the subject *s* security level dominates the level of object *o*, i.e., $\lambda(s) \geq \lambda(o)$. Although the Biba model provides methods for information integrity, it suffers from Trojan Horse problem because it allows users to read up and write down.

Latter, Sandhu et al. [40] proposed a lattice-based mandatory approach that combines both BLP and Biba models in order to achieve confidentiality and integrity purposes. This security approach has the advantage of making multi-level secure systems unsusceptible to Trojan Horse attacks.

Historically, MAC models are very useful for systems that have valuable objects and work in a hostile environment. A good implementation of MAC models is very effective in ensuring confidentiality and integrity for systems that are in a high danger by a warfare's spy, such as military systems and intelligence agencies. Furthermore, mandatory models can also be useful to cohabit with other access control models to protect highly classified data in hierarchical organizations such as banking systems.

However, the rigidity of MAC models makes them not widely accepted by many commercial organizations. MAC policies considerably restrict users' activities and prevents security officers from flexible modification of underlying policies because access is always controlled based on data classification. Users' ability to control data is restricted over unnecessarily highly data classifications causing a low system productivity. Additionally, the administration of security levels in MAC policies necessitates a significant involvement of hierarchically classified agents and outsourced components. Moreover, MAC policies cannot support least privileges and dynamic Separation of Duties (SoD).

Ross Anderson [41] argued that the implementation of MAC systems requires the entire operating system and many related utilities to be considered as trusted agents executing outside the MAC framework. These trusted components bypass MAC rules in order to maintain security policies at a high cost. Thus, these components should be verified to prevent improper access to policies, and developing access control mechanisms to protect against inappropriate access of these components.

2.1.3 Role-Based Access Control Model

Role Based Access Control (RBAC) has emerged as a powerful and generalised approach to access control and it is a promising alternative of MAC and DAC [42, 21]. RBAC has been extensively implemented in several applications at different levels due to its inherited beneficial features such as flexibility, intuitiveness, and ease of administration. By configuring roles, RBAC can express a wide range of security polices including discretionary and mandatory ones [36]. RBAC has been used in a number of commercial software products such as Windows Server 2003 and Oracle database for managing authorizations.

The reference model for role-based access control is RBAC96 [36]. Figure 2.1 shows the entities and relationships of RBAC96. RBAC96 consists of four sub-models, increasingly adding features to the basic model. These models define the four primary building components: users, permissions, roles, and session, and the relationships among these sets. Roles represent job functions within an organization. Roles are the most steady components because organization's activities or functions usually change less frequently. In contrast, users and permissions associated with roles are transitory. In the following we discuss the sub-models of RBAC96.

Core RBAC (*RBAC0*): The primary concept of RBAC is that permissions are assigned to roles

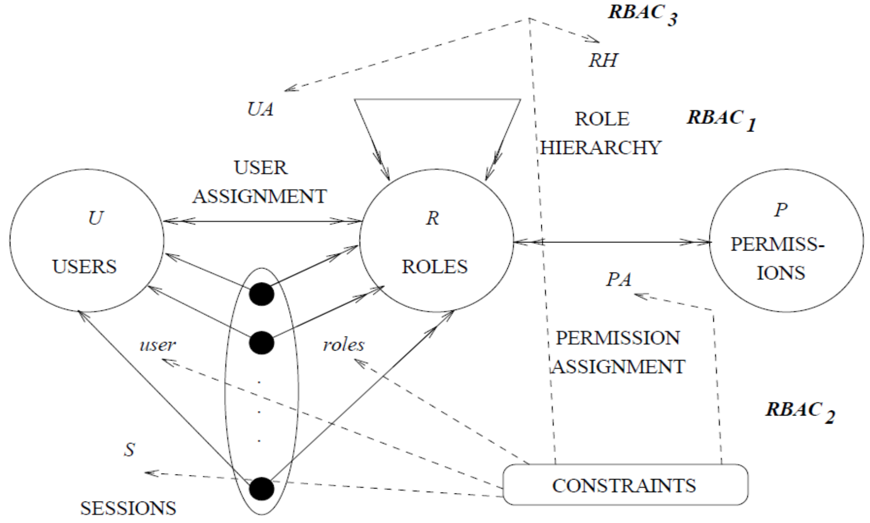


Figure 2.1: RBAC96 Model

and users access permissions through an appropriate role membership. *RBAC0* defines many-to-many relations, namely, user-role assignment, permission-role assignment, and user-session assignment. Users can be assigned several roles, a role can have as many permissions, and a permission can be assigned to multiple roles. A user can instantiate multiple sessions at the same time and each session should belong to a single individual.

Hierarchical RBAC (*RBAC1*): *RBAC1* defines the notion of role hierarchy, which model the line of authority and responsibility in an organization. With role hierarchy, the roles and permissions assignment are implicitly given via senior roles, and as such, greatly simplifies the management of authorizations. Mathematically, role hierarchy is a partial order relation, which is reflexive, transitive, and anti-symmetric. In role hierarchy, powerful (or senior) roles at the top of hierarchy, subsume all permissions associated with less powerful (or junior) roles at the bottom. Joshi et al. [43] have defined two classes of hierarchy, namely, permission-inheritance hierarchy where a senior role inherits permissions from junior roles, and role-activation hierarchy where members of senior roles are permitted to activate junior roles.

Constrained RBAC (*RBAC2*): *RBAC2* supports a collection of constraints, including Separation of Duties (SoD), cardinality, and pre-requisite constraints. The SoD principle reduces the risk of a fraud or collusion by spreading the responsibility and authority over multiple users. The static SoD (SSoD) and dynamic SoD (DSoD) constraints are two kinds of SoD [44]. SSoD prevents conflicting roles to be assigned to the same individual while DSoD prohibits the same individual from activating conflicting roles simultaneously. SoD is also defined between permissions to prevent the assignment of conflicting permissions to a single role. Cardinality constraints enforce the security principle "least privileges"; a certain subset of permissions needed to perform a job are only assigned to a role. Pre-requisite constraints control users' competency and appropriateness. That is, a user should be assigned to role *A* in order to be assigned to role *B*.

Consolidated RBAC (*RBAC3*): It combines *RBAC0*, *RBAC1*, and *RBAC2* in one model, which creates several issues. A policy conflict may occur due to subtle interaction between role hierarchies and SoD constraints. For example, a senior role should not inherit two conflicted junior roles in order to prevent its members to access the permissions associated with those junior roles. Cardinality constraints might be breached through role hierarchies too. All of these issues need to be carefully considered when designing a security policy.

Despite the improvement of the authorization management by RBAC, with all support of hierarchy, constraints, assignments, and activation features, the administration complexity is potentially an issue for large systems. Moreover, with data abstraction in RBAC, it is not easy to control the sequences of operations through permissions, a more sophisticated control needs to be incorporated in RBAC [36]. Traditional RBAC96 do not take into account location and time information while performing access control.

2.2 Extended Role-Based Access Control Models

With the advent of wireless and mobile devices, new applications make use of the spatio-temporal information in order to provide better functionality. However, traditional RBAC models do not take into account contextual information into access control. In our earlier work [8], we proposed a trust-based RBAC model for pervasive computing systems. Users are evaluated for their trustworthiness levels before they are assigned to different roles. Roles are associated with a trust range indicating the minimum trust level that a user needs to attain before it can be assigned to that role. A permission is also associated with a trust range indicating the minimum trust level a user in a specific role needs to attain to activate the permission. To determine the authorization between a user and a role, a user's trust value is evaluated based on each role context separately. The trust relationship between a user and the system in the role context depends on three factors: properties, experience, and recommendations. Each of these components has a value between *[0,1]* and sum of these components is *1*. The semantics of our model is expressed in graph-theoretic notations that allows us to formulate precise semantics for the model. However, this model cannot be used for specifying spatio-temporal RBAC policies needed by many mobile applications.

The follows elaborate on research work for extending RBAC to provide spatial, temporal, and spatio-temporal access control.

2.2.1 Temporal Role-Based Access Control Model

Earlier studies have proposed temporal authorization models that are not based on users' roles. Bertino et al. [45] proposed a temporal authorization model that extends authorizations with temporal constraints. In this model, an authorization is associated with a temporal expression identifying the periods of time in which the authorization applies. Further, it also permits the specification of derivation rules for expressing temporal dependencies among authorizations. Gal and Atluri [46] have proposed a Temporal Data Authorization Model (TDAM) that can be seen as a complementary model to the one in [45]. TDAM expresses time-based policies based on the temporal attributes of data such as transaction time. However, since these models are for non-RBAC policies, they cannot express temporal constraints on roles such as temporal constraints on role enabling and disabling.

In the temporal domain, Bertino et al. [47] have proposed a time-dependent, role based access control model, named Temporal RBAC (TRBAC). The TRBAC model expresses temporal constraints on role enabling and disabling, temporal dependency, trigger priority, and run-time requests. Thus, users can only assume roles when they are periodically enabled. However, the model does not consider temporal constraints on user-role and role-permission assignments. Furthermore, this model lacks the definition of separation of duties constraints, cardinality constraints, and role hierarchy relationships in the temporal domain.

Latter, Joshi et al.[43] have proposed a more general temporal RBAC (GTRBAC) that subsumes TRBAC features and handle a wider range of temporal constraints that are overlooked by prior temporal models. The GTRBAC model associates temporal information with role hierarchy and SoD constraints, and different forms of temporal dependency constraints.

2.2.2 Spatial Role Based Access Control Model

Covington et al. [48] have introduced Generalized RBAC (GRBAC) to adopt environmental aspects into RBAC . It makes a distinction between two classes of roles termed as Environment Role and Object Role. The former role class expresses a favourable system state in some positions, and the latter class specifies necessary objects properties. Here, the GRBAC model incorporates properties such as role activation, role hierarchy, and separation of duties into environment roles. However, the set of environment roles would be extremely large for most applications.

Researchers have also incorporated geographical information into RBAC. Hansen and Oleshchuk extended RBAC with spatial information in their model termed as spatio-RBAC (SRBAC) [49]. This model defines location based access control policy for wireless networks. With SRBAC model, users are permitted to assume roles in order to access application resources not only based on their credentials, but also on their current physical positions. The user position, however, has not semantically meaningful “logical position“, but simply a geometric value; the position of a user is defined by a cell-phone access point. The model also does not support physical locations containment and overlapping which make the model incapable for specifying important spatial constraints.

Bertino et al. [50] have proposed a GEO-RBAC model which allows a role activation based on a user’s location with regard to logical positions associated with roles. In this model, spacial information is captured by Geographical Information System (GIS) standards. Here, user-role and permission-role assignments are not pertained to spatial information. The model lacks the definition of spatial constraints on separation of duties or role hierarchies too. Furthermore, a single location extent is associated with each role to specify the boundary at which the role is in active space.

Ray et al. [51] have proposed a Location Aware RBAC model (LRBAC) which can handle location overlapping while processing access requests. The authors have described how location information is associated with RBAC components to simplifies the specification of location based on security policies. The LRBAC model considers not only location constraints on role activations, but also associates locations with role and permission assignments. The LRBAC model defines role hierarchy and a role-role relationships regardless of spatial information.

However, none of the aforementioned security models have addressed the combined impact of both spatial and temporal information on RBAC components in making access decisions.

2.2.3 Spatio-Temporal Role Based Access Control Model

In spatio-temporal applications, access to resources is contingent based on location and time information that are appended with the access requests. Recently, researchers have investigated spatio-temporal access control policies' needs and designed some authorization models based on the RBAC paradigm.

Chandran and Joshi [52] have developed a LoT-RBAC model which incorporates both location and time into RBAC. The LoT-RBAC model combines both characteristics of GTRBAC [43] and GEO-RBAC [50]. Here, a role can only be enabled on the satisfaction of temporal constraints and users are entitled to activate a temporally enabled role once their current location satisfies the location constraints associated with that role. It borrows the concepts of the distinction between physical and logical locations from the GEO-RBAC model and the temporal context from the GTRBAC model. However, the model does not address the spatio-temporally aware of role hierarchy and separation of duties constraints.

Samuel et al. [16] have proposed a spatio-temporal RBAC model which adds spatial sensitivity to GTRBAC [43] to support spatial-temporal constraints. Once the temporal constraints associated with GTRBAC components are satisfied, the spatial constraints are enforced. This includes the association of spatial information with temporal role enabling, user-role assignment, role-permission assignment, and role-activation. However, the model ignores the impact of spatio-temporal information on RBAC permissions. Whenever and wherever a role is enabled, all the permissions associated with that roles can be executed regardless of any spatio-temporal constraints. Furthermore, the model only supports spatiality-based role hierarchy and separation of duties constraints.

Aich et al. [10] have proposed spatio-temporal RBAC model which is formalized based on propositional logic and it is termed as STARBAC. The logical operations specified in STARBAC have been shown to be useful in expressing a number of real-world policies including spatio-temporal aware of role enabling-disabling, role activation-deactivation, and roles assignment, and permissions assignment. However, the model does not support spatio-temporal constraints on separation of duties neither on role hierarchies.

Later, Aich et al. [12] extended the capabilities of their STARBAC model [10] in a model termed as Enhanced Spatio-temporal STARABAC (ESTARBAC). The ESTARBAC model is based on the idea of spatial and temporal extents which guides the definition of roles and permissions in the model. The concept of spatio-temporal extent is the basis of specifying spatio-temporal constraints including separation of duties which is not considered in their prior model [10]. However, ESTARBAC does not consider spatio-temporal constraints on user-role assignments, permission-role assignments, role hierarchies, and dynamic separation of duties. Our proposed models eliminates these shortcomings. Although the goal of our work has similar concern of the combining effect of location and time on RBAC components, our models can express real-world constraints at a more granular level that are not possible in a such model. The proposed models substantially differs from those models in the sense that it considers distinctive features such as spatio-temporal constrains on pre-requisite and post-requisite relationships as well as on action triggers.

Chen and Crampton [13] have introduced graph-based formulation to describe the semantic of three spatio-temporal RBAC models, each of which addresses certain spatio-temporal requirements. These models [13], like others, treat the location and time as separate entities. Unlike our model, neither of these models impose spatio-temporal constraints on moving objects and separation of duties relationships nor consider the interaction between role hierarchies and separation of duties for determining access. Furthermore, the work in [13] did not provide methods to identify inconsistencies and conflicts in a policy graph model. We argue that using three different models to specify a policy is non-trivial and the interoperability among various features is hard to capture and validate. The notion of trusted entities and eliminating constraints on RBAC entities may introduce ambiguities and they are contrary to standard RBAC semantics. Our work rectifies these shortcomings by studying the impacts of spatio-temporal information on RBAC components and providing validation and enforcement approaches.

In contrast of using three separate models, and trusted entities, eliminating constrains in [13], the containment and universal STZone (e.g., <anywhere, anytime>) in our model flexibly allow a user to express a similar set of spatio-temporal requirements in one intuitive model. A subtle interaction between role hierarchy and separation of duties may authorize a user to activate some conflicting roles or access conflicting permissions, our model concerns with this problem. Furthermore, trusted entities (trusted users or roles) add more entities to the model and the management of these entities is non-trivial since they are extremely privileged entities that bypass model constraints. That is, a trusted role might be pointed to inherit conflicting junior roles that may be exploited to commit a fraud or penetrate a system security. The proposed model mainly addresses this point by reducing such complexity and eliminating inconsistencies in a policy specification and evaluation of access requests as well as condensing the number of entities that need to be managed in a model.

Another approach for specification of spatio-temporal RBAC policy has been recently proposed by Ray and Toahchoodee [9]. The model is termed as STRBAC and it improves the spatio-temporal models proposed by Chandran and Joshi [52] and Samuel et al. [16]. The STRBAC model considers spatio-temporal constraints that are not possible in those models. The authors discussed the interaction impact of location and time information on RBAC components in making access decisions. Here, different forms of spatio-temporal (weak, standard, and strong) constraints are expressed on user-role assignment, permission-role assignment, role activation, role hierarchy, and separation of duties. Later, an extended spatio-temporal RBAC (STRBAC) is proposed by Toachoodee and Ray [11]. The extended STRBAC improves the expressiveness in their prior STRBAC model [9]. The earlier STRBAC is enhanced to incorporate the transfer operations concept and the concept of delegation chain.

Though the STRBAC model supports many features, separating *role enabling* and *role allocation* concepts are vague. That is, a user can activate a role in a position and time instance where a role is allocated and enabled. The user, however, cannot activate a role if the role is allocated but not enabled and vice versa. The relation between user context and these concepts is not obvious. Our proposed models allow roles to be activated by users in some spatio-temporal zones where roles are available based on the role availability concept. Moreover, the STRBAC model lacks the definition of the positions overlapping and containment, a user request at a given point of time is always attached to a single logical location. Role hierarchies are defined at a single level of inheritance lacking the transitivity feature of privileges.

Generally speaking, most of the existing spatio-temporal models do not consider locations and intervals overlapping and containment. A user at a given point of time is considered to be in a single point of location and time elements. Furthermore, treating time and spatial information in isolation during the access process causes inconsistencies. That is, a user might assume a role on the satisfaction of locations and temporal constraints in apart, but not on the acceptable spatio-temporal point. Such formalizms also increase the number of entities in a policy resulting in implications of policy specifications and analysis.

The proposed models fill these gaps by extending earlier models with a number of spatio-temporal features and introduces the spatio-temporal zone notion that are associated with RBAC components. Our models support the definition of entities pre-requisite, post-requisite, and triggers that are not formalized in its counterparts. In our models, during the access, in addition to locations overlapping and containment, temporal intervals overlapping and containment are also considered in combination through the definition of STZzones operators. Furthermore, the STZone and entity availability concepts simplify the specification of the impact of the time and location factors on RBAC components, and also improve the performance as well as streamline the policy verification. Different classes (types) of zones allow the specification of temporal, spatial, and strong policy requirements with no need to change the formal semantic of the spatio-temporal model. Users of our model can selectively model as many requirements from their perspective and in different domains using a single model.

Note that, here has been some work on so-called “provisions” (i.e., conditions that must hold or actions that must be executed before an access is authorized) [53, 54, 55, 56]. Such conditions seem related to the kind of spatio-temporal pre-requisite constraints that must be satisfied to grant a role to a user. These access control models have formally specified provision and obligation requirements, but in the none-role based access control. We also argued that the spatio-temporal pre-requisite constraints are not considered in the previous studies on spatio-temporal RBAC models. In particular, the models in [53, 54, 55, 56] cannot express spatio-temporal pre-requisite constraints on assignments of roles and permissions, which our model does.

Furthermore, in these models, provision and obligation authorization have conflict in their semantics. For example, in most of these models, except for the $UCON_{ABC}$ model [56], provisions impose conditions that must be fulfilled (e.g., in the present/past) prior to authorizing an access, and obligations refer to commitments that a subject is bound to carry out after access (in the future) with/without certain time frames. However, in $UCON_{ABC}$ model [56], obligations refers to the provision authorizations described in [53, 54, 55]. Our proposed pre-requisite constraints seem to have a closer meaning to provision authorization, but our semantic is different.

The provision authorizations in [53, 54, 55, 56] are proposed as an extension that make the access control more flexible than merely accept/deny access requests in order to support the proliferating e-commerce and business to business (B2B) requirements. For example, a user might be authorized for a certain purchase if that user is registered in a system, and the user can make another purchase if the user has completed the first purchase reviews within 30 days. In contrast to our model, the pre-requisite conditions bind certain relationships between roles and permissions to ensure that a user has required responsibilities and qualifications. For example, a user should have been trusted to play a certain role in order to be authorized to become a member of a more critical role than the former one. Furthermore, our pre-requisite semantic focuses on the results of the provisional authorizations, but not on how and when certain actions are performed.

2.3 Analysis of Role-Based Access Control Models

Along the development of novel RBAC models, researchers have also provided some automated verification approaches for RBAC policies. They based the analysis approaches on the formal specification languages with tool supports. The tool support is important for eliminating human errors and enhancing the performance. These approaches vary in complexity and in the way of modeling and verifying RBAC with respect to the notations inherited in those languages. The following sections discuss a number of formal verification mechanisms with toolboxes support for analyzing RBAC policies.

2.3.1 $\mathbb{Z}$-*EVES* Approaches

The formal language $\mathbb{Z}$ has been used to analyze RBAC in some studies [57, 51]. It was shown that $\mathbb{Z}$ can express RBAC components and constraints. The authors have used $\mathbb{Z}$ to specify and verify the consistency of RBAC systems using a theorem prover. This approach is based on the development of a state-based verifiable model representing RBAC components, constraints, and state-transition operations. The software toolkit $\mathbb{Z}$/*EVES* is used for modeling and theorem proving.

The state-based verifiable model of RBAC is given by describing the RBAC state and the set of operations over that state. The state schema of RBAC defines its sets and relations along with invariants. A number of operation schemas are defined to specify events. There are three states: *initial state* is the starting state of the system, *secure/consistent state* is a state in which all security requirements are satisfied. The consistent *state-transition* is produced by a proper system transition. The authors showed how to verify the consistency of the RBAC state under different operations. The theorem stated that the system remains in secure states if and only if state transitions start from secure states and operations comply with state-transition constraints.

However, the $\mathbb{Z}$-based analysis approach lacks a complete support of automated analysis useful to detect inconsistent states. With theorem prover $\mathbb{Z}$-*EVES*, it is none-trivial to prove theorems. It requires human-intervention to give some proof facts to the machine. Additionally, the $\mathbb{Z}$-*EVES* prover is only capable of proving simple theorems. The formal semantic of the prover is very difficult for users to understand. It is hard to know what action should be taken when the prover fails. In these studies, a limited number of security constraints (e.g., cardinality and separation of duties) are verified. Verifying the interactions among various features in a state-transition RBAC model is not addressed.

2.3.2 UML/OCL Approaches

Ray et al. [58] proposed a security analysis method of RBAC using the Unified Modeling Language (UML) [14] and Object Constraint Language (OCL) [22]. Following this approach, we could visualize some conflicts in RBAC constraints.

Sohr et al. [59] have employed the USE tool [60] to validate a classic RBAC policy that are specified in UML/OCL. The authors demonstrate how the USE tool can be utilized to validate the correctness of some non-temporal properties in a conceptual UML/OCL model of the RBAC policy. They validate the user-role assignment property by generating snapshots and validate them against a UML policy model. The USE has been shown useful for uncovering errors and identifying missing constraints in RBAC policies. We extend this approach to verify spatio-temporal policies.

Our UML/OCL verification approach is distinctive since it specifies and verifies a set of spatio-temporal properties. In particular, here are some differences with prior UML/OCL approaches: first, prior approaches have verified a classic RBAC policy against some authorization constraints regardless of the impact of spatio-temporal factors; second, we verify the metamodel of UML against a set of model instances that are automatically and manually generated; third, we consider the specification of the core STZone entity and examine its impact on model entities and relationships; fourth, conflicts in the model due to subtle features interaction such as between hierarchy and separation of duties are restricted and checked in the model; fifth, we opt to use class associations in the model rather than traditional binary associations to simplify the definition of OCL expressions; we also develop an approach to generate sub-object models to eliminate irrelevant entities from the validation of certain properties. Later in this research proposal, we will demonstrate the effectiveness of our approach through the validation of a number of spatio-temporal properties in a real-world application.

Despite all the advantages of the light-weight formalisms and validation using the UML/OCL approach, some limitations should be considered. With UML/OCL approach, it is non-trivial to capture dynamic properties of RBAC. The USE/UML approach cannot ensure that all conflicts are detected; however, it can uncover conflicts only in certain scenarios of the users' choices. Therefore, we cannot put a complete reliance on the USE/UML approach to prove that the design is correct. Moreover, UML/OCL approach is not suitable to check strict temporal and liveness properties.

We argue that UML/OCL is very useful approach to streamline the modeling of security policies and perform a sanity check. Applying as many heuristics may enhance the process of conflict detection by uncovering most common critical specification errors. Afterword, we can apply model checking techniques to guarantee the detection of remaining conflicts. Strictly speaking, the USE/UML approach is a reliable and an effective approach for designing security policies as a primary task before testing and deployment of the RBAC policies using model checking.

2.3.3 Alloy Approaches

Samuel et al. [16] have used Alloy [15] for verifying security properties in their RBAC model. They showed how to analyze spatio-temporal constraints on user-role assignment, permission-role assignment, user-role activation, role hierarchy, and separation of duties. Their verification approach does not consider the analysis of the subtle features interaction which might cause a harm of an application security.

Toahchoodee and Ray have also considered the use of Alloy to verify a number of security properties in their spatio-temporal RBAC model in three consecutive studies [61, 17, 62]. In the first study [61], they specified various spatio-temporal features in Alloy. The Alloy analyzer is used to detect a potential conflicting constraints in RBAC policies. In the following study [17], Alloy is also used to specify and verify properties in their spatio-temporal RBAC model that are not addressed in the prior study [61]. In the third study [62], the UML language is used to specify access control requirements of a real-world application, and then UML2Alloy transformation tool [63] is utilized to transform the UML model to Alloy specifications in order to verify some RBAC operations.

The main motivation of the transformation is to take advantages of the verification tools capabilities provided by the target formal language for checking certain properties. However, the correctness of the model strongly depends on the transformation algorithms. Transformation between modeling languages is challenging since it requires deep understanding of the semantics of the involved languages and their objectives. Furthermore, the current transformation approaches use informal validation of the transformation algorithms correctness; they first create a policy in the source language and informally show how it is transformed and compatible to the target language. However, a formal proof that the transformation is correct and complete is required, otherwise, the models in the source language are not guaranteed to be semantically equivalent to the models in the target language. As a result, the verification process of the target model might not reveal errors that are present in the source model, or might discover faults that are not actually present in the source model. Our specification approach of UML/OCL models mitigates the transformation issues by employing the prevalent USE tool.

The Alloy approach, however, is not without issues besides the transformation issues. It is not easy to analyze and understand system behavior using Alloy. In particular, it is difficult to identify undesirable states that a system might enter under different situations. Furthermore, in terms of verifying real-time systems, Alloy is not commonly used to verify real-time systems. Alloy does not support temporal logic for specifying temporal safety, liveness, and reachability properties. Real-valued variables cannot be modeled in Alloy because it does not support real-numbers. In Alloy approaches, time intervals are represented by a set of atoms which are static in nature (indivisible, immutable, and un-interpreted). Additionally, with Alloy, it is hard to decide whether a policy holds or not when the Alloy analyzer cannot find a counterexample of a property in question within a certain scope. That is, when a counter example cannot be found in a model, this does not mean the policy model is consistent.

2.3.4 Petri Nets Approaches

Researchers have also proposed alternative verification methods based on model checking techniques such Colored Petri Nets (CPN) [64] or Timed-Automata (TA) [24]. These approaches can perform exhaustive search in a model space to verify security properties including concurrent and distributed actions that are infeasible to verify with Alloy and UML/OCL approaches.

The CPN approach has been proposed by Laborde et al. [65] for modeling and verifying network security policies expressed in RBAC. The authors analyzed security properties in a network security policy like confidentiality, integrity, and availability of services. Rakkay et al. [19] have also showed how CPN can model classic RBAC policy and checks for some access control requirements. With a help of the CPN-tool, some security conflicts are captured in undesirable CPN control states (Places).

Shafiq et al. [18] have proposed CPN based verification framework to check the correctness of event-driven RBAC policies for real-time applications. The authors illustrated how to examine the violation of cardinality and separation of duties constraints that might take place in three actions namely: user-role assignment/de-assignment, role enabling/disabling, and role activation/deactivations. However, the authors have only checked non-temporal RBAC properties. Additionally, the authors have ignored the analysis of conflicting constraints.

The CPN analysis technique is also utilized by Toahchoodee and Ray [11] to verify security conflicts in a policy expressed by their spatio-temporal model. The verification approach is twofold: a security policy is first transformed to an Access Control Graph (ACG) policy that is proposed by Chen and Crampton [13], then the ACG policy is mapped to a number of isolated CPN nets in order to perform automated analysis. The authors performed a security analysis for a real-world application scenario and they uncovered some errors such as isolated entities infeasible paths between model entities.

Though, the structural decomposition of the security model into multiple CPN nets reduces the state-space explosion problem, it is hard to say if the entire system is free from ambiguities because the interaction between features across different modules remains unchecked. It is not clear whether these CPN modules check independent properties. Furthermore, the spatio-temporal information is approximately represented by tokens of a product color of two string variables representing time and location factors. Furthermore, a number of auxiliary places are added in the CPN model to capture tokens representing policy violations of certain properties; this may cause a drastic increase of the model state-space.

Although CPN supports the exhaustive search, it has some issues for analyzing spatio-temporal properties. CPN approaches approximate the time rather than using explicit real variables. We believe that it is ultimately counterproductive when use approximate variables for reasoning about systems whose correct functioning depends on critical real-time constraints. With CPN approaches, queries for checking some policy requirements are expressed by *ML* language which is a domain specific language of the CPN tool. This language is hard to use to formulate properties. Unlike temporal logics such as Linear Temporal Logic (LTL), *ML* specification language does not have rules and symbolisms such as ’eventually’ and ’always’ keywords that are frequently used to express temporal properties. Moreover, current CPN approaches do not utilize the timed Petri Net (TPN) extension for verifying temporal properties in RBAC. In TPN nets, real-valued variables are associated with each token, arc expression, and transition, and as such, results in extreme state-space size. Consequently, the state space-explosion problem becomes an perpetual problem for a large number of timed tokens, arcs, and transitions [66, 67].

Godray et al. [68] made a comparison between the use of the timed-automata and timed Petri Net models in the context of temporal validation of real-time systems. The authors investigated two factors, namely, the modeling formalism expression power and modeling formalism analysis power. They concluded that both validation approaches have quite similar expressive power; however, timed-automata improves usability and intuitiveness upon the TPN approach. That is, the graph representation of a quite large-scaled system in CPN may become too complex to be useful and understandable. In terms of performance power, the timed-automata approach improves upon TPN approach even though the optimization options supported by timed-automata is not used. Since timed-automata algorithms are proven to have high performance, some researchers proposed approaches for the structural transformation for TPN models to the timed-automata models for analyzing temporal behavior [69, 70, 71, 72].

Another an approximate time analysis approach has been recently proposed Uzun et al. [73]. They have proposed an analysis approach for Temporal-RBAC (TRBAC) using a discrete time model. The authors proposed to split the analysis problem of TRBAC into simpler RBAC subproblems to apply analysis techniques for traditional RBAC. In particular, their approach composes multiple RBAC subproblems in order to simply search for particular reachable states. Our UPPAAL approach primarily differs from this work since we assume continuous time model. We are analyzing continues time properties while considering users' mobility in GSTRBAC. The cost of continues time analysis has been shown to be expensive, thereby we proposed a number of techniques to confine this analysis cost. While performing the security analysis of some GSTRBAC properties, our analysis approach decomposes the analysis problem into subproblems based on those properties of our interest. We use a real-world mobile based DDSS system to evaluate our approach. Our experimental results show that our analysis approach is both feasible and flexible.

A number of algorithms and computational analysis of those algorithms are provided too. Furthermore, we have checked some critical temporal properties (e.g., bounded liveness and triggers) that have not been considered in the Uzun work.

2.3.5 Timed-Automata Approaches

Researchers have also proposed a Timed-Automata (TA) approach as an alternative exhaustive search approach to CPN for checking temporal properties in RBAC. Mondal et al. [74, 75, 67] have used timed-automata for specifying properties in TRBAC [47] and GTRBAC [43] models. The authors have verified temporal properties using toolbox UPPAAL [31]. In the first study [74], the authors have illustrated how to verify temporal constraints on the role activation in TRBAC timed-automata. In the subsequent study [75], the authors verified additional temporal properties. Both studies have assumed that a user can activate one role at a time, and that user should deactivate that role in order to activate another role. However, in RBAC, a user is authorized to activate multiple roles simultaneously.

In the latter study, Mondal and Sural [67] have extended their earlier timed-automata approaches to check a wider number of temporal properties in the GTRBAC model. In this approach, users are classified into four distinct categories: a single role activation, a multiple role activation at different time instants, a simultaneous role activation, and a mutual exclusion role activation. In real-world applications, however, most often users are authorized for performing all of these role-activation categories in the same policy. Furthermore, these approaches do not consider the verification of role hierarchies with multiple levels of inheritance. This is important to mitigate conflicting junior roles activation at critical points of time. The authors also ignored the use of the state-space reduction techniques supported by UPPAAL. Thus, their analysis performance degrades drastically with increasing number of model features and temporal constraints.

None of these timed-automata approaches addressed the verification of spatio-temporal constraints of RBAC policies. In our approach, we use the timed-automata to model the behavior of systems whose correct functioning depends crucially upon spatiality and temporality considerations. The combined impact of time and location on RBAC entities necessitated the design of multi-interactive timed-automata that respond to location and time changes. Here, each RBAC entity is described by an interactive timed-automaton which reacts to spatio-temporal environment changes by possibly leaving its current state.

The proposed timed-automata approach allows users to verify a number of spatio-temporal constraints that are non-trivially and inadequately addressed in the existing approaches. In our approach, it is possible to check some complex spatio-temporal properties such as bounded liveness, atomic actions, urgent actions, pre-requisite post-requisite, triggers, and features interactions. Moreover, users can be in multiple role-activation categories at the same time. We also employ a number of optimization techniques for reducing the state-space explosion problem in our timed-automata model. Some of these techniques are supported by UPPAAL algorithms while others are proposed by us such as careful design structure and entities-dependability techniques. To the best of our knowledge, most of the existing RBAC analysis approaches do not apply state-space reduction techniques. As a proof-of-concept, we evaluate our analysis approach performance for a real-word mobile application.

2.4 Enforcing Role-Based Access Control Policies

The enforcement of spatio-temporal RBAC policies has not received significant attention from the RBAC researchers. In the past two decades, most of the research on RBAC primarily focus on providing policy models and analysis approaches, but rarely consider the implementation details of RBAC models. The enforcement mechanism of RBAC policies in real-world applications provides additional information about the practical use of the RBAC models. The enforcement mechanism answers a number of questions such as what are the cost effectiveness and the system's requirements in order to apply such models in various applications. To the best of our knowledge, such questions require us to develop a framework for implementing the RBAC policy models, and more specific in mobile environment which is one of the main focus of our dissertation.

Very little research appears in addressing the challenges raised by enforcing spatio-temporal RBAC policies in practical applications. The enforcement mechanism needs to integrate the access control component into a typical application architecture. Access control models traditionally define policies from an abstract perspective and enforcement mechanisms describe how the policies are actually implemented in the context of the application. The enforcement of spatio-temporal access control models in mobile applications introduces a number of interesting implementation challenges that have not been addressed successfully yet. Verifying the time and location of a mobile user while he is accessing some authorized resources is non-trivial. The followings elaborate the earlier RBAC enforcement approaches and compare them with our enforcement mechanism.

Kirkpatrick and Bertino [76] enforced access control according to the user's context. This work, as does our work, it checks a user claim to a certain context. The authors developed protocols to authenticate users' claims based on the GEO-RBAC model [50] which supports spatially-aware access control policy. In particular, these proctorial validate users' locations in accordance to the spatial conditions associated with roles. Their spatially-aware system is developed based on the assumption that a number of location devices are preloaded in known physical areas to provide a location proof for users. The location devices are installed in immovable structure, such as room's wall, and broadcast a very restricted range of 10 cm in radius. The authors also presumed that malicious administrators are unable to remove the location devices and reinstall them in false locations; these devices might be fixed inside a wall. In their approach, a user retrieves an evidence of being in a room or a building from a location devise using a cell phone equipped with NFC reader technology. Then, the user presents the location evidence with relevant credentials in the access request to the system manger component. At a system side, the user claim is authenticated and then compared with locations associated with the requested roles in the RBAC policy.

Our work differs from that of Kirkpatrick and Bertino in the following ways. First, Kirkpatrick and Bertino approach does not support the needs for spatio-temporal RBAC polices in which access decisions depend on the location and time associated with all RBAC entities, which our work supports.

Second, the location devices in their approach are immobile, and as such, users are restricted to be present in a room with a proximate distance from those devices in order to get location proof. In contrast, our approach supports mobile users in the space and users, without such restraint, can make access from any nomadic devices associated a trusted location system.

Third, we support the temporary suspension of the user's access for a certain period of time the user is moving out of a valid zone, this feature is not supported in the work of Kirkpatrick and Bertino. Our event-based architecture allows a user to automatically resume all of the previous rights at a time the user moves back to the valid zone. Thus, our design enhances the performance upon this previous work by alleviating the cost of re-activation of the previous roles for the continuing usage of resources.

Fourth, for continuity of access, their work mandates a user to initiate a conformation protocol every fixed period of time to proof his location although the user remains in the same valid position. This approach, however, adds burden on the user's side and also on the verifier side. Unlike such approach, our event-based model requires a user to send a notification of access termination or suspension to the system at the moment the user moves to an invalid zone. With this design, we are able to reduce the number of communication passes as well as operations on both sides of client and servers.

Fifth, Kirkpatrick and Bertino informally analyzed the security of their design. We argue that such analysis cannot be considered complete unless it is formal. We, on the other hand, develop a threat model to determine possible attacks on our proposed design, and next we formally analyze those attacks using model finder Alloy [15]. Our enforcement model also has a number of design characteristics for computational and communication efficiency as well as for implementation flexibility, which this prior work did not attempt to provide. For example, if the application component could not authenticate the user's login information, it immediately sends a rejection response to the user without consulting with the policy component. Our design also enhances the application performance by alleviating the involvement of communication overhead with a third party, such as a separate location device, for providing authentication information.

Finally, their design assumes a single access request at time, a single recourse manger to check the user request, and a single role manger to map the user to a set of roles. Furthermore, the resource manger is inseparable part of the system architecture so that a designer must consider it when they develop their system. However, this approach requires a developer to make a major configuration in order to adopt a policy model and it is not suitable for widely deployed distributed architecture. In contrast to such design, our aim is to provide general and flexible unified enforcement model by separating a policy component from the point of use to make it possible to be integrated in many applications. Additionally, our design considers multiple authorization components that can provide the required level of access control. In our design, a system forwards access requests to one of the available authorization servers in order to check whether the resource access can be granted or not, and as such, the system is oblivious about the underlying polices. This design decision adds implementation flexibility and protects from malicious insiders given that the policy is managed by security officers in an isolated manner.

In one of our earlier works [77], we proposed a spatio-temporal role-based access control model for mobile applications and discuss a simple centralized architecture for enforcing policies adhering to this model. This model is useful for implementing policies for systems that have a single access decision point, a resource handler node, and a single data resource base. Our subsequent work extends the earlier approach along different dimensions. First, our proposed software architecture model supports distributed environment by incorporating multiple points of resources access, access decision, and databases for application data and policy. Henceforth, the current access control protocols considers multiple endpoints of interactions for processing the protocol passes. In addition, we specify the space and computation capabilities for system devices implementing our architecture design.

Second, we specify a number of architecture characteristics such as model generality (e.g., centralized and distributed systems), efficiency (e.g., using small number of passes and operations), and separating access decision from point of use to integrate our model with many applications. For example, the feature of generating access decision by application node without consulting the authorization node in the architecture excludes unneeded extra passes.

Third, we develop a self-contained algorithm for generating an authorization-token for a particular user request, which ensures no SoD conflicts can occur between new and current users' authorized roles and permissions. We also develop a threat model to identify possible attacks on our proposed design as well as to assess the applied countermeasures to tackle these attacks. This threat model is decomposed from three threat models: threat model for architecture nodes, the exchanged data threat model, and the treat model for the authorization protocol. As such, the common attacks identified by the threat model can be formally analyzed.

Last but not least, we provide a formal analysis approach to uncover vulnerabilities in the protocol design. We resort to the use of model finder Alloy that rigorously checks whether some attackers can break our protocol. Our analysis approach is twofold, it makes a sanity check that the protocol behaves as expected, and then it tends to check whether some attackers can exploit the design to launch an attack. Using Alloy, we model the correct behavior of the protocol design and instruct Alloy to generate an instance of the success of an attack. The analysis approach demonstrates the soundness of our proposed authentication protocols in the context of some well-known attack methods, and it might be followed as a general analysis approach for RBAC authorization protocols.

Enforcement has also been discussed in the context of XACML (eXtensible Access Control Markup Language) [78]. The XACML framework describes XML-based language for managing access to resources and provides a structure for enforcing access control policies.

In a usage scenario of XACML, a subject sends a resource access request to the Policy Enforcement Point (PEP), which is the entity protecting that resource. The PEP creates a request based on the subject's credentials and other relevant information and sends it to the Policy Decision Point (PDP). The PDP forms a request to the Policy Information Point (PIP) for retrieving information relevant to that request. Once the PDP has determined the access decision, the decision is returned to the PEP, which then allows or denies access to the requester.

Sometimes PEP forward requests to PDP via the Context Handler point, which generates authorization requests based on the resource request details from a user. The Context Handler point fetches the current users' contextual information and incorporates this information into that authorization requests and then it queries PDP.

Our architecture differs from the XACML structure in relation to the use of the PIP and Context Handler points. We have decided to initially exclude these points to allow for a simpler architecture and minimal communications overhead. Typically, PEP communicates with PDP through Context Handler point when the user's contextual information needs to be considered for making access decision. However, this mechanism introduces additional communication steps.

In our work, the authorization module needs to contact the recourse module, which imposes additional delay in the access decision. Since the current user's spatio-temporal zone is incorporated into the access request packages sent by users, there is no need to intermediate the Context Handler point in between PEP and PDP. In its place, the authorization module is only required to authenticate a users' zones initiated by the resource module. Moreover, the communication overhead in our architecture is also reduced by closely coupling PIP and PDP into a single principle. Additionally, this module acts as PDP for evaluating the spatio-temporal zone and role membership claims, and in combination with the resource policy itself, the authorization module acts as the PIP. In our architecture, the authorization module manages access based on the underlying access control policy whilst the resource module authenticates a user's subscription in a system.

Moreover, existing XACML formulation of RBAC policies termed as XACML RBAC profiles (RB-XACML) [79, 80] do not support the impact of spatio-temporal information on RBAC components (RBAC entities and relationships). These RB-XACML profiles only consider the contextual information associated with subjects making resource access requests. Moreover, associating spatio-temporal information with RBAC components in those RB-XACML profiles is a non-trivial task and there is no a clear approach for formally verifying the RB-XACML profiles before deploying the such policies.

Chapter 3

Model Specification and Verification using UML and OCL

Role-based access control (RBAC) models have been receiving growing attention as they provide access control security through a proven and increasingly predominant authorization model. One of the main advantages of the RBAC in comparison with other authorization models is the ease of its security administrations [81]. Using RBAC, organizations are capable to model security from their unique perspective. RBAC models are policy neutral [21]; they can support different access control policies including mandatory and discretionary through the appropriate role configuration. Security principles such as least privileges, Separation of Duties (SoD), and pre-requisite authorizations are proven to be adequately specified using classical RBAC model [36].

With growth of the network technology and increasing use of mobile devices (e.g., smart phones) new security requirements pose new security challenges. For mobile applications, access decisions are influenced by spatio and temporal information of both subjects and objects in the applications. Since classical RBAC model does not take into account environmental factors in making access decisions, some studies have extended RBAC in the spatio-temporal domain to satisfy mobile policy requirements [9, 10, 16]. Generally, these models compose spatio properties defined in spatio RBAC extensions [50, 51, 49] with the temporal properties specified in temporal RBAC extensions [82, 47, 43]. However, in spatio-temporal polices access depends simultaneously on both location and temporal information associated with access requests and with the system entities. With spatio-temporal RBAC models, spatio-temporal information are associated with RBAC components, and proper matching of these information authorizes the use of application services.

Our first contribution in this dissertation is the development of an access control model, which we refer to a generalized spatio-temporal RBAC (GSTRBAC). This model is built on the top of RBAC and it improves upon many of its counterpart models by supporting richer security features and has a powerful expressiveness via the notions of spatio-temporal zones (*STZone*) and role availability. The concept of *STZone* is an extent to RBAC entities, such as to a role which is referred to a role zone. *STZones* is a set of pairs for locations and intervals (e.g., Cartesian product of set of locations and intervals) where and when RBAC entities are available for use. Unlike existing spatio-temporal RBAC models, spatio-temporal information are not only associated with application users and roles, instead, they are associated with permissions and objects stored in computing machines at different physical locations. Further, the notion of *STZone* helps us to flexibly specify a variety of spatio-temporal policies at more granular level than the prior spatio-temporal models. For information security, we assume that the location devices signalling the current users' locations are tamper proof.

Some difficulties in specifying security properties in existing spatio-temporal RBAC models originated from the isolated association of temporal and spatio information with RBAC components. Strictly speaking, a role is associated with a set of locations and with a separate set of intervals at which that role can be assumed. Every time a user sends an access request, the user context information is evaluated with the set of locations first and then with set of intervals or vice versa. This process might introduce inconsistency as a result of inaccurate matching of spatio and temporal information of the role. Additionally, security features like pre-requisite memberships, objects accessibility, and pre-requisite are not supported by previous models.

Putting our model in use, the security artifact simply determines a set of *STZones* for a system and then associates them with RBAC entities. In these *STZones*, we say that roles, permissions, and objects are spatio-temporally accessible. This process alleviates the effort and errors from combining spatio and temporal information for each access request. Additionally, various operations are defined on *STZones* to flexibly model many security properties including user-role assignment, permission-role assignment, role activation,role hierarchy, SoD, and pre-requisite in the context of spatio-temporal domain. For example, the containment and equality operations refer to some points shared by two distinctive *STZones*. In other words, containment and equality operations are specific type of overlapping between *STZones* points.

The semantics of the GSTRBAC model is defined using the Unified Modeling Language (UML) [14] and Object Constraint Language (OCL) [22], though we believe that other formalisms can also be used for such specification. UML is a general-propose language that has been considered as the de facto standard in modeling software. UML has a set of graphical notations that specify static as well as dynamic aspects of software systems. The graphical diagrams of UML make it intuitive and appealing to software developers. Moreover, UML could be used in all phases of software development process and has many specifications and analysis tools [23] that software engineers could utilize. For its prevalence and its support for software development process, we believe that specifying GSTRBAC model in UML makes the integration of access control into software development process more feasible. Consequently, we decided to use UML and its constraint language, the Object Constrain Language (OCL), for specifying different security properties supported by our model.

In our specification approach of GSTRBAC, UML class diagram provides a structural view of the proposed model entities and relationships, whereas OCL expresses different spatio-temporal constraints in the model. The fact that *STZone* entity is defined as an aggregate class in the UML model which encapsulates both location and interval classes, simplifies the definition of various spatio-temporal constraints in OCL. Further, model entity relationships are describe as class in order to facilitate the specifications of spatio-temporal operations and constraints. The definition of *STZone* as a class aims also to reduce the effort for the validation of policies specified using the GSTRBAC model. Presently, we will demonstrate the practical implementation of our model through the specification of a spatio-temporal policy for a real-world application.

With the all features supported by the GSTRBAC model, GSTRBAC needs to be analyzed at the model level and application level for conflicts and inconsistencies. Some works have used multiple formalizms in order to specify and verify the semantics of their models [9, 10, 12, 16]. The advantages of the verification tools capabilities motivated this choice. Thus, this method requires the use of different languages, thereby transformation between modeling and verification languages, such as Alloy [15], Colored Petri Nets (CPN) [64], or Timed-Automata (TA) [24], is necessary. However, the transformation is challenging since it requires a formal proof of the mapping correctness and completeness.

To the best of our knowledge, there are very few approaches that are based on transformation in which the transformation process is formally proven to be correct. Some of these techniques use a mathematical approach to prove the transformation that is based on a technique called bi-simulation presented in Robin Milner book [83]. Additionally, this technique is not applicable in all situations because there are parts in the transformation process that makes bi-simulation undecidable. It is worth noting that these techniques, that use bi-simulation, are not specific to the specification and analysis of access control policies.

Our specification approach mitigates the transformation issues by using the prevalent UML/OCL modeling and validation USE tool [23]. Here, spatio-temporal policies are defined as instances of the GSTRBAC class model. UML-based approach is used for both the specification and the verification of the GSTRBAC policies. The fact that UML is the standard modeling language in software industry makes our approach even more suitable to be used. Therefore, industry security designer using UML are relieved from learning new formal language just to do the verification. This design decision makes our approach more applicable than other approaches in the industrial context.

In this chapter, we discuss the specification and verification of the proposed access control models using the UML/OCL languages. Section 3.1 discusses the representation of spatial, temporal, and spatio-temporal information in our model. Section 3.2 introduces UML/OCL formalism of the proposed model. In Section 3.3, we discuss a lightweight verification approach based on UML/OCL. Section 3.4 presents the specification and validation of a military application using the USE tool.

3.1 Location and Time Representation

In our model, each entity and relation is associated with spatio-temporal information. Before describing these associations in details, we show how spatio-temporal information is represented in our models.

3.1.1 Location Representation

Our model considers two types of locations: physical and logical locations. In real world, the *physical location* is a collection of physical points in the space in which objects are localized by means of three-dimensional coordinates [50, 52]. The physical points are represented by set P, where $P = \{p_1, p_2, \ldots, p_m\}$, such that m is the number of physical points.

However, in software systems, the absolute space is not the most appropriate structure for performing access control. Alternatively, the symbolic representation is better conceived by programmers and system administrators than the absolute one. The symbolic representation of a physical position is referred as a logical location. Here, each logical location identifies a place of interest such as *Teller Office*, *Classroom*, *CS-Department*, etc.

We define a logical location called *anywhere* that contains all other locations. Each application can describe logical locations at different granularity levels. For example, some permissions might be applicable on the entire state whereas other permissions are only applicable to people in the city. A location is defined as minimal if it represents the smallest level of granularity supported by the application. In other words, the size of the smallest location in **P** corresponds to the *minimal location granularity* of the application. For example, in the organization Software Development Corporation, we may have **L** = { *MainBuilding, TestingOffice, DirectorOffice, DevelopmentOffice* }. The *MainBuilding* houses the three offices in separate floors of the building. In this case, the minimal location granularity is one floor. In other words, *TestingOffice*, *DirectorOffice*, and *DevelopmentOffice* are minimal logical locations, but *MainBuilding* is the containment location.

Since logical locations formed from a set of physical points in the space, we define a one-to-one mapping function T to perform the translation. We define the set L to determine all logical locations in a system from which protected resources are accessed.

Definition 1 [Mapping Function T**]** *T is a total function that converts a logical location into a subset of physical points in the space.*

- *$T : L \to 2^P$, where P is the set of physical points and L is the set of logical locations.*

Different operations can be performed on logical locations namely locations *containment* $\subseteq$, *equality* $=$, and location overlapping $\cap$.

- **Locations Containment** $l_j \subseteq l_i$**:** Logical location l_j is said to be contained in another location l_i, i.e., $l_j \subseteq l_i$: if all physical points of location l_j are physical points of location l_i, i.e., $T(l_j) \subseteq T(l_i)$

- **Locations Equality $l_i = l_j$:** Two logical locations l_i and l_j are equal, i.e., $l_i = l_j$ if their physical points are similar, i.e., $T(l_i) = T(l_j)$

- **Locations Overlapping $l_i \cap l_j$:** Two logical locations l_i and l_j are said to be overlapping, i.e., $l_i \cap l_j$ if they share some physical points, i.e., $T(l_i) \cap T(l_j) \neq \phi$

3.1.2 Time Representation

The periodic and duration behavior are represented based on the definition of the periodic expression introduced by Bertino et. al. [45]. A periodic expression is represented by two formalisms: the symbolic and mathematical formalisms. The symbolic formalism is expressed as a tuple of two elements $< [begin, end], P >$, where P is an expression that refers to a set of time intervals, and $[begin, end]$ is the scope defines the lower and upper bounds of P expression. The expression P is defined based on the calendar notion by Niezette and Stevenne [84]. Calendars are formalized by the Hours, Days, Months, Years sets.

For example, the periodic expression $Years + 7.Months \triangleright 3.Months$ represents the set of intervals starting at month seventh of every year, and having a duration of three months (e.g., summer time). The date expression is used to represent the symbolic boundaries *[begin,end]*, i.e., *[1/1/10, 12/31/10]*. For example, the periodic time *<[1/1/10, 12/31/10], Everyday + 8.Hours ▷ 8.Hours* > specifies the set of time intervals with a duration of 8 hours, each starting at 8 a.m. (e.g., *[8 AM to 4 PM]*) every day for a year, 2010.

Although symbolic expressions are convenient for users, they are not easy to manipulate by application processes. Therefore, symbolic expressions given by administrators need to be translated into a mathematical format that is easily handled by programs. Mathematically, a periodic expressions is a set of intervals whiten certain date boundaries. Elements of an interval are time instants represented by natural numbers $\mathbb{N}$. A *time instant* is one discrete point on the time line.

A time interval is a set of consecutive time instants which can be represented in the form of $d = [t_s - t_e]$, where t_s, t_e represent time instants and t_s precedes t_e on the time line if $t_s \neq t_e$. We use the notation $t_i \in d$ to mean that t_i is a time instant in the time interval d. The exact granularity of a time instant is application dependent. Suppose the granularity of time instant in an application is one minute. In this case, time interval *[3:00 a.m. - 4:00 a.m.]* consists of the set of time instants $\{$ *3:00 a.m., 3:01 a.m., 3:02 a.m., . . . , 3:59 a.m., 4:00 a.m.* $\}$.

The *minimal time granularity* of an application refers to the size of the smallest time interval used by the application. For example, in the Software Development Corporation, we may have the following intervals that are of interest: **I** = $\{i_1, i_2, i_3, i_4\}$, where $i_1 = [8a.m. - 5p.m.]$, $i_2 = [8a.m. - 12p.m.]$, $i_3 = [12p.m. - 1p.m.]$, and $i_4 = [1p.m. - 5p.m.]$. The minimal time granularity pertaining to this application is one hour, which is interval i_3. Furthermore, i_2, i_3, and i_4 are considered minimal, but i_1 is not as it contains at least one other interval.

We define a time interval called *always* that includes all other time intervals. Each application should be able to express different types of temporal intervals. The set of all time intervals of interest to the application is defined by **I**. Set **I** has all intervals corresponding to those in the periodic expressions. In order to manipulate intervals, the *Instant* function determines the subset of time instants for an interval.

- $\mathit{Instant} : I \rightarrow 2^{\mathbb{N}}$

We define the following three operations that can be performed on time intervals.

- **Intervals Containment** $d_j \subseteq d_i$**:** Time interval d_j is said to be contained in interval d_i , i.e., $d_j \subseteq d_i$, if all time instants of interval d_j are time instants of interval d_i, i.e., $\mathit{Instant}(d_j) \subseteq \mathit{Instant}(d_i)$

- **Intervals Equality** $d_i = d_j$**:** Two intervals d_i and d_j are equal , i.e., $d_i = d_j$, if their time instants are similar, i.e., $\mathit{Instant}(d_i) = \mathit{Instant}(d_j)$

- **Intervals Overlapping** $d_i \cap d_j$**:** Two time intervals d_i and d_j are said to be overlapping, i.e., $d_i \cap d_j$, if they have time instants in common, i.e., $\mathit{Instant}(d_i) \cap \mathit{Instant}(d_j) \neq \phi$

3.1.3 Spatio-Temporal Zone (STZone)

A spatio-temporal zone is an abstract logical unit that encapsulates both spatial and temporal information. STZone is a one of the core components of the proposed model, which is linked to model entities. The spatio-temporal zone is written as STZone. For example, a spatio-temporal zone z_i is represented as a pair: $< l_i, d_i >$ where l_i represents a spatial component and d_i represents a temporal one. In our model, the entity availability is proportional to spatio-temporal zones that define where and when that entity is accessible.

Definition 2 [Spatio-Temporal Zone] *A spatio-temporal zone STZone is a pair of the form $< l, d >$ where l and d represent the logical location and the time interval respectively. An example of a spatio-temporal zone can be, z =*<Home Office, [6 p.m. - 8 a.m.]>.

The set of all spatio-temporal zones in a system is STZones, where $STZones = \{z_1, z_2, \ldots, z_m\}$. An example of a spatio-temporal zone set is $\{$ < *HomeOffice, [6 p.m. - 8 a.m.]*>, <*DeptOffice, [8 a.m. - 6 p.m.]*> $\}$. A spatio-temporal zone $< l, d >$ is specified at *minimal granularity* if *l* and *d* are specified at minimal location granularity and minimal temporal granularity respectively. In other words, STZone $< l, d >$ has the location and time interval that do not contain other location or interval respectively in the context of the applications. Formally, in STZone $< l, d >$, location *l* is minimal *iff* $\neg \exists \ l' \in P, \ l' \neq l$ such that $l' \subseteq l$, and time interval *d* is minimal *iff* $\neg \exists \ d' \in I, \ d' \neq d$ such that $d' \subseteq d$.

Different zone contents define special classes (types) of constraints, such as temporal, spatial, and strong constraints. The *universal zones* content defines strong constraints that should hold at any time and in any locations, i.e $z_u =< anywhere, anytime >$. The second content of zones is *temporal zones* that expresses temporal constraints that should hold at any location but during a certain period of time *i*, i.e $z_i =< anywhere, i >$. The *location zones* content specifies the location constraints that should hold at any time but in location *l*, i.e., $z_l =< l, anytime >$.

Furthermore, the *ZInt* and *ZLoc* return the interval and location in a zone, respectively. These functions are important to elaborate the content of a zone in order to define granular constraints. STZones, *ZInt*, and *ZLoc* are formally defined as following:

- $STZones \subseteq I \times L$

- $ZInt : STZones \rightarrow I$

- $ZLoc : STzones \rightarrow L$

In the following, we define a set of operations that can be performed on zones:

- **Spatio-temporal zone Containment** $z_j \subseteq z_i$**:** Spatio-temporal zone $z_j =< l_j, d_j >$ is said to be contained in zone $z_i =< l_i, d_i >$, i.e., $z_j \subseteq z_i$, if they have both intervals containment and locations containment, i.e., $d_i \subseteq d_j$ and $l_i \subseteq l_j$.

- **Spatio-temporal Equality** $z_i = z_j$**:** Two zones $z_i =< l_i, d_i >$ and $z_j =< l_j, d_j >$ are equal, i.e., $z_i = z_j$, if their time intervals and logical locations are similar, i.e., $d_i = d_j$ and $l_i = l_j$.

- **Spatio-temporal zone Overlapping** $z_i \cap z_j$**:** Two zones $z_i =< l_i, d_i >$ and $z_j =< l_j, d_j >$ are said to be overlapping $z_i \cap z_i$ if their time intervals and logical locations are overlapping, i.e., $d_j \cap d_i$ and $l_j \cap l_i$.

We define the following predicate to evaluate the relation between zones. Predicate $containedZones(z, z^{'})$ is true when zone z and zone $z^{'}$ are related by one of the zones' operators.

- $containedZones(z, z^{'}) \;\Rightarrow\; z = z' \;\vee\; z \subseteq z'$

Consider the following example of STZones containment. $<$ *FortCollinsOffice, May2011* $> \subseteq$ $<$ *Colorado, Year2011* $>$ since *T(FortCollins)* $\subseteq$ *T(Colorado)* and *May2011* $\subseteq$ *Year2011*. However, $<$ *FortCollins, May2011* $> \nsubseteq <$ *Colorado, Year2010* $>$ since *May2011* $\nsubseteq$ *Year2010*. Similarly, $<$ *FortCollins, May2011* $> \nsubseteq <$ *Nevada, Year2011* $>$ because *T(FortCollins)* $\nsubseteq$ *T(Nevada)*.

3.2 A Generalized Spatio-Temporal Access Control Model

The GSTRBAC model is designed on top of the classical RBAC model [36]. In our model, access is controlled by spatio-temporal information. Model entities and relationships are associated with spatio-temporal zones. For example, a system administrator can take a back up from his/her office during working hours. At home, after working hours, the same administrator is only allowed to perform some updates (e.g., installing software).

To perform a spatio-temporal access control, a location and time device should be attached to users and objects. We assume a GPS standard is installed in a computing machine, which signals the current location of entities, and time reader device which returns the current local time.

Furthermore, information sent by these devices is trusted because these devices are considered tamper-proof; malicious programs cannot fraud spatio-temporal zones.

3.2.1 Model Entities

This section describes how spatio-temporal zone component is integrated into user, role, permission, and object components. We exclude the sessions from our model because STZones associated with users have the same concept and at a more granular level than sessions. In conformance with session termination, whenever a user move-out of the early declared zone, the user's rights are revoked .

Users:

Two kinds of users are considered in GSTRBAC: human users or agents operating on behalf of humans. Unlike standard RBAC, users are mobile in nature. *Users* is the set of all the users identifiers in a system, i.e., $Users = \{u_1, u_2, \ldots, u_n\}$, where n is the number of users in a system.

Access requests are typically instantiated by users to activate roles or use some permissions of active roles. The spatio-temporal zone associated with a user gives the user's current location and time. We assume that each valid user, interested in doing some location-sensitive operations, carries a locating device that is able to track his location. The location of a user changes with time.

Upon receiving a user access request, a system processes the spatio-temporal information to decide whether to allow or deny access. For each request, a typical question to be asked is in the form of "Is a user allowed to perform a certain action on an object from his/her current zone? " This question is formulated in our model as a predicate to be approved before allowing access. We define the *currentzone* function to give the current *user zone* from which a user can use system's resources or services.

- $currentzone : Users \longrightarrow STZones$

Note that, time and location can have different levels of granularity. For example, the current time can be expressed as *12:00:05 p.m.* or *12:00 p.m.* Similarly, a user's current location can be Fort Collins or it can be Colorado. The user's current location and time information will be used for making access decisions. Let us illustrate why the notion of minimality must be associated with the user's spatio-temporal zone. Suppose permission is valid in a certain zone. If we do not use the concept of minimal, then it is possible that the user zone may partially overlap with the permission zone. In such a case, should we give access or deny access? On the other hand, if we use the concept of minimal, then the user's zone will either be within the permission zone or outside it. In such cases, we know whether to give or deny access. Consequently, we require the minimal temporal and location be used to express the spatio-temporal zone associated with a user.

Consequently, we require the minimal temporal and location be used to express the spatio-temporal zone associated with a user. For example, the minimal physical location of a user in the 110 CS lab inside the computer science (CS) building, is the three co-ordinates of the 110 CS lap which is contained in the physical points of the CS building.

There are some challenges for capturing users' zones; since users are mobile, their locations change over time. The changes in the user zones may block the execution of some authorized roles at the time a user moves into an invalid position or the access duration ends. Furthermore, it is also important to resume the user's privileges when the user go back to its valid zone. STZone provides a kind of intelligence to contiguously track the status of the current active roles. It is also used to trigger exceptions to revoke roles at the time a user moves in an invalid zone. These issues are a major concern when policies are enforced in a system. Chapter 5 discusses some protocols that handle the environmental changes based on the STZone concept. Most of the existing spatio-temporal models do not address the environmental changes in the access requests at the run time.

Objects:

Objects are similar in nature to mobile users. The set *Objects* has all the objects' identifiers in an access control policy, i.e., $Objects = \{o_1, o_2, \ldots, o_n\}$. Files are examples of objects which contain or receive information and they are stored in different locations. The computing machines where objects are stored are associated with trusted devices that send locations and timestamps information. For example, tellers can only review a customer file at a bank only during working hours. The function *ozones* determines the subset of *object zones* where an object is accessible.

- $ozones : Objects \rightarrow 2^{STZones}$

Operation/Activity: Operations or activities are similar to functions in a program that are invoked to perform certain actions on objects. A single operation can be performed on one or more objects. For example, money transfer from one account to another is an abstract operation that updates multiple account information. The set of all operations in a system is *Operations*.

Roles:

The set of all roles in a system is defined by set *Roles*, i.e., $Roles = \{r_1, r_2, \ldots, r_n\}$. In GSTRBAC, a role is available for activation or assignment in some predefined spatio-temporal zones which are referred to as *role zones*. For example, a person can activate the role of nurse only when she/he is in a hospital and during the working hours (day-time or night-time). To express such requirement, we define a total function *rzones* that maps each role to a set of spatio-temporal zones.

- $rzones : Roles \rightarrow 2^{STZones}$

Permissions:

The set of all permissions identifiers is termed as *Permissions*, i.e $Permissions = \{p_1, p_2, \ldots, p_n\}$. In our model, a permission is an abstract view of the combination of objects and operations. Following functions determine the objects and operations associated with a permission.

- $permObjects : Permissions \longrightarrow 2^{Objects}$
- $permOperations : Permissions \longrightarrow 2^{Operations}$

Spatial-temporal zones are also associated with permissions to defined users' access mode. Each *permission zone* specifies the location and time in which a permission can be used. For example, a nurse-on-duty is allowed to give medications to a patient and write in the patient's file in a specific ward and during the night. The *pzones* function returns the set of *permission zones* associated with a permission.

- $pzones : Permissions \rightarrow 2^{STZones}$

3.2.2 UML/OCL Specifications of GSTRBAC

This section presents the specification of the GSTRBAC model in UML/OCL. The structural concept of the GSTRBAC model is specified using UML class diagram with OCL constraints. The behavioral aspects are modeled using pre-conditions and post-conditions of OCL operations. The spatio-temporal constraints on GSTRBAC entities and relationships are specified in OCL invariants. We first define the conceptual class diagram model describing GSTRBAC components. Then, we define OCL expressions to specify spatio-temporal constraints and operations. Before starting the discussion of the GSTRBAC formalism, we provide a brief overview of UML and OCL languages.

Overview of UML and OCL

Unified Modeling Language (UML):

UML is a family of modeling languages through which we can specify, visualize, and document objects of software systems [14]. UML has a set of intuitive graphic notations which can sufficiently model various components in a software system. Its importance in the software industry is attributed to its use in all phases of a software development process. It helps a software engineer to confine and understand the structure of a system and how they interact with each other to accomplish functional requirements.

UML includes functional, static and dynamic models. The first model specifies the functional requirements of a software system using use-case diagrams. The second is the static model which represents the structural view of data in a system at the conceptual, requirement, and implementation levels. System entities are replaced by object classes in the UML static diagram. In the static diagram model, object classes are defined in terms of names, attributes, and behavior while associations among classes are annotated by multiplicity, constraints, and role names. Associations have different forms including specialization, generalization, and aggregation relations. In the third model, the behavior of a system is modeled using dynamic diagrams such as collaboration, sequence, and state-chart diagrams.

Object Constraint Language (OCL):

OCL is a declarative/textual language that was designed specifically to enable UML developers to write constraints that are not possible to specify in a diagrammatic manner [22]. It enforces constraints on class values and relationships with conformance to system requirements. Collections are the main building blocks of OCL which include sets (no duplicates), bags (duplicates), and sequences (duplicates, ordered). Various class invariants and operation restrictions can be represented by OCL to control the structure and behavior in class diagram instances.

The expressions in OCL are constructed in four parts: context identifies the limited situation in which the expression is valid; property is a characteristic of the context which might be an attribute or set if the context is an object class; operation can be arithmetic, set manipulations, or qualifies a property; and keywords specify conditional expressions, in particular, attributes or collections (e.g., if, then, else, and, or, not, implies).

GSTRBAC Class Model

The UML class diagram model in Figure 3.1 defines the conceptual structure of the GSTRBAC model. GSTRBAC entities such as zones, users, roles, permissions, objects, and activities are represented by classes. Most of the entities classes are associated with the STZone class in order to determine the zones where these entities are spatially and temporally available. All entities are defined by single compact classes except for permissions and STZones entities. Permissions are represented by superclass Permission of subclasses Object and Activity.

The STZone class is the core class in the class diagram model that is also connected to GSTRBAC relationships. The STZone class is a superclass that encapsulates the Location and the TimeInterval classes. *zcontainment* is a reflexive association that specifies the requirement that a zone can be contained in other zones. The query *containedZones()* operation in class STZone returns the set of zones that are contained in a certain zone. GSTRBAC relationships are modeled using association classes which are transformed to normal classes following the modeling guidelines in [14, 22].

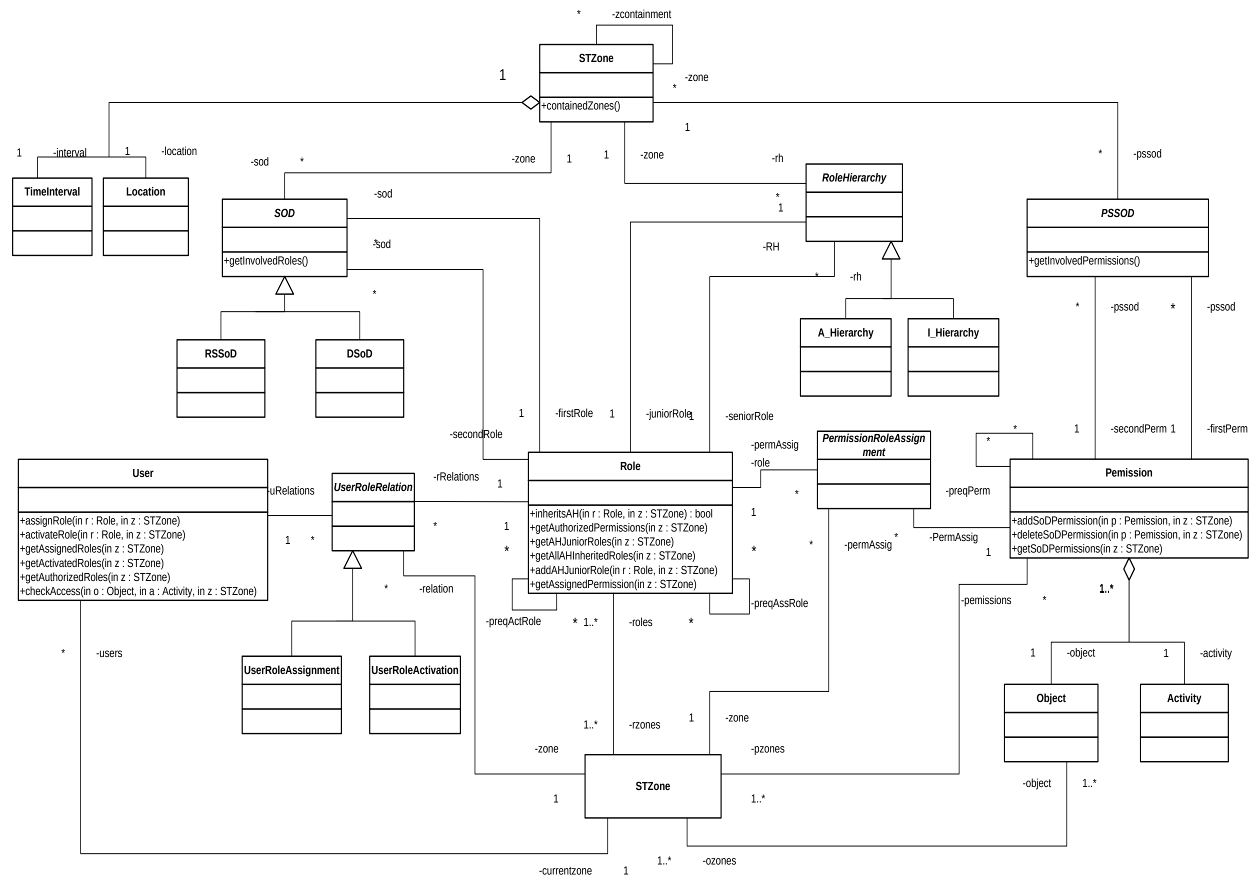


Figure 3.1: UML Class Model for GSTRBAC

Model Relationships

The relationships in GSTRBAC are spatio-temporal dependent. They are expressed in GSTRBAC class diagram by association classes between entities. These association classes have a binary relationship with the STZone class to define the spatio-temporal zones in which those relationships' entities are available. This way of modeling simplifies the specification of spatio-temporal constraints in OCL operations and invariants. Each association class has both association and class properties. Their instances are links that have attribute values as well as references to other objects.

A number of OCL expressions are defined to describe the spatio-temporal constraints on different relationships in the UML model of GSTRBAC. OCL constraints validate the correctness of relationships' instance by checking the association with STZone objects. For example, if entities in a relationship are available in a particular STZone, then the OCL constraint returns true, otherwise it denies the relationship from taking place in that zone. The complete UML/OCL GSTRBAC code generated by USE tool is shown in Appendix A.

User-Role Assignment:

User-role assignment is location and time dependent. That is, a user can be assigned to a role once the role is spatio-temporally available. This relationship is represented in the GSTRBAC class digram via subclass *UserRoleAssignment* of association class *UserRoleRelations*. *UserRoleRelations* is linked to class STZone to specify the spatio-temporal constrains. Furthermore, user-role assignment should only happen once in each zone. The following OCL operation describes the spatio-temporal constraints on assigning a user to a role.

```
context User::assignRole(r: Role, z:STZone): UserRoleAssignment
pre:  r.rzones -> includes(z)
pre:  self.currentzone -> includes(z)
pre:  self.getAssignedRoles(z)-> excludes(r)
post: self.getAssignedRoles(z)-> includes(r)
```

The preconditions respectively verify that role *r* is available in zone *z*, current user *u* is in zone *z*, and role *r* is not already assigned to user *u* in zone *z*. The OCL query *getAssignedRoles(z)* operation determines the subset of assigned roles to user *u* in STZone *z*. The post-condition asserts that a user-role assignment instance has been created between user *u* and role *r* in zone *z*.

User-Role Activation:

A user can activate a role if the role is spatio-temporally available and it is already assigned to that user. For example, the role of doctor trainee can only be activated in a hospital during the training period. The difference with spatio-temporal user-role assignment is that the assignment of roles does not mean roles are being used, but the activation refers to the usage of roles in a specific zone. The dual checking of activation zones services as a second line of defence for any error in the first place of user-role assignment.

User-role activation is specified in the GSTRBAC class diagram via subclass *UserRoleActivation* which has an association with the STZone class to define the zones where the role activation is allowed. Spatio-temporal constraints on a user-role activation operation are specified in OCL *activateRole()* operation.

```
context User::activateRole(r: Role, z:STZone): UserRoleActivation
pre:  self.currentzone->includes(z)
pre:  r.rzones->includes(z)
pre:  self.getAssignedRoles(z)-> includes(r)
pre:  self.getActivatedRoles(z)-> excludes(r)
post: self.getActivatedRoles(z)->includes(r)
```

The first precondition checks that user *u* is currently in zone *z*, the second one ensures that role *r* is available in zone *z*, user *u* assignment to role *r* in zone *z* is checked in the third precondition, and the last one verifies that role *r* is not already in active state by user *u* in zone *z*. The post-condition ensures the creation of *UserRoleActivation* instance between user *u* and role *r* in STZone *z*.

User-Object Access:

The goal of authorization models is protecting the access to system's objects from unauthorized users. In the spatio-temporal domain, a user access to protected objects from specific locations and intervals. For example, a secure bank policy might require that a teller is allowed to have a read and write access to teller files from the teller booth and during working hours. This feature is not explicitly defined in existing spatio-temporal models. These models merely define spatio-temporal constraints on roles and permissions.

The process of a user object access is three steps: first, the user activates a role; second, the user accesses an appropriate permission associated with that role; third, the user accesses to the object through that permission. The zones associated with the user, role, permission, and object should be equivalent or have the zones containment relation. The *checkAccess()* boolean operation checks whether a user is allowed to access an object in a specific zone.

```
context User::checkAccess(o:Object,a:Activity,z:STZone):Boolean
post: result = getActivatedRoles(z)->
```

```
collect( r | r.getAuthorizedPermissions(z))-> asSet()->
exists( p | p.object=o and p.activity=a and o.zones-> includes(z))
```

The OCL query returns true only if a user has activated role *r* in zone *z*, which has permission *p* to access object *o* in that zone.

Permission-Role Assignment:

Permissions are assigned to roles in specific spatio-temporal zones. For example, a permission to open a cashier drawer in a store should only be assigned to a salesman role and during the day-time.

Permission-role assignment is specified in the GSTRBAC class model by association class *PermissionAssignment* that is also linked to the STZone class. The spatio-temporal constrain on assigning a permission to a role is specified by the OCL *assignPermission()* operation.

```
context Role::assignPermission(p:Permission,z:STZone):
PermissionAssignment
pre:   p.pzones->includes(z) and self.rzones->includes(z)
pre:   self.getAssignedPermissions(z)-> excludes(p)
post:  self.getAssignedPermissions(z)-> includes(p)
```

The first precondition checks that zone *z* belongs to the permission *p* and role *r* zones. The second precondition checks that permission *p* has not yet assigned to role *r* in zone *z*. The OCL operation *getAssignedPermissions(z)* returns the subset of permissions assigned to role *r* in STZone *z*. The result of this operation is a new instance of *PermissionAssignment* that links role *r*, permission *p*, and STZone *z*.

Role Hierarchy

In role hierarchy (RH), higher-level roles (i.e., senior roles) dominate lower-level roles (i.e., junior roles). Role hierarchy spans the scope of permission acquisition and role activation over explicit assignment. For example, a project supervisor role inherits permissions from a programmer role via role hierarchy in addition to its assigned permissions. Furthermore, RH prevents the problem of multiple permissions assignment to roles.

The permission-inheritance Hierarchy (*I-Hierarchy*) and the role-activation hierarchy (*A-Hierarchy*) are two distinct subtypes of role hierarchy [43, 85]. In permission-inheritance hierarchy, members of senior roles can access the inherited permissions without the need to activate the junior roles. In role-activation hierarchy, members of senior roles are not implicitly authorized to permissions of junior roles, they need to activate those junior roles.

In our model, *I-Hierarchy* and *I-Hierarchy* are spatio-temporal dependent. The spatio-temporal role hierarchy only holds in specific locations and time intervals. In Figure 3.1, the aggregate *Role-Hierarchy* class is linked to the STZone class to specify the set of zones where role hierarchies are authorized. For the sake of simplicity, the zone where a senior role inherits a junior role is considered to be the same zone associated with senior and junior roles. The following shows how different forms of role hierarchies are defined and constrained in the UML/OCL specification.

Permission-Inheritance Hierarchy:

Here, location and time constraints should be satisfied in order of a senior role to inherit permissions from a junior role. For example, a software company policy might require that a software project supervisor can only inherit permissions of a test engineer in the testing department during the day-time.

The permission-inheritance is represented by *I-Hierarchy* class. Each instance of the *I-Hierarchy* class draws the path between a senior role, a junior role, and a zone objects. Whenever a new junior role is added to role hierarchy, a new instance of *I-Hierarchy* class is created if and only if the spatio-temporal zone constraints are satisfied. The following OCL operation expresses the spatio-temporal constraints on adding junior roles to *I-Hierarchy*.

```
context Role::addIHJuniorRole(r:Role,z:STZone): I-Hierarchy
pre: self.rzones->includes(z) and r.rzones->includes(z)
pre:  self.getIHJuniorRoles(z)-> excludes(r)
post: self.getIHJuniorRoles(z)-> includes(r)
```

The first precondition ensures that senior and junior roles are available in zone *z*. The second precondition checks that junior role *r* is not in the set of inherited junior roles of the context role.

Operation *getIHJuniorRoles(z)* returns the set of junior roles of the context role in zone *z*.

The delete operation of a junior role from *I-Hierarchy* in a particular zone can be defined in the similar manner to the addition operation. The following OCL operation defines the deletion of an *I-Hierarchy* instance.

```
context Role::deleteIHJuniorRole(r:Role, z:STZone)
pre: self.getIHJuniorRoles(z) -> includes(r)
post: self.getIHJuniorRoles(z)-> excludes(r)
```

The precondition checks that role *r* is in the subset of junior roles of the context senior role. The *getIHJuniorRoles(z)* operation returns the junior roles of the context role in zone *z*. The post-condition asserts that the *I-Hierarchy* instance is removed from the object model.

I-Hierarchy is a partial order and anti-symmetric in the context of spatio-temporal zone. That is, not all roles have senior roles or junior roles, and the relation between a senior role and a junior role is only one direction in a particular zone. Such requirement is enforced in *I-Hierarchy* by the following OCL invariant.

```
context r1,r2: Role
inv IHierarchy_Cycle_Constraint: not STZone.allInstances->
exists(z|r1. inheritsIH(r2,z) and r2.inheritsIH(r1,z)and r1<>r2)
```

This OCL prevents the creation of any cycles of *I-Hierarchy* instances between roles. Operation *inheritsIH(r,z)* returns true if a role is a junior role directly or indirectly of a context role. It evaluates the inheritance relation between roles through multiple levels of *I-Hierarchy*. OCL expressions do not have a primitive for the transitive closure like Alloy. We have implicitly defined the transitive closure using a recursive OCL operation. The OCL iterates construct allows such operation to be executed. OCL boolean operation *inheritsIH(r,z)* is defined as following.

```
inheritsIH(r:Role,z:STZone): Boolean =
if (self.getIHJuniorRoles(z)->includes(r)) then true
else self.getIHJuniorRoles(z)-> exists(j | j.inheritsIH(r,z))
endif
```

Permissions are either explicitly assigned to a role or implicitly acquired by a role via *I-Hierarchy*. The following OCL *getAuthorizedPermissions(z)* operation gives the subset of authorized permissions for a context role in STZone *z*.

```
context Role::getAuthorizedPermissions(z:STZone): Set(Permission)
Post: result= self.getAssignedPermissions(z)->
union(self.getAllIInheritedRoles(z)->
collect(r | r.getAssignedPermissions(z)))->
asSet()
```

Query *getAllIInheritedRoles(z)* uses boolean operation *inheritsIH(z)* in order to get the subset of junior roles of the context role in STZone *z*. Some of the existing spatio-temporal models totally lack the definition of such requirements or inaccurately considered them; they either define a single level of role hierarchies or define unambiguous predicates that ignores enabling conditions on intermediate roles and/or permissions.

```
getAllIHInheritedRoles(z:STZone): Set(Role)= Role.allInstances->
select(r | self.inheritsIH(r,z))-> asSet()
```

Role-Activation Hierarchy:

A restricted spatio-temporal role activation hierarchy allows members of senior roles to activate junior roles in predefined spatio-temporal zones. For example, members of a chairman role in a computer science department can only activate a role of a faculty member during the semester and inside the department building.

Role-activation hierarchy is represented by class *A-Hierarchy*, which has two binary associations with class Role and one with class STZone. The OCL operations of adding and deleting junior roles from *A-Hierarchy* are defined in the same way of the *I-Hierarchy*. The OCL operation of adding an *A-Hierarchy* instance is defined as following.

```
context Role::addAHJuniorRole(r:Role,z:STZone): A-Hierarchy
pre: self.rzones->includes(z) and r.rzones->includes(z)
```

```
pre:  self.getAHJuniorRoles(z)-> excludes(r)
post: self.getAHJuniorRoles(z)-> includes(r)
```

Operation *getAHJuniorRoles(z)* returns the junior roles of the context role in zone *z*. The OCL deletion operation of deleting a junior role from *A-Hierarchy* is specified as follows.

```
context Role::deleteAHJuniorRole(r:Role, z:STZone)
pre: self.getAHJuniorRoles(z) -> includes(r)
post: self.getAHJuniorRoles(z)-> excludes(r)
```

The cycling condition among different roles in *A-Hierarchy* is enforced in the same way in *I-Hierarchy*.

```
context r1,r2: Role
inv AHierarchy_Cycle_Constraint: not STZone.allInstances->
exists(z| r1.inheritsAH(r2,z) and
r2.inheritsAH(r1,z)and r1<>r2)

inheritsAH(r:Role,z:STZone): Boolean =
if (self.getHJuniorRoles(z)->includes(r)) then true
else self.getAHJuniorRoles(z)->
exists(j | j.inheritsAH(r,z)) endif
```

Users can activate roles explicitly through user-role assignment or implicitly via *A-Hierarchy*. OCL query *getAuthorizedRoles(z)* gets the authorized roles for a context user in STZone *z*.

```
context User:: getAuthorizedRoles(z:STZone): Set(Role)
post: result= self.getAssignedRoles(z)->
union(self.getAssignedRoles(z)->
collect(r| r. getAllAHInheritedRoles(z))->
asSet())
```

The result of this OCL query is a subset of roles that can be activated by a context user. OCL query operation *getAllAHInheritedRoles(z)* returns a subset of junior roles inherited by a context role in zone *z*.

3.2.3 Model Constraints

Spatio-temporal constraints are an important part of our access control model; they are a powerful techniques for expressing at a higher level organizational policy. Spatio-temporal constraints in our model are classified into two categories, namely, prohibition and pre-requisite constraints [86].

Prohibition Constraints

Prohibition constraints in our model prohibit any a GSTRBAC component from doing something which is not allowed to do in some undesirable spatio-temporal zones. An example of the spatio-temporal prohibition constraints is spatio-temporal SoD. In the following subsections, we discuss different forms of spatio-temporal SoD in our model.

Separation of Duties:
Static SoD (SSoD) and dynamic SoD (DSoD) constraints are two variations of SoD in RBAC [86]. SSoD prevents the assignment of conflict roles or permissions, whereas DSoD forbids the activation of conflict roles. That is, role SSoD (RSSoD) constraints are applied to user-role assignment, permission SSoD (PSSoD) constraints are applied to permission-role assignment, and DSoD to user-role activation

In our model, spatial and temporal factors are applied to the SoD constraints. Consider this motivation example, the same individual cannot be assigned to roles of student and teaching assistant for the same class and during a semester in a specific department. Spatio-temporal SoD constraints are defined using the zone notion. For the sake of simplicity, the SoD constraints are applied to conflicting roles or permissions in the same zones where conflicting entities are available. Here below we discuss how to specify spatio-temporal SoD in OCL expressions.

Role SSoD:
Sometimes the same user should not be assigned to some roles in a specific location and for a certain duration. For example, an individual should not be assigned to billing clerk and account receivable clerk roles in the same time at a specific trade corporation. Subclass RSSoD in Figure 4.1

links pairs of conflicting roles with a zone. Thus, roles that are related by RSSoD in a specific STZone should not have an *UserRoleAssignment* association instance with a single user in that STZone. The following OCL invariant forbids the instantiation of user-role assignment between conflicting roles.

```
context User
inv RSSOD_Constaint: STZone.allInstances->
forAll( z | not self.getAssignedRoles(z)->
exists(r1,r2 | r1.getSSoDRoles(z)->includes(r2)))
```

This OCL invariant ensures that the context user must not have user-role assignments with r_1 and r_2 roles that are related by RSSoD in zone z. Query operation *getSSoDRoles(z)* returns the conflicting role with a certain role in zone z.

This RSSoD invariant prevents the direct user-role assignment of conflicting roles in particular zones. However, this constraint might be violated through *I-Hierarchy*. RSSoD violation might occur when conflicting junior roles are inherited by a senior role in a critical zone. Thus, members of that senior role can have access to conflicting roles.

This RSSoD violation is encountered by propagating RSSoD between junior roles upwards *I-Hierarchy*. That is, a role must not inherit conflicting junior roles in some critical zones. For example, a billing supervisor role must not be a senior role of the two conflicting billing clerk and account clerk roles at the same time and in the same accounting department. The following OCL constraint enforces such requirement.

```
context User
 inv RSSOD_RH_Constraint: STZone.allInstances->
 forAll( z | not self.getAuthorizedRoles(z)->
 exists(r1,r2 | r1.getSSoDRoles(z)->includes(r2)))
```

This OCL invariant restricts a user from having conflicting roles through *I-Hierarchy*.

Permission SSoD:

Spatio-temporal PSSoD prevents conflicting permissions to be assigned to a role in particular locations and at specific intervals. For example, a loan officer is not permissible to issue a loan request and approve it in the bank building during the day-time.

Subclass PSSoD expresses this requirement by relating conflicting permissions with zones. Therefore, any two permissions instances have the PSSoD relation in a specific zone must not have an *PermissionsAssignment* association in that zone. The following OCL constraint prevents the assignment of conflicting permissions in undeniable zones.

```
context Role
inv PSSOD_Constaint: STZone.allInstances->
forAll( z | not self.getAssignedPermissions(z)->
exists(p1,p2 | p1.getPSSoDPermissions(z)->includes(p2)))
```

This OCL invariant states that conflicting permissions p_1 and p_2 must not have permission-role assignment instance in zone z. Query *getPSSoDPermissions(z)* specifies the conflicting permissions by PSSoD in zone z.

This OCL invariant protects against the direct assignment of conflicting permissions to roles, however, it can be violated through *I-Hierarchy*. Thus, the subset of permissions authorized to a role by direct assignment or indirect *I-Hierarchy* in some zones must not have conflicting permissions. The following OCL invariant prevents the violation of PSSoD via *I-Hierarchy*.

```
context Role
inv PSSOD_RH_Constraint: STZone.allInstances->
forAll( z | not self.getAuthorizedPermissions(z)->
exists(p1,p2 | p1.getPSSoDPermissions(z)->includes(p2)))
```

DSoD:

DSoD defines mutual exclusion roles at run-time. Spatio-temporal DSoD states that an individual should not activate conflicting roles at some intervals and in some locations. For example, a user might be assigned to both roles of cashier and cashier supervisor, where a supervisor is responsible for the correction of a cashier's cash drawer in a store and during day-time. However, if a user playing the cashier role attempts to switch to the cashier supervisor role, that user must drop the cashier role and close the cashier drawer before assuming the cashier supervisor role.

In Figure 4.1, subclass DSoD links conflicting activation roles with zones in which the role activation is forbidden. Whenever an instance of DSoD is created between two conflicting roles in a zone, *UserRoleActvation* instances between these roles cannot be generated. The following OCL constraint prevents the creation of *UserRoleActvation* between roles related by DSoD.

```
context User
inv DSOD_Constaint: STZone.allInstances->
forAll( z | not self.getActivatedRoles(z)->
exists(r1,r2 | r1.getDSoDRoles(z)->includes(r2)))
```

Operation *getDSoDRoles(z)* returns the conflicting activation roles in zone *z* that must not be simultaneously activated.

Note that a user can activate roles via user-role assignment or membership of senior roles in *A-Hierarchy*. The later activation might authorize a user to activate conflicting junior roles. Suppose that a specialist doctor role is defined as a senior role of the doctor-on-night-duty and doctor-on-day-duty roles in a hospital during night-time (10 pm to 10 am) and day-time (7 am to 7 pm) respectively. Thus, there is a partial containment time interval (7 am to 10 am) where both roles are available. Thus, members of the specialist doctor role might simultaneously activate the doctor-on-night-duty and doctor-on-day-duty roles via *A-Hierarchy* in the partial containment intervals.

Such DSoD violation cannot occur in our model. Query operation *getActivatedRoles(z)* returns the current subset of active roles by a context user in zone *z* that are either assumed by user-role assignment or *A-Hierarchy*, and are not in activation conflicts. Most of the existing spatio-temporal models did not (granularly) address the violation of SoD via role hierarchies.

Obligation Constraints

Sometimes an organizational policy enforces a component in RBAC to do (or being) something which is allowed by that policy. This motivation is known as pre-requisite constraints [86]. Sandhu et al. [36] have defined authorization constraints that mandate a user to have a combination of user-role assignment or user-role activation for some roles. For instance, a user should have been assigned role r_1 in order to be assigned role r_2.

Pre-requisite Constraints:

In our model, the pre-requisite constraints are location and time dependant which are defined using the STZone concept. For example, in order for a user to be authorized to a role in a specific STZone, the same user must have assumed another role in combination in that STZone. Two types of spatio-temporal pre-requisite constraints are specified in GSTRBAC. These pre-requisite constraints are not supported by existing spatio-temporal models.

Pre-requisite on User-Role Assignment:

Spatio-temporal pre-requisite constraints indicate that the assignment of a critical role necessitates the assignment of some less critical roles in some favorable zones. For example, a role of nurse-on-night-duty at a hospital can be assigned in an urgent care unit if and only if a role of nurse is already assigned in the same urgent unit and during night-time.

Therefore, the user-role assignment instances can only be created if and only if the pre-requisite constraints are satisfied. The following OCL constraint restricts the creation of *UserRoleAssignment* instances.

```
context User
inv Prerequiste_URAssign: STZone.allInstances->
forAll(z | Role.allInstances->
forAll(r1 | (self.getAssignedRoles(z)->includes(r1)) implies
(self.getAssignedRoles(z) ->
includesAll(r1.getPreqAssRoles()))))
```

This OCL constraint states that for all zones, if a user is assigned to role r_1 in zone z, this implies that all the pre-requisite roles are assigned to that user. Query operation *getPreqAssRoles()* returns the assignment pre-requisite roles needed for assigning a certain role.

Pre-requisite on Permission-Role Assignment:

This constraint states that some critical permissions can be assigned to roles in specific zones if and only if some pre-requisite permissions are already assigned to those roles. This constraint is a dual form of the pre-requisite user-role assignment constraint.

For example, some banking policies enforce that a teller role can be assigned the permission of read-customer file if and only if it has been assigned the permission of read-directory in a file server machine located inside a bank building and during day-time. This pre-requisite constraint is specified using the OCL expression in a similar way to the pre-requisite constraint on role assignment.

```
context Role
inv Prerequist_PRAssign: STZone.allInstances->
forAll(z | Permission.allInstances->
forAll(p1 | (self.getAssignedPermissions(z)->
includes(p1)) implies
(self.getAssignedPermissions(z) ->
includesAll(p1.getPrerequisitePermissions())))))
```

Pre-requisite on User-Role Activation:

With this constraint, a user should have a certain combination of roles in user-role activation in some zones. For example, a student role must be activated in a department during a semester time in order to activate a teaching assistant role for a particular class. This principle also enforces the legislation that an individual must be registered as a student in a department in order to be granted a teaching assistant position.

Thus, the creation of *UserRoleActivation* instances is constrained by this pre-requisite constraint in some critical zones. That is, whenever a user requests a user-role activation creation for a critical role in a specific zone, that user should have already created a user-role activation for some pre-requisite roles in that zone. The following OCL invariant restricts the generation of *UserRoleActivation* instances on the satisfaction of the pre-requisite role-activation constraints.

```
context User
inv Prerequist_URActiv: STZone.allInstances->
forAll(z | Role.allInstances->
forAll(r1 | (self.getActivatedRoles(z)->
includes(r1)) implies (self.getActivatedRoles(z)
->includesAll(r1.getPreqActRole()))))
```

Query operation *getPreqActRole()* returns a subset of pre-requisite activation roles needed to activate role *r* by user *u* in zone *z*.

3.2.4 Discussion

This section discusses some challenges that are encountered during the specification of the GSTRBAC model in UML/OCL. At the beginning, the class diagram model is designed to capture the semantic of GSTRBAC. Then, OCL constraints are incrementally defined in the context of the GSTRBAC class model in order to restrict model instances. OCL invariants as well as pre-conditions and post-conditions are defined based on the zone concept. Entities in the UML class diagram are linked with the STZone class association to specify the zones in which these entities are available.

For modeling relationships, we have tried different ways to link relationships with spatio-temporal zones. The relationships among entities are first specified using binary associations. For example, user-role assignment is specified using a binary association between the User class and the Role class. This binary association is named as *UserRoleAssignment* with one-to-many multiplicity.

However, modeling relationships in this way does not explicitly reflect the fact that GSTRBAC relationships are spatio-temporal dependent. For instance, binary association *UserRoleAssignment* between the User and Role classes in this manner does not express that the relationship is only valid in particular spatio-temporal zones. Furthermore, the operations in these classes are much harder to define and validate.

For example, the *assignRole(r:Role, z:STZone)* operation in the User class assigns a user to role *r* in STZone *z*. It is difficult to define such operation with a binary association due to the fact that *UserRoleAssignment* is zone dependent. Such definition requires the creations of links between class User and class STZone as well as between class Role and class STZone and writing additional OCL constraints in order to prevent ambiguities in the class model. To overcome this problem, one must define an attribute on the binary association of the role assignment that specify the zone in which the assignment holds. This choice makes binary associations between entities act as association classes. Thus, we decided to employ association classes between model entities to specify GSTRBAC relationships. These association classes reduce the complexity of policy specification and validation to a great extent. This technique is not followed by existing UML/OCL specification of RBAC.

As such, the specification of GSTRBAC relationships via association classes mirrors spatio-temporal constraints and reduces the complexity of defining operations and OCL constraints. With our approach, each relationship is represented by association class that is linked to two entities classes as well as to the STZone class. Consequently, operations defined in the context of the relationships classes becomes easier to define and check.

According to modeling guidelines of UML [14, 22], it is recommended that associations are represented by classes in such case. Another option was using ternary associations, however, ternary associations are usually not useful unless the multiplicity is many on all ends [14, 22]. In our case, multiplicity of the class STZone end is one in all these association classes. Therefore, we opted to use association classes and transformed them to normal object classes. Instances of association classes precisely capture the requirement that GSTRBAC relationships are STZone dependent. Moreover, association classes make OCL constraints more expressiveness and easy analyzed than binary associations.

After the GSTRBAC class model has been specified, OCL expressions are developed and tested on the model through the USE tool. This includes OCL invariants, pre-/post-conditions, and operations. The process of defining OCL expressions is incrementally performed on the class model. For each spatio-temporal constraint, a number of instances are created and tested until the constraint is satisfied. We utilized the basic mode of USE to create some good and bad instances manually [23]. Good instances are those that conform to a particular OCL constraint while bad instances violate it. If a good instance of the model violates the constraint, then the constraint is not properly specified. If a bad snapshot does not violate the constraint, then the model does not precisely capture our intuition of what the model should and should not allow.

For example, to validate the correctness of user-role assignment, we created good and bad snapshots. A good scenario articulates a state that a user is assigned a role in a valid zone while in a bad scenario a user is assigned a role in a bad zone. Then, we fed these snapshots to USE and checked whether the tool returns the expected result for each scenario. We have followed this approach to uncover and correct faults in the UML/OCL specification of GSTRBAC. We also utilized the auto-generation mode of USE for automatic snapshot generation [23].

Consider the incremental process that we followed for defining the *activateRole(z)* operation. First *activateRole(z)* is executed in order to generate an instance of *UserRoleActivation* class that links USER, Role, and STZone classes. This *activateRole(z)* operation has two pre-conditions (*pre: self.currentzone*$-$ $>$ *includes(z)*) and (*pre: r.rzones*$-$ $>$ *includes(z)*) that check whether the current user zone and role zone conform the correct activation zone. We instructed the USE tool to execute *avtivateRole(z)* and the result shows that a user in question can activate a role in zone *z* in which the role is available.

Then, we tried to execute the activation operation with a role that is spatio-temporally available but it is not assigned to a user. The activation operation successfully allowed the user to activate unassigned role. As such, the activation operation violated our GSTRBAC model specification that enforces role assignment before allowing users to activate roles. Therefore, we added precondition (*pre: self.getAssignedRoles(z)*$-$ $>$ *includes(r)*) to check role assignment conditions before role activation and re-executed the operation again. As a result, the activation operation fails because the user tries to activate an unassigned role in a valid zone.

Furthermore, the pre-requisite role-activation constraint is also missed in the first place. As such, the *activateRole(z)* operation successfully allowed a user to activate a role regardless to the activation of pre-requisite roles. In consequence, we added the required pre-requisite activation invariant and *activateRole(z)* is tested again. Then, operation *activateRole(z)* failed because the user has not activated the pre-requisite roles. The rest of the OCL expressions are developed following the same incremental process for the role activation operation.

3.3 A UML/OCL Model Validation Approach

The proposed model in Section 3.2 is well-suited for many spatio-temporal policies, however, by itself does not guarantee a system remains secure in all situations. The model supports the specification of various spatio-temporal features that might interact in a subtle way results in inconsistencies and conflicts. A potential flaw in the policy because of an inconsistency or incompleteness in the authorization constraints causes an erratic behavior or exposes the underlying resources to many security breaches. As such, it is important to analyze a policy model before putting it in use.

At application level, a policy state keeps changing over time due to users' and administrators' activities, or due to the environmental changes such as zone updates. A security administrator might add a new entity to a system or a user might assume or drop some roles. A policy state is a particular configuration of entities and relationships in a system. These kind of changes might change a policy state to a valid or invalid state. Thus, a policy designer should ensure the correctness of a security policy under different states before it is widely deployed. A manual analysis of access control policies is tedious and error-prone. A study by Jha et al. [87] concluded that model checking approaches are promising for security analysis. Tool support reduces the probability of human mistakes and enhances analysis performance.

Our second contribution in this dissertation is proposing an automated verification approach of our model. The analysis approach is performed at model-level to ensure a proper specification of model constrains and it does not raise any conflicts or inconsistencies. We also perform analysis at application level to ensure that systems using our model are well protected form security violations. We base our analysis approach on applying the constraint analyzer USE tool [23]. USE is an interactive environment helps us to model and validate a system specified in UML/OCL models. It facilitates the validation of security properties expressed in OCL invariants, pre-, and post- conditions against some test scenarios, i.e object models and snapshots adhering to a UML model. Such test scenarios could be automatically generated by USE tool which makes the verification process is automated.

Thus, following our validation approach, a security verifier is not mandated to generate the test scenarios manually, instead, s/he can impose the USE tool to do that by specifying procedures generating such scenarios. Once USE provides the object diagram and sequence diagram illustrating the operations leading to the system state in question, we can check the validity of some security constraints. Further, we can also evaluate some security queries described in OCL expressions useful for navigating and exploring the system state.

We witnessed from the existing verification approaches that checking of spatio-temporal RBAC policies is not easy to perform. Some of which suffers from a large number of model states due to location and time conditions, and others lacks the verifications of some spatio-temporal properties such as checking access to mobile objects. Furthermore, checking the interaction between model features are not easy to handle following the existing approaches. In our approach, the concept of a spatio-temporal zone streamlines the complexity of analysis by abstracting the environmental information and their impact on model components. Instead of checking location availability and then temporality or the reverse, our approach checks only zone availability which reduces the verification time and space. The follows provides details about our UML/OCL analysis approach. We first give an overview of the USE tool.

3.3.1 USE Constraint Analyzer

UML Specification Environment (USE) [23] is a constraint analyzer tool with the ability of simulating UML models. USE can be used to analyze structural properties of class diagrams. Structural properties are those concerning about the configuration of system's entities at particular point of time (i.e., a snapshot). Unlike behavioral properties that pertain to multiple system's states or sequence of snapshots, structural properties check the attributes of objects, links between objects, and invariants associated with objects. For behavioural properties, USE provides a support for checking operations expressed in OCL. It allows a user to interactively simulate the behavior of an operation by entering command that changes the state of objects. The state at the end of a simulation is checked by USE to determine if the operations' post-conditions hold.

Another advantages of the USE tool, it can automatically generate instances that conform to a class model. Gogolla et al. [23] suggest a scripting language that automates the process of generating instances, which can be used to automatically check a number of instances. USE can also be used to check whether a specific instance conforms to a model or not. This method requires the manual generation of such instances. We use both approaches in our validation approach.

In practice, USE tool is used to validate UML/OCL models as follows. The tool takes two inputs: Class digram model with OCL constraints and Object diagram (model instance). Firstly, a UML class model with OCL constraints that characterize valid application states. This input is represented textually in a file describing different classes, operations, associations, attributes, invariants, pre-conditions and post-conditions. Secondly, an UML object diagram describes a particular state of a system and it is constructed manually by the verifier of the system. Given these two inputs, the tool makes some static analysis and returns whether the object diagram is consistent with the class diagram and OCL constraints. If the object diagram is one possible instantiation of the class model, then the tool returns a positive result. If not, a negative feedback is returned indicating that the object model does not conform to the class model. The tool also shows the constraints that are not satisfied in the object diagram. USE has been extended to support the auto-generation of object diagrams that are in conformance with a class model [23]. In the auto-generation approach, the construction of system states is performed in a declarative manner in which a developer checks the properties that a system is expected to satisfy.

3.3.2 The UML/OCL Verification Framework

GSTRBAC policies are verified using two validation modes as shown in Figure 3.2. The first mode we termed as Use-basic use mode that allows manual creation of a model instance and then verifies that instance against the class model. The second validation mode we termed as Use-auto generation mode that allows the automatic generation of model snapshots and verifies them. The following two sections elaborates how these analysis methods are used in our approach.

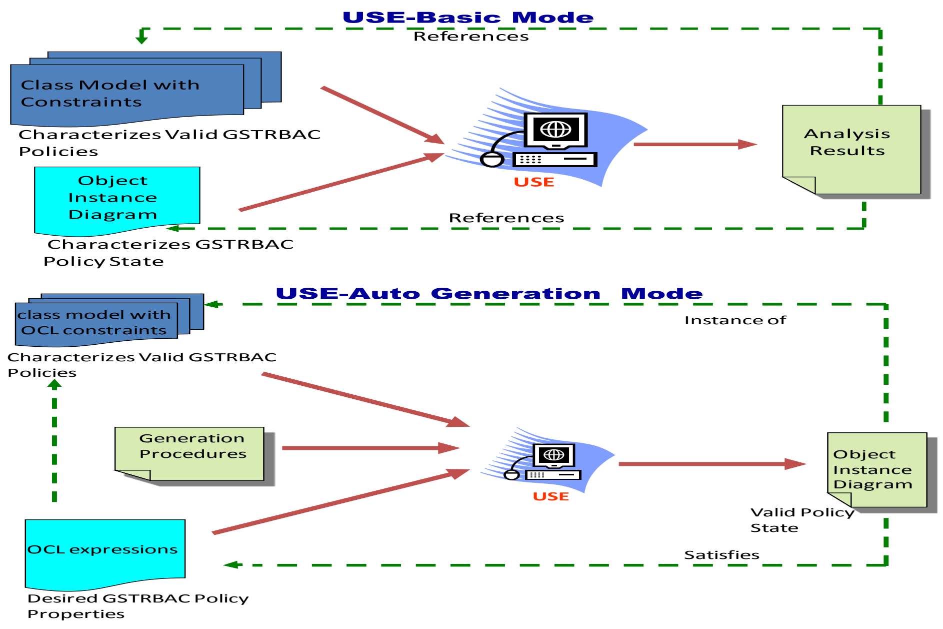


Figure 3.2: The UML/OCL Verification Approach Framework

USE-Basic mode

In the USE-basic mode, a security verify can check a particular policy configuration. A security verifier feeds two inputs to the USE tool: UML class model with its OCL constraints that characterizes a valid GSTRBAC policy, and an object model of a GSTRBAC policy which is generated manually. The GSTRABC class model with the OCL authorization constraints are defined in a textual format as input to the USE tool. The USE tool performs the validation to determine errors in the model instances or in the model design.

Based on a policy verifier understandability of what is valid and invalid, the verifier chooses bad and good GSTRBAC policy states. USE returns whether a policy state conforms to the GSTRBAC class diagram model or not. In case the policy state does not satisfy the model, USE indicates the list of constraints that are violated. Depending on the result, a verifier either increases the correctness of the model or debug the model to find an inconsistency.

However, analyzing the entire policy object model for a particular property is inconvenient and ambiguous because the model has many entities and some of them might not be related to properties in question. Furthermore, we should only focus on interesting states that reveals some problems in the policy and ignore the repeatable or regular states. Therefore, we define the following coverage criterion based on the idea of abstraction and partition.

For each property in our model, we generate some instances covering a possible association of these entities only. For instance, in order to check the correctness of some OCL constraints on user-role assignment, we should only concern about instances of user, role, and zones. As such, we ignore the generation of instances that are irrelevant to the user-role assignment relations such as permissions, objects, and activities. This process supports the concept of separations of concern and also reduces the search space. Instead of analyzing the entire model, a set of sub-object models are derived from the policy object model and verified one-by-one into the USE tool. However, the manual approach is tedious and time consuming. Thus, we develop Algorithm 1 to generate these sub-object models based on each property in the policy.

Algorithm for Generating sub-Object Models:

Algorithm 1 takes the inputs of Access Control Specification (ACS), GSTRBAC-UML, and property *p* that is being verified, and then it produces sub-object model *m*. It first checks whether property *p* is supported by the policy or not; if not, then the algorithm will halt in line 42. Sub-object model *m* is initialized from *GSTRBAC_UML* class model with no instances. Lines 5-16 adds entity's instances to sub-model *m* that are pertained to property *p*. Some dependent properties (p') are evaluated in Lines 21-24 based on the set (*entSet*) derived from property *p*. In case both p' and *p* share some common entities, dependent property p' entities are added to sub-model *m* in lines 25-38. The resulting object model *m* should only include entities and relationships that are dependent on *p*.

Example:

The following example illustrates how Algorithm 1 generates a sub-object model for checking user-role assignment. Consider an access control specification that has the following configuration: $2:4:2:2$, which means 2 users, 4 roles, 2 permissions, and 2 zones. The policy is formally

Algorithm 1: Constructing sub-Object Model for p Property.

```
input  : Access Control Specification (ACS), GSTRBAC-UML Class Diagram, and Property $p$
output: m sub-Object Model
if $p \in ACS$ // ensure that the property $p$ is supported by policy $ACS$.
then
    Let: $m = createObjectModel(GSTRBAC\_UML)$;  // Function that instantiates object model $m$ from class model $GSTRBAC\_UML$ with no
      instances of the classes.
    $entSet = entities(p)$;  // obtain the entities involved in the property $p$
    Initialization: $instanceSet = \phi$;   // initialize the set of instances.
    foreach $ent \in entSet$ // adding property $p$ to $m$ model
    do
        foreach $e \in ent$ // for any element $e$ in the set $ent$
        do
            if $isNotEntityInstance(e, m)$ // if no $e$ instance in the object model $m$.
            then
                Add: $addNewInstance(e, m)$;  // Function adds $e$ instance to the object model$m$.
                Add: $instanceSet = instanceSet \cup \{e\}$;  // add the instance $e$ to the set of instances $instanceSet$.
            end
        end
    end
    $relationInstances = creatRelationInstances(p, instanceSet)$;  // added relationships defined by $p$ to $relationInstances$ set.
    foreach $r \in relationInstances$ do
        Add: $addRelationInstance(r, m)$;  // add new relationship $r$ defined by property $p$ to model $m$
    end
    foreach $p' \in ACS$  $and$  $p \neq p'$ // $p'$ is another property supported by ACS
    do
        $entSet' = entities(p')$;  // extract the entities used by property $p'$
        if $entSet' \subseteq entSet$ // check whether $p'$ is a dependent property on $p$.
        then
            Initialization: $instanceSet' = \phi$;
            foreach $ent' \in entSet'$ // adding property $p'$ to $m$ model
            do
                foreach $e' \in ent'$ // any element $e'$ in the set $ent'$
                do
                    Add: $instanceSet' = instanceSet' \cup \{e'\}$;  // add instance $e'$ to the set $instanceSet'$.
                end
            end
            $relationInstances' = creatRelationInstances(p', instanceSet')$;  // add relationships defined by $p'$ to the $relationInstances'$ set.
            foreach $r' \in relationInstances'$ // adding relation instances defined by $p'$ property to the $m$ model
            do
                Add: $addRelationInstance(r', m)$;  // add relationship $r'$ to the model $m$.
            end
        end
    end
    $return(m)$;
else
    Print: display("The property $p$ is not supported")
end
```

described as following:

- $ACS = \{Users, Roles, Permissions, STZones,$ $UserRoleAssignment, PermissionAssignment,$ $RH, RSSoD, rzones, pzones\}$. Note that many more relationships can be also considered in this ACS.
- $Users = \{u_0, u_1\}$
- $Roles = \{r_0, r_1, r_2, r_3\}$
- $Permissions = \{p_0, p_1\}$
- $STZone = \{z_0, z_1\}$

- $rzones = \{(r_0, z_0), (r_1, z_0), (r_2, z_0), (r_3, z_1)\}$
- $pzones = \{(p_0, z_0), (p_1, z_0), (p_1, z_1)\}$
- $UserRoleAssignment = \{(u_0, r_0, z_0), (u_1, r_1, z_0)\}$
- $PermissionAssignment = \{(r_0, p_0, z_0), (r_1, p_1, z_0)\}$
- $RH = \{(r_0, r_2, z_0)\}$
- $RSSoD = \{(r_0, r_1, z_0)\}$

Algorithm 1 takes the policy sets and relationships in ACS as well as GSTRBAC class model as input and generates sub-object model *ura-Model* as shown in Figure 3.3. The *ura-Model* sub-object model in Figure 3.3 excludes Permission and RH instances because these instances are not dependent on the *UserRoleAssignment* property. The only dependent property on *UserRoleAssignment* is role SSoD. Role SSoD is an important property that should be considered in *ura-Model* to detect any conflicting roles assignment.

In addition, roles that are not assigned to any user are also considered in *ura-Model* to detect the isolated entities. We can observe that role SSoD between r_0 and r_1 is not violated since these roles are assigned in zone z_0 to users u_0 and u_1 respectively. Roles r_2 and r_3 are isolated entities; they are not linked to any User instances. However, the informal observation of the *ura-Model* graph does not guarantee the model is free from errors. Thus, *ura-Model* should be validated against OCL constraints in GSTRBAC class model using the USE tool.

USE-Auto Generation mode

With the USE auto-generation mode, a security verifier can automatically generate some snapshots reflecting the allowed policy changes. The USE tool then does the validation process and returns the analysis results. With this mode, a security verifier investigates possible GSTRBAC policy states that are allowed by a policy model. By observing some of allowed possible states, the verifier can identify errors in the UML class model and OCL constraints. Moreover, the security verifier would be able to investigate possible GSTRBAC policy changes leading to new states.

We write some generation procedures to automatically generate instances of the class model. The output of these procedures are GSTRBAC policy instances that satisfy particular properties of

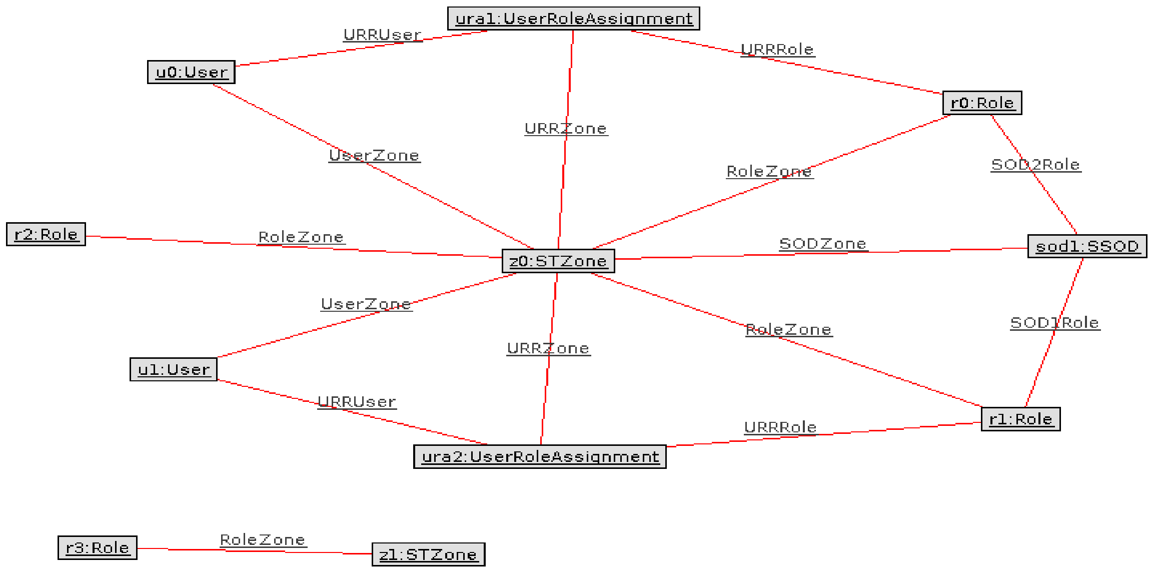


Figure 3.3: Algorithm 1 Output: sub-Object Model *ura-Model*

our interest. In USE tool, the Snapshot Sequence Language (ASSL) aids the construction of system states in the automatic mode. Interested readers can consult reference [23] for more information about ASSL. Consider the definition of the following *generateUsers* and *generateUserRoleAssignment* ASSL procedures:

```
  procedure generateUsers(count:Integer)
  var theUsers:Sequence(User);
  begin
  the Users:=CreateN(User,[count]);
  end;
```

This ASSL procedure creates as many users as indicated by the parameter count. The CreateN expression returns a sequence of newly created users.

```
procedure generateUserRoleAssignment(count:Integer)
var theURAs:Sequence(UserRoleAssignment),
aUser:User, aRole:Role, aZone:STZone;
begin
  theURAs:=CreateN(UserRoleAssignment,[count]);
  for asign: UserRoleAssignment in [theURAs]
  begin
   aUser:=Try([User.allInstances->asSequence->
   select(u| u.relations->isEmpty())]);
   aRole:=Try([Role.allInstances->asSequence->
   select(r| r.relations->isEmpty())]);
```

```
    aZone:=Try([STZone.allInstances->asSequence]);
    Insert(URRUser,[aUser],[asign]);
    Insert(URRRole,[aRole],[asign]);
    Insert(URRZone,[aZone],[asign]);
  end;
end;
```

The above ASSL procedure assigns users to roles in particular zones through the generation of objects of the class *UserRoleAssignment*. The *URAs* command creates *aUser* is of type User, *aRole* is of type Role, and *UserRoleAssignment* objects. The procedure generates as many *UserRoleAssignment* objects as indicated by the *count* parameter. The *Try* command create links between generated entity objects and the *UserRoleAssignment* objects. The following script instructs the USE tool to generate a model snapshot of *UserRoleAssignment* ASSL.

```
use>open GSTRBAC.use
use>gen start GSTRBAC.assl generateUsers(7)
use>gen start GSTRBAC.assl generateRoles(5)
use>gen start GSTRBAC.assl generateZones(2)
use>gen start GSTRBAC.assl generateUserRoleAssignment(4)
use>gen start GSTRBAC.assl generateUserRoleActivation(2)
use>gen result
use>gen result accept
```

We omit some commands of this script that show the results of the snapshot generation process. For example, after the generation of a user instance, The *gen result* and *gen accept* shows the generation result. This script automatically generates 7 users, 5 roles, 4 *UserRoleAssignment*, and 2 *UserRoleActivation* objects as shown in the Figure 3.4. We follow the same approach for the auto-generation of the rest of GSTRBAC entities and relationships

Typically, the result of the policy validation leads to the following consequences: First, the policy might permit undesired states because the constraints are too weak. Second, some acceptable states are rejected because the authorization constraints are strongly overlapped. For the fist case, the constraints are strengthened while the second case the conflicts are resolved. In the following, we briefly elaborate the type of problems that are expected to be uncovered in a GSTRBAC policy. In the subsequent sections, we discuss how they are detected using our analysis approaches

- **Inaccessible Entity:** A user is not connected to a role, a role is not assigned to any user, a permission is not linked to role

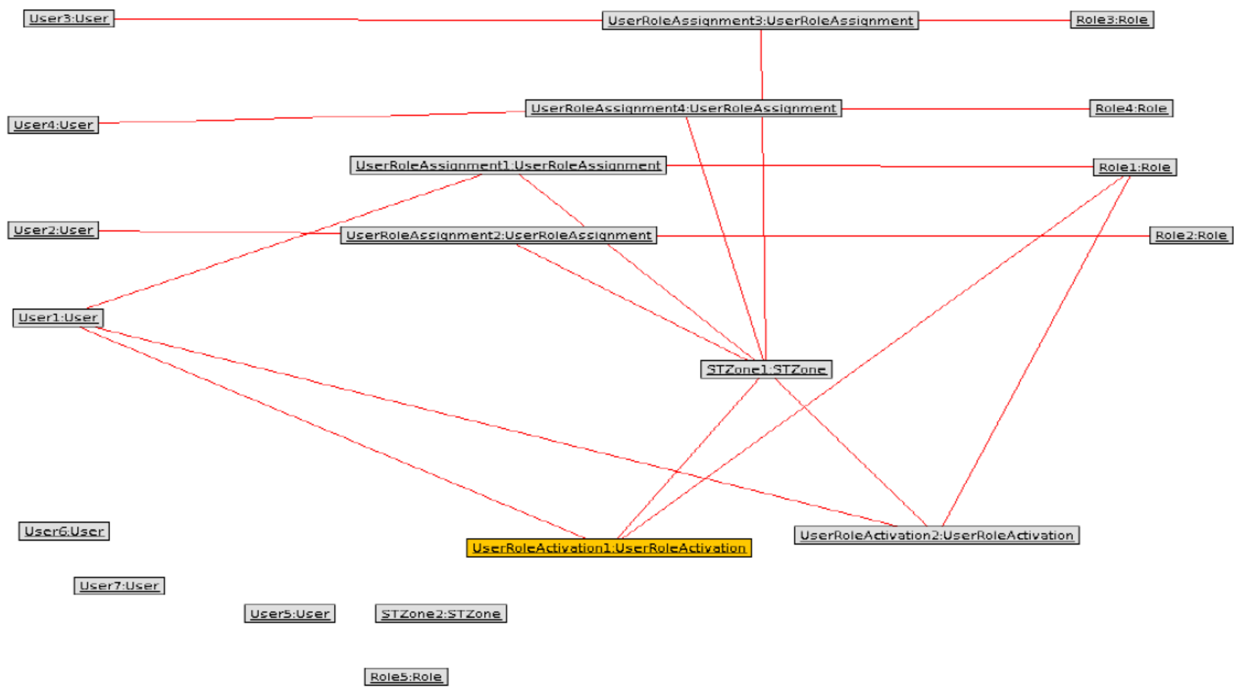

User3:User
UserRoleAssignment3:UserRoleAssignment
Role3:Role
User4:User
UserRoleAssignment4:UserRoleAssignment
Role4:Role
UserRoleAssignment1:UserRoleAssignment
Role1:Role
User2:User
UserRoleAssignment2:UserRoleAssignment
Role2:Role
User1:User
STZone1:STZone
User6:User
UserRoleActivation1:UserRoleActivation
UserRoleActivation2:UserRoleActivation
User7:User
User5:User
STZone2:STZone
Role5:Role

Figure 3.4: Auto Generation Instance of Role Assignment and Activation

- **Missing Constraints:** unauthorized users might access to some roles or permissions because of the missing conditions.
- **Conflicting constraints:** A user might penetrate policy constraints through the benefit from other constraints (e.g access conflicting roles via role hierarchy), it is not easy to uncover.
- **Violating Constraints:** A security violation might occur due to some errors in the policy specification; for example, a user incidentally access to roles from invalid zones, a user access conflicting roles, or conflicting permissions are assigned to the same role

3.4 A Military Application Scenario

This section illustrates how a spatio-temporal policy is represented in GSTRBAC and analyzed following our verification approach. For each security property, a number of instances are generated and checked through the USE tool.

Consider a software production system which is spatio-temporally oriented. This system is utilized to manage the development of a military software project. Software engineers use that system to share the development and maintenance of the project's files. It is a secure system that prevents users from improper access to the project's files. These files are sensitive to exposure to the public or even to some software programmers. Therefore, the software development system implements an access control mechanism to protect against any kind of a tamper or an exposure of those files.

The project's files are stored in computer machines inside a secure building, and the access to those files is location and time dependent. For example, a software engineer can copy project files from the software development office and during the day-time, but the same user is not permitted to do that from home. The access control policy of the department is informally defined as following:

- The department has six users: Bob, Ben, Alice, Rachael, Clare, and Sam
- The department has six job titles: a software engineer, a software programmer, a test engineer, a software supervisor, a test supervisor, and a project leader.
- The software engineer can be as a software programmer or a test engineer.
- The software programmer can read, update and backup files of a project inside the office building and during working hours (8 am to 6 pm). The same software programmer can continue working on the project from home and during the night-time (6 pm, 8 am) , but he is restricted to take backup.
- The test engineer writes test cases for each software project and runs the tests for some project files inside the testing office building and during working hours.
- The test engineer can write test cases from home and during night-time, but he is conditioned to run the test cases.
- The software programmer supervisor is responsible for reading logs of the programs, writing the business requirements, and developing software models in the software development office.
- The test engineer supervisor is in charge of reading, testing, and approving test cases developed by testing engineers as well as reviewing bug history report.
- The project leader reads reports of supervisors, writes the project plan, and approves project development phases only from the director office and during office hours.
- The following defines restrictions on these job functions:
    - The project programmer and test engineer should be essentially a software engineer.
    - The software engineer can work as a test engineer or a programmer in different projects, and from different buildings where these projects are held. However, these two job functions cannot be assumed by the same individual in the same project.
    - The permission to test and write the same program must not be authorized for the same job function.
    - The software engineer supervisor can work as a software programmer and he has a complete knowledge of programmers' logs.

- The test engineer supervisor has access to the rights of test engineers.
- The project leader can only be a director of one project in the same building.
- The project leader can be a supervisor when the supervisor is in a business trip.

3.4.1 Policy Specification

The access control policy of the secure software development system is specified in the GSTRBAC model as follows:

- $ACS = \{Users, Roles, Permissions, Objects, STZones, rzones, pzones, UserRoleAssignment, PermissionRoleAssignment, RH, RSSoD, PSSoD\}$
- $Users = \{Bob, Ben, Alice, Rachael, Clare, Sam\}$
- $TimeIntervals = \{i_1 = (8am, 6pm), i_2 = (6pm, 8am)\}$.
- $Locations = \{Home, DevelopmentOffice, TestingOffice, DirectorOffice, DepartmentBuilding\}$.
- $STZones = \{z_0 = (DepartmentBuilding, i_1), z_1 = (user - home, i_2), z_2 = (DevelopmentOffice, i_1), z_3 = (TestingOffice, i_1), z_4 = (DirectorOffice, i_1)\}$. Zone z_0 represents the containment zone of the zones z_2, z_3, and z_4.
- $Roles = \{SoftwareEngineer(SE), SoftwareProgrammer(SP), TestEngineers(TE), ProgrammerSupervisor(PS), TestSupervisor(TS), ProjectLeader(PL), \}$
- $rzones = \{(SE, z_0), (SE, z_2), (SP, z_1), (SP, z_2), (TE, z_1), (TE, z_3), (PS, z_2), (TS, z_3), (PL, z_4)\}$
- $Objects = \{ProjectFiles(obj_1), TestFiles(obj_2), ProgramersLogs(obj_3), TestLogs(obj_4), ProgrammerSupervisourReport(obj_5), TestSupervisorsReports(obj_6), \}$.
- $ozones = \{(obj_1, < z_1, z_2 >), (obj_2, < z_1, z_3 >), (obj_3, z_2), (obj_4, z_3), (obj_5, z_4), (obj_6, z_4)\}$
- $Activities = \{(a_1 = read), (a_2 = write), (a_3 = copy), (a_4 = run), (a_5 = review)\}$
- $\{Permissions = ReadFile(P_1 = (a_1, obj_1)), WriteFiles(P_2 = (a_2, obj_1)), CopyFiles(P_3 = (a_3, obj_1)), WriteTest(P_4 = (a_2, obj_2)), RunTest(P_5 = (a_4, obj_2)), ReviewTestLog(P_6 = (a_5, obj_4)), ReviewProgramLogs(P_7 = (a_5, obj_3)), ReadSupervisorLogs(P_8 = (a_1, obj_5))\}$
- $pzones = \{(P_1, z_1), (P_1, z_2), (P_2, z_1), (P_2, z_2), (P_3, z_2), (P_4, z_1), (P_4, z_3), (P_5, z_3), (P_6, z_3), (P_7, z_2), (P_8, z_4)\}$
- Model relationships:
  - $UserRoleAssignment = \{(Ben, SP, < z_1, z_2 >), (Bob, PS, z_2), (Alice, PL, z_4), (Clare, TS, z_3), (Rachael, TE, < z_1, z_3 >), (Sam, SE, < z_1, z_2 >)\}$

- $PermissionAssignment = \{(SP, P_1, < z_1, z_2 >), (SP, P_2, < z_1, z_2 >), (SP, P_3, z_2),$
  $(TE, P_4 < z_1, z_3 >), (TE, P_5, z_3), (TS, P_6, z_3), (PS, P_7, z_2), (PL, P_8, z_4)\}$
- $I\_Hierarchy = \{(SP, PS, z_2), (TE, TS, z_3), (PL, PS, z_0),$
  $(PL, TS, z_0)\}$

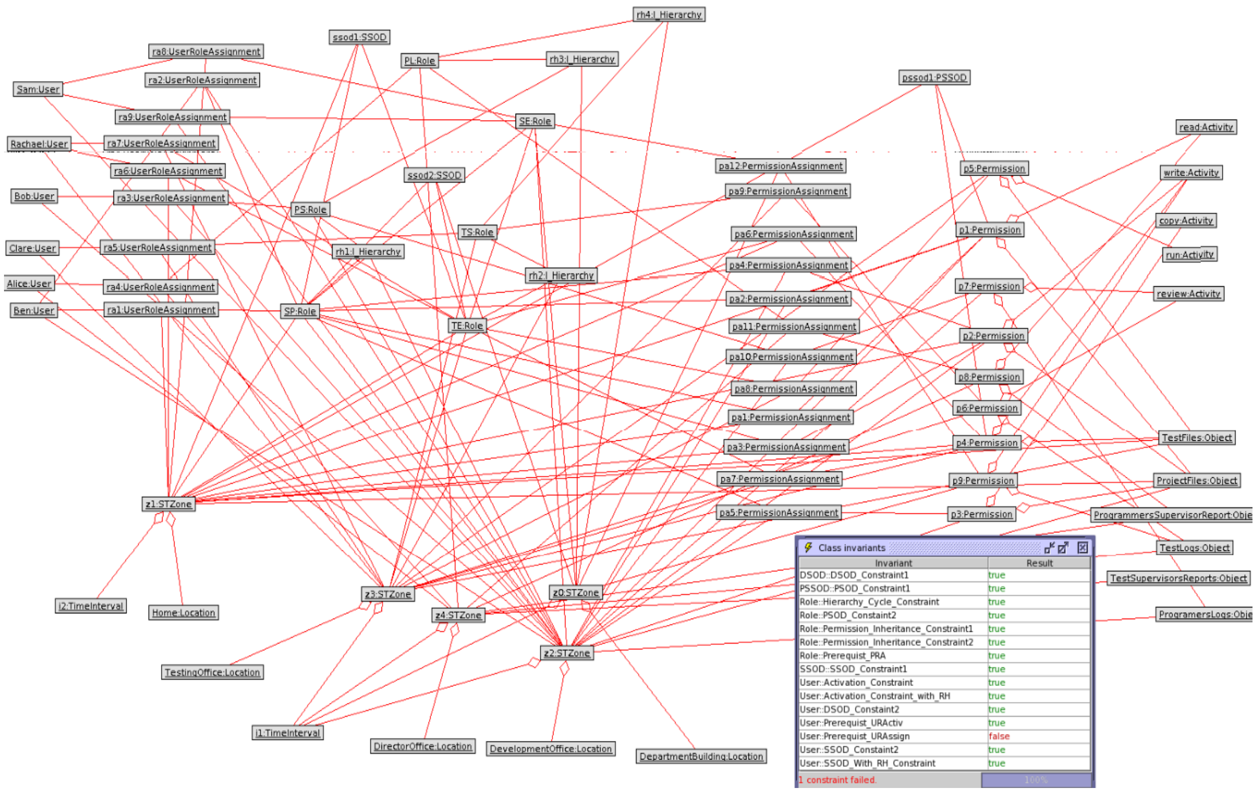
- Constraints:

- $RSSoD = \{(SP, TE, z_0)\}$.

- $PSSoD = \{(P_2, P_4, z_0)\}$.

- $PreqAssRoles = \{(SP, SE, z_0), (TE, SE, z_0)\}$ The software programmer and test engineer roles can only be assigned if a user is already assigned the software engineer role.

Figure 3.5 shows the GSTRBAC policy of our application example. The policy is an instance of the GSTRBAC class diagram. However, a complete visualization of the policy reveals a very crowded object diagram due to the number of entities and relationships.



Invariant	Result
DSOD::DSOD_Constraint1	true
PSSOD::PSOD_Constraint1	true
Role::Hierarchy_Cycle_Constraint	true
Role::PSOD_Constaint2	true
Role::Permission_Inheritance_Constraint1	true
Role::Permission_Inheritance_Constraint2	true
Role::Prerequist_PRA	true
SSOD::SSOD_Constraint1	true
User::Activation_Constraint	true
User::Activation_Constraint_with_RH	true
User::DSOD_Constaint2	true
User::Prerequist_URActiv	true
User::Prerequist_URAssign	false
User::SSOD_Constaint2	true
User::SSOD_With_RH_Constraint	true

1 constraint failed. 100%

Figure 3.5: Software Development Policy Specified in USE Tool

Therefore, the policy is graphically represented in Figure 3.6 using a graph-theoretic notation of RBAC which is introduced by Chen and Crampton [13]. We extend the *access control graph* of RBAC to include the concept of spatio-temporal zones. Since permissions are aggregation of objects and activities, we ignore the representation of objects and activities in Figure 3.6 to avoid the crowding.

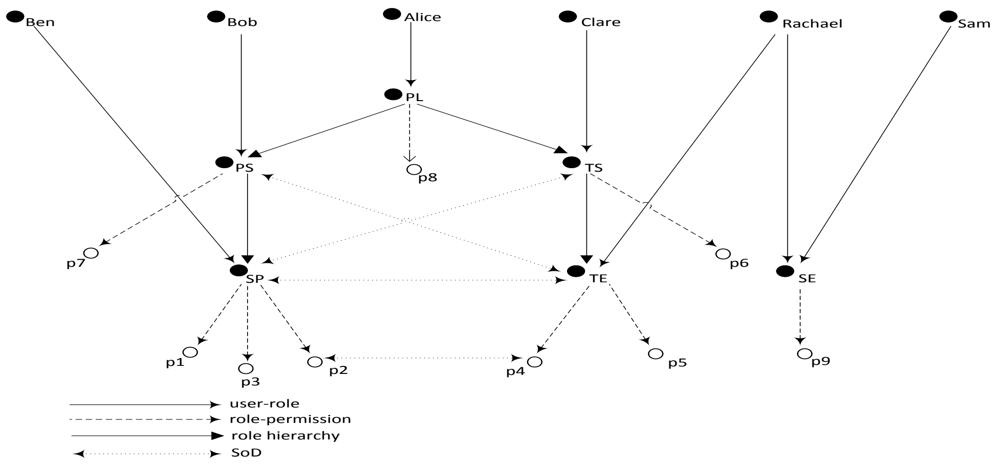


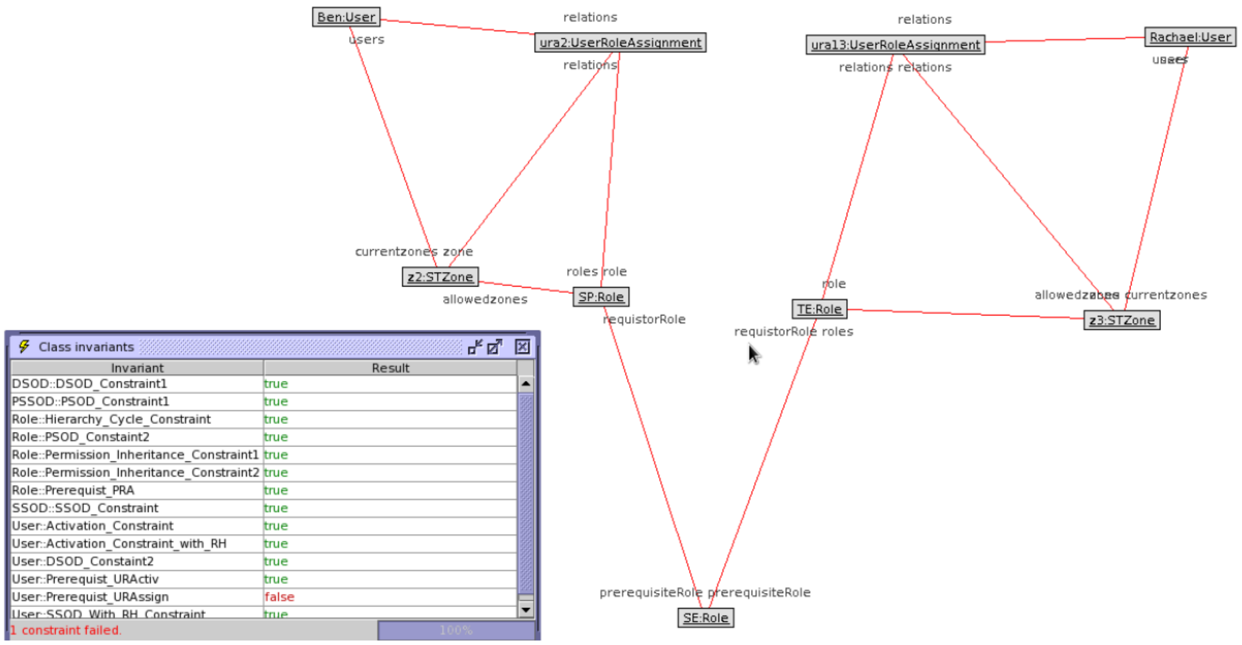
Figure 3.6: Access Control Graph for Software Development System Policy

3.4.2 Policy Verification

In the following, we examine some important security properties in the policy example using our analysis approach. We check the user-object access property which is a harder to validate and it is not considered in many of the existing spatio-temporal analysis approaches. Here, the current policy configuration is analyzed using the USE-basic mode and a number of snapshots are automatically generated and checked through the USE auto-generation mode. The result of the analysis is either confirms or refutes the policy model constraints.

user-role assignment:

Figure 3.5 shows that our policy fails the pre-requisite condition (e.g., *Prerequisite-URAssign* invariant) of the user role assignment. This means that our policy has an error that needs to be corrected. Ben is assigned to *SP*, but not assigned to pre-requisite *SE* role. Referring to the access control graph, Ben is not assigned to *SE* role. This policy error is depicted in Figure 3.7. Note that in Figure 3.7 we have instructed the tool to show the objects and links that we are interested in this property. Irrelevant objects and links that do not pertain to the pre-requisite constraint are ignored.



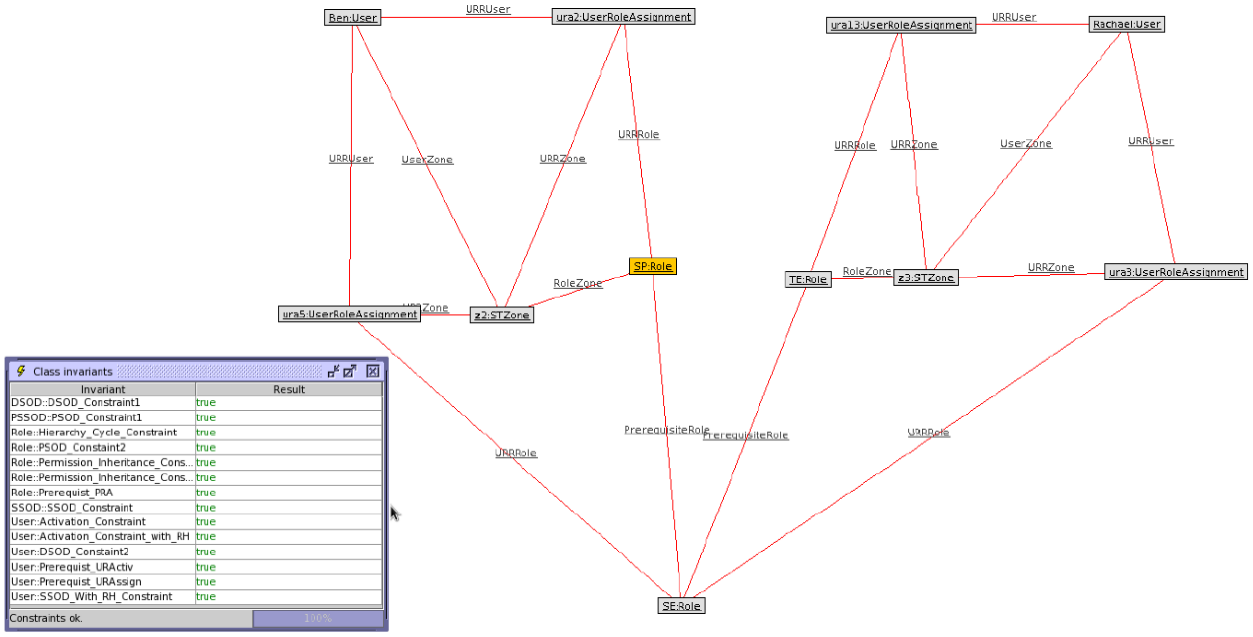
Class invariants

Invariant	Result
DSOD::DSOD_Constraint1	true
PSSOD::PSOD_Constraint1	true
Role::Hierarchy_Cycle_Constraint	true
Role::PSOD_Constaint2	true
Role::Permission_Inheritance_Constraint1	true
Role::Permission_Inheritance_Constraint2	true
Role::Prerequist_PRA	true
SSOD::SSOD_Constraint	true
User::Activation_Constraint	true
User::Activation_Constraint_with_RH	true
User::DSOD_Constaint2	true
User::Prerequist_URActiv	true
User::Prerequist_URAssign	false
User::SSOD_With_RH_Constraint	true

1 constraint failed. 100%

Figure 3.7: Violating Pre-requisite Conditions in User-Role Assignment

We update the policy accordingly and check if it violates this constraint. Now, Ben is assigned to *SE* role and the *Prerequisite-URAssign* constraint is not violated as shown in Figure 3.8.



Class invariants

Invariant	Result
DSOD::DSOD_Constraint1	true
PSSOD::PSOD_Constraint1	true
Role::Hierarchy_Cycle_Constraint	true
Role::PSOD_Constaint2	true
Role::Permission_Inheritance_Cons...	true
Role::Permission_Inheritance_Cons...	true
Role::Prerequist_PRA	true
SSOD::SSOD_Constraint	true
User::Activation_Constraint	true
User::Activation_Constraint_with_RH	true
User::DSOD_Constaint2	true
User::Prerequist_URActiv	true
User::Prerequist_URAssign	true
User::SSOD_With_RH_Constraint	true

Constraints ok. 100%

Figure 3.8: Satisfying Pre-requisite Conditions in User-Role Assignment

Suppose that now a security administrator wants to assign Bob to role *PS* role in zone z_3 = *(TestingOffice, (8am,6pm))* which is an invalid zone. The USE tool detects the error since the assignment operation *assignRole(r:Role,z:STZone)* fails to be invoked as shown in Figure 3.9. The sequence diagram on the right shows that Bob is already assigned to role *PS* in zone z_2. The red arrow in the sequence diagram indicates that the zone pre-condition of the assign operation is not satisfied and hence the operation is not invoked.

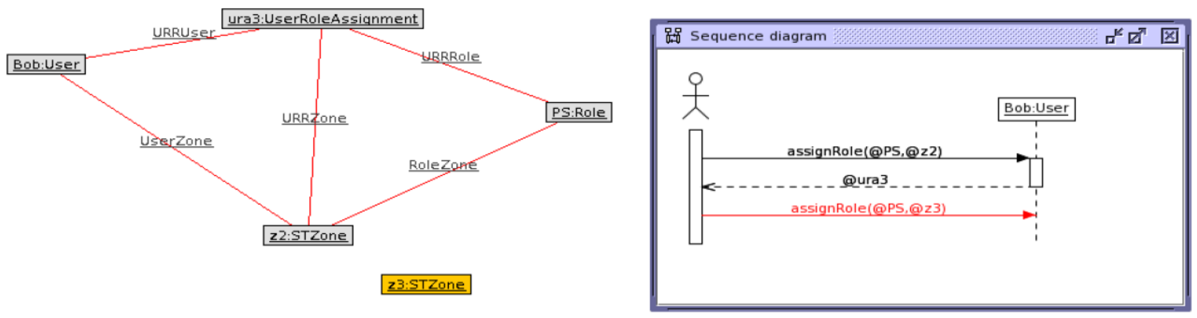


Figure 3.9: Detecting Role Assignment in an Invalid Zone

permission-role assignment:

Suppose a security administrators tries to assign permission p_6 to role *TS* in zone z_3 in which both the role and the permission are available. Therefore, the assignment operation *assignPermission(*p_6*,*z_3*)* is successful. When the administrator invokes the same operation in zone z_2, the operation fails and no assignment is allowed. The USE simulates both scenarios and detects the error as shown in Figure 3.10.

Referring to the policy, Clare can access to permissions of the junior role *TE* in addition to permission of senior role *TS* in zone z_3. Figure 3.11 shows the subset of permissions that Clare can have which are actually a combination of the permissions assigned to *TS* (permission p_6) and to junior role *TE* (permissions p_4 and p_5).

Figure 3.11 shows that Clare is assigned to Role *TS* in zone z_3 (e.g., *Clare.getAssignedRoles(*z_3*)*). Clare is only allowed to activate *TS* role in the same zone (e.g., *Clare.getAuthorizedRoles(*z_3*)*). Role *TS* is assigned to the permission p_6 (e.g *TS.getAssignedPermissions(*z_3*)*) and it is authorized

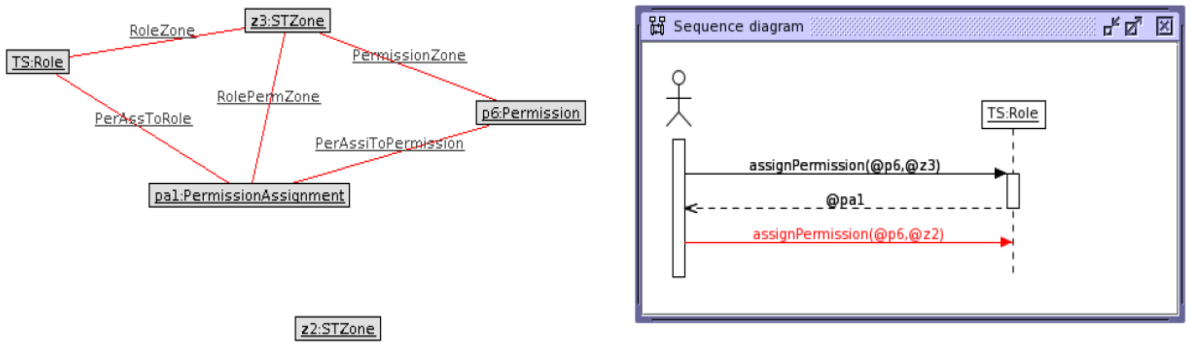
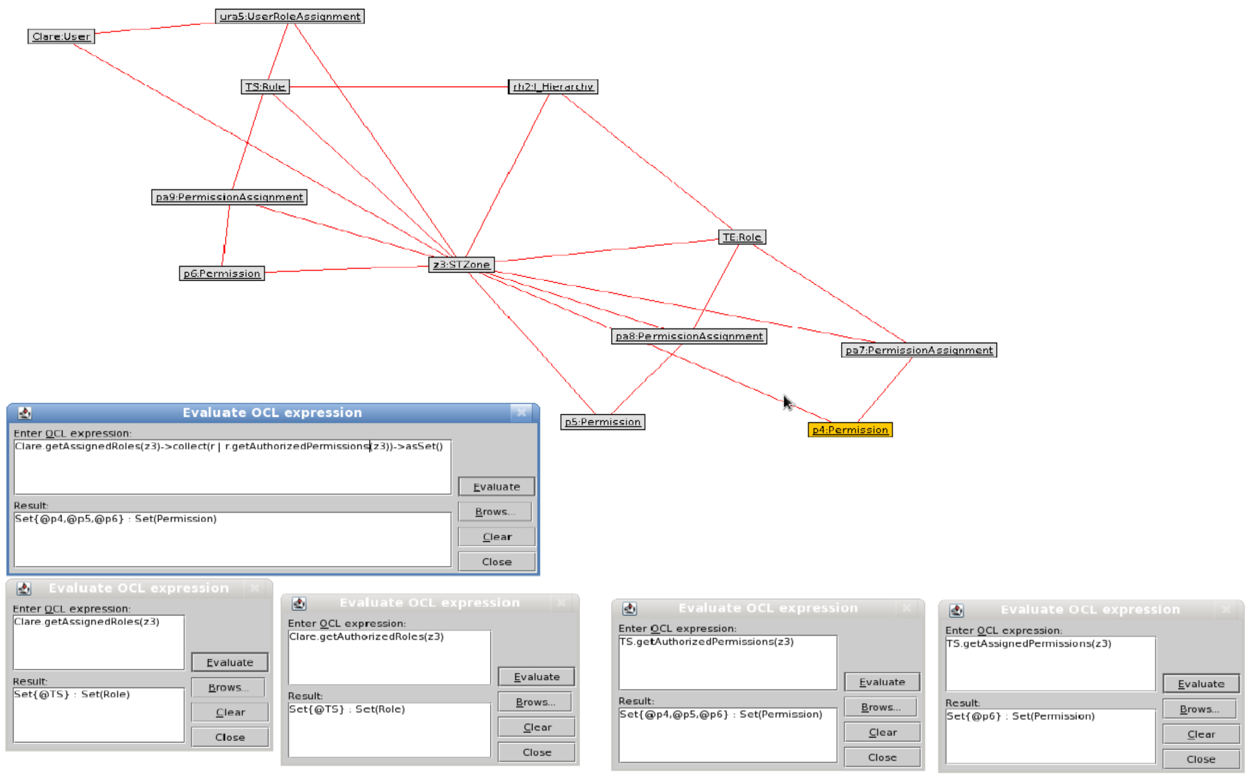


Figure 3.10: Detecting Permission Assignment in an Invalid Zone

to p_4 and p_5 permissions via junior role *TE* (e.g *TS.getAuthorizedPermissions(*z_3*)*).



```
Enter OCL expression:
Clare.getAssignedRoles(z3)->collect(r | r.getAuthorizedPermissions(z3))->asSet()
Result:
Set{@p4,@p5,@p6} : Set(Permission)
```

```
Enter OCL expression:
Clare.getAssignedRoles(z3)
Result:
Set{@TS} : Set(Role)
```

```
Enter OCL expression:
Clare.getAuthorizedRoles(z3)
Result:
Set{@TS} : Set(Role)
```

```
Enter OCL expression:
TS.getAuthorizedPermissions(z3)
Result:
Set{@p4,@p5,@p6} : Set(Permission)
```

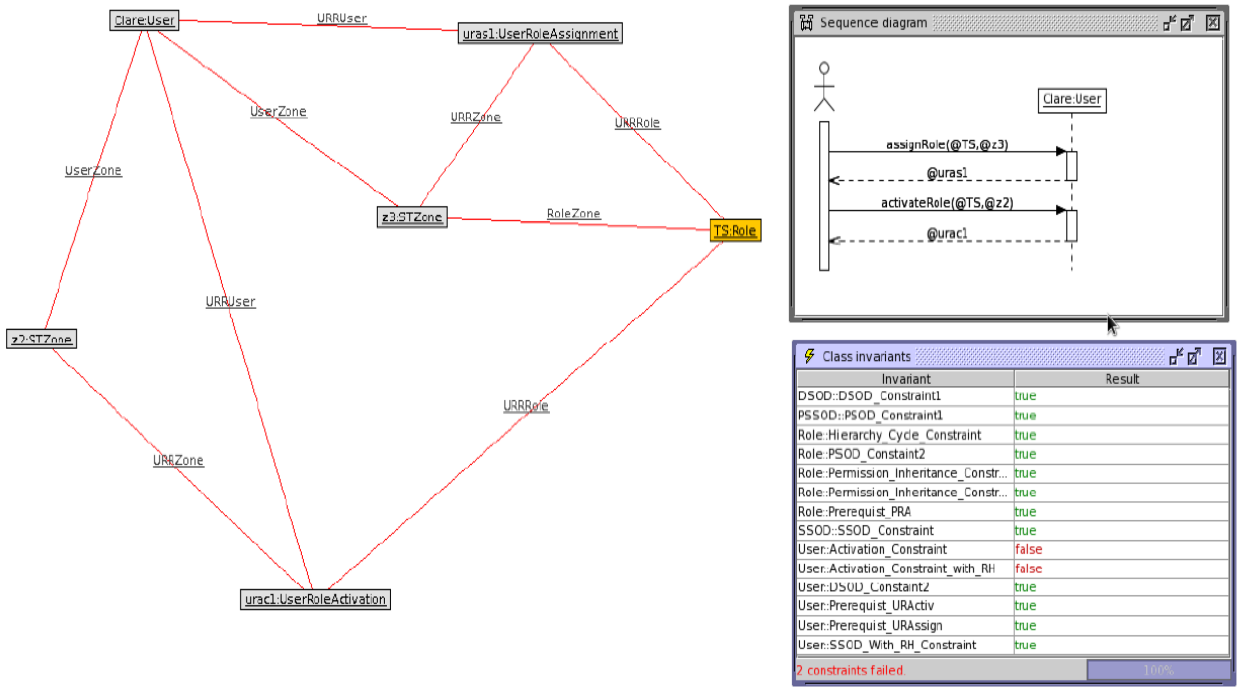
```
Enter OCL expression:
TS.getAssignedPermissions(z3)
Result:
Set{@p6} : Set(Permission)
```

Figure 3.11: User Access to Authorized Permissions

user-role activation:

User role activation operation (*activateRole(r:Role,z:STZone)*) is an example of the user-oriented changes of the policy. A security verifier might instruct the USE tool to automatically generate some user operations and check if the resulting system states violate the policy model. Suppose the USE tool generates a system state in which Clare was able to activate role *TS* in zone z_2 which is not assigned to her in zone z_2. This situation should not happen because Clare is only authorized to *TS* in zone z_3.

In Figure 3.12, the USE tool shows that this activation does not violate the activation constraint. After investigating this undesired state, there was a missing pre-condition that checks the zone availability in the *activateRole(r:Role,z:STZone)* operation. By adding this condition in the activation operation, this undesired state should not reachable anymore.



Invariant	Result
DSOD::DSOD_Constraint1	true
PSSOD::PSOD_Constraint1	true
Role::Hierarchy_Cycle_Constraint	true
Role::PSOD_Constaint2	true
Role::Permission_Inheritance_Constr...	true
Role::Permission_Inheritance_Constr...	true
Role::Prerequist_PRA	true
SSOD::SSOD_Constraint	true
User::Activation_Constraint	false
User::Activation_Constraint_with_RH	false
User::DSOD_Constaint2	true
User::Prerequist_URActiv	true
User::Prerequist_URAssign	true
User::SSOD_With_RH_Constraint	true

2 constraints failed.

Figure 3.12: Detecting The Activation of Unassigned Role

Suppose that Clare deactivates role *TS* and she tries to activate it in zone z_2 over again. The activation operation fails because the pre-condition in the *activateRole(r:Role,z:STZone)* operation is not satisfied. Figure 3.13 shows the scenario where the resulting state is not allowed after updating the missing constraint in the model.

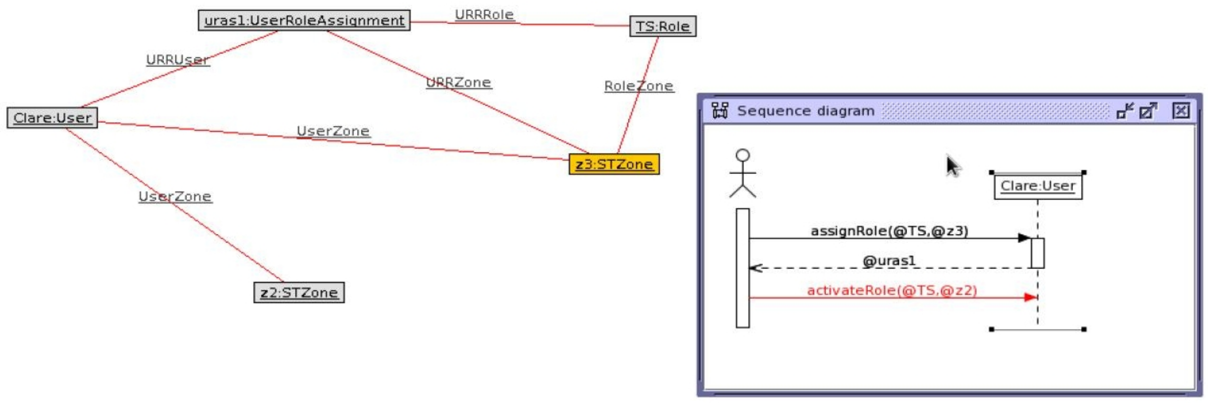


Figure 3.13: Detecting Role Activation in an Invalid Zone

Role SoD:

Suppose the policy administrator instructs the USE tool to generate an assignment relation between Ben and role *TE* in the zone z_1. Note that Ben is already assigned to role *SP* which conflicts with role *TE* in zone z_1. Surprisingly, the assignment was successful and the new system state violates the *RSSoD* constraint.

Investigating this problem, we discover that the assignment operation is missing a pre-condition for checking the conflicting assignment of roles. Thus, the missing pre-condition is added to the assignment operation. Now, Figure 3.14 shows that the assign operation cannot by invoked in case of conflicting roles.

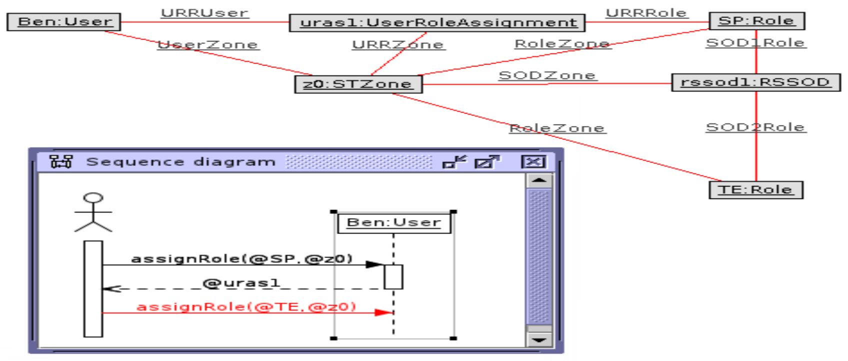
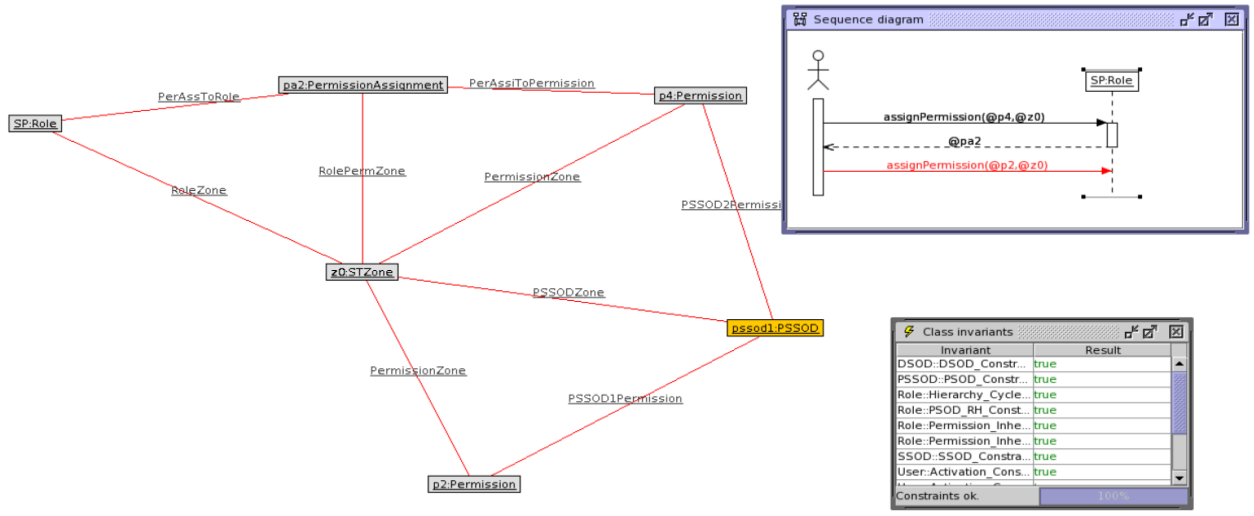


Figure 3.14: Detecting Incorrect Assignment of Conflicting Roles

Permission SSoD:

Suppose an administrator tries to assign permission p_4 to role *SP* in zone z_0. This assignment operation should fail because of the PSSoD between permissions p_4 and p_2 in zone z_0. The *assignPermission(p:Permission,z:STZone)* operation has a pre-condition that checks the existence of conflict between permissions. Consequently, the administrator is not able to assign permission p_4 to role *SP* as shown in Figure 3.15.



Invariant	Result
DSOD::DSOD_Constr...	true
PSSOD::PSOD_Constr...	true
Role::Hierarchy_Cycle...	true
Role::PSOD_RH_Const...	true
Role::Permission_Inhe...	true
Role::Permission_Inhe...	true
SSOD::SSOD_Constra...	true
User::Activation_Cons...	true

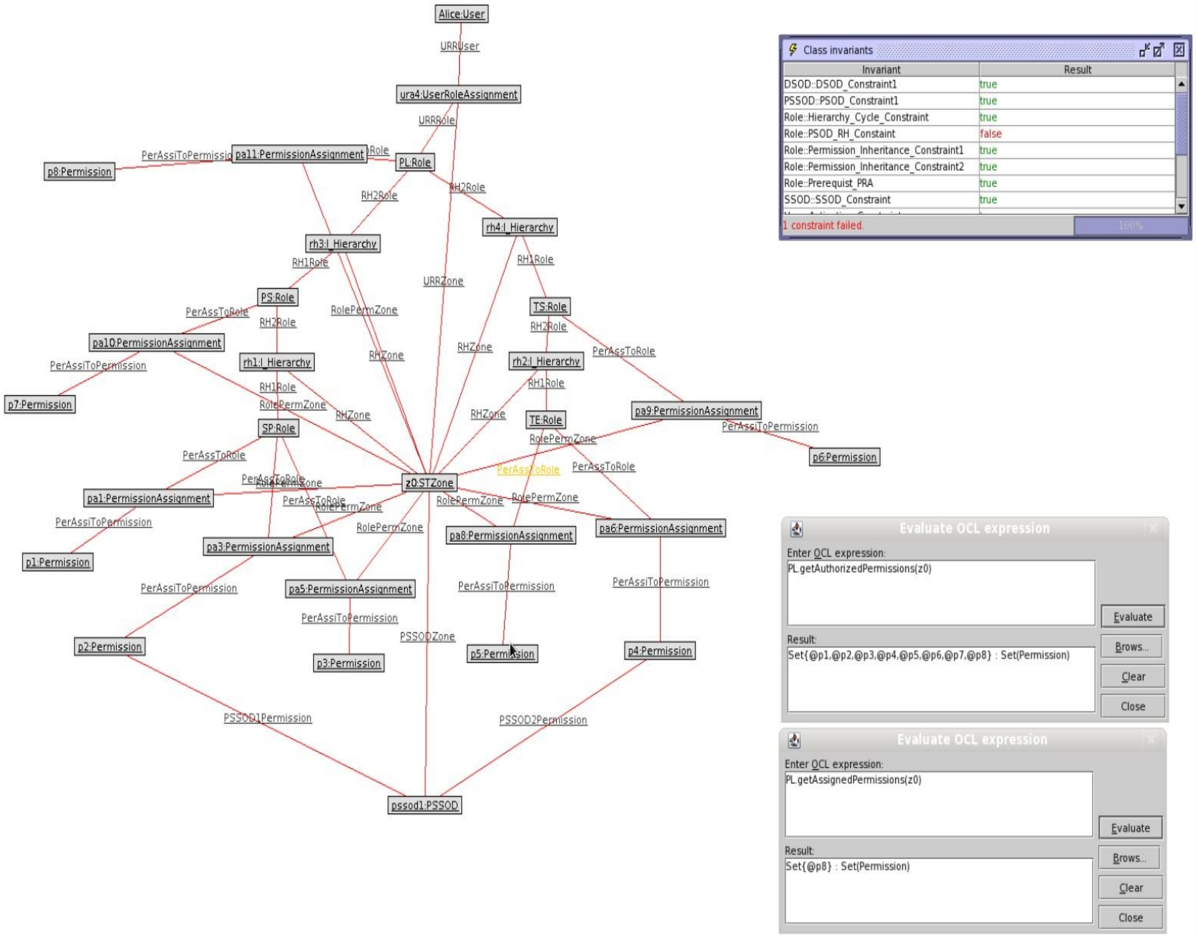
Constraints ok. 100%

Figure 3.15: Detecting Incorrect Assignment To Conflicting Permissions

Role hierarchy:

The access control graph shows that *PL* inherits conflicting junior roles *SP* and *TE* in zone z_0 through multiple levels of hierarchy. The *SP* and *TE* junior roles are respectively assigned conflicting permissions p_2 and p_4. As a result, members of senior role *PL* can assume conflicting roles and commit a fraud.

Our model considers *PSSOD-RH-Constraint* constraint in the presence of multiple level of hierarchy. Figure 3.16 shows that Alice can access to permissions p_2 and p_4 authorized to role *PL* through role hierarchy resulting in *PSSOD-RH-constraint* violation. We consider this violation as a security warning. The security administrator is warned that he is assigning Alice a too powerful role. Then, it is up to the security designer to revise the policy or pay more attention when role PL is assigned.



Class invariants

Invariant	Result
DSOD::DSOD_Constraint1	true
PSSOD::PSOD_Constraint1	true
Role::Hierarchy_Cycle_Constraint	true
Role::PSOD_RH_Constaint	false
Role::Permission_Inheritance_Constraint1	true
Role::Permission_Inheritance_Constraint2	true
Role::Prerequist_PRA	true
SSOD::SSOD_Constraint	true

1 constraint failed.

Evaluate OCL expression

Enter OCL expression:
```
PL.getAuthorizedPermissions(z0)
```
Result:
```
Set{@p1,@p2,@p3,@p4,@p5,@p6,@p7,@p8} : Set(Permission)
```

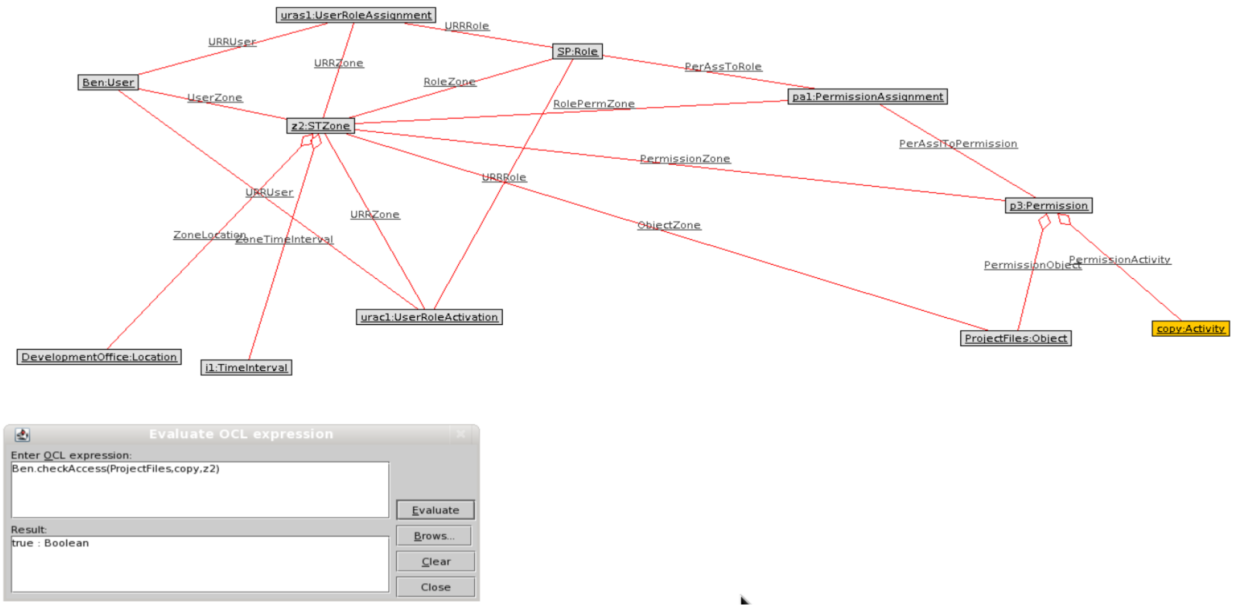
Evaluate OCL expression

Enter OCL expression:
```
PL.getAssignedPermissions(z0)
```
Result:
```
Set{@p8} : Set(Permission)
```

Figure 3.16: Conflicting Between RH and SoD

Checking user access to objects:

Figure 3.17 shows that Ben is allowed to copy project files via permission p_3 in zone z_2. Ben has activated role SP in order to access permission p_3 in zone z_2. The *checkAccess* operation returns true as shown in the OCL expression evaluation box in the bottom of Figure 3.17.



Enter OCL expression:

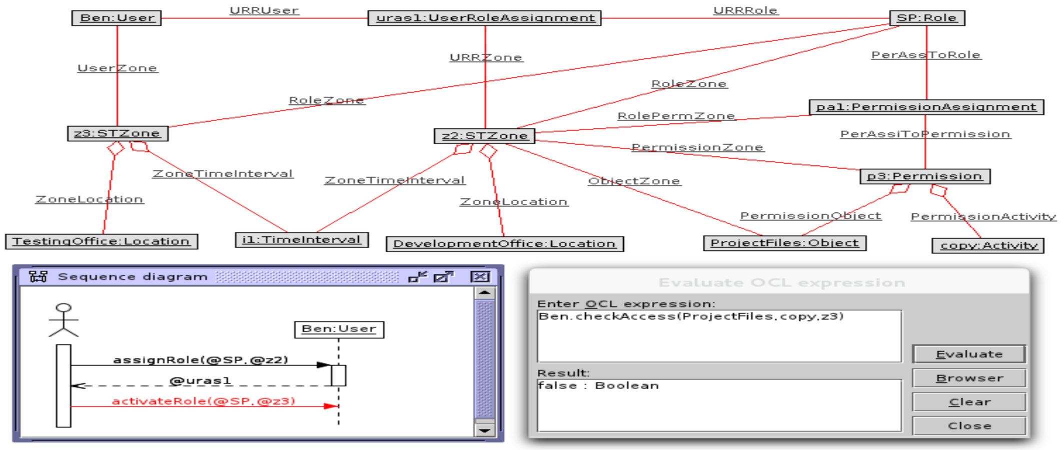
Ben.checkAccess(ProjectFiles,copy,z2)

Result:

true : Boolean

Figure 3.17: Object Access from a Valid Zone

Suppose now Ben moves to zone z_3 = *(TestingOffice, 8am to 6pm)* and tries to copy the *ProjectFiles* object again. However, Ben should not be allowed to access that object because role *SP* is not available for activation in zone z_3. Figure 3.18 shows that the object access constraint is correctly specified in the policy.



Enter OCL expression:

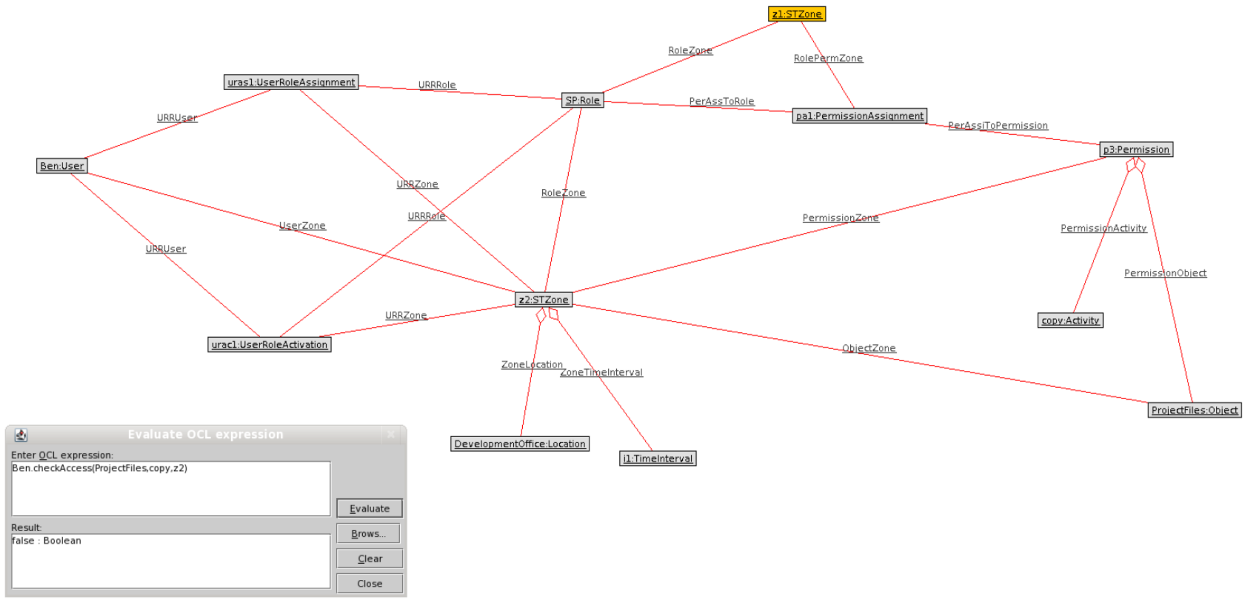
Ben.checkAccess(ProjectFiles,copy,z3)

Result:

false : Boolean

Figure 3.18: Detecting Object Access from an Invalid User Zone

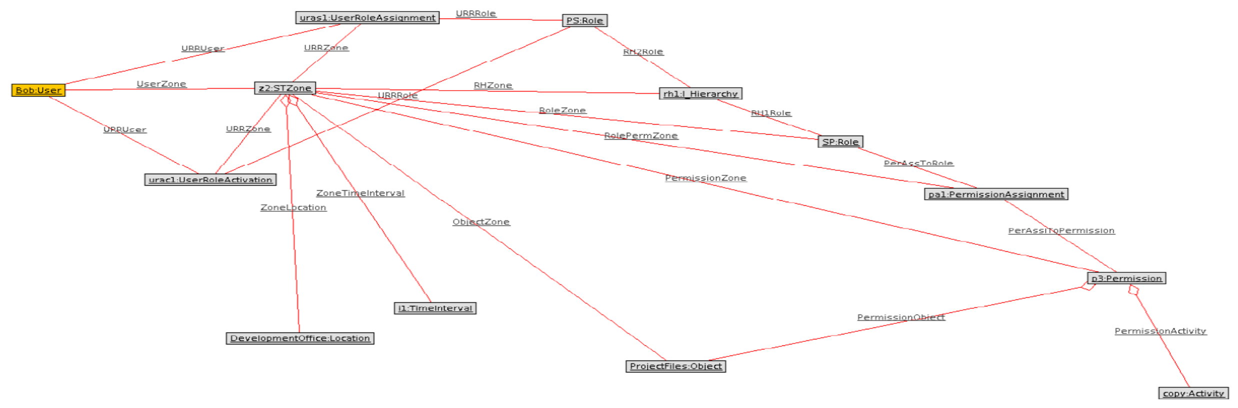
Consider a policy update in which a security administrator assigns permission p_3 to role *SP* in zone z_1. Thus, Ben can activate role *SP* in z_2, but he is not able to access the *ProjectFiles* object anymore because p_3 is not available in zone z_2. Figure 3.19 confirms that Ben is not able to access object *ProjectFiles*.

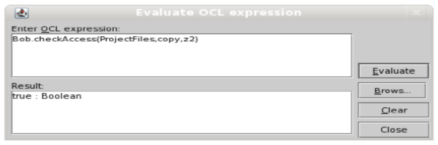


```
Enter OCL expression:
Ben.checkAccess(ProjectFiles,copy,z2)

Result:
false : Boolean
```

Figure 3.19: Detecting Object Access from Incorrect Permission Zone

We would like to check if Bob is able to copy object *ProjectFiles* zone z_2. Bob is a member of senior role *PS* of junior role *SP* in zone z_2, and permission p_3 is assigned to *SP* the same zone. Figure 3.20 shows that Bob can copy object *ProjectFiles* through permission-inheritance hierarchy, which is authorized by the policy.





Enter OCL expression:
Bob.checkAccess(ProjectFiles,copy,z2)
Result:
true : Boolean

Figure 3.20: User Access Objects Through RH

Chapter 4

Model Specification and Verification using Predicate Logic

In this chapter, we present an extended spatio-temporal model that also realizes the promise of RBAC, including simplifying the management of authorizations, a policy neutral, and easy-to-customize for expressing access control policies. This model aims to provide the right information or services to the right individuals in the right places and at the correct time. The formal semantic of the model is defined using predicate logic. The correct behaviour of an application using our model is subject to the approval of model constraints. Here, RBAC entities and relationships are also appended with some spatio-temporal zone entities which control their availability.

This model extends GSTRBAC with post-requisite and triggers features as well as it considers different zone operators at the access time. While performing spatio-temporal access control, the extended model considers various relations between STZones entities (e.g., containment, equality, and overlapping) in a simplified manner. The concept of triggers allows one to enforce *persistent* spatio-temporal access control while access rights to some resources are being used. That is, an access should be terminated at the moment a user moves to an invalid zone. Most of the works on spatio-temporal RBAC perform control before the access; once the access is authorized, there is no control on the correct use of resources.

Additionally, the extended model expresses spatial, temporal, and strong constraints using the zone classes (types). Each of the zone classes can flexible specify a particular domain requirement. Unlike existing spatio-temporal models, the same predicate can express different domain requirements without the need to define a certain predicate or use a specific model for each domain requirement. This feature significantly reduces the number of predicates in our model. As such, this model is referred as a consolidated model since it can support multi-dimensional policy requirements.

Our model is developed to be adopted by non-real-time systems, and also for real-time critical and sensitive systems. Non-real times systems cannot guarantee responses will meet the deadlines, even fast responses are standard. In contrast, in real-time systems, the response to an event, such as termination triggers of a user access, must be guaranteed within strict time constraints. A real-time response is usually in the order of seconds or sometimes milliseconds. An example of real-time constraint in a spatio-temporal policy is the time in which a user must release a role before the role activation deadline, or at the moment the user leaves a certain locations. Missing deadlines in these systems can cause a complete system failure. Therefore, analyzing spatio-temporal features in a continues time model is important to guarantee the information security for hard real-time systems. A correct behaviour of such systems depends on the correct function of real-time constraints at each time tick of a system.

We, therefore, propose a rigorous analysis approach for verifying hard real-time polices while considering spatiality using timed-automata. The timed-automata approach is useful to model and verify real-time properties that cannot be addressed in non-real-time analysis techniques. Examples in our spatio-temporal policy model include the bounded liveness, atomic actions, urgent actions, triggers, and user access termination features.

In this chapter: Section 4.1 shows the formalism of the extended model using predicate logic; a proof-of-concept case study for specifying mobile-based healthcare system DDSS is described in Section 4.4; Section 4.5 introduces a state-space based verification approach using timed-automata; Section 4.6 describes the analysis of the DDSS policy using UPPAAL and also presents the analysis results.

4.1 An Extended Spatio-Temporal Access Control Model

In this model, we use the representation of time, location, and spatio-temporal zone information presented in Chapter 3. The following describes the formalism of the model using first predicate logic.

4.1.1 Entities and Relationships

Spatio-temporal constraints are predicates which applied to model components to determine if their value is acceptable or not. This section describes how different classes of spatio-temporal constraints are defined for the RBAC components.

Users:

The set of users in a system is defined by the *Users* set. The one-to-one function *UserZone* gives the current user spatio-temporal zone.

- $UserZone : (u : Users) \rightarrow STZones$

Roles:

The set of all role identifiers in a system is denoted as *Roles*. The function *Rzones* maps each element in the set *Roles* to a subset from the power set STZones.

- $Rzones : Roles \rightarrow 2^{STZones}$

User-Role Assignment:

This spatio-temporal assignment constraint is formalized in this model using $can\_AssignRole(u,r,z)$ predicate. The predicate $can\_AssignRole(u,r,z)$ holds when the current user zone z is one of the zones in which role r is available. The predicates for expressing temporal, spatial, or strong constraints on user-role assignment can be defined in a similar manner using the temporal, spatial, and universal zones respectively. The user-role assignment is defined in the UA_z relation. Furthermore, the $assigned\_roles(u,z)$ function defines the set of assigned roles to a user, and function $assigned\_users(r,z)$ gives the set of users assigned to a role in a certain zone.

- $can\_AssignRole(u,\ r,\ z) \Rightarrow UserZone(u) = z \wedge\ \exists\ z' \in Rzones(r) \wedge containedZones(z, z')$

- $UA_z \subseteq Users \times Roles \times STZones$, where all the relation elements satisfies the zone condition. Formally, $\forall(r,u,z) \in UA_z$ the predicate $can\_AssignRole(u,\ r,\ z)$ must be true.

- $assigned\_roles :\ (u : Users\ ,\ z : STZones)\ \rightarrow 2^{Roles}$, this maps the user u in zone z to subset of roles. Formally, $assigned\_roles(u,z) = \{r \in Roles \mid (u,r,z) \in UA_z\}$

- $assigned\_userss : \ (r : Roless \ , \ z : STZones) \ \rightarrow 2^{Users}$, this maps role r in zone z to subset of users. Formally, $assigned\_userss(r,z) = \{u \in Users \mid (u,r,z) \in UA_z\}$

Role Enabling:

Once a role is enabled in a spatio-temporal zone, the role can be activated. The predicate $can\_EnableRole(r,z)$ specifies the spatio-temporal role enabling constraints. It checks whether role r can be enabled in zone z. This predicate is true if and only if zone z is one of the valid zones attached to role r.

The temporal, spatial, and strong role enabling constraints can be expressed in a similar manner using the temporal, spatial, and universal zones respectively. The function $enabled\_roles(z)$ specifies the current enabled roles in zone z.

- $can\_EnableRole(r,z) \Rightarrow \exists \ \ z^{'} \in Rzones(r) \ \wedge \ containedZones(z,z^{'})$

- $enabled\_roles : (z : STZones) \times 2^{Roles}$, maps a zone to a set of roles that are in enabled state. This is formally written, $enabled\_roles(z) = \{r \in Roles \mid can\_EnableRole(r,z)\}$

Whenever a role is disabled in a zone, that role cannot be used to acquire the permissions associated with it. $disabled\_role(z)$ gives the set of roles that are disabled in zone z and should not be accessed by its members.

- $disabled\_roles : (z : STZones) \times 2^{Roles}$, maps a zone to a set of roles that cannot be accessed. Formally, $disabled\_roles(z) = \{r \in Roles \mid \neg can\_EnableRole(r,z)\}$

User-Role Activation:

In order for a user to activate a role, the predicate $can\_ActivateRole(u,r,z)$ should be satisfied. This predicate holds if and only if the current user zone is one of the zones in which role r is in enabled state. Additionally, role r should be assigned to user u in zone z. The temporal, spatial, and strong role activation constraints can be defined in a similar manner using temporal, spatial, and universal zones, respectively.

- $can\_ActivateRole(u,r,z) \Rightarrow r \in enabled\_roles(z) \ \ \wedge \ \ r \in assigned\_roles(u,z)$

- $active\_roles : (u : Users\ ,\ z : STZones) \rightarrow 2^{Roles}$, maps a user u in zone z to a subset of active roles. Formally, $active\_roles(u,z) = \{r \in Roles \mid can\_ActivateRole(u,r,z)\}$

Objects:

The *Objects* set defines the objects in a system. The function $Objzones(obj)$ associates each object in the set *Objects* with a subset of spatio-temporal zones.

- $Objzones : Objects \rightarrow 2^{STZones}$

Operations:

A single operation can be performed on one or more objects. The set of all operations in a system is referred to *Operations*.

Permissions:

The set of all permissions in a system is defined by the *Permissions* set. Permissions are categorized into human and agent permissions corresponding to human and agent roles. Some spatio-temporal zones are also associated with permissions. The function *Pzones* determines the subset of the power set STZones in which a permission can be accessed.

- $Pzones : Permissions \rightarrow 2^{STZones}$

Each permission is a cross product between the set of objects and operations inside some zones. The following functions define the allowed objects for a permission in some, and set of permissions that can be preformed on an object, set of operations associated with an object, respectively.

- $Permissions = 2^{(Operations \times Objects \times STZones)}$, the set of permissions.

- $permObjs : (p : Permissions\ ,\ z : STZones) \rightarrow 2^{Objects}$ maps each permission to set of objects in particular zone. Formally, $permObjs(p,z) = \{obj \in Objects \mid \exists\ z' \in Objzones(obj) \wedge containedZones(z,z')\}$

- $objectPerms : (obj : Objects, z : STZones) \rightarrow 2^{Permissions}$

- $permOpr : (p : Permissions\ ,\ z : STZones) \rightarrow 2^{Operations}$

Permission-Role Assignment:

The predicate $can\_AssignPerm(r,p,z)$ expresses the spatio-temporal constraints on permission-role assignment. This predicate is true when zone z of permissions p is one of the zones in which role r is available. The temporal, spatial, and strong constraints on permission-role assignments can be expressed in a similar manner through temporal, spatial, and universal zones respectively. The permission-role assignment is a many-to-many relation defined by PA_z. The function $assigned\_perms(r,z)$ determines all the explicitly assigned permissions to role r in zone z, and function $used\_roles(p,z)$ determines the set of roles using a permission within a specific zone.

- $can\_AssignPerm(r,p,z) \Rightarrow z \in Rzones(r) \ \wedge \ \exists \ z' \in Pzones(p) \ \wedge \ containedZones(z,z')$
- $PA_z \subseteq Roles \times Permissions \times STZones$
  , so that $\forall(r,p,z) \in PA_z$ the predicate $can\_AssignPerm(r,p,z)$ should be true.
- $assigned\_perms : (r : Roles \ , \ z : STZones) \rightarrow 2^{Permissions}$, maps a role r in zone z to a set of permissions. Formally: $assigned\_perms(r,z) = \{p \in Permissions \mid can\_AssignPerm(r,p,z)\}$

User-Permission Authorization:

The predicate $can\_AuthorizePerm(u,p,z)$ evaluates whether user u is authorized to permission p in zone z.

- $can\_AuthorizePerm(u,p,z) \Rightarrow \exists \ r \in active\_roles(u,z) \ \wedge \ p \in assigned\_perms(r,z)$
- $user\_authorized\_perm : (u : User \ , \ z : STZones) \rightarrow 2^{Permissions}$, it maps user u in zone z to a set of permissions. This is formally written $user\_authorized\_perms(u,z) = \bigcup_{r \in active\_roles(u,z)} \{p \in Permissions \mid p \in assigned\_perms(r,z)\}$

4.1.2 Role Hierarchy

The following shows the formalisms for role hierarchy RH, the permission-inheritance hierarchy RH_I, and role-activation hierarchy RH_A. The functions $juniorRoles_I(r,z)$ and $juniorRoles_A(r,z)$ respectively give the junior roles of role r in zone z in RH_I and RH_A .

- $RH \subseteq Roles \times Roles \times STZones$

- $RH_I \subseteq RH$, $RH_A \subseteq RH$, and $RH_I \cap RH_A = \phi$

- $juniorRoles_I : (r : Roles, z : STZone) \rightarrow 2^{Roles}$, it gives a subset of junior roles in zone z. Formally, $juniorRoles_I(r,z) = \{r' \in Roles \mid (r,r',z) \in RH_I\}$

- $juniorRoles_A : (r : Roles, z : STZone) \rightarrow 2^{Roles}$, return a subset of junior activation roles in zone z, so that $juniorRoles_A(r,z) = \{r' \in Roles \mid (r,r',z) \in RH_A\}$

Now, different forms of permission-inheritance and role-activation hierarchies are defined based on the zone concept.

Permission-Inheritance Hierarchy (RH_I):
In this model, the spatio-temporal RH_I is written as $\succeq_{I,z}$, where role r inherits r' role in zone z is denoted by $r \succeq_{I,z} r'$, if and only if all permissions of role r' are permissions of role r. If roles r and r' are not related by RH_I in zone z, it means there is no dominance relation between these two roles in zone z, i.e., $r \not\succeq_{I,z} r'$. The function $authorized\_perms(r,z)$ returns a subset of permissions that are either assigned to role r or inherited by role r in zone z.

- $(r \succeq_{I,z} r') \Rightarrow z \in Rzones(r) \ \wedge \ \exists z' \in Rzones(r') \ \wedge \ containedZones(z,z')$

- $authorized\_perms : (r : Roles \ , \ z : STZones) \rightarrow 2^{Permissions}$, the mapping of role r in zone z onto a set of permissions in the presence of RH_I hierarchy. This is formally written, $authorized\_perms(r,z) = \{p \in Permissions \mid (p \in assigned\_perms(r,z)) \vee [\ r' \in juniorRoles_I(r,z)\ ,\ \ p \in authorized\_perms(r',z)\ ]\}$.

Now, all permissions authorized to an active role, either assigned or inherited, are authorized to all the role's members. The user-authorized permissions is revised as following:

- $can\_AuthorizePerm(u,p,z) \Rightarrow \exists \ r \in active\_roles(u,z) \ \wedge \ p \in authorized\_perms(r,z)$

Sometimes a policy allows a senior role to inherit permissions from junior roles within a particular time interval regardless of the locations implication. For example, a policy of a health care system may allow a doctor on duty to inherit permissions of a nurse on duty during critical situations. Like spatio-temporal permission-inheritance hierarchy, the temporal permission-inheritance hierarchy can be simply expressed in the temporal zones.

Furthermore, location permission-inheritance hierarchy might be needed in a policy to allow permissions to be inherited in some specific locations despite the consequences of time. For example, an account auditor can inherit permissions of the accountant only in the bank. Such class of permissions-inheritance hierarchy can be expressed via the spatial zones. The strong permissions-inheritance can also be expressed in the same way in the universal zones.

Role-activation Hierarchy (RH_A):

The spatio-temporal role-activation hierarchy is referred to $\succeq_{A,z}$. Roles r and $r^{'}$ are related by RH_A in zone z if and only if $r \succeq_{A,z} r^{'}$. The function $user\_autorized\_roles(u,z)$ determines a subset of roles that are authorized to user u in zone z via either user-role assignment or RH_A.

- $r \succeq_{A,z} r^{'} \;\Rightarrow\; z \in Rzones(r) \;\wedge\; \exists\; z^{'} \in Rzones(r^{'}) \;\wedge\; containedZones(z,z^{'})$

- $user\_autorized\_roles : (u : Users\;,\; z : STZones) \;\rightarrow\; 2^{Roles}$, maps user u in zone z to subset of authorized roles that can be activated. We formally define this as, $user\_autorized\_roles(u,z) = \{r \in Roles \mid (r \in assigned\_roles(u,z)) \vee [r \;\in\; juniorRoles_A(r^{'},z),\; r^{'} \in active\_roles(u,z)\,]\}$.

Based on the RH_A relation, the user activation predicate is revised :

- $can\_ActivateRole(u,r,z) \Rightarrow r \in enabled\_roles(z) \;\wedge\; r \in user\_autorized\_roles(u,z)$

Timed RH_A authorizes members of a senior role to activate junior roles for a certain time interval. For example, an instructor can activate the role of teaching during the lab time. The time RH_A can be expressed using timed zones in a similar manner to the spatio-temporal RH_A. Location RH_A allow members of an activate senior role to activate junior roles only in certain locations regardless to the temporal implication. For example, members of the specialist physician role can activate the primary care physician role in the emergency room whenever needed. The location RH_A is constrained by spatial zones in the same way to spatio-temporal RH_A

4.2 Model Constraints

4.2.1 Separation of Duties (SoD)

The relation $SSoD_r$ determines the set of conflicting roles that should not be assigned to the same individual in some spatio-temporal zones. In a similar way to $SSoD_r$, the relation $SSoD_p$ defines the conflicting permissions that are not allowed to be assumed by the same role in some spatio-temporal zones. The conflicting activation roles in undesirable spatio-temporal zones are defined by the $DSoD_r$ relation. The functions $ssod\_Roles(r,z)$ and $dsod\_Roles(r^{'},z)$ determine the conflicting roles in zone z with role r and $r^{'}$ respectively, and $ssod\_Perm(p,z)$ determines the conflicting permissions with permission p in zone z.

- $SSoD_r \subseteq Roles \times Roles \times STZones$
- $DSoD_r \subseteq Roles \times Roles \times STZones$, and $SSoD_r \cap DSoD_r = \phi$
- $SSoD_p \subseteq Permissions \times Permissions \times STZones$
- $ssod\_Roles : \ (r : Roles, z : Stzone) \ \rightarrow \ 2^{Roles}$
- $dsod\_Roles : \ (r : Roles, z : Stzone) \ \rightarrow \ 2^{Roles}$
- $ssod\_Perms : \ (p : Permissions, z : Stzone) \ \rightarrow \ 2^{permissions}$

In the following, we describe the formalism of the various forms of SoD constraints in this model.

Role SSoD:

The conflict between roles in some spatio-temporal zones is defined by the relation $SSoD_r$.

- $\forall u \in Users \bullet (r, r^{'}, z) \in SSoD_r \ \Rightarrow \ \neg(\ r \in assigned\_roles(u,z) \ \bigwedge \ r^{'} \in assigned\_roles(u,z) \ )$

In some applications, the same user cannot be assigned to conflicting roles for a specific duration. For example, in an university, an individual can be a chairman of more than one department, but not at the same time interval. This temporal $SSoD_r$ requirement can be enforced in a similar semantic to spatio-temporal $SSoD_r$ by using temporal zones. The location $SSoD_r$ requires that the conflict between roles only takes place in a certain physical locations. For example, a bank security policy reduces the possibility of committing a fraud by prohibiting the same individual from being assigned to auditor and teller roles in the teller room. Similar to spatio-temporal $SSoD_r$, the spatial zones can be used to define location $SSoD_r$.

Permission SSoD:

The mutual exclusive relation between permissions is defined by $SSoD_p$ as following.

- $\forall r \in Roles \bullet (p, p', z) \in SSoD_p \Rightarrow \neg(p \in authorized\_perms(r, z) \bigwedge p \in authorized\_perms(r, z))$

Timed $SSoD_p$ limits the distribution of powerful permissions only for a certain period of time. For example, the same software engineer programmer should not be given the permissions to write a code and test the same code at the same time. This requirement can be enforced by temporal $SSoD_p$ using the temporal zones in the same way of the spatio-temporal $SSoD_p$. The spatial $SSoD_p$ prohibits the same role from having conflicting permissions in some locations. For example, a doctor assistant cannot be authorized to write a patient report and approve it in the same ward; the report should be signed by a specialist doctor. The spatial $SSoD_p$ can be expressed using the spatial zones.

Role DSoD:

In this model, a pair of mutual exclusive roles related by $DSoD_r$ in a spatio-temporal zone are not allowed from being activated by the same user.

- $\forall u \in Users \bullet (r, r', z) \in DSoD_r \wedge r, r' \in authorized\_roles(u, z) \Rightarrow \neg(r \in active\_roles(u, z) \bigwedge r' \in active\_roles(u, z))$

Temporal $DSoD_r$ prevents the temporally conflicted roles from being activated by the same individual at some time intervals. A motivating example of this principle is that a doctor can activate either a day-time or night-time doctor roles. Temporal $DSoD_r$ can be simply expressed in our model through the association of temporal zones with the conflicting roles in $SDoD_r$. Spatial

$DSoD_r$ emphasizes that conflicting roles cannot be activated by the same individual in some locations. An intersecting example of this property is that for many companies, it is not acceptable to authorize an individual to be acting the roles of production engineer and quality engineer in the same engineering department. In our model, spatially conflicting roles are associated with spatial zones in $DSoD_r$.

4.2.2 Pre-requisite, Post-requisite, and Trigger Constraints

Some applications might require the automatic execution of certain actions due to the occurrence of some events which necessitate the pre-requisite, post-requisite, trigger constraints among actions. The spatio-temporal pre-requisite constraint enforces that a role should be in a certain state before allowing an action to be taken on another role within specific time and location. We refer to the former role as a *pre-requisite role* and the second role as a *succeeding role*.

The spatio-temporal role post-requisite constraint requires a role to remain in an active state in a certain spatio-temporal zone as long as some related roles are active. An example where this property is practical is that the role of doctor-on-training in a surgery room during a surgical operation can be activated if and only if the role of senior doctor-on-duty is already activated in the same room during the same surgical time. This means that the trainee doctor can only practice in presence of a senior doctor. Thus, the senior doctor is forbidden to deactivate his/her role in the surgery room during the surgical operation time if there is an active trainee doctor is involved in that surgical operation.

Therefore, the spatio-temporal post-requisite constraints between roles are more restrictive form than the spatio-temporal pre-requisite constraints where disabling of the pre-requisite roles are not restricted. We refer to the doctor-on-training role as *post-requisite* role and the senior doctor role as *primary role*.

The spatio-temporal role triggers are another important constraint referring to the automatic execution of certain actions because of the occurrence of an event such as enabling a role. In the event of role enabling, another role should be enabled in a particular location and for a certain duration. The former role is referred to as *stimuli role* and the later role is referred to as *triggered role*. An example of this property is that enabling the role of surgeon in a surgery room triggers the role of anaesthesiologist to be enabled in that room during the surgery. The following discusses some forms of spatio-temporal role pre-requisite, post-requisite and trigger constraints in our model.

Pre-requisite Constraints

Pre-requisite on Role-Enabling:

An access control policy might require that a role can be enabled if and only if other roles are already enabled in a certain location and time. For example, in a retail store, a cashier supervisor reviews a cashier drawer before a designated cashier start to work. Thus, a role of cashier is enabled if and only if the role of cashier supervisor is already enabled in the same store and at the same time. To capture this requirement in our model, we define the function $pre\_EnableRoles(r,z)$ which retrieves the set of roles that should be already enabled in order to enable role r in zone z. It is important to note that the disabling of any of the *pre-requisite roles* does not mean that role r should be disabled. The role enabling pre-requisite in temporal and spatial domains can be formalized using the same function.

- $\forall (r', z') \in pre\_EnableRoles(r,z) \bullet can\_EnableRole(r,z) \Rightarrow r' \in enabled\_roles(z')$

Pre-requisite on Role-Disabling:

Sometimes before a role can be disabled, other critical roles must be disabled. $pre\_DisableRoles(r,z)$ defines the set of roles that must be disabled in certain zones prior to disabling role r in z. $can\_Disable(r,z)$ predicate defines when role r can be disabled in zone z.

- $\forall (r', z') \in pre\_DisableRoles(r,z) \bullet can\_Disable(r,z) \Rightarrow r' \notin enabled\_roles(z')$

Pre-requisite on Role Assignment:

We define the function $pre\_AssinRoles(r,z)$ in order to obtain the pre-requisite roles that should be already assigned to a user in order to assign role r in zone z to a user. The de-assignment of any of *pre-requisite roles* to a user does not imply that the assignment of role r must be revoked. The temporal, spatial, and strong role assignment pre-requisite can be defined by this function.

- $\forall (r', z') \in pre\_AssignRoles(r,z) \bullet can\_AssignRole(u,r,z) \Rightarrow r' \in assigned\_roles(u,z')$

Pre-requisite on Role Deassignment:

Some critical roles must be deassigned from a user before deassigning other roles. The function $\mathit{pre\_DeassignRoles}(r,z)$ gives the set of roles that must be deassigned before deasigning role in zone z. The predicate $\mathit{can\_DeassignRoles}(u,r,z)$ says whether or not role r can be deassigned for the user u in zone z.

- $\forall(r^{'},z') \in \mathit{pre\_DeassignRoles}(r,z) \bullet \mathit{can\_DeassignRole}(u,r,z) \Rightarrow r^{'} \notin \mathit{assigned\_roles}(u,z')$

Pre-requisite on Role Activation:

The function $\mathit{pre\_ActivateRoles}(r,z)$ determines the set of *pre-requisite* roles that have to be previously activated by a user in zone z in order to activate r. This security principle does not enforce the deactivation of roles in case of some or all of the *pre-requisite* roles are deactivated. The temporal, spatial, and strong role activation pre-requisites are also defined over the same function.

- $\forall(r^{'},z') \in \mathit{pre\_ActivateRoles}(r,z) \bullet \mathit{can\_ActivateRole}(u,r,z) \Rightarrow r^{'} \in \mathit{active\_roles}(u,z')$

Pre-requisite on on Role Deactivation:

Some critical roles must be deactivated before deactivating other less important roles. The function $\mathit{pre\_DeactivateRoles}(r,z)$ gives the set of roles that must be deactivated before deactivating role r in zone z. The predicate $\mathit{can\_DeactivateRole}(u,r,z)$ states whether role r can be deactivated from user u in zone z. The following constraint captures this.

- $\forall(r^{'},z') \in \mathit{pre\_DeactivateRoles}(r,z) \bullet \mathit{can\_DeactivateRoles}(u,r,z) \Rightarrow r^{'} \notin \mathit{active\_roles}(u,z')$

Triggers for Performing Automated Operations

Role Trigger:

In our model, the role trigger principle enforces that enabling or disabling a role triggers other roles to be enabled or disabled respectively, in some predefined locations and durations. The trigger event is defined by symbol $\longmapsto$. For example, in a medical information system, the role nurse-in-training is enabled in a hospitable for a predefined duration whenever the role of day-time-nurse is enabled. Furthermore, disabling the day-time-nurse forces the role nurse-in-training to be disabled immediately or after some fixed durations depending on the information sensitivity.

In other words, a nurse in training role can access to a system as long as a member of role day-time-nurse is present in the system. Another example for role activation triggers, the activation of doctor-on-night-duty in a hospital at the night-time imposes the role of nurse-on-night-duty to be activated.

The relations $\mathit{triggered\_EnableRoles}(r,z)$ and $\mathit{triggered\_ActiveRoles}(r,z)$ give the triggered roles that need to be respectively enabled or activated in zone z. This means that the event of enabling role r in zone z, i.e., $\mathit{enable}(r,z)$, triggers ($\longmapsto$) the roles in $\mathit{triggered\_EnableRoles}(r,z)$ to be enabled. The events of disabling a role (i.e., $\mathit{disable}(r,z)$), activating a role (i.e., $\mathit{activate}(r,z)$), and deactivating a role (i.e., $\mathit{deactivate}(r,z)$) in zone z are defined in the same way of the enabling trigger. The following conditions must be satisfied whenever a *stimuli role* gets enabled or activated by someone in predefined zones. Note that, the *termination triggers* enforce a system to deactivate a role at the moment the current user's zone violates the STZone conditions associated with that role.

- $\forall r' \in \mathit{trriger\_EnableRoles}(r,z) \bullet \mathit{enable}(r,z) \longmapsto \mathit{enable}(r',z)$
- $\forall r' \in \mathit{trriger\_DisableRoles}(r,z) \bullet \mathit{disable}(r,z) \longmapsto \mathit{disable}(r',z)$
- $\forall r' \in \mathit{trriger\_ActiveRoles}(r,z) \bullet \mathit{activate}(r,z) \longmapsto \mathit{activate}(r',z)$
- $\forall r' \in \mathit{trriger\_DeactiveRoles}(r,z) \bullet \mathit{deactivate}(r,z) \longmapsto \mathit{deactivate}(r',z)$

Triggers for Spatio-Temporal Zone Change of User:

In the spatio-temporal model, we may need some actions to take place when some events of interest happens. We can formalize triggers for this purpose. For example, if a user moves out of a spatio-temporal zone, his role must be automatically deactivated. The triggers are described using the notation $\mathit{event} \overset{\mathit{cond}}{\longmapsto} \mathit{action}$, where *event* triggers *action* if the predicate *cond* is true.

Let $\mathit{ZoneChange}(u,z,z')$ represent the event where the spatio-temporal zone associated with the user u changes from z to z'.

Role Deactivation: $\mathit{ZoneChange}(u,z,z')\{\xrightarrow{r \in \mathit{active\_roles}(u,z) \wedge \neg \mathit{can\_ActivateRole}(u,r,z')}\}$
$\mathit{DeactivateRole}(u,r,z')$

Role Deassign: $ZoneChange(u,z,z')\{\xrightarrow{r \in assigned\_roles(u,z) \wedge \neg can\_AssignRole(u,r,z')}\}$
$DeassignRole(u,r,z')$

Triggers for Enforcing Pre-requisite Constraints:

We define a number of concurrent constraints for guaranteeing role pre-requisites:

Role Enabling: $EnableRoles(r',z) \mapsto EnableRole(r,z)$ where $r' \in pre\_EnableRole(r,z)$

Role Disabling: $DisableRole(r',z) \mapsto DisableRole(r,z)$, where $r' \in pre\_DisableRole(r,z)$

Role Activation: $ActivateRole(u,r',z) \mapsto ActivateRole(u,r,z)$, where $r' \in pre\_ActiveRoles(r,z)$

Role Deactivation: $DeactivateRole(u,r',z) \mapsto DeactivateRole(u,r,z)$, where
$r' \in pre\_DeactivateRole(r,z)$

Assign Role: $AssignRole(u,r',z) \mapsto AssignRole(u,r,z)$, where $r' \in pre\_AssignRole(r,z)$

Deassign Role: $DeassignRole(r',z) \mapsto DeassignRole(r,z)$, where $r' \in pre\_DeassignRoles(r,z)$

Post-Requisite Constraints

Post-requisite on Role Enabling:

Spatio-temporal post-requisite enforces a role to remain enabled in a specific location and time as long as other roles are in an enabled state. An example where this constraint is required is that a primary-care-nurse role can be enabled inside a ward during the treatment time if and only if the primary-care-physician role is already enabled. Due to the fact that the primary-care-physician must follow up the treatments given by the primary-care-nurse to the patient, the primary-care-physician role should remain in an enabled state as long as the primary-care-nurse work is going on. The function *post_EnableRoles*(r,z) gives the set of post-requisite roles for role r. This function can also define the temporal and spatial role enabling post-requisite using respectively temporal and spatial zones.

- $\forall\ r' \in post\_EnableRoles(r,z) \bullet r' \in enabled\_roles(z) \Rightarrow r \in enabled\_roles(z)$

Post-requisite on Role-Assignment:

With spatio-temporal role assignment post-requisite, a role cannot be deassigned to a user while another critical role is still assigned to the same user in some zones. This example shows where this constraint is useful, a corporation may have a policy that the account auditor role can be assigned to a user if and only if the accountant has already assigned in the accountant office during the auditing period. Additionally, the role accountant cannot be designed while the account auditor is assigned to the same user. $post\_AssignRoles(r,z)$ gives the set of all the post-requisite assignment roles in zone z which requires the assignment of role r to the same user. The temporal, spatial, and strong role assignment post-requisites are defined using the same function.

- $\forall\ r^{'} \in post\_AssignRoles(r,z)\ \bullet\ r^{'} \in assigned\_roles(u,z)\ \Rightarrow\ r \in assigned\_roles(u,z)$

Post-requisite Role Activation:

This constraint prevents a user from deactivating a role as long as another critical role is in an active state by the same user in undesirable zones. For example, once a user activate the role of teller and access a confidential room in a bank during the working hours, the user must has already activated the role of bank clerk. Furthermore, the same user cannot deactivate the role of bank clerk while that user is playing the teller role in the bank during the working hours. Our model express such requirements using the $post\_ActiveRoles(r,z)$ function which gives the set of post-requisite roles of role r in zone z. This function also expresses the temporal and spatial role activation post-requisites.

- $\forall r^{'} \in post\_ActiveRoles(r,z)\ \bullet\ r^{'} \in active\_roles(u,z)\ \Rightarrow\ r \in active\_roles(u,z)$

4.3 The Graph Model Representation

Figure 4.1 summarizes the primary components of the proposed models. In this figure, the models' entities are represented by oval shapes, and the single directional and bi-directional arrows respectively represent one-to-one and many-to-many relations between those entities. Furthermore, the bi-directional arrow with double ends (filled triangles) represents multiple relations between the same entities, and it is used to reduce the crowd in the graph model. Most entities in the graph are connected to the STZone entity to express the spatio-temporal constraints. The cylinder shape depicts the spatio-temporal constraints on model relationships.

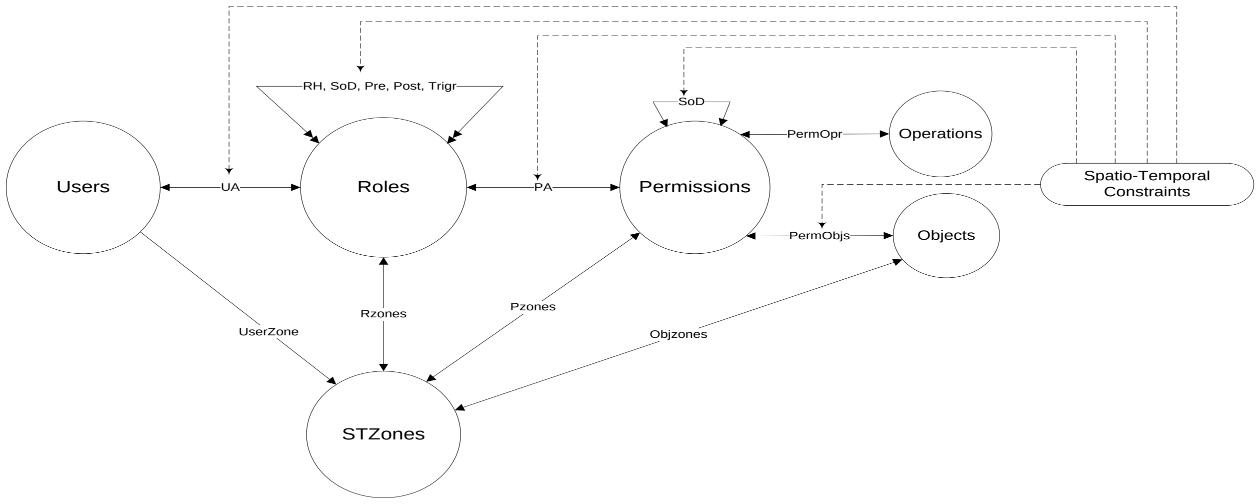


Figure 4.1: Consolidated Spatio-Temporal RBAC Model

4.4 Real-World Mobile Application DDSS

As a proof-of-concept, we specify an extended cell-phone version of Dengue Decision Support System (DDSS) using our approach. DDSS was developed in a collaborative effort between Colorado State University (CSU), Universidad Autonoma de Yucatan (UAY), Merida, Mexico, and Servicios de Salud de Yucatan (SSY), Merida, Mexico. DDSS is a management information system aims to improve prevention, surveillance, and control of the dengue vector-borne disease in constrained geographical environments.

DDSS aids end-users in the prediction and quick response to outbreaks of the dengue disease. This includes Vector Control (VC) and Vector Surveillance (VS) team members to collect information regarding the surveillance and control tasks. Additionally, public health officials including clinic and hospital physicians, laboratory technicians, and epidemiologists interact with the system for collecting disease data during the course of control or surveillance mosquito vector. Control and surveillance information are structured in the form of a flexible hierarchy using a located-in relationship. For example, a city is located-in a jurisdiction, a jurisdiction is located-in a municipality, a municipality in turn is located-in a state, which is located-in a country. Thus, DDSS database resides in different geographical areas, and it is remotely accessed by users via smartphones while they are working in different surveillance areas.

DDSS has the following job functions: Personal managers are responsible for assigning roles, tasks, and privileges to users. Clinicians review patient personal information (e.g., names, gender, date of birth), premises of the patient (e.g., residence, and optionally work or school), past hospitalization and treatment information (e.g., clinic, physicians, disease), and clinical findings (e.g., presence/absence of fever, nausea, or headache). Laboratory technicians collect laboratory test data including type of samples, method(s) used to test the samples, framework for interpreting test results, and interpreted results. Epidemiologists access patient and laboratory test information with regard to plan and evaluate health safety standards and programs in order to improve surveillance and public health strategies.

VC team members spray houses in certain infected areas. VS team members perform mosquito collection and testing tasks intended for developing insecticide resistance methods. VC and VS members are provided with needed materials to use during the course of dengue vector surveillance and control. Material managers (MM) at state or city provide required materials to teams and update materials inventories data in their vicinity. Vector manager (VM) creates a list of tasks to be performed by VC and VS teams. Figure 4.2 illustrates the operational architecture of DDSS.

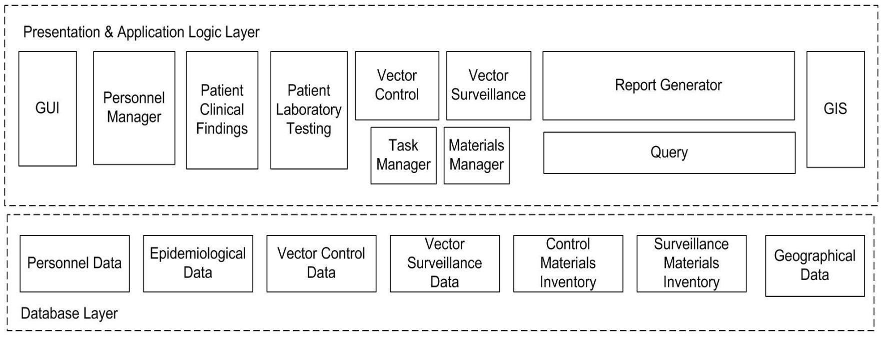


Figure 4.2: DDSS Operational Architecture

With the old version of DDSS, VC and VS teams go house-to-house and collect data in paper-based forms, and then data is entered into the system. However, entering data from paper sheets has many disadvantages. Paper-based data collection is time-consuming, error-prone, degrades the system performance (i.e., late responses), and these forms might easily get lost or damaged. Furthermore, much of the collected information is never entered into the system, and henceforth the analysis precision is lost.

Electronic data collection via cell-phones can solve the paper-based forms problem. It expedites emergency responses and tracking the disease evolution. With the help of cell-phone, data is instantaneously collected and transmitted to DDSS in a reliable manner. In case of network access is not available, data is stored in cell-phones and transmitted later. Figure 4.3 shows the architecture of cell-phone based DDSS.

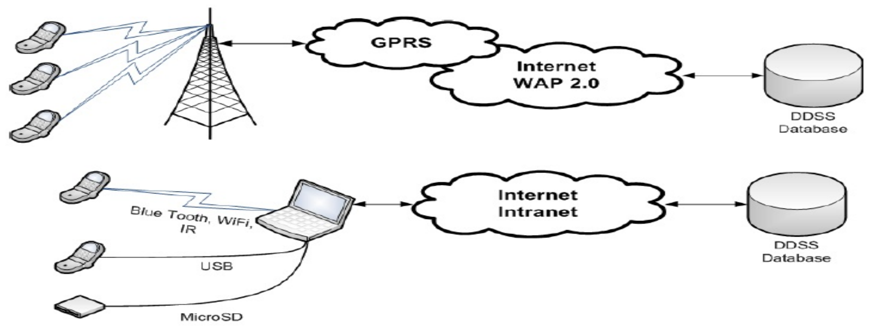


Figure 4.3: Cell-Phone DDSS Architecture

Security Requirements for Mobile-based DDSS

Since DDSS is accessed by nomadic devices from diverse geographical locations and during different time intervals, spatiality and temporality are important security measures that must be considered. Healthcare information of dengue patients should be protected from unauthorized access. Healthcare professionals are only allowed to access their patients' records in the same areas where the dengue case occurred and during the course of dengue.

Furthermore, a spatio-temporal evidence that VC and VS teams performing their tasks in the right place and time is also mandatory for the credibility of data and disease control. Geographical hierarchy in DDSS imposes restrictions on both permission-inheritance, role-activation hierarchies, role assignments, and activation relationships. For example, a clinician working at the state level can inherit permissions from the clinician at the city level. Therefore, a spatio-temporal policy should be in place to define these requirements in DDSS. Consider the following DDSS access control policy specified by our model:

- **Users:** Consider the following DDSS users.

  $Users = \{Dan, Alice, Cliar, Tom, Sam, Yue, Lura\}$

- **Roles:** We only consider job functions at state *State* and city *CityA*.

  $Roles = \{PersonalManeger(PM), StateHosbitalClinician(SHC), CityHosbitalClinician(CHC), StateEpidemiologist(SE), CityEpidemiologist(CE), CityMaterialManager(CMM), CityVM(CVM), VCTeamMember(VCT), VSTeamMember(VST)\}$

- **Permissions:**

  $Permissions = \{$ *ReadPatientRecord*(p_1), *UpdatePatientRecord*(p_2), *ReadPatientPrimse*(p_3), *UpdatePatientPrimse*(p_4), *ReadLaboratoryTest*(p_5), *UpdateLaboratoryTest* (p_6), *ReadUsersData*(p_7), *UpdateUsersData*(p_8), *ReadControlMaterial*(p_9), *UpdateControlMaterial*(p_{10}), *ReadVectorInfo*(p_{11}), *UpdateVectorInfo*(p_{12}), *ReadScheduledTasks*(p_{13}), *UpdateScheduledTasks*(p_{14}), *DispatchVectorCase*(p_{15}), *CollectVectorInfo*(p_{16}), *SprayHouses*(p_{17}) $\}$

- **Objects:**

  $Objects = \{\ PersonalUserData(obj_1),\ PatientClinicalData(obj_2),\ PatientLaboratoryData(obj_3), VectorData(obj_4),\ PatientPremise(obj_5),\ MaterialsInventoryData(obj_6)\ \}$

- **Spatio-temporal zones:**

  STZones = \{z_0:<*State,DayTime*>, z'_0:<*State,NightTime*>, z_1:<*MainOffice,DayTime*>, z'_1:<*MainOffice,NightTime*>, z_2:<*StateClinic,DayTime*>, z_3:<*CityClinic,DayTime*>, z_4:<*StateEpo,DayTime*>, z_5:<*CityEpo,DayTime*>, z_6:<*MainWarehouse,DayTime*>, z_7:<*CityWarehouse,DayTime*>, z_8:<*VMainOffice,DayTime*>, z_9:<*VCityOffice,DayTime*>, z_{10}:<*City,Dayime*>, z'_{10}:<*City,NightTime*>\} , *DayTime = [8 a.m, 5 p.m]* and *NightTime = [6 pm, 10 a.m]* and zone z_0 is the zone that contains all other zones.

- **Role zones:**

  $Rzones = \{\ (PM, z_1),\ (PM, z'_1),\ (SHC, z_2),\ (SHC, z_3),\ (CHC, z_3),\ (CHC, z_{10}), (SE, z_4),\ ,(SE, z_5),\ (CE, z_5),\ (CE, z_{10}),\ (CMM, z_7),\ (CMM, z_{10}), (CVM, z_9),\ (CVM, z_{10}),\ (VCT, z_{10}),\ \ (VST, z_{10})\ \}.$

- **Permissions zones:**

  $Pzones = \{\ (p_1, z_2),\ (p_2, z_3),\ (p_3, z_5),\ (p_4, z_5),\ (p_5, z_3),\ (p_6, z_3),\ (p_7, z_1),\ (p_8, z_1),\ (p_9, z_6), (p_{10}, z_7),\ (p_{11}, z_8),\ (p_{12}, z_9),\ (p_{13}, z_9),\ (p_{14}, z_9),\ (p_{15}, z_9)\ (p_{16}, z_{10}),\ (p_{17}, z_{10})\}$

- **Object zones:**

  $Objzones\ =\ \{(obj_1, z_1),\ (obj_2, z_3),\ (obj_3, z_3),\ (obj_4, z_9), (obj_5, z_5),\ (obj_6, z_7)\}$, most of these objects can be also accessed in state zone z_0 and city zone z_{10} with limited permissions such as read only.

- **User-Role Assignments:**

  $UA_z = \{(Dan, PM, z_1),\ (Dan, PM, z_1'),\ (Alice, SHC, z_2), (Alice, SHC, z_3),$
  $(Clair, SE, z_4),\ (Clair, SE, z_5),\ (Tom, CMM, z_7),\ (Tom, CMM, z_{10}),$
  $(Tom, CVM, z_9),\ (Tom, CVM, z_{10}),\ (Sam, CVM, z_9),\ (Sam, CVM, z_{10}),$
  $(Yue, VCT, z_{10}),\ (Lura, VST, z_{10})\}$

- **Permission-Role Assignments:**

  $PA_z = \{(PM, p_7, z_1), (SHC, p_1, z_2),\ (CHC, p_2, z_3),\ (SE, p_3, z_4),\ (CE, p_4, z_5),\ (CMM, p_{10}, z_7),$
  $(CVM, p_{12}, z_9),\ (VCT, p_{13}, z_{10}),\ (VCT, p_{17}, z_{10}),\ (VST, p_{13}, z_{10}), (VST, p_{16}, z_{10})\ \}$

- **Permission Object access:** Table 4.1 shows the zones where the permissions access objects.

Table 4.1: Permission Object Access Zones

Permission	Object	zone	Permission	Object	zone
p_1	obj_2	z_3	p_2	obj_2	z_3
p_3	obj_5	z_4	p_4	obj_5	z_5
p_5	obj_3	z_3	p_6	obj_3	z_3
p_7	obj_1	z_1	p_8	obj_1	z_1
p_9	obj_6	z_6	p_{10}	obj_6	z_7
p_{11}	obj_4	z_8	p_{12}	obj_4	z_9
p_{13}	obj_4	z_9	p_{14}	obj_4	z_9
p_{15}	obj_4	z_9	p_{16}	obj_4	z_{10}
p_{17}	obj_4	z_{10}			

- **Role-Activation Hierarchy:** The job functions in DDSS are extremely hierarchical.

  $RH_A = \{(SHC, CHC, z_3),\ (SE, CE, z_5)\}$

- **Role Enabling Triggers:** The enabling of roles VCT or VST triggers role CMM to be enabled in order to provide control or surveillance materials.

  $enable(VCT, z_{10})\ \longmapsto\ enable(CMM, z_7)$
  $enable(VST, z_{10})\ \longmapsto\ enable(CMM, z_7)$

- **Separation of Duties:** In DDSS policy has the following spatio-temporal SoD constraints:

  – The role of clinician conflicts with role epidemiologist.

  – Material manager and vector manager are conflicting roles.

  – The vector control and vector surveillance are conflicting roles.

  – The updating patient information and patient premise permissions are conflicting permissions.

- The permission to update vector case information conflicts with update vector material permission.
- The permissions of spray a house and collect vector information are conflicting permissions.

Note that most of the conflict between roles and permissions occurs in the same zone or in the containment zones. For example, role VC and role VS cannot be assumed by a user in the same city. Additionally, SoD constraint is inherited upwards in RH_A. For example, SoD conflict between *CHC* and *CE* roles is propagated to their senior roles *SHC* and *SE*.

- $DSoD_r = \{(CHC, CE, z_{10}),\ (CMM, CVM, z_{10}),\ (VCT, VST, z_{10})\}$
- $SSoD_p = \{(p_2, p_4, z_{10}),\ (p_{10}, p_{12}, z_{10}),\ (p_{16}, p_{17}, z_{10})\ \}$

The DDSS policy is graphically visualized in Figure 4.4.

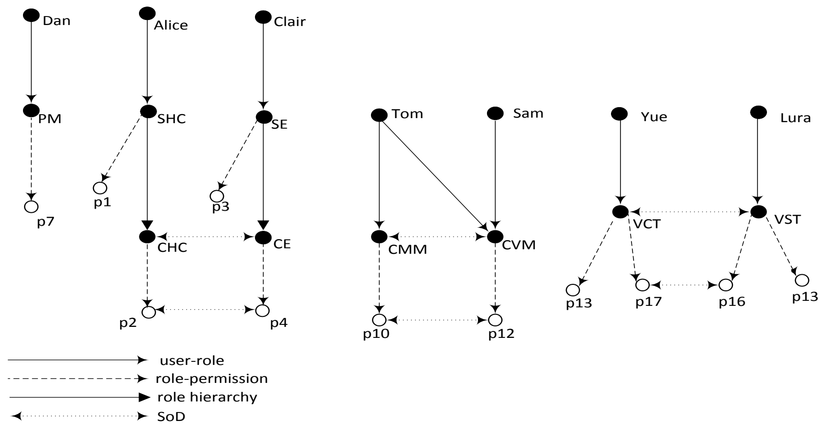


Figure 4.4: Access Control Graph for Mobile DDSS Policy

4.5 A Timed-automata Verification Approach

Due to the nature of strict continuous time in the proposed model, the USE approach is not appropriate to explicitly verify temporal liveness, reachability or safety properties. Usually these properties are expressed in Linear Temporal Logic (LTL) or Time Computational Tree Logic (TCTL) [32, 33]. Therefore, we propose a timed-automata approach supporting TCTL to analyze spatio-temporal policies expressed by our extended model. To the best of our knowledge, existing studies on RBAC have not considered a continuous time modeling language to explicitly verify the impact of spatial and temporal parameters on RBAC components. Furthermore, we consider some important properties for mobile applications that have not addressed yet. Examples include, bounded liveness, atomic actions, urgent actions, pre-requisites, post-requisites, triggers, and granular features interactions. We also provide some effective techniques to condense the state-space explosion problem.

In this approach, temporal constraints are represented by global real variables, whereas the spatiality constraints are specified using shared discrete variables and control states. The nature of spatio-temporal information requires that the timed-automata model must be configured in a different way from merely handling temporal information. Here, a timed-automaton maintains a continuous interaction with two environmental parameters that cause state transitions. The time parameters are uniform and they continuously advance at the same pace, whereas locations of entities are divers and may change or remain the same over time. If there is a location change, it will be handled individually for each entity.

With the timed-automata approach, we build a formal model '**M**' describing the behavior of a mobile policy under verification, the correctness of temporal property 'P' is expressed with TCTL, and then a model-checker UPPAAL is used to automatically decide whether '**M**' satisfies 'P' or not. If successful, UPPAAL reports *"a property is satisfied"*. If unsuccessful, it reports *"a property is not satisfied"*, hence a counterexample trace helps the user to identify the source of the errors.

TCTL has many rules and symbols that allow us, in a feasible manner, to specify a variety of temporal properties. In a spatio-temporal domain, a typical safety property would be "A role should never be activated in some undesirable STZones". Along with safety properties, it is also important to check for some liveness properties to ensure that the system is functioning properly. For instance, a classic liveness query would be "An enabled role will eventually be activated". Moreover, reachability properties are also needed to perform a sanity check for validating the basic behavior of the model. For instance, it makes sense to ask "Is it possible for user *u* to activate role *r* in a certain zone?". Figure 4.5 illustrates the flow of the timed-automata approach.

In the following: Section 4.5.1 provides an overview of timed-automata language and model checking UPPAAL; Section 4.5.2 discusses the timed-automata model of the DDSS policy; Section 4.5.3 describes a set of timed-automata algorithms for defining a policy model; we discuss a number of optimization techniques for alleviating the state-space explosion problem in Sec-

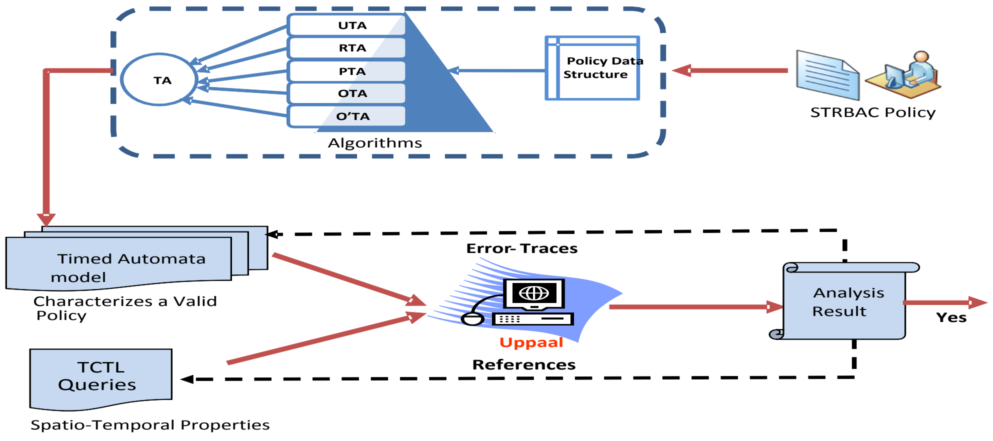


Figure 4.5: The Timed-Automata Verification Approach Framework

tion 4.5.4.

4.5.1 Overview of Timed-Automata and UPPAAL

Timed-Automata:

The theory of timed-automata was proposed by Alur and Dill [24] for modeling and verification of real-time systems. Timed-automata is a finite state machine (a finite directed graph containing a set of nodes/locations connected through edges) augmented with non-negative real-valued variables which model logical clocks in a system. A node in the timed-automata represents a control state and edges are annotated with actions, constraints, and variable updates. Clocks are initialized to zero when a system is started, and they advance synchronously at the same pace. Guards on edges are expressed using clocks and/or discrete variables to constrain the behavior of a timed-automata model. System transitions are represented by edges that are taken whenever clock values satisfy guards on that edges. Clock values might be reset to zero whenever a specific transition is taken. Furthermore, control states (locations) are associated with guards termed as invariants that should be satisfied to stay in such states.

Syntactically, timed-automaton *TA* is a 6-tuple $< L, l_0, C, A, E, I >$ where: L is a set of finite control states, l_0 is the initial location, C is a set of clock variables, A is a set of synchronized actions, E is a finite set of directed edges also called transitions between locations, I is a function that assign locations to invariants, i.e., $I : L \rightarrow B(C)$, and $B(C)$ represents the clock constraints over the set C.

The clocks can take any non-negative real value in $\mathbb{R}_{\geq 0}$. The clock valuation function v, $v : C \rightarrow \mathbb{R}_{\geq 0}$, which maps each clock variable c in the set C to a non-negative real value $v(c) \in \mathbb{R}_{\geq 0}$. The set of all clock valuations is $\mathbb{R}^C$. After some delay $d \in \mathbb{R}_{\geq 0}$ the clock valuation $v(c) + d$ represents the clock valuation of clock c after delay d, and the system may remain in the same state or move to another state. If the next control state is assigned to an invariant, then the clock valuation should satisfy it in order to fire an incoming transition to that state. When the a clock valuation $v(c)$ satisfy a guard transition g, it is formally denoted by $v(c) \vDash g$. Similarly, when the clock valuations satisfy a location invariant $I(l)$ after some delay $d \in \mathbb{R}_{\geq 0}$, it is written as $v(c) + d \vDash I(l)$.

The set of directed edges are formally defined as $E \subseteq L \times A \times B(C) \times 2^C \times L$. The first element L is the outgoing location, the last element L is the target location, A is an action, $B(C)$ is a guard, and 2^C is the a set of clock need to be reset. When $e \in E$ this means that $e = (l_1, a, g, r, l_2)$ such that e is a transition between two control states such as l_1 and l_2 that has an action a and guard g boolean expression involving clocks and other variables that should be satisfied to get the transition enabled, and finally r is a set of clocks need to be reset.

Systems are typically modeled in a single timed-automata model that is composed from a network of timed-automata operating in parallel over a set of clocks, variables, and actions. The parallelism in timed-automata helps to model distributed and concurrent systems efficiently. These timed-automata interact with each other through synchronization channels. This means that changing the state of a timed-automaton by firing a transition might lead to a change of the state of a relevant timed-automaton. For example, two timed-automata T_1 and T_2 can synchronize over two edges e_1 and e_2 labelled with $a_1!$ and $a_2?$ respectively. Once a guard on edge e_1 is evaluated to true, the transition is taken which will cause the transition on edge e_2 to be taken simultaneously if the guard on e_2 is also satisfied. The former automaton T_1 is refereed as "sending" automaton, and the later automaton T_2 is refereed as "receiving" automaton (i.e., there might be more than one receiving automata). In such synchronization action, the transition in the sending automata is performed before those of the receiving automata.

The state of a system modeled as a network of timed-automata is determined by the active locations in each timed-automaton and the values of all clock variables and discrete variables. A transition to a new system state results from a time transition (affects clock valuation, control states, or both) or action transition that are taken only if transition guards and state invariants are evaluated to be true. In particular, a time might elapse without affecting the control states or other variables, it only causes an update of all clock valuations by the same rate reflecting a new system state. The timed-automata language has some supporting toolboxes, such as UPPAAL [31], for performing automated analysis.

UPPAAL Tool:

The model checker UPPAAL [31] is an extensive toolbox which provides the ability to create, simulate, and analyze timed-automata models, and it offers additional features. In particular, UPPAAL is a useful tool for verifying systems modeled as a network of connected parametrized timed-automata acting in parallel. With UPPAAL, we model a system using the UPPAAL editor, simulate it to ensure it behaves as anticipated using the UPPAAL simulator, and verify the system with respect to a set of properties using the UPPAAL verifier. Each parametrized timed-automaton in the model is represented in UPPAAL by a template, and that template has a local declaration pane for variables, channels, constants, and functions. Processes are instantiated for all templates in a model during the simulation and verification.

UPPAAL extends the timed-automata with additional features that enable users to model the behaviors of various types of real time systems. It allows the declaration of different types of data structures including bounded integer variables, one-dimensional arrays, urgent synchronization channels, urgent locations, and user defined functions. Data can be either declared locally (i.e., for a single automata) or globally for all automata in a model. A global clock variable *t* can be used by all automata in a model to synchronize their actions.

Locations in a timed-automaton can be labelled as committed or urgent locations to disallow the passing of time in these locations (i.e., locations must be left without delay). Consequently, if the current active location of a timed-automaton is the urgent or committed location, then the next system transition must involve one of the outgoing edges of that location. These two types of locations are used to express atomic transactions that involve more than a single transition to be taken at the same time (i.e., enabling arbitrary number of roles at the same time). Such a requirement is modeled by labelling intermediate locations as urgent or committed locations connected by transitions representing an atomic transaction.

Edges can also be labelled with urgent synchronization channels to guarantee that those edges are taken without delay as soon as they are enabled. The urgent synchronization channels help to model urgent actions like disabling a role once a spatio-temporal constraint is not satisfied. When several transitions are interleaving, an urgent edge is non-deterministically taken over other edges. This feature reduces the interleaving transitions (i.e., edges from different automata may interleave), thus speeding up the verification process. This feature significantly reduces the number of branches in the state-space graph.

UPPAAL checks properties that are expressed by TCTL. TCTL is the most appropriate logic to express temporal properties as it has operators like "A role will be eventually activated," "An object will always be accessed ," and so on. This query language consists of state formulae and path formulae. State formulae is an expression that describes a property that we need to check in each system state while the path formulae quantifies over the paths and traces in which states are visited. TCTL expressions use two operators to describe state formula, $<>$ indicates over some states and $[]$ indicates at all states. For example, the state formulae φ could be a simple expression, like *Role.Activated*, that is true whenever a role timed-automaton is in a location "Activated" that represents the activation state of a role. Two operators are used in the path formulae, *A* means in all paths and *E* means in some paths. Path formulae is used to specify spatio-temporal *Reachability*, *safety*, and *liveness* properties in our analysis approach.

Reachability properties check whether a given property is satisfied in some reachable states along a path starting from the initial state. For example, it makes sense to ask whether it is possible for a role to be activated by a user in an available spatial-temporal zone. Usually, reachability properties are performed during a system design to validate the sanity of a model behavior. The reachability properties are expressed using the expression $E <> \varphi$, where φ is the state formulae, and path formulae $E <>$ states that there is a path where some states satisfy the state formulae φ.

Safety properties ask that something bad will never happen. In other words, all system states must not violate the safety conditions. For example, a role will never be activated in an undesirable spatial-temporal zone. The safety properties are expressed as follows, $A[]\varphi$, which means that in all paths, all states must satisfy the state formulae φ, if one state evaluated to be false, then the system is considered insecure.

liveness properties verify whether something will eventually happen starting from some states. For example, once a role is assigned, it will be eventually activated by a user in some desirable spatio-temporal zones, otherwise, that role has no meaning in the system and it shouldn't be defined and assigned (e.g., we call such role an isolated/useless entity). The liveness properties can be expressed in the form of *leads to* written as $\varphi \rightsquigarrow \Omega$. It means that whenever the state formulae φ is satisfied in a state, then eventually the state formulae Ω will be satisfied in some states along the path.

Alternatively, the liveness is expressed in the path formulae $A <> \varphi$ meaning that the state formulae φ is eventually satisfied in some states starting from the initial state. UPPAAL also supports a bounded liveness property that ensures that a property of interest not only holds eventually, but within a certain upper time-limit. The use of bounded liveness properties instead of unrestricted liveness properties provides more information and leads to a granular timed verification in case deadlines must be satisfied. The bounded liveness properties are expressed in the form of $\varphi \rightsquigarrow (\Omega \wedge x \leq t')$, where x is a clock variable. This property states that whenever φ is satisfied, Ω must hold before t' time unit.

4.5.2 Timed-Automata Model

The first step of our verification is translating a spatio-temporal policy to a timed-automata model. The result of the translation is a model of a network of timed-automata. Each timed-automaton represents the behavior of an entity at a particular spatio-temporal zone. That is, each timed-automaton in the analysis model represents an entity STZone (e.g., role zone, permission zone, or object zone). Thus, a number of timed-automata are instantiated for each user, role, object, and permission in a system. Clocks express temporal constraints on enabling transitions (i.e., Guards) and control states (i.e., *Location's invariant*) while discrete (integer type) variables express spatial constraints. The state of each entity is identified in a timed-automaton by the current active control state as well as the clock and discrete variables values. In the following, we will discuss how timed-automata are constructed for each entity in the sets of Users, Roles, Permissions, and Objects of the DDSS policy. We use some entities of DDSS to illustrate how timed-automata are defined for the purpose of analyzing the DDSS policy.

Role Observer Timed-Automata O'TA:

Global clock t is used by all timed-automata to express temporal constraints. For each role zone, we develop a timed-automata that describes the behavior of the role for the corresponding zone. We also develop a Role Observer Timed-Automaton (*O'TA*) that is responsible for clock reset, enabling and disabling of all roles. *O'TA* explores the set *Rzones* to determine the zones where a role can be enabled. Nodes in the *O'TA* represent the states signifying the enabling and disabling of roles in zones.

Figure 4.6 shows a partial *O'TA* for role *SHC*, *CHC*, and *PM*. Recall that the roles *PM* is enabled in STZones z_1:<MainOffice,*[8 - 17]*> and z'_1:<MainOffice,*[18 - 22]*>, and *SHC* and *CHC* are enabled in z_3:<CityClinic,*[8 - 17]*>. The *Init* node signifies the state where all the roles are disabled. The outgoing transition from initial location *Init* to location ER_{SHC} with sending action $enable\_SHC[CityClinic]!$, enables role SHC at time instant $t_1 = 8$. Firing this transition changes the state of the receiving role SHC timed-automaton at zone z_3. Node ER_{SHC} represents the state where role *SHC* is enabled.

Since roles *CHC* and *PM* are enabled at the same time instant, thereby intermediate nodes ER_{SHC} and ER_{CHC} are labeled as *committed* to allow simultaneous enabling of these roles. These locations allows more than a single action to be taken at the same time. The transitions enabling roles *CHC* and *PM* are labeled with sending actions $enable\_CHC[CityClinic]!$ and $enable\_PM[MainOffice]!$, respectively. Thus, these action will change the states of roles *PM* and *CHC* timed-automata at zone z_3. Node ER_{CHC} and ER_{PM} represent the state where role *CHC* and *PM* are respectively enabled.

O'TA remains in node ER_{PM} as long as the clock invariant $t < t_1^{'}$, (note that, $t_1^{'} = 17$). At time $t == t_1^{'}$, roles *SHC*, *CHC*, and *PM* are simultaneously disabled by transitions labeled with synchronous actions $disable\_SHC[CityClinic]!$, $disable\_CHC[CityClinic]!$, and $disable\_PM[MainOffice]!$, respectively. The states, DR_{SHC}, DR_{CHC}, and ER_{PM}, express that roles *SHC*, *CHC*, *PM* are disabled, respectively.

Subsequent transitions enable and disable role PM at zone z_1' in nodes ER_{PM}' and DR_{PM}'. At time $t == t_2$, i.e., $t_2 = 18$, PM timed-automaton gets enabled by the $enable\_PM'[MainOffice]!$ action and the control of *O'TA* moves to node ER_{PM}'. Role PM remains enabled in node ER_{PM}' until the moment $t == t_2'$, i.e., $t_2' = 22$, PM is disabled by action $disable\_PM'[MainOffice]!$. Location DR_{PM}' represents the state that role PM in zone z_1' is in disabled state.

On the event of firing the input transition to location *Init* coming from location DR_{PM}', at $t = t_3$ (note that, $t_3 = 24$), the function $reset\_clock()$ resets clock variable t. The nodes that are not committed are labeled with invariants that strictly express the start and end of time intervals at which roles are enabled or disabled.

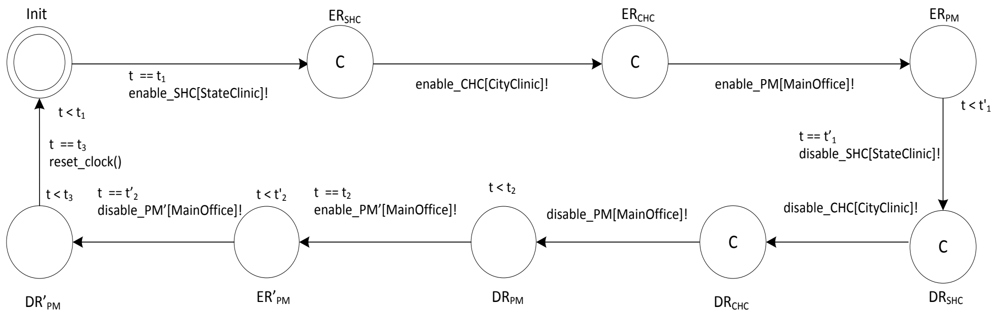


Figure 4.6: Role Observer Timed-Automata

Role Timed-Automata *RTA*:

For each role in set *Roles*, a parametrized automaton is constructed for each role zone in set *Rzones* to capture the states at which the role is enabled, disabled, activated, or deactivated at that zone. In the DDSS policy, we have *16* roles timed-automata. Note that, a role might be at the same time in different states at different role zones. The timed-automaton in Figure 4.7 represents the enable, disable, active, inactive states of role CHC at role zone $z_3 =< CityClinic, [8 - 17] >$.

Node *DisabledInCityClinic* is the initial state of role CHC where it is disable, and control state *EnabledInCityClinic* is the target state of the *DisabledInCityClinic* state that represents the enable state of role CHC. Edges between these two control states synchronize with sending edges in *O'TA* (see Fig. 4.6) through receiving action $enable\_CHC[CityClinic]?$. Firing this action changes the state of role CHC at zone z_3 to the enable state.

The receiving action $activate\_CHC[CityClinic]?$ activates role CHC. This action synchronizes with a sending activation action at user Alice timed-automaton (see Fig. 4.10). Firing this action changes the state of role CHC from an enabled to an active state. Note that, a role might be assigned to multiple users in zone z_3. Thus, any subsequent activation of role CHC in *ActivatedInCityClinic* does not change the state of role CHC. The self-loop transitions at *ActivatedInCityClinic* indicate that role CHC remains in an active state as long as there is at least one user is using role CHC.

The $Cont\_CHC[CityClinic]$ variable records the number of users who activate role CHC. When no user is using role CHC, role CHC goes back to the *EnabledInCityClinic* state. Role CHC goes back from the *EnabledInCityClinic* state to the *DisabledInCityClinic* state at the end of the enabling duration through *urgent* synchronization action $disable\_CHC[$
$CityClinic]?$. Since CHC is only accessed by Alice via RH_A, these self-loop edges are not needed, but we present them for the clarity of discussion.

The *committed* $Aperm_2$ node between *EnabledInCityClinic* and *ActivatedInCityClinic* are added to capture the permissions p_2 access at the time role CHC is activated. The sending $access\_p_2[CityClinic]!$ action synchronize with a receiving action at the corresponding permission timed-automaton of p_2 in zone z_3 (see Fig. 4.8). The *committed* $Eperm_2$ node is also added between *ActivatedInCityClinic* and *EnabledInCityClinic* to release of permission p_2 when role CHC is deactivated by the user. The sending channel $exit\_p_2[CityClinic]!$ has a corresponding receiving action at permission p_2 timed-automaton.

The deactivation of role CHC is represented by *urgent* synchronization channel $deactivate\_$
$CHC[CityClinic]?$. Thus, if a user suddenly moves to an invalid location while a role is in an active state by that user, that role must be immediately deactivated. Similarly, when a role activation time is expired, role CHC must be instantly revoked from the user. Such requirements are expressed by

the deactivation triggers at user timed-automata (e.g., *urgent* actions and location invariants (see Fig. 4.10)). Timed-automata for other roles are composite in a similar manner to CHC.

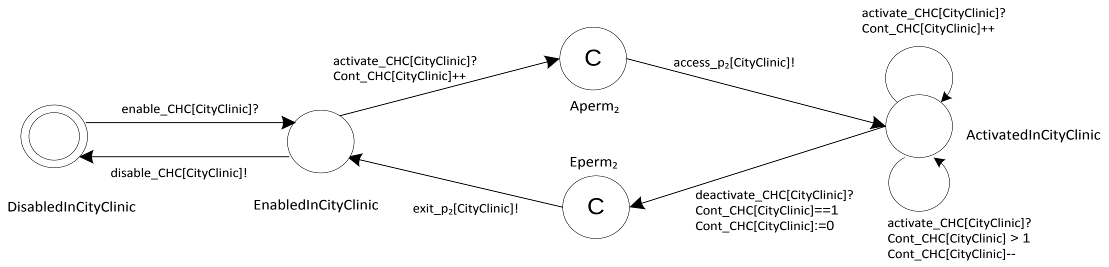


Figure 4.7: Timed-Automata of Role CHC at zone z_3

Permission Timed-Automata *PTA*:

A permission timed-automaton is created for each permission zone in set *Pzones*. Therefore, the DDSS policy has *17* permissions timed-automata. Each permission timed-automaton models the behaviour of a permission at a certain permission zone. Figure 4.8 shows the timed-automaton of permission p_2 in zone z_3. Here, the states of a permission are represented by control states *UnacquiredInCityClinic* and *AcquiredInCityClinic* by a role. The receiving action $access\_p_2[CityClinic]?$ coming from initial node *UnacquiredInCityClinic* represents a request from CHC role timed-automaton, (see Fig. 4.7), to access permission p_2. The action of exit permission p_2 is represented by receiving action $exit\_p_2[CityClinic]?$ going from node *AcquiredInCityClinic*.

Once the access transition is taken, the permission p_2 state moves to the *AcquiredInCityClinic* control state. Permission p_2 remains in the *AcquiredInCityClinic* state as long as some roles are using it. The variable $Cont\_p_2[CityClinic]$ controls the number of roles that are currently using permission p_2 in zone z_3.

The *committed* node $Aobj_2$ is added between *UnacquiredInCityClinic* and *AcquiredInCityClinic* to express that the associated object *obj*2 (see Fig. 4.9) with permission p_2 is accessed in zone z_3. Object *obj*2 is accessed by a sending action $access\_obj_2[CityClinic]!$ that has a corresponding receiving action at timed-automaton p_2. The *committed* $Eobj_2$ node between *AcquiredInCityClinic* and *UnacquiredInCityClinic* nodes with sending actions $exit\_obj_2[CityClinic]?$ represent the exit from object obj_2. We follow the same approach to instantiate timed-automata of other permissions in the *Permissions* set.

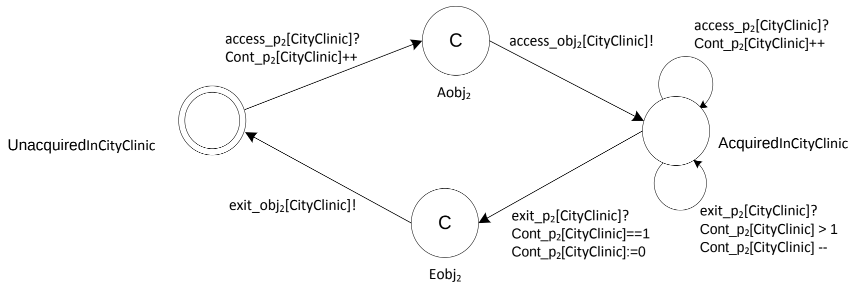


Figure 4.8: Timed-Automata of Permission p_2 at zone z_3

Object Timed-Automata OTA:

An object timed-automaton is generated for each object zone in set *Objzones*. In total, the DDSS policy has *6* objects timed-automata. The timed-automaton of object obj_2 in Figure 4.9 has two alternating control states, initial node *UnaccessedInCityClinic* and node *AccessedInCityClinic* in zone z_3. Receiving action $access\_obj_2[CityClinic]?$ represents the access to object obj_2 by permission p_2, while the action of leaving the object is defined by $exit\_obj_2[CityClinic]?$. Object obj_2 remains in *AccessedInCityClinic* node as long as it is being accessed by at least one permission. The $Cont\_obj_2[CityClinic]$ variable tracks the number of permissions accessing object obj_2. Other objects timed-automata are composite in the same way of object obj_2.

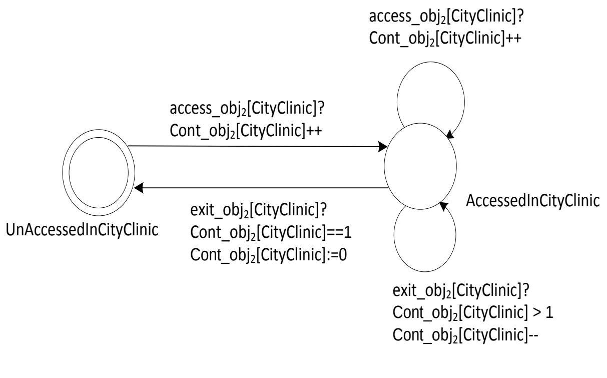


Figure 4.9: Timed-Automata for Object obj_2 at zone z_3

User Timed-Automata UTA:

A user timed-automaton is created for each user in set *Users*. DDSS has 7 users timed-automata. Each user timed-automaton has an initial node and a set of nodes representing the behaviour of a user at a particular zone where some roles are assigned. The set of user-role assignment, UA_z, determines the number of such nodes. For example, when $(u,r,z),(u,r',z') \in UA_z$, then two nodes, for example L_z and $L_{z'}$, are created in user u timed-automaton that captures the states of activating roles r and r' at zones z and z', respectively. The configuration of edges at each nodes varies based on relations RH_A, $DSoD_r$, pre-requisite enabling, post-requisite enabling, etc.

A partial user timed-automaton *UTA* in Figure 4.10 models the behavior for user Alice at zone z_3. Control state *UserElsewhere* is an initial state that covers all the zones in which user Alice is not permitted to access a system. Node *UserInCityClinic* represents that user Alice is in zone z_3 The edges between these control states represent that Alice moves in and out of zone z_3, and they are controlled by timed guards using global clock variable t.

At time $t >= t_1$, Alice moves to zone z_3, which is captured by node *UserInCityClinic*. At control state *UserInCityClinic*, user Alice can activate and deactivate role SHC through synchronization channels $activate\_SHC[CityClinic]!$ and $deactivate\_SHC[CityClinic]!$ and during time interval $[t_1,\ t_1']$. Firing a transition with $activate\_SHC[CityClinic]!$ or $deactivate\_$
$SHC[CityClinic]!$ changes the state of the role SHC timed-automaton. The deactivation actions are defined as *urgent* actions. Such action is always has a high priority to be taken once it is enabled.

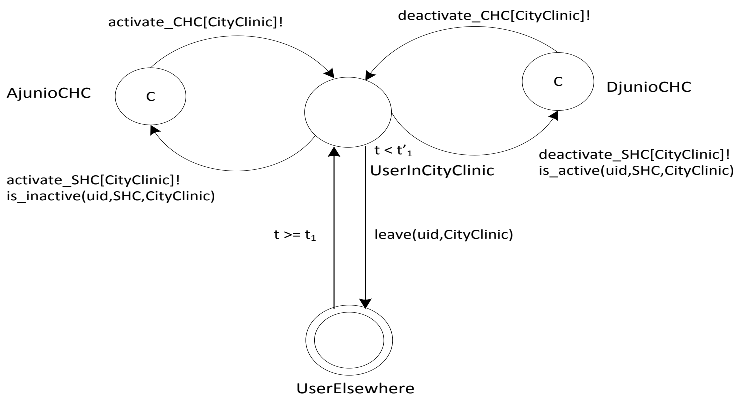
The boolean $is\_inactive(uid,SHC,CityClinic)$ function ensures that user Alice has not yet activated role SHC, the boolean $is\_active(uid,SHC,CityClinic)$ function checks that user Alice has already activated role SHC. The variable *uid* stores the identifier of user Alice. The boolean $leave(uid,CityClinic)$ function ensures that all roles are deactivated when user Alice leaves position *CityClinic*. Alice can deactivate role SHC at any time before the activation time expires. In case Alice leaves node *CityClinic* without deactivating the active role SHC, a system enforces such role to be deactivated. This requirements are expressed by associating invariant $t < t_1'$ with node *UserInCityClinic* to urgently deactivate role SHC in case a user does not do that.

In the DDSS policy, RH_A defines that role SHC is a senior role of junior role CHC in zone

z_3. Thus, Alice can have full access to the role CHC privileges in zone z_3 through the role activation of SHC. That is, the activation of role SHC allows Alice to activate role CHC to access some resources in the clinic. As such, we capture such requirements by associating some *committed* nodes with the *UserInCityClinic* node. The *committed AjuniorCHC* node and activation edges between *AjuniorCHC* and *UserInCityClinic* nodes allow user Alice to activate junior role CHC at the time senior role SHC is activated. The edge between *AjuniorCHC* and *UserInCityClinic* with sending action $activate\_CHC[CityClinic]!$ activates role CHC right after activating role SHC. The *committed DjuniorCHC* node and deactivation transitions connected to it enforce the deactivation of junior role CHC at the time senior role SHC is deactivated. That is, firing edge between *DjuniorCHC* and *UserInCityClinic* with sending action $deactivate\_CHC[CityClinic]!$ deactivates CHC at the moment role SHC is deactivated.

A user must not activate two roles that have the $DSoD_r$ constraints in some zones. In the DDSS policy, Tom is assigned to CMM and CVM roles in zone z_6 which have the $DSoD_r$ conflict in this zone. A timed-automaton in Figure 4.11 shows that Tom cannot simultaneously activate roles CMM and CVM in node *UserInCity*, he is only allowed to activate at most one role. That is, once the activation transition of role CMM is taken at *UserInCity*, the activation transition of role CVM is disabled and vice versa. The activation transition of roles CMM and CVM are respectively associated with boolean functions $dsd(CMM, City)$ and $dsd(CVM, City)$ to satisfy this requirement. These functions determine whether some conflicting roles are in active state or not. For example, role CMM cannot be activated if $dsd(CMM, City)$ returns true.

Note also that, a user may attempt to activate a role that may have some pre-requisite activation constrains in some zones. In such scenario, we define some boolean functions to evaluate these constraints in the policy. Therefore, the role activation transition must be guarded by a boolean function to represent the pre-requisite constraint. We will elaborate how these functions are defined and associated with edges in the definition of algorithms constructing a timed-automata model.



activate_CHC[CityClinic]!
deactivate_CHC[CityClinic]!
AjunioCHC
C
C
DjunioCHC
t < t'₁
UserInCityClinic
deactivate_SHC[CityClinic]!
is_active(uid,SHC,CityClinic)
activate_SHC[CityClinic]!
is_inactive(uid,SHC,CityClinic)
t >= t₁
leave(uid,CityClinic)
UserElsewhere

Figure 4.10: Timed-Automata for User Alice

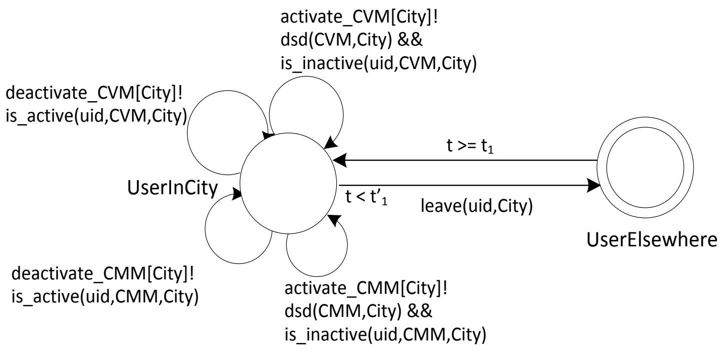


Figure 4.11: Timed-Automata for User Tom

4.5.3 Algorithms for Constructing Timed-Automata Model

Now we describe a set of algorithms for constructing a number of timed-automata for each user, role, permission, and object as well as observer timed-automaton in a system. Furthermore, we also provide the time-computation complexity for each algorithm. Table 4.2 summarizes the input data structure in the form of sets and relations that are used in these algorithms for constructing a timed-automata model for a system.

Algorithm 2 for constructing role O'TA:

Algorithm 2 explores role's zones for each role r in *Roles* and determines the start time t_s and end time t_e at which role r is enabled and disabled. The $enable\_r[S_i]!$ and $enable\_r[S_i]!$ actions are ordered in the $record[]$ array based on time instants when these actions should be taken.

Table 4.2: Input Data Structure for Timed-Automata Algorithms

N	Data Set	Description
1	*Users*	Set of users in a system
2	*Roles*	Set of roles in a system
3	*Objects*	Set of protected objects in a system.
4	*Permissions*	Set of permissions in a system.
5	$Rzones(r)$	Returns the set of spatio-temporal zones associated with role r
6	$Pzones(p)$	Returns the set of spatio-temporal zones associated with permissions p
7	$Objzones(obj)$	Returns the set of spatio-temporal zones associated with object obj
8	$assigned\_perms(r,z)$	Returns the set of permissions associated with role r in zone z
9	$used\_roles(p,z)$	Gives the set of roles using permission p in zone z.
10	$permObjs(p,z)$	Returns the set of objects associated with permission p in zone z
11	$objectPerms(obj,z)$	Defines the set of permissions using an object in zone z
12	$ZLoc(z)$	Returns the physical location in zone z
13	$ZInt(z)$	Returns the time interval in zone z
14	$assigned\_roles(u,z)$	Returns the set of roles assigned to user u in zone z
15	$assigned\_users(r,z)$	Gives the set of assigned users to role r in zone z
16	$junior_I(r,z)$	Gives the junior roles of role r in RH_I at zone z
17	$junior_A(r,z)$	Returns the junior roles of role r in RH_A at zone z
18	$dsod\_Roles(r,z)$	Returns dynamically conflict roles with role r in zone z
19	$pre\_EnableRoles(r,z)$	Returns pre-requisite enabling roles for role r in zone z
20	$post\_EnableRoles(r,z)$	Returns the enable post-requisite roles of role r in zone z
21	$Cont\_r[S]$	Count the number of users assuming role r in location S in a zone
22	$Cont\_[S]$	Count the number of roles acquiring permission p in location S in a zone
23	$Cont\_obj[S]$	Determine the number of permissions accessing object obj in location S in a zone

Lines 20-35 explore elements in $record[]$ orderly and construct nodes and edges needed to enable or disable each role. Based on the actions in $record[]$, each node can be enabling node ER_i or disabling node DR_i. A node is marked as a *committed* node only if a role is enabled or disabled at the time instant when the preceding role in $record[]$ is enabled or disabled. The *non-committed* nodes are associated with invariants to enforce enabled edges to be taken without delay.

The number of roles (N_r) and role's zones (N_z) define the time complexity of Algorithm 2 which is $O(N_r * N_z)$. Algorithm 2 constructs one time automata. Therefore, total number of timed-automata constructed in a network timed-automata model is N_{TA}, where $N_{TA} = N_{RTA} + N_{PTA} + N_{OTA} + N_{UTA} + 1$, where N_{RTA} is the number of roles automata, N_{PTA} for the permissions automata, N_{OTA} is the number of objects automata, and N_{UTA} is the number of users automata .

Algorithm 2: Constructing Role Observer Timed-Automaton

```
    input   : Data Structure
    output  : Role Observer Timed-Automata “$O'TA''$
1   foreach $r \in Roles$ do
2       $i = 0$;
3       foreach $z \in Rzones()$ do
4           if $assigned\_users(r, z) \neq \phi$ // role $r$ is assigned to users in zone $z =< s, d >$
5           then
6               $S_i = ZLoc(z)$;
7               $D = ZInt(z)$ ;
8               $t_s = firstInst(Instant(D)$ ;
9               $t_e = lastInst(Instant(D)$;
10              $record[i++] =< enable_r[S_i]!, t_s >$ ;
11              $record[i++] =< disable_r[S_i]!, t_e >$ ;  // $firstInst()$ and $lastInst()$ functions return first and last time instants,
                    respectively
12          end
13      end
14  end
15  $sort(secondField(Record), >)$;  // sort the list of records ascendantly on the basis of the second field time
16  $n = i$;
17  $prevTime = 0$;
18  Let: a Timed-Automata $O'TA = \langle L, L_0, C, A, E, I \rangle$;
19  Initialization: $L = \{L_0\}, I = \phi, L_0 = Init, C = \{t\}, A = \phi, E = \phi$;
20  foreach i = 0 to i = (n-1) do
21      $L = L \cup \{L_{i+1}\}$; // $L_{i+1}$ can be $ER_i$ or $DR_i$
22      $currentTime = secondField(Record[i+1])$;
23      if $(prevTime == currentTime$ // committed nodes
24      then
25          $commit(L_{i+1})$ ;
26          $g = \phi$ ;
27          else
28              $g = (x == secondFiedl(Record[i+1]))$ ;
29              $I = I \cup \{(L_{i+1} \rightarrow (x < secondField(Record[i+1]))\}$ ;  // associate an invariant with not "Committed" nodes
30          end
31      end
32      $a = firstField(Record[i+1])$;
33      $A = A \cup \{a\}$ ;
34      $E = E \cup \{(S_i, a, g, \phi, L_{i+1})\}$ $prevTime = currentTime$ ;
35  end
36  $E = E \cup \{(L_{i+1}, a, g, reset\_clock(), L_0)\}$; // $reset()$ function to resets global clock $t$ to 0
37  $return(O'TA)$;
```

Algorithm 3 for constructing RTA:

A number of parametrized timed-automata are constructed for each role r in the *Roles* set to capture its behaviour at some role zones. Nodes set L is initialized by $DisabledInS_i$ and $EnabledInS_i$ nodes; actions and edges connecting these nodes are respectively defined in the A and E sets. Then, node $ActivatedInS_i$ is added to set L and actions $activate\_r[S_i]?$ and $deactivate\_r[S_i]?$ are added to set A.

The *perms* set stores all permissions assigned or inherited by role r in zone z. For each permission p_m in set *perms*, nodes $Aperm\_p_m$ and $Eperm\_p_m$ are added to L, where m counts the number of permissions, and actions to access and leave these permissions are added to E. To allow simultaneous access to these permissions, nodes $Aperm\_p_m$ and $Eperm\_p_m$ are marked as *committed* nodes using the $commit()$ function.

Once set L is complete, the edges in set E are used to connect these nodes. Lines 27-31 consider the case in which role r is assigned to a single user while lines 32-41 structure the timed-automaton edges in case of multiple users are assigned to role r. Then, Algorithm 3 goes back to create another role timed-automaton for the subsequent zone of role r.

The time complexity of Algorithm 3 primarily depends on the number of roles, role's zones, and role's permissions. Therefore, the time complexity of Algorithm 3 is $O(N_r * N_z * (N_{rh} + N_p))$, where N_r is the number of application roles, N_z represents the maximum number of role's zones, N_{rh} is the maximum number of junior roles, and N_p is the upper bound of the role's permissions. When Algorithm 3 halts, it constructs N_{RTA} role timed-automata, i.e., $N_{RTA} = N_z \times |Roles|$.

Algorithm 3: Constructing Roles Timed-Automata

```
input   : Data Structure
output  : A Set of Parametrized Roles Timed-Automata "RTA"
$RTA = \phi$;
foreach $r \in Roles$ do
    $i = 0$;
    foreach $z \in Rzones(r)$ do
        if $assigned\_users(r,z) \neq \phi$ // role $r$ is assigned to some users in zone $z =< s,d >$
        then
            $S_i = ZLoc(z)$;
            Let: Parametrized Timed-Automata $RTA(S_i) = \langle L, L_0, C, A, E, I \rangle$;
            Initialization: $L = \{DisabledInS_i, EnabledInS_i\}, L_0 = \{DisabledInS_i\}, I = \phi, C = \phi, A = \{enable\_r[S_i], disable\_r[S_i]\}$,
                $E = (DisabledInS_i, enable\_r[S_i], \phi, \phi, EnabledInS_i), (EnabledInS_i, disable\_r[Li], \phi, \phi, EnabledInS_i)$ ;
            $L = L \cup \{ActivatedInS_i\}$;
            $A = A \cup \{activate\_r[S_i]?, deactivate\_r[S_i]?\}$;
            $perm = assigned\_perms(r,z)$;  // the set of permissions associated with role $r$ in $z$ zone
            if $junior_I(r,z) \neq \phi$ // role $r$ is a senior role in role $RH_I$
            then
                foreach $r' \in junior_I(r,z)$ do
                    $perms = assigned\_perms(r', z)$;
                end
            end
            $m = 0$;
            foreach $p_m \in perms$ do
                $commit(Aperm_m, Eperm_m)$ ;  // commit function marks nodes as "Committed" nodes
                $L = L \cup \{Aperm_m,\ Eperm_m\}$;
                $urgentSynChannel(exit\_p_m[S_i])$;  // function to make this action an urgent action
                $A = A \cup \{access\_p_m[S_i]!, exit\_p_m[S_i]!\}$;
                $m++$;
            end
            if $|\ assigned\_users(r,z)\ | = 1$ // role $r$ is assigned to a single user in $z$ zone
            then
                $E = E \cup \{(EnabledInS_i, activate\_r[S_i]?, \phi, \phi, Aperm_0), (Aperm_0, access\_P_0[S_i]!, \phi, \phi, Aperm_1),$
                    $(Aperm_1, access\_p_1[S_i]!, \phi, \phi, Aperm_2), \ldots, (Aperm_{m-1}, access\_p_{m-1}[S_i]!, \phi, \phi, Aperm_m),$
                    $(Aperm_m, access\_p_m[S_i]!, \phi, \phi, ActivatedInS_i)\}$;  // activation and access edges
                $E = E \cup \{(ActivatedInS_i, deactivate\_r[S_i]?, \phi, \phi, Eperm_m),$
                    $(Eperm_m, exit\_p_m[S_i]!, \phi, \phi, Eperm_{m-1}), \ldots, (Eperm_1, exit\_p_1[S_i]!, \phi, \phi, Eperm_0),$
                    $(Eperm_0, exit\_p_0[S_i]!, \phi, \phi, EnabledInS_i)\}$; // deactivation and exit edges
            end
            else
                Let $g$, a guard which is a conjunction of predicates and boolean expressions involving clocks ;
                $Cont\_r[S_i] = 0$;
                $u = Cont\_r[S_i]++$;
                $u' = Cont\_r[S_i]--$;
                $g = Cont\_r[S_i] > 1$;
                $g' = Cont\_r[S_i] == 1$;
                $E = E \cup \{(EnabledInS_i, activate\_r[S_i]?, \phi, u, Aperm_0), (Aperm_0, access\_P_0[S_i]!, \phi, \phi, Aperm_1),$
                    $(Aperm_1, access\_p_1[S_i]!, \phi, \phi, Aperm_2), \ldots, (Aperm_{m-1}, access\_p_{m-1}[S_i]!, \phi, \phi, Aperm_m),$
                    $(Aperm_m, access\_p_m[S_i]!, \phi, \phi, ActivatedInS_i), (ActivatedInS_i, activate\_r[S_i]?, \phi, u, ActivatedInS_i)\}$;  // activation and
                    access edges
                $E = E \cup \{(ActivatedInS_i, deactivate\_r[S_i]?, g, u', ActivatedInS_i), (ActivatedInS_i, deactivate\_r[S_i]?, g', u', Eperm_m),$
                    $(Eperm_m, exit\_p_m[S_i]!, \phi, \phi, Eperm_{m-1}), \ldots, (Eperm_1, exit\_p_1[S_i]!, \phi, \phi, Eperm_0),$
                    $(Eperm_0, exit\_p_0[S_i]!, \phi, \phi, EnabledInS_i)\}$;  // deactivation and exit edges
            end
            $RTA \longleftarrow RTA \cup \{RTA(S_i)\}$;  // adds $RTA(S_i)$ to set RTA
            $i++$;
        end
    end
end
$return(RTA)$;
```

Algorithm 4 for constructing PTA:

Algorithm 4 constructs a number of timed-automata for each permission p in set *Permsiosns* in a quite similar manner to RTA automata. The permission p timed-automaton is initialized by appropriate nodes, actions, and edges as shown in lines 4 - 16. Lines 10-16 explore a set of objects that can be accessed by permission p in S_i. For each object, two *committed* nodes and two edges connecting these nodes are added to set L and set E, respectively. Lines 17-20 and 22-30 structure the timed-automaton based on the number of roles that can acquire permission p. Algorithm 4 repeats the same steps to create another permission timed-automaton for permission p behaviour in another zone.

The key factors of Algorithm 4 time complexity are the number of permissions, permissions' zones, and permissions' objects. The time complexity of Algorithm 4 is $O(N_p * N_z * N_{obj})$, where N_p is the maximum number of permissions, N_z is the upper bound of the permission's zones, N_{obj} is the maximum number of the permission's objects. The total number of permission timed-automata constructed by Algorithm 4 is N_{PTA}, where $N_{PTA} = |N_z| \times |Permissions|$.

Algorithm 5 for Constructing OTA:

Algorithm 5 elaborates the steps of constructing a number of timed-automata for each object *obj* in set *Objects* at each object zone. Lines 1-21 create the nodes and actions that controls the sate of object *obj* at location S_i. The structure of the object *obj* timed-automaton at location S_i is defined by the number of permissions authorized to access object *obj*. Thus, lines 30-34 and 36-44 alternate the object *obj* timed-automaton structure for a single permission or multiple permissions accessing object *obj*.

The number of objects in set *Objects* and objects' zones in set *Objzones* are the key factors to determine the time complexity of Algorithm 5. The time complexity of Algorithm 5 is $O(N_{Obj} * N_z)$, where N_{Obj} is the number of objects and N_z is the maximum number of object's zones. Algorithm 5 constructs $N_{OTA} = |N_z| \times |Objects|$ object timed-automata.

Algorithm 4: Constructing Permissions Timed-Automata

```
input   : Data Structure
output  : A Set of Parametrized Permissions Timed-Automata "PTA"
$PTA = \phi$;
foreach $p$ in Permissions do
    $i = 0$ ;
    foreach $z$ in $Pzones(p)$ do
        if $used\_roles(p, z) \neq \phi$ // permission $p$ is used by some roles in zone $z =< s, d >$
        then
            $S_i = ZLoc(Z)$ ;
            Let: Timed-Automata $PTA(S_i) = \langle L, L_0, C, A, E, I \rangle$;
            Initialization: $I = \phi, L_0 = \{UnacquiredInS_i\}, C = \phi, A = \phi, E = \phi$;
            $L = L \cup \{UnacquiredInS_i, AcquiredInS_i\}$;
            $A = A \cup \{access\_p[S_i]?, exit\_p[S_i]?\}$;
            $m = 0$;
            foreach $Obj_m \in PermObjs(p, z)$ do
                $commit(Aobj_m, Eobj_m)$;  // commit function marks these nodes as "Committed" nodes
                $L = L \cup \{Aobj_m, Eobj_m\}$;
                $A = A \cup \{access\_obj_m[S_i]!, exit\_obj_m[S_i]!\}$;
                $m++$;
            end
            if $|used\_roles(p, z)| == 1$ then
                $E = E \cup \{(UnacquiredInS_i, access\_p[S_i]?, \phi, \phi, Aobj_0),$
                    $(Aobj_0, access\_obj_0[S_i]!, \phi, \phi, Aobj_1), (Aobj_1, access\_obj_1[S_i]!, \phi, \phi, Aobj_2),$
                    $\ldots, (Aobj_{m-1}, access\_obj_{m-1}[S_i]!, \phi, \phi, Aobj_m), (Aobj_m, access\_obj_m[S_i]!, \phi, \phi, AquiredInS_i)\}$;  // acquiring and
                    access edges
                $E = E \cup \{(AcquiredInS_i, exit\_p[S_i]?, \phi, \phi, Eobj_m),$
                    $(Eobj_m, exit\_obj_m[S_i]!, \phi, \phi, Eobj_{m-1}), (Eobj_{m-1}, exit\_obj_{m-1}[S_i]!, \phi, \phi, Eobj_{m-2}),$
                    $\ldots, (Eobj_1, exit\_obj_1[S_i]!, \phi, \phi, Eobj_0), (Eobj_0, exit\_obj_0[S_i]!, \phi, \phi, UnaquiredInS_i)\}$; // unacquiring and exit edges
            end
            else
                Let $g$, a guard which is a conjunction of predicates and boolean expressions involving clocks ;
                $Cont\_p[S_i] = 0$;
                $u = Cont\_p[S_i]++$ ;
                $u' = Cont\_p[S_i]--$ ;
                $g = Cont\_p[S_i] > 1$ ;
                $g' = Cont\_p[S_i]--$ ;
                $E = E \cup \{(UnacquiredInS_i, access\_p[S_i]?, \phi, u, Aobj_0), (Aobj_0, access\_obj_0[S_i]!, \phi, \phi, Aobj_1),$
                    $(Aobj_1, access\_obj_1[S_i]!, \phi, \phi, Aobj_2), \ldots, (Aobj_{m-1}, access\_obj_{m-1}[S_i]!, \phi, \phi, Aobj_m),$
                    $(Aobj_m, access\_obj_m[S_i]!, \phi, \phi, AquiredInS_i), (AquiredInS_i, access\_p[S_i]?, \phi, u, AquiredInS_i)\}$;  // acquiring and access
                    edges
                $E = E \cup \{(AcquiredInS_i, exit\_p[S_i]?, g, u', AcquiredInS_i), (AcquiredInS_i, exit\_p[S_i]?, g', u', Eobj_m),$
                    $(Eobj_m, exit\_obj_m[S_i]!, \phi, \phi, Eobj_{m-1}), (Eobj_{m-1}, exit\_obj_{m-1}[S_i]!, \phi, \phi, Eobj_{m-2}),$
                    $\ldots, (Eobj_1, exit\_obj_1[S_i]!, \phi, \phi, Eobj_0), (Eobj_0, exit\_obj_0[S_i]!, \phi, \phi, UnaquiredInS_i)\}$;  // unacquiring and exit edges
            end
            $PTA \longleftarrow PTA \cup \{PTA(S_i)\}$; // adds $PTA(S_i)$ to set PTA
            $i++$;
        end
    end
end
$return(PTA)$;
```

Algorithm 6 for constructing UTA:

Since a user can only be in a certain location at a time, Algorithm 6 constructs one timed-automaton for each user who is assigned to at least one role. For each physical position S_i from which a user can activate roles, control state *UserInS*$_i$ is added to set L. Two edges are added to connect initial node *UserElsewhere* and node *UserInS*$_i$. g and g' are conjunction of boolean expressions forming guards needed to activate and deactivate a role in position S_i.

The structure of a user timed-automaton at each *UserInS*$_i$ node differs based on the authorized roles and activation constraints. At node *UserInS*$_i$, self-lop activation and deactivation edges are added to set E. These edges are primarily fired on the satisfaction of temporal guards. Additionally, roles available at node *UserInS*$_i$ might have pre-requisite and DSoD constraints that should be

Algorithm 5: Constructing Objects Timed-Automata

```
input   : Data Structure
output  : A Set of Parametrized Objects Timed-Automata “$OTA$”
1  $OTA = \phi$;
2  foreach $obj$ in $Objects$ do
3      $i = 0$ ;
4      foreach $z$ in $Objzons(obj)$ do
5          if $ObjectPerms(obj, z) \neq \phi$ // object $obj$ is accessed by permissions in location $S_i$
6          then
7              $S_i = ZLoc(z)$;
8              Let: Timed-Automata $OTA(S_i) = \langle L, L_0, C, A, E, I \rangle$;
9              Initialization: $I = \phi, L_0 = \{UnaccessedInS_i\}, C = \phi, A = \phi, E = \phi$;
10             $L = L \cup \{UnaccessedInS_i, AccessedInS_i\}$;
11             $A = A \cup \{access\_obj[S_i]?, exit\_obj[S_i]?\}$;
12             $m = 0$;
13             if $|ObjectPerms(obj, S_i)| == 1$ then
14                 $E = E \cup \{(UnaccessedInS_i, access\_obj[S_i]?, \phi, \phi, AccessedInS_i),$
                   $(AccessedInS_i, exit\_obj[S_i]?, \phi, \phi, UnaccessedInS_i)\}$;
15             end
16             else
17                 Let $g$, a guard which is a conjunction of predicates and boolean expressions involving clocks ;
18                 $Cont\_obj[S_i] = 0$;
19                 $u = Cont\_obj[S_i] + +$;
20                 $u' = Cont\_obj[S_i] - -$;
21                 $g = Cont\_obj[S_i] > 1$;
22                 $g' = (Cont\_obj[S_i] == 1)$;
23                 $E = E \cup \{(UnaccessedInS_i, access\_obj[S_i]?, \phi, u, AccessedInS_i), (AccessedInS_i,$
                   $access\_obj[S_i]?, \phi, u, AccessedInS_i), (AccessedInS_i, exit\_obj[S_i]?, g, u', AccessedInL_)$,
                   $(AccessedInS_i, exit\_obj[S_i]?, g', u', UnaccessedInS_i)\}$;
24             end
25             $OTA \longleftarrow OTA \cup \{OTA(S_i)\}$; // adds $OTA(S_i)$ to set OTA
26             $i + +$;
27         end
28     end
29 end
30 $return(OTA)$
```

satisfied too. In the case of activating a senior role, activation and deactivation *committed* nodes as well as edges connecting them to $UserInS_i$ are added for each junior role.

The time complexity of Algorithm 6 depends on the number of users, roles, roles’ zones and depth of RH. Algorithm 6 complexity is $O(N_u * N_{ur} * N_z * N_{rh})$, where N_u is the number of users, N_{nr} is the number of assigned roles, N_z is the number of role’s zones, and N_{rh} is the number of junior roles. Algorithm 6 constructs N_{UTA}, $N_{UTA} = |Users|$.

Algorithm 6: Constructing Users Timed-Automata

```
input    : Data Structure
output   : A Set of Users Timed-Automata: "UTA"

$UTA = \phi$ foreach $u \in Users$ do
    Let: Timed-Automata $UTA_u = \langle L, L_0, C, A, E, I \rangle$;
    Initialization: $L = \phi, I = \phi, C = \{t\}, l = \phi, A = \phi, L_0 = \{UserElsewhere\}$, $t$ is a global clock ;
    Let $g$, and $g^{'}$ two guards which is a conjunction of boolean expressions involving clocks ;
    $Addedzones = \phi$; // refers to the set of zones that has already created a node for it in user $u$ timed-automaton
    foreach $r \in assigned\_roles(u, z)$ do
        $i = 0$;
        if $can\_ActivateRole(u, r, z)$ // user $u$ can activate role $r$ in zone $z =< s, d >$
        then
            if $z \notin Addedzones)$ // zone $z$ has not yet defined for user $u$
            then
                $Addedzones \longleftarrow Addedzones \cup \{z\}$;
                $S_i = ZLoc(z)$ ;
                $L = L \cup \{UserInS_i\}$;
                $g = x >= firstInst(Instant(ZInt(z)))$;   // gives first time instance in instances of a time interval
                $E = E \cup \{(UserElsewhere, \phi, g, \phi, UserInS_i)\}$;
                $A = A \cup \{activate\_r[S_i]!, deactivate\_r[S_i]!\}$;
                $g = leave(u, S_i)$, is evaluated to true if user $u$ can leave location $S_i$;
                $E = E \cup \{UserInS_i, \phi, g, \phi, UserElsewhere\}$;
                $I = I \cup \{(UserInS_i \rightarrow x < lastInst(Instant(ZInt(z))))\}$ ;   // associate an invariant with node $UserInS_i$
            end
            $g = is\_inactive(u, r, S_i)$; // evaluates that $u$ has not activated $r$ in location $S_i$
            $g^{'} = is\_active(u, r, S_i)$; // evaluates that $u$ has activated $r$ in location $S_i$
            $g^{'} = t < lastInst(Instant(ZInt(z)))$; |; |; // gives last time instance in instances of a time interval
            if $post\_EnableRoles(r, z) \neq \phi$ // role $r$ has post-requisite roles
            then
                $g^{'} = g^{'} \wedge formPostGurad(u, post\_EnableRoles(r, z))$; // role $r$ should not be deactivated while post-requisite
                    role are still active in $z$ zone
            end
            else
                $g^{'} = g^{'} \wedge formDeactvateGuard(r, S_i)$; // user $u$ is only allowed deactivate active roles
            end
            if $pre\_EnableRoles(r, z) \neq \phi$ // role $r$ has spatio-temporal activation pre-requisite constraint
            then
                $g = g \wedge formPreGuard(r, z))$ ;
                $E = E \cup \{(UserInS_i, activate\_r[S_i]!, g, u, UserInS_i), (UserInS_i, deactivate\_r[S_i]!, g^{'}, u, UserInS_i)\}$;
            end
            else if $dsod\_Roles(r, z) \neq \phi$ // role $r$ is in $DsD$ with other roles
            then
                $g = g \wedge formDSoDGuard(r, z))$ ;
                $E = E \cup \{(UserInS_i, activate\_r[S_i]!, g, u, UserInS_i), (UserInS_i, deactivate\_r[S_i]!, g^{'}, u, UserInS_i)\}$;
            end
            else if $RH_A(r, z) \neq \phi$ // role $r$ is a senior role in role activation hierarchy relation
            then
                $j = 0$;
                foreach $r_j \in junior_A(r, z)$ do
                    if $can\_ActivateRole(r_j, z)$ then
                        $A = A \cup \{activate\_r_j[S_i]!, deactivate\_r_j[S_i]!\}$;
                        $commit(Ajunior\_r_j, Djunior\_r_j)$;   // commit function marks these nodes as "Committed" nodes
                        $L = L \cup \{Ajunior\_r_j, Djunior\_r_j\}$;
                        $j + +$;
                    end
                end
                if $|junior_A(r, z)| = 1$ then
                    $E = E \cup \{(UserInS_i, activate\_r[S_i]!, g, u, Ajunior\_r_0)$,
                    $(Ajunior\_r_0, activate\_r_0[S_i]!, \phi, u, UserInS_i)$,
                    $(UserInS_i, deactivate\_r[S_i]!, g^{'}, u, Djunior\_r_0)$,
                    $(Djunior\_r_0, deactivate\_r_0[S_i]!, \phi, u, UserInS_i)\}$;
                end
                else
                    $E = E \cup \{(UserInS_i, activate\_r[S_i]!, g, u, Ajunior\_r_0)$,
                    $(Ajunior\_r_0, activate\_r_0[S_i]!, \phi, u, Ajunior\_1)$, $(Ajunior\_1, activate\_1[S_i]!, \phi, u, Ajunior\_2)$,
                    $\ldots$ $(Ajunior\_j - 1, activate\_j - 1[S_i]!, \phi, u, Ajunior\_j)$,
                    $(Ajunior\_j, activate\_j[S_i]!, \phi, u, UserInS_i)$,
                    $(UserInS_i, deactivate\_r[S_i]!, g^{'}, u, Djunior\_r_J)$,
                    $(Djunior\_r_j, deactivate\_r_j[S_i]!, \phi, u, Djunior\_r_{j-1})$,
                    $(Djunior\_r_{j-1}, deactivate\_r_{j-1}[S_i]!, \phi, u, Djunior\_r_{j-1})$, $\ldots$ ,
                    $(Djunior\_r_1, deactivate\_r_1[S_i]!, \phi, u, Djunior\_r_0)$,
                    $(Djunior\_0, deactivate\_r_0[S_i]!, \phi, u, UserInS_i)\}$;
                end
            end
            else
                $E = E \cup \{(UserInS_i, activate\_r[S_i]!, g, u, UserInS_i)$,
                $(UserInS_i, deactivate\_r[S_i]!, g^{'}, u, UserInS_i)\}$;
            end
            $i + +$;
        end
    end
    $UTA \longleftarrow UTA \cup \{UTA_u\}$; // adds $UTA_u$ to set UTA
end
$return(UTA)$;
```

4.5.4 Alleviating The State-Space Explosion Problem

It is a well-known fact that the state-space explosion is an perpetual problem in model checkers. In our case, the number of temporal conditions increases the state-space size. Here, we present some techniques for reducing the likelihood of the state-space explosion problem.

Analysis Optimizations

UPPAAL supports a number of optimization techniques that are useful for improving the analysis performance. Our approach makes use of these optimization techniques and follows a set of design recommendations described in [31]. With a single clock the state-space size is significantly reduced. Resting variables when they are no longer needed reduces the state-space and the occurrence of deadlocked states.

We make part of the model executing in atomic steps (i.e., atomicity) in order to reduce the number of interleaving edges. The *committed* nodes are used to achieve the atomicity. Atomicity is also important to model concurrent actions such as role triggers. Urgent channels also decrease the number of interleaving paths in state-space and model actions that should be taken without delay such as deactivating roles.

Information Encoding

To carry out the verification of a policy in UPPAAL, we use settings supporting the optimization of state-space. We utilize these parameters: State-space Representation- use Difference Bound Matrix (DBM) to store reachable states which increases the memory usage, State-space Reduction- aggressive option that reduces the number of states stored in the memory, Search Option- Search Order-breadth first order for a quick search in the state-space, Trace Option- by default produces symbolic traces, and 16 MB hash table size.

Temporal Properties Refinement

A careful design structure can decrease the state-space size. Our modeling pattern has many supporting facts related to the intuitiveness, motivation, and state-space optimizations. Typically, the number of temporal conditions drastically expand the state-space while the number of processes (i.e., timed-automata at run time) with non-temporal variables do not considerably impact on the state-space size. Most of the security queries can be verified in almost a constant time irrespective of the number of non-timed processes. Therefore, our design decision was reducing the number of temporal constraints as possible.

In our analysis approach, temporal conditions are mainly confined in the Observer and User timed-automata. Furthermore, the number of these timed-automata is usually limited; a single timed-automaton is constructed for each user and only one Observer automaton is used. Our model has a larger number of timed-automata with no timed edges and nodes. That is , the parametrized timed-automata for roles, permissions, and objects are designed without temporal guards and their temporal behaviour are captured by the Observer and User timed-automata.

Sub-Models Verification

Analyzing the entire network of timed-automata model for a particular property has a negative impact on the analysis performance and it might cause the state-space explosion problem. In such case, the UPPAAL engine explores a large number of paths and states in order to check a property. Moreover, the state-space of the network of timed-automata model has some states and paths that are not dependent on a property in question. Thus, exploring such paths and states unnecessarily requires excessive searching time.

Therefore, we propose a technique, which we refer to as the *entities–dependability* technique, for reducing the state-space size. Our reduction pattern replaces the problem of verifying a large system to hopefully feasible abstracted sub-systems. It generates a sub-automata model from the network of timed-automata models in order to verify a certain property. The resulting sub-automata model has only the dependent entities that are pertinent for checking a property. Thus, the state-space of the sub-automaton model has a subgroup of states and paths that are important to search in order to know whether a property holds or not.

Our reduction approach is formally defined as follows. For analyzing a policy, we need to check a set of properties, i.e., $P = \{p_1, p_2, \ldots, p_n\}$ where n is the number of properties in time-automata model $\mathbf{M}$ representing the policy. Each property in set P checks the correctness of a certain relationship between some entities. For example, property p_i may check the correctness of user-role assignment in certain spatio-temporal zones. If model $\mathbf{M}$ satisfies all the properties in set P, i.e., $p_1 \wedge p_2 \wedge \cdots \wedge p_n$, then we say that the policy is free from ambiguities, otherwise we investigate model $\mathbf{M}$ with respect to the properties that do not hold.

With our approach, for verifying properties in P, a number of sub-automaton models are derived from model $\mathbf{M}$ based on the set of properties in P. For example, for verifying property p_i, sub-automata model $\mathbf{M}_i$ is instantiated which has all the timed-automata processes related to property p_i. Therefore, the UPPAAL engine explores the state-space of sub-automaton model $\mathbf{M}_i$ instead of model $\mathbf{M}$. In case all the properties are satisfied in some sub-automaton models, then the policy is considered consistent. In case one of the properties does not hold, the sub-automaton model concerning that property is examined.

Algorithm 7 forms sub-automaton model $\mathbf{M}_i$ for verifying property p_i in timed-automata model $\mathbf{M}$. Lines 1-10 construct model $\mathbf{M}_i$ from timed-automata $\mathbf{M}$ for each entity set in the *entSet* set. The $entities(p_i)$ function returns all entities' sets involved in property p_i. For example, if property p_i investigates the correctness of user-role assignment, the sets (i.e., *ent*) in *entSet* are Roles and Users. The $gets()$ function copies some timed-automata from model $\mathbf{M}$ to model $\mathbf{M}_i$ in the regard of property p_i, and the $TA()$ function returns timed-automata of an entity in the *ent* set. For example, if the entity *el* is role r_0, then the $TA()$ function gives a number of timed-automata concerning about the behavior of role r_0 in different zones.

Lines 11-22 updates the edges and temporal-guards in timed-automaton m' in $\mathbf{M}_i$ to conform property p_i. Edges that synchronize m' with timed-automata that are not in $\mathbf{M}_i$ are removed and m' is reconstructed in lines 11-21. Functions $remove()$ and $reconstruct()$ respectively deletes edges connecting to unwanted timed-automata and then reconfigures timed-automata in M_i. The $sending()$ function determines the timed-automaton initiating the actions that change the states of the $\mathbf{M}_i$ model. Function $guard()$ adds guards and invariants to the sending m' where are needed. Once Algorithm 7 terminates, it produces $\mathbf{M}_i$ concerning about checking property p_i.

Algorithm 7: Constructing sub-Automaton Model $\mathbf{M}_i$ for Property p_i.

```
    input  : Timed-Automata model $\mathbf{M}$, property $p_i \in P$
    output: sub-Automaton model $\mathbf{M}_i$
1   $autom(M_i)$; // sub-routine creates timed-automata model $\mathbf{M}_i$
2   $M_i \longleftarrow gets(O'TA, M)$; // returns O'TA from $\mathbf{M}$ and add it to $\mathbf{M}_i$
3   $entSet = entities(p_i)$; // extracts the entities sets involved in the property $p_i$
4   foreach $ent \in entSet$ // for every entity set in entSet
5   do
6       foreach $el \in ent$ // for any element $el$ in the set $entSet$
7       do
8           $M_i \longleftarrow gets(TA(el, M), M)$; // adds timed-automata of entity $el$ to $\mathbf{M}_i$
9       end
10  end
    // at this step sub-model $\mathbf{M}_i$ is created
11  foreach $m \in M$ and $m \notin M_i$ do
12      foreach $m' \in M_i$ do
13          foreach $e \in connect(E', E)$ // $E$ and $E'$ are set of edges in $m$ and $m'$
14          do
15              $remove(e, E')$; // deletes the edge $e$ from $E'$
16              $reconstruct(E', L', m')$ // updates the automaton $m'$
17          end
18      end
19  end
20  foreach $m' \in sending(M_i)$ do
21      $guard(g, E')$ // creates temporal guards and invariants to $m'$
22  end
23  $return(M_i)$;
```

Example:

The following example helps to illustrate the reduction technique. Consider a simple timed-automata model $\mathbf{M}$ for a DDSS policy in the left rounded rectangle of Figure 4.12, $M = UTA_{Tom} \parallel RTA_{CMM} \parallel PTA_{p_{10}} \parallel OTA_{obj_6} \parallel O'TA$. In this model, role CMM can only be activated by user Tom in time interval *[t_1 - t'_1] = [8:00 am - 5:00 pm]* and from node *CityWarehouse*. At the moment role CMM is activated, permission p_{10} is acquired and object obj_6 is accessed.

Suppose now a policy verifier tends to check property p_1 that validates the correctness of the spatio-temporal relation between role CMM and permission p_{10} in the $\mathbf{M}$ model. Following Algorithm 7, sub-automata model $\mathbf{M}_1$ is derived from $\mathbf{M}$ as shown in the right rounded rectangle of Figure 4.12. In this sub-model, timed-automata UTA_{Tom} and OTA_{obj_6} are excluded from sub-model $\mathbf{M}_1$. Furthermore, the edges and guards are also reconstructed for RTA_{CMM} and $PTA_{p_{10}}$ in a consequence of removing the user and object timed-automata. Note that, the observer timed-automaton remains intact because it is not directly related to the required changes for checking property p_1. Once model $\mathbf{M}_1$ is constructed, a number of queries representing property p_1 are checked against $\mathbf{M}_1$ using UPPAAL. In the subsequent section, we will evaluate the effectiveness of our approach in terms of the number of states and searching time.

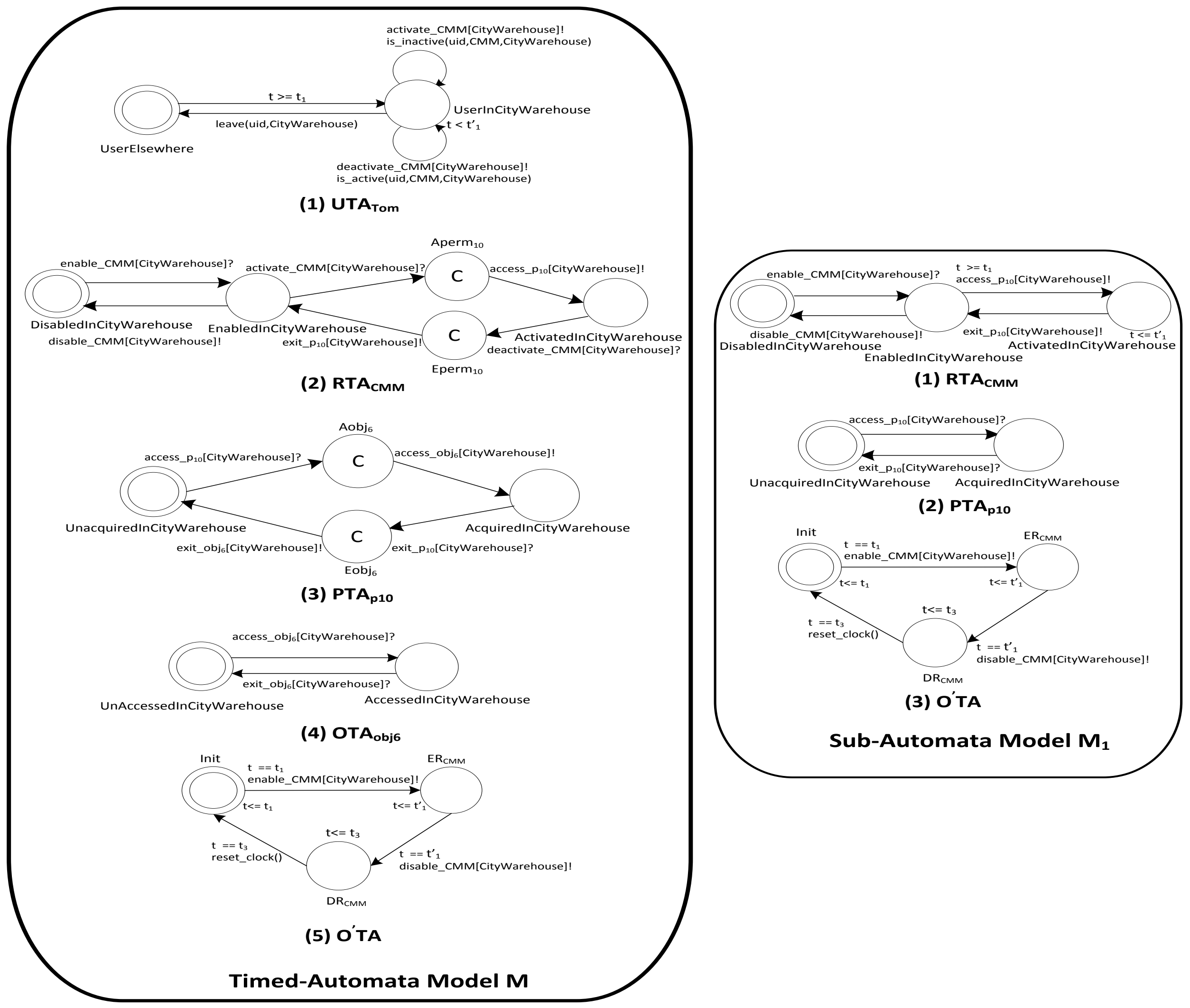


Figure 4.12: State-Space Reduction Example

4.6 DDSS Policy Analysis

In the analysis of the DDSS policy, we apply our algorithms in Section 4.5.3 to construct timed-automata model **M**. **M** is composite from a number of timed-automata operating in parallel, i.e., $\mathbf{M} = \mathbf{M}_0 \parallel \mathbf{M}_1 \parallel \mathbf{M}_2 \parallel \cdots \parallel \mathbf{M}_n$, where n is the number of timed-automata. The total number of timed-automata in **M** is computed by $N_{TA} = N_{RTA} + N_{PTA} + N_{OTA} + N_{UTA} + 1.$, where, $N_{UTA} = 7$ for users automata, $N_{RTA} = 16$ for roles, $N_{PTA} = 17$ for permissions, $N_{OTA} = 6$ for objects, and 1 for $O'TA$, in total, $N_{TA} = 47$ automata in **M**.

The partial UPPAAL data types and functions of the DDSS timed-automata model **M** is shown in Appendix B. The verification is carried out on the Windows platform using 4GB RAM with Intel(R) Core(TM) 2Due CPU. Furthermore, our optimization techniques in Section 4.5.4 are applied for improving the analysis performance.

We first simulate model **M** using UPPAAL to gain confidence that it behaves as anticipated and it has no deadlocked states. In the model state-space, the execution path might halt in a deadlocked state and it does not reach to the final/accepted state in the path. Such a problem usually occurs in model checking approaches because of incorrect specifications of a system. For example, the execution cannot continue in a path that has a role remains in *Active* state and never goes back to the *Inactive* state even when no user is using it. UPPAAL has the advantages over existing RBAC model checking approaches as it provides a built-in query (*deadlock*) to automatically uncover deadlocked states in the model state-space, and as such, eliminates human involvement overhead.

Once the simulation reveals no problems, the UPPAAL verifier is utilized to check whether the DDSS policy satisfies some properties or not. Here, a number of sub-automata models are instantiated using Algorithm 7 for checking those properties. For each property, certain queries are written using *TCTL* and checked against a sub-automata model concerning about that property. It is important to note that all the queries are checked in UPPAAL with contiguous time intervals. In other words, UPPAAL records the events of the DDSS system for each single clock tick and creates the timed state-space for all model states.

Along the verification of queries, the UPPAAL verifier also shows the memory space usage (or number of states explored) and the time needed to explore these states in order to answer each query. Our next goal is to specify some spatio-temporal properties in *TCTL* and verify them. These properties are represented as reachability, safety, and liveness queries.

```
(Q)uery1: A[] not deadlock
```

Query 1 is a safety query that checks for deadlocked states. The query fails at a deadlocked state in which role VCT halts in the active state and never goes to the successor inactive state. The counter example trace shows that the action of deactivating VCT role sometimes interleaves with other actions, causing that the deactivation transition is not taken at the moment user Yue leaves the patient premise in the city or the activation time expires.

This means that the policy does not enforce Yue or the system to deactivate VCT role at the time the role zone is violated. To fix this inconsistency, we define the action of deactivating role VCT as an urgent action that must be taken at the time it is enabled. Thus , it will be given a high priority over the other transitions when they are interleaving. An alternative solution is to associate the *Activated* nodes in VCT timed-automata with an invariant. This invariant enforces the enabled deactivation transition to be taken once the clock valuation is not satisfied. In this manner, all deactivation actions of roles are defined as urgent actions to express the enforcement of immediate roles deactivation in invalid zones.

```
(Q)uery2: A[] CMM(MainWarehouse).Enabled imply (x>=8 && x < 17)
```

Query 2 is a positive safety property that indicates a role can only be enabled in a favourable zone. It checks that role CMM is only enabled in valid zone z_7. The verification result shows that the property is satisfied in all paths and at any state.

```
(Q)uery3: A<> SE(StateEpo).Activated and (x>=8 && x=<17)
```

Query3 is a general liveness query guarantees that the DDSS system is doing something good as well. The query expresses that a role should eventually be activated at favourable zones. We checked that role SE can eventually be activated in zone z_4.

```
(Q)uery4: CMM(CityWarehouse).Enlabled && x>=8 -->
          CMM(CityWarehouse).Disabled && x>17
```

Query 4 specifies a bounded liveness property for role enabling. The query is satisfies that once role CMM is correctly enabled, it will be eventually disabled in location *CityWarehouse* before time instant 17.

```
(Q)uery5: A[] SHC(StateClinic).Activated && SHC_u[Alice.uid]== 1 imply
        Alice.UserInStateClinic and (x>=8 and x<=17)
```

Query 5 represents a positive safety property of spatio-temporal constraints on user-role activation. It explicitly checks that role SHC is activated by Alice inside the *StateClinic* location and during the day-time. The result shows that the property is satisfied along all baths and states.

```
(Q)uery6: E<> p2(CityClinic).Acquired and (x>=8 and x<=17)
```

Query 6 applies a reachability query of permission access in some favourable zones. The query asks whether or not permission p_2 can be acquired by a role in favourable zone z_3. The query result shows that there are some states (at least one) where permission p_2 is accessed. This security requirement is very important to check because a permission might never used. For permissions: p_5, p_6, p_8, p_9, p_{11}, p_{14}, and p_{15}, such query is not satisfied because these permissions are not assigned to roles. In such case, a security officer should exclude/assign these permissions in the policy.

```
(Q)uery7: E<> obj2(CityClinic).Accessed and
          (p2(CityClinic).Acquired and (x> 8 && x<17))
```

The query specifies a reachability property for the correct object access in some favourable zones. The query checks whether object obj_2 can be accessed by permission p_2 in zone z_3. This property is satisfied, the object is correctly accessed.

```
(Q)uery8: A[] forall(i:uid)(CMM(City).Activated
        and CMM_u[i] == 1) and forall(j:uid)(CVM(City).Activated
        and CVM_u[j] == 1) and (x >= 8 && x < 17)imply  i !=  j
```

Query 8 is a safety property that checks for the violations of DSoD constraint. The query search for the states that violate the DSoD constraint between CMM and CVM roles in zone z_{10}. The results shows that at all reachable states, DSoD constraint is satisfied.

```
(Q)uery9: E<> SE(CityEpo).Activated and SE_u[Clair.uid] == 1 and
          CE(CityEpo).Activated and (CE_u[Clair.uid]==1
          and (x>=8 and x<=17))
```

Query 9 applies the reachability properties to spatio-temporal constraints on RH_A. It checks the activation of junior role CE of senior role SE in zone z_5. The policy specifies that whenever Clair activates SE, she is allowed to activate CE. This property is satisfied at some reachable states. From this query, another important security property can be also derived, senior role SE must not be deactivated while junior role CE is in active state.

```
(Q)uery10: VCT(City).Enabled --> (CMM(CityWarehouse).Enabled
           and (x>=8 && x<=17))
```

This bounded liveness query checks for the correctness of the role enabling triggers in some favourable zones. The property of interest is that the enabling of the VCT role to perform a certain control task triggers the enabling of role CMM to provide the required materials, the property is satisfied.

Table 4.3 compares the average number of the explored states and the consumed time for checking quires: Q_2, Q_3, Q_4, . . . , and Q_{10}, for the entire policy model **M** versus the sub-automata models that are generated by Algorithm 7. In our analysis scenario, we have four models, timed-automata model **M** and three sub-automata models: $\mathbf{M}_1$ for checking the interaction between users and roles, $\mathbf{M}_2$ for verifying the roles and permissions relationships, and $\mathbf{M}_3$ concerning about the access to objects via permissions.

Table 4.3: Evaluation of the State-Space Reduction Technique

Properties	**Timed-Automata Model**		**Sub-Automata Models**		**Search-Space Reduction %**	
	Explored States	**Elapsed Time**	**Explored States**	**Elapsed Time**	**States%**	**Time%**
Q_2 :**M vs M**$_1$	*95616*	*7,252ms*	*5184*	*749ms*	*94%*	*89%*
Q_3 :**M vs M**$_1$	*30*	*156ms*	*5*	*73ms*	*83%*	*53%*
Q_4 :**M vs M**$_1$	*26*	*628ms*	*4*	*102ms*	*85%*	*84%*
Q_5 :**M vs M**$_1$	*95616*	*5,703ms*	*5184*	*853ms*	*94%*	*85%*
Q_6 :**M vs M**$_2$	*1019*	*230ms*	*1*	*12ms*	*99%*	*94%*
Q_7 :**M vs M**$_3$	*1028*	*175ms*	*1*	*11ms*	*99%*	*93%*
Q_8 :**M vs M**$_1$	*95616*	*7.018ms*	*5184*	*644ms*	*94%*	*90%*
Q_9 :**M vs M**$_1$	*622*	*193ms*	*30*	*79ms*	*95%*	*59%*
Q_{10} :**M vs M**$_1$	*95616*	*6,968ms*	*5184*	*866ms*	*94%*	*87%*
Reduction Average					*93%*	*81%*

Sub-automata model $\mathbf{M}_1$ has timed-automata for users (7 timed-automata) and roles (16 timed-automata), in total, it is 24 automata including the observer automata. Thus, the number of automata in $\mathbf{M}_1$ is reduced to *48%* from the number of automata in **M**. $\mathbf{M}_2$ includes on 34 timed-automata, 16 automata for roles, 17 automata for permissions, and the observer automata. The number of automata in $\mathbf{M}_2$ is *28%* smaller than the number of automata in **M**. $\mathbf{M}_3$ has 17 permissions automata, 6 objects automata, and the observer automata, thereby in total 24 automata. The number of automata in $\mathbf{M}_3$ is reduced to *49%* in comparison to the number of automata in

M. Table 4.3 shows that the average number of explored states is reduced by *93%* and the average elapsed time is decreased by value of *81%* in the sub-models in a comparison to check those queries against the policy model **M**. Consequently, these values demonstrate the effectiveness of the proposed reduction technique in verifying spatio-temporal properties in our model.

Chapter 5

The Enforcement Mechanism of Our Models

Chapter 3 describes how a spatio-temporal policy is formally specified using our UML/OCL model, but the practical viability of our model need to be studied to ensure that it provides the required level of access control. That is, our access control model defines policies from high-level perspective, and enforcement mechanisms describe a useful implementation architecture. In the pertinent literature, there are little works that have considered the enforcement of spatio-temporal RBAC policies in real-world applications. Most of existing researches on RBAC focus more on modeling and analysis of access control models. Such models might be either error-prone or not flexible to configure when used. We believe that the developments of new applications and RBAC models have given rise to a number of interesting implementation problems.

Therefore, a policy enforcement mechanism of our model predicts potential challenges and bottleneck performances with hints to possible answers. For example, an architecture model can inform us how systems are configured to implement access control policies and what design decisions make our model more efficient to use. Additionally, implementing our model in a prototype system and analyzing the access decisions response time provide a natural way to evaluate the suitability of the model. With respect to this aspect, our third contribution in this dissertation focuses on addressing a primary problem of enforcing spatio-temporal access controls in the mobile environment. We introduce a platform-independent architecture model for designing applications enforcing a spatio-temporal policy specified by our model.

This chapter describes the proposed enforcement mechanism of our access control models in mobile applications. Section 5.1 introduces an implementation architecture; we discuss the security guarantees of our architecture in Section 5.3; Section 5.2 describes a number of protocols used by our architecture; and Section 5.5 discusses an empirical study on a prototype implementation of the proposed architecture.

5.1 Software Architecture Model

In the previous sections, we addressed the high-level definitions and formal analysis of GSTRBAC polices. Yet, we did not consider the enforcement mechanism of the policy at the application level. This section describes a platform-independent implementation architecture, which maps the high-level GSTRBAC policy definition to the enforcement mechanism in mobile applications.

In our implementation architecture, some elements of the usage control model $UCON_{ABC}$ [56] are integrated into GSTRBAC model for checking access requests. $UCON_{ABC}$ is a family of models describe a number of methods for checking access requests. The pre-authorization (preA) in $UCON_{ABC}$ indicates that the resource access verification must be performed before the requested resource is exercised. The ongoing-authorizations (onA) defined by $UCON_{ABC}$ indicates that a system continue to enforce access control while the resource is being accessed.

Since our access control model supports the spatiality and temporality access control, the enforcement of the GSTRBAC policy introduces a number of challenges; we mainly focus on two of them. First, the integrity of the current user STZone should be maintained. A system should authenticate a user claim of his STZone by a secure manner before processing the access request. Second, capturing the user STZone changes after approving a user access request. When a current user STZone becomes invalid (i.e., due to the locations changes or the access duration ends on), all granted user's privileges should be revoked immediately.

The first challenge is addressed by the STZone-proof provided by a trustworthy STZone reader component planted in a user's mobile computer. For each access request, the STZone reader component at the client side constructs the STZone-proof and timestamp at which the proof is issued. The second challenge is addressed by enforcing the ongoing-authorizations (onA) of $UCON_{ABC}$. We define a listener component at the user's mobile that gets notifications at the time the current mobile's STZone deviates from the valid zone associated with a user mobile. The listener component requests the service termination at the time a user moves into an invalid STZone.

Figure 5.1 depicts the proposed implementation architecture for enforcing GSTRBAC in a mobile application. Typically, the design is developed with aim of having a feasible tradeoff between security, functionality, and ease-of-use. Users might become disgruntled with the heavy security constraints complicating their work for no substantial reasons, resulting in bad outcomes for enterprises. From a high-level perspective, the design is based on separating the security policy from point of use as well as granting authorization-tokens (ATs) for managing users' access while in move. The architecture in Figure 5.1, consists of three core components, the request composition module (RCM), resource access module (RAM), and authorization control module (ACM) which are stand alone programs. The request composition module is installed at a user mobile device while the resource access and authorization control modules are implemented in servers which may or may not be colocated depending on the implementation details. The application data and policy data can be kept in a single local database server or maintained in distributed servers. In case the relevant data is not stored locally, the modules must intercommunicate in the form of remotely distributed servers.

Architecture components are either locally or remotely interconnected to data repositories in order to perform their tasks. In the presentation of the proposed implementation architecture, we discuss our design characteristics, the computation and the space capabilities, the architecture modules, and the authorization algorithm.

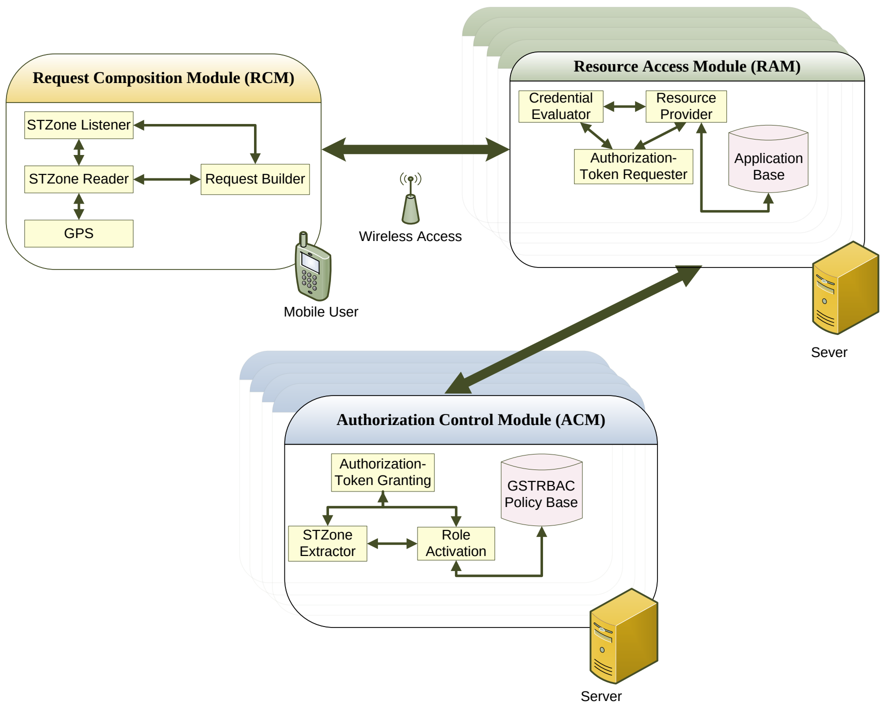


Figure 5.1: Implementation Architecture of GSTRBAC Policy in Mobile Applications

5.1.1 Assumptions

Here are some realistic considerations in a mobile application architecture enforcing a GSTRBAC policy model:

- The access control model is implemented as a separate component linked to the target application to provide a proper access control.
- The architecture implements the client server paradigm, in which the client side sends requests to the server side, and the server processes the requests and sends back the responses.
- Architecture components are implemented by tamper-proof and trusted software. In particular, RCM component is installed in a secure element of a mobile-device, which control access to signed applications. Software product vendors are in charge of using up-to-date preventive techniques to protect software data and functionality across multiple applications.
- The key aspect STZone is implemented as object information; its attributes are the location and time information needed for checking a user access.
- A public key cryptography and a hash algorithm *MD5* should be installed in users' mobile phones as well as RAM and ACM servers. These algorithms at the entire architecture components must be identical. The design decision of employing the public key cryptography in preference to the secret key scheme is because the public key scheme has many advantages for building secure authentication protocols. The secret key cryptosystem is faster, but it suffers from many security threats such as replay and reflection attacks on the authentication exchanges.
- The strength of the cryptography and hash algorithms, and the secure distributions of certificates, depend on the implementation details and sensitivity of applications' documents.
- A user public certificate ties a user with a certain mobile device; the mapping between a user and a mobile device is one-to-one relationship. That is, a user cannot use his colleague's cell-phone to access an application.

- The GSTRBAC model is defined based on the assumption that a tamper-proof location system such as GPS or WLAN-based systems [1] is installed in a user mobile device. This assumption is based on the fact that smartphones with a tamper-proof GPS receivers [88] onboard are becoming increasingly common. Most of today's smartphones, such as the iPhone/iOS, Bada, Windows Mobile 7, and Android, are fully GPS-Enabled.
- Users' membership is controlled by strong passwords that are resistant to many password cracking tools. Furthermore, if users store their passwords in their cell-phones, these passwords should be properly encrypted by some keys that are transparently hoarded in a secure element of the cell-phones. Stolen cell-phones are also protected from improper usage of local sensitive information by employing a password protection tool or a biometric measure such as a fingerprint reader.
- Architecture components clocks have proper synchronization within time windows that are controlled by a time synchronization protocol. That is, the clocks are defined so that they synchronously advance at the same pace in each architecture component.
- The data consistency at all architecture components (i.e., distributed RAM and ACM servers) is guaranteed by applying some underlying protocols for consistent data services [89, 90, 91]. These protocols support the data integrity, concurrency control, and recovery from crashes and communication failures across interconnected databases.

5.1.2 Design Characteristics

Before we discuss our implementation architecture model in Figure 5.1, we introduce a number of desired characteristics that our architecture design should satisfy. The following elaborate on each of these design requirements:

- Keeping the architecture as general as possible in order to easily incorporate it in various mobile applications. In practice, it is desirable to have multiple authorization and resource manager servers within mobile applications. The architecture can be implemented in a local centralized paradigm or a distributed environment.

- The design architecture should maintain practical computational and communication efficiency. With computational efficiency, the number of operations required to execute the protocols are kept to the minimum. We reduced the number of cryptographic operations needed to securely exchange messages to the least amount. With communication efficiency, the number of passes (communication steps) for performing the protocols are kept as small as possible. Therefore, our implementation does not make for inconvenience to users and a lowering the targeted application effectiveness.

- In order to improve the performance, the design should reduce the number of components involved in the access control. To fulfil this design goal, our architecture does not relay on third parties to provide the STZone proof for a user, authorization tickets, secrets (e.g., keys or hash values), or any other forms of trusted information. For example, the STZone proof is issued locally by a trusted software component that is embedded in the user device.

- Due to the constraint environment of the mobile devices, the number and type of operations performed at the client side should be small. In our design, most of the computations overheads are placed on the server side which are assumed to have powerful space and computational capabilities. Furthermore, we only allow the operations that consume a small amount of space and computation overhead to be performed at the mobile devices.

- For ongoing/persistent access checking, the continuous STZone confirmation should not drastically increase the communication messages consuming a significant amount of computation power at a user mobile. In our design, the STZone listener component, at the user client side, sends an access termination request if and only if the current user STZone is invalid. In such a way, the user does not need to continuously or periodically sends STZone conformations after some fixed period of time to maintain the access.

- The key feature of our architecture is the separation between the application layer and the policy layer. We design two distinctive modules, one is for the authorization enforcement and another one for making access decisions. This design choice adds flexibility for integrating GSTRBAC policies with various applications. In practice, ACM maps a user to proper roles and permissions. Conversely, RAM enforces the authorizations granted by ACM without any knowledge about the underlying policies. As such, our design protects against insider attackers by isolating the authorization decision from the point of use.

- The users' anonymity must to be preserved. The ACM servers should not learn any information about users' credentials such as the users' devices, passwords, and public keys. Here, users' credentials are only meaningfulness to the resource server to validate the users' registrations in the application. Thus, such information is not forwarded to ACM during the resource access. Furthermore, ACM cannot match the user name with roles since GSTRBAC policies are defined based on users' identifiers.

- Protocols in our architecture should be secure. Our protocols are designed to establish secure connections over insecure channels. As such, these protocols employ various countermeasures protecting against most common security attacks.

- Architecture protocols should have a significant performance while performing access control. This goal is achieved in our protocol design by reducing the number of passes between the RAM and ACM servers. That is, in case a user is not authenticated at a RAM server, the access rejection response is sent directly to a user aside from consulting with a ACM server. In similar manner, the suspension of the user's authorizations while on the move is concisely handled by the RCM and RAM components.

- A user must not be restricted to be in a proximate place such as a room, a department, or a building in order to have an access; these physical quarters may have pre-installed location devices to provide users' locations. In the proposed architecture model, a user can freely move and access resources with no restraints of being a near by (i.e., few centimetres or meters) to a location device signalling his current location.

- The implementation should also consider freezing authorizations of moving users. Users' privileges should not be revoked at the time a user suddenly leaves a valid location since that user might return after a short period of time and before the access duration is timed out. This design objective is defined to tackle the redundancy problem of redoing the role activation steps all over again. Such problem causes pointless resource and time consumption as well as inconvenience for users. Authorization suspicion is primarily one of our techniques for getting better implementation performance. That is, in case a user suddenly leaves a valid position, a client sends a request to freeze or suspend user access for a certain time window computed by the client and properly synchronized with the RAM server. Once the user gets back to his/her valid position, that user can effortlessly resume his work. It is worth mentioning that the ACM component is not involved in the freezing authorization procedure.

5.1.3 Architecture Modules

In the definition of our architecture modules, we incorporate some relevant key points of the general-purpose access control policy language XACML (eXtensible Access Control Markup Language) [78].

Now, we describe the proposed architecture modules in terms of their accountabilities:

- *RCM* - is responsible for forming a user access request and maintaining the access while the rights are being exercised. The *Request Builder* component in RCM creates a resource access request package and includes the current user STZone and user's credentials (e.g., user identifier, password, timestamp, and device identifier) with the request, then sends it to one of the available RAM servers. The *Request Builder* enquires the current user STZone from *STZone Reader*. The *STZone Reader* component in turn reads the current mobile device time and gets the location from GPS data component storing the location information. *STZone Reader* tracks current user's STZone and sends notifications of invalid STZones to the *STZone Listener* component that requests an access termination request to the server side.

- *RAM* - is an intermediate server between a user's mobile and the ACM server, which is primarily responsible for handling the application resources to the users. It typically acts like PEP as it is in charge of communicating with the application resource base. RAM receives the user request on one-hand, and it consults the user's authorizations with ACM server on the other hand. The *Credential Evaluator* component checks user credentials stored in the application base server. The *Authorization-Token Requester* component requests a user's access rights from one of the available ACM servers. The *Resource Provider* component provides access to some object information requested by users.

- *ACM* - is accountable for the policy evaluation and issuing a new authorization-token for every user request. ACM acts as PIP to provide required information for evaluating a user request and as PDP to make an access decision and forward it to RAM. ACM communicates with the policy base in order to issue users' authorizations-tokens. The *Role Activation* component maps a user to an appropriate set of roles and permissions based on its STZone provided by the *STZone Extractor* component. *STZone Extractor* is the component implements the mapping function *T* discussed in Chapter 3. Furthermore, the *Role Activation* updates the policy state for each activated role. The *Authorization-Token Granter* component is the mean that allowed us to concretize the ongoing access rules. The *Authorization-Token Granter* component is responsible for granting the authorization-token for using a particular role. Note that, a user's role can be activated if and only if all the following conditions are satisfied: (a) the role is not already active, (b) the role can be activated in the given STZone, and (c) no conflicting roles are in active state.

The authorization-tokens are like a tickets which equip client applications by a "kind of intelligence" that made them able to manage their own access rights. That is, a user will have an ongoing access till the point in time the current user STZone violates the authorized STZone in the authorization-token that is stored in the user's device. The format of the authorization-token *AT* may varies depending on the implementation details of a system. In our protocols, the user's authorization-token should have the current activated role, a set of authorized permissions, and the current user STZone.

An authorization-token is formally defined as $AT = (ID_u, ID_{ut}, r, P, STZone)$ where ID_u refers to a user identifier, ID_{ut} is a token's identifier, r is a requested role to be activated, P is a set of the authorized permissions associated with role r, and STZone defines where and when these privileges are valid. The set of activated roles by a user and the set of authorized permissions are respectively defined by these functions: $ActiveRoles : (u : Users, z : STZones) \rightarrow 2^{Roles}$, and $AuthorizedPerms : (u : Users, z : STZones) \rightarrow 2^{Permissions}$. With a distributed RAM and ACM paradigm, ACM broadcasts the new user's authorization-token to all RAM servers to keep the access control consistent. In addition, ACM also broadcasts this update to a list of ACM servers to preserve the state of the policy.

For every activation request, ACM grants or denies *authorization-tokens* using Algorithm 8. A user request is approved if and only if the requested role does not violate the spatio-temporal policy rules. If a user requests to activate an unauthorized or a conflicting role, ACM does not issue a new authorization-token for that role, instead, a rejected access response is sent to the user. Correspondingly, for having conflicting permissions, the access request should be rejected. Algorithm 8 descries how the authorization-tokens (ATs) are either computed or denied; it is invoked by ACM for each activation request. As shown in Algorithm 8, mutually exclusive roles are also checked before issuing an new authorization-token.

Algorithm 8 takes in three inputs: user u, role r, and *STZone* z: u is the user who wants to activate role r in zone z. Line 1 initializes the permission P to the null set. The algorithm proceeds only if r has not already been assigned to u in zone z. If the user already has authorization tokens, a check is made to ensure that no conflicting roles or permissions are activated – this is done in Lines 4 – 24. If these conditions are satisfied, the role r can be activated for user u in zone z. The set of active roles for user u in zone z is updated in Line 25 to include the role r. In Line 26, P is set to the permissions associated with role r in zone z. Lines 27 and 28 generate the *authorization-token AT*. The algorithm produces as output the *AT* if the role can be activated, otherwise a reject message is sent to the user.

With distributed and replicated *RAM* and *ACM* servers, *ACM* broadcasts the new *authorization-token* to all the *RAM* and *ACM* servers to keep the access control consistent.

5.1.4 Computational Capabilities and Storage Space

The user mobile device is assumed to operate in a restricted environment in terms of a computational power and a space capacity. Nevertheless, in modern mobile Operating Systems (OSs), the processor computation power is very advanced and the storage space is increasingly extended with permanent memories of 8GB, 16GB, 32GB, etc.

A user's mobile is assumed to execute a public key cryptography algorithm as well as hashing

Algorithm 8: Generate *authorization-token AT*

```
input  : User $u$, STZone $z$, and Role $r$
output: authorization-token $AT$ or Reject
$P \longleftarrow \phi$;
if $r \in active\_roles(u, z)$ // $r$ is already activated by $u$
or $r \notin assigned\_roles(u, z)$ // $r$ is assigned to $u$ in $z$
then
    if $userToken(u) \neq \phi$ // $u$ has some authorization-token
    then
        foreach $r' \in active\_roles(u, z)$ // checks conflicting roles
        do
            if $r' \in RSoD(r, r', z)$ // a role conflict is exist
            then
                print(Request is Rejected + Details);
                exit;
            end
        end
        foreach $p \in assigned\_perms(r, z)$ // checks conflicting permissions
        do
            foreach $p' \in assigned\_perms(r, z)$ do
                if $p' \in PSoD(p, p', z)$ // a permission conflict exists
                then
                    print(Request is Rejected + Details);
                    exit;
                end
            end
        end
    end
    $active\_roles(u, z) \longleftarrow active\_roles(u, z) \cup \{r\}$; // updates role $r$ state
    $P \longleftarrow assigned\_perms(r, z)$ ;
    $ID_{ut} = generateID(AT)$; // generate a sequence number of user's token
    $AT \longleftarrow (u, ID_{ut}, r, P, z)$; // creates authorization-token $AT$
    $return(AT)$;
end
else
    print(Request is Rejected + Details);
end
```

algorithm *MD5*. It is also supposed to store a user's private key PrK_u, the RAM's public key, and a list of authorization-tokens. Such data should not consume a large volume from the memory, and it should be stored in a secure component of the user's smartphone; it should only be reachable by licensed applications.

On the servers' side, the servers' computation power and storage capacity are not restricted since they are usually very powerful machines in comparison with personal computers (PC). Servers are designed to have a high-speed memory, can handle a very high volume input and output (I/O), deal with massive databases, and emphasize high throughputs in order to perform as many computational tasks on behalf of the clients.

A RAM server maintains a list of users' public key certificates denoted by *UCerts*. A user certificate must be deleted from *UCerts* list in case the user's key is compromised. It also preserves a list of authorization-tokens that is referred to as the *UATokens* list. When the authorization-token expires, it must be removed from *UATokens* list. The integrity of this data must be properly maintained from threats such as data is accidentally or maliciously modified, altered, or destroyed.

Since the implementation paradigm supports the distributed environment, RAM also stores a list of public keys certificates of distributed ACM servers referred by the *ACCert* list. Each public key in the *ACCert* list is used to communicate with a particular ACM. In other words, all the connections between the RAM and RCM servers or between the RAM and ACM servers are peer-to-peer connection controlled by designated public keys. RAM should also performs the public key cryptography and hashing.

We do not place a certain storage capability on the ACM servers. Each ACM server only stores its private key and a list of public keys certificates for distributed RAM servers. Since ACM is connected with GSTRBAC policy base, it updates the policy base with each new request.

5.2 Access Control Protocols

We systemically define a number of resource usage protocols incorporating spatio-temporal constraints. These protocols securely govern the way our architecture components interact in order to permit the resource usage under various circumstances; they are fully automated. This section discusses the protocols' prelude and define how these protocols work.

5.2.1 Protocols Prelude

We expect the following essential settings are already carried out in order to operate the protocols. All architecture principles should have agreement on some public keys that are used to encrypt messages and verify signatures. A user must have registered with a system and has created a password. When a user register in a system, a trusted Certificate Authority (CA) issues a pair of public and private keys for that user. A user's certificate has a user's public key PuK_u, a user's smartphone identifier ID_s, user identifier ID_u, and user password P_u. The Certificate Authority (CA) must have already deployed each RAM's public key certificate to every ACM and RCM entities in a system, and similarly for ACM servers.

The communication passes are controlled by timestamps that record the time at which a protocol exchange is created. Therefore, the protocol endpoints should have already adjusted their clocks to a certain amount of synchronization throughout a time management protocol. The time should be approximately the same at every interconnecting endpoints in our protocols. The size of the time frame at a recipient endpoint is left up to the application implementers, it might depend on the criticality of the requested resources.

Figure 5.2 describes the communication exchanges of the resource usage protocols implemented by a single RCM client, RAM server, and ACM server. Suffixes associated with the communication messages indicate the order of steps in the protocols. Message $\mathbf{M}_i$ indicates step i of the protocol. Table 5.1 enumerates the notations used in the description of the protocols.

In Figure 5.2, the RAM and ACM servers are implemented in two distinct centralized servers. RAM with application base are executed in a sever denoted by the resource server (RS). ACM along with the policy base are applied in a server that is called the authorization server (AS). The user component RCM is implemented in a mobile device referred to by (RC). The protocols endpoints' identifies are sent in clear in order to allow the receiving endpoints to allocate the senders' key certificates. The channel between user mobile phone and RAM server are wireless network (e.g., GPRS, UMTS, WLAN, Internet WAP 2.0). We do not put any assumptions about the communication channel between RAM server and ACM server, it can be either wired or wireless.

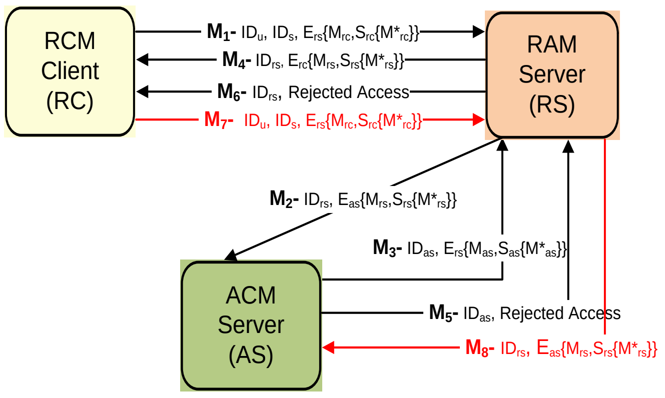

RCM Client (RC)
RAM Server (RS)
ACM Server (AS)
M_1- ID_u, ID_s, $E_{rs}\{M_{rc},S_{rc}\{M^*_{rc}\}\}$
M_4- ID_{rs}, $E_{rc}\{M_{rs},S_{rs}\{M^*_{rs}\}\}$
M_6- ID_{rs}, Rejected Access
M_7- ID_u, ID_s, $E_{rs}\{M_{rc},S_{rc}\{M^*_{rc}\}\}$
M_2- ID_{rs}, $E_{as}\{M_{rs},S_{rs}\{M^*_{rs}\}\}$
M_3- ID_{as}, $E_{rs}\{M_{as},S_{as}\{M^*_{as}\}\}$
M_5- ID_{as}, Rejected Access
M_8- ID_{rs}, $E_{as}\{M_{rs},S_{rs}\{M^*_{rs}\}\}$

Figure 5.2: Communication Steps of the Resource Usage Protocols

An Initial Access Control Protocol

The steps of the basic resource usage protocol for handling users' requests are performed as following:

- **Resource Usage Request** $[RCM \xrightarrow{M_1} RAM]$: A user mobile sends an encrypted and digitally signed access request package M_1 to RAM as shown in Figure 5.2. RCM creates an access

Table 5.1: The Notations of the Resource Usage Protocols

Symbols	**Interpenetration**
ID_x	Identifier of party x
ID_s	User's device identifier
PuK_x	Public key of party x
PrK_x	Private key of party x
Ts_x	Timestamp computed by party x
P_u	User password
P_u^*	One-time password
$H(P_u, Ts_x)$	Computes P_u^*
M_x	Package payload created by party x
$E_x\{S\}$	Encryption of sequence S by PuK_x
M_x^*	Message checksum generated by party x
$H(M_x, Ts_x)$	Computes M_x^*
$S_x\{M_x^*\}$	Signing M_x^* by PrK_x
$A \xrightarrow{M_i} B$	Party A sends package M_i to party B
Tw	Time window
AT	Authorization-token
ID_{ut}	Authentication-token identifier
"Close"	Keyword indicates deletes user's AT
"Freeze"	Keyword indicates suspends user's AT

request payload $M_{rc} = (ID_u, ID_s, P_u^*, STZone, r, Ts_{rc})$, where ID_u is the user identifier, ID_s is the device identifier, $P_u^* = H(P_u, Ts_{rc})$ is the user one-time password, $STZone$ is the current user zone, r is the requested role, and Ts_{rc} is the timestamp at which M_{rc} is created. RCM computes the hash value $M_{rc}^* = H(M_{rc}, Ts_{rc})$ and signs it using user's private key PrK_u, i.e., $S_{rc}\{M_{rc}^*\}$. RCM composes the package M_1 which has ID_u and ID_s in clear. At the resource server, RAM uses ID_u and ID_s to lookup for the corresponding the user's certificate. RAM decrypts M_1 using its private key and verifies the digital signature using user public key PuK_u; it recomputes the message checksum and compares it with the one in M_1. Then, RAM validates the user registration in the application using user ID_u and P_u in his/her certificate.

- **User AT Request** $[RAM \xrightarrow{M_2} ACM]$: If the user is authenticated, RAM forwards an encrypted and signed request package M_2 to the ACM server in order to issue the user's authorization-token AT. RAM forwards the AT request payload $M_{rs} = (ID_{rs}, ID_u, STZone, r, Ts_{rs})$, where Ts_{rs} is the payload's timestamp. RAM computes the hash value and signs it using its private key PrK_{rs}. Then, RAM encrypts the M_{rs} along with digitally signed signature $S_{rs}\{M^*_{rs}\}$ using ACM public key PuK_{as}. At the authorization server, ACM decrypts package M_2 and validates the digital signature using the public key of RAM PuK_{rs}. Nevertheless, if the user authentication fails, RAM sends access rejection response M_6 to the user without the need to communicate with the ACM, and as such, speeds up the implementation performance.

- **User AT Response** $[ACM \xrightarrow{M_3} RAM]$: Once message M_2 is authenticated, ACM replays the user's AT in M_3 package only if the user has the rights to activate a requested role. ACM maps the user ID_u and his/her current user *STZone* with appropriate rights in the policy. Once the user's request is approved, ACM sends an encrypted and signed response acceptance package M_3 to RAM, which includes M_{as} payload as well as a digital signature $S_{as}\{M^*_{as}\}$ signed by ACM private key PrK_{as}. Payload $M_{as} = (ID_{as}, ID_{ut}, ID_u, AT, Ts_{as})$, has user's identifier ID_u, user's authorization-token AT, token identifier ID_{ut}, and timestamp Ts_{as}. However, if the requested role is not approved by the policy, then an access rejection response M_5 is sent to the RAM server.

- **Forwarding User AT** $[RAM \xrightarrow{M_4} RCM]$: After authenticating message M_3 from ACM, RAM stores a copy of the user's authorization-token AT in the *UATokens* list along with the token identifier ID_{ut}, user identifier ID_u, and device identifier ID_s. Then, RAM forwards a signed and encrypted response package M_4 to RCM. The response package has payload $M_{rs} = (ID_{rs}, ID_u, ID_s, ID_{ut}, AT, Ts_{rs})$ as well as a digital signature $S_{rs}\{M^*_{rs}\}$ signed by the private key PrK_{rs}. Note that the user's authorization-token is bound to particular user ID_u and device ID_s. At user side, if M_4 package from RAM is authenticated, RCM stores the authorization-token AT in a secure element of the user's cell-phone. However, in case the access request is rejected by ACM, RAM directly forwards rejection response M_6 to RCM.

The basic resource usage protocol thwarts a user from having conflicting roles or permissions. The SoD constraints are enforced by ACM. As shown in Algorithm 8, a user is not granted a new token for role r if it conflicts with at least one of the current user's active roles, and also if the associated permissions with role r conflicting with one of the current user's permissions.

An Ongoing Access Control Protocol

The resource usage protocol supports (preA) principle of $UCON_{ABC}$ for checking access requests before resources are actually used. However, in our model, we need to revoke access whenever a user moves out of a valid STZone. The basic protocol does not check the users' access after resources are being granted. Therefore, we define the ongoing access protocol that applies the (onA) principle of $UCON_{ABC}$.

The first implementation choice was that STZone conformations are continuously or periodically sent to RAM servers while users' rights are being exercised. In this manner, a user connects with RAM and send a conformation of current STZone, or RAM server requests a conformation from a user. The STZone conformation is automatically performed by client software without a need to personal reminders. In response, RAM servers check the confirmations and either maintain the users' access or terminate them. However, in this design choice, there is a significant overhead on the user side and on the server side. Our design choice intends to minimize the work load primarily on the users and preferably on the servers.

Our alternative choice is that a user device sends a notification of terminating an access only if the current user STZone becomes unacceptable. Therefore, there is no need to send a conformation if the current user STZone is valid. The problem of extra work load and computations at both the client side and the server side are resolved in our design. At the client side, the software engineer solution, "Observer" design pattern [92], is implemented to fulfil this objective. In this design solution, The *STZone Listener* component gets periodic updates by the *STZone Reader* component about the spatio-temporal coordinates of the user. Whenever the current user location or time does not satisfy the information in a user's authorization-token, the *STZone Listener* component revokes the user's authorization-token and requires *Request Builder* to request an access termination. *STZone Listener* must be tamper-proof component and transparent to users, it is only accessed by signed applications.

The initial resource access protocol is revised in order to define the ongoing/persistent resource usage protocol. In this protocol, two communications steps are added to the basic resource usage protocol. In Figure 5.2, the communications messages M_7 and M_8 describe the additional exchanges needed to implement the ongoing access protocol.

- **Terminating User Access** $[RCM \xrightarrow{M_7} RAM]$: RCM sends the termination access request M_7 to RAM at the time the current user STZone becomes invalid. It creates a termination message payload $M_{rc} = (ID_u, ID_s, ID_{ut}, P_u, Close, Ts_{rc})$ where the keyword "Close" indicates the termination of access. RCM concatenates M_{rc} with the user digital signature $S_{rc}\{M^*_{rc}\}$ and encrypts M_7 with RAM public key PuK_{rs}. The user's authorization-token is deleted at the client side in order to terminate the user access.

- **Revoking User Privileges** $[RAM \xrightarrow{M_8} ACM]$: After authenticating the sender of M_7 package, RAM reads the keyword "Close" and then use ID_u, ID_s, and ID_{ut} to lookup for the user's authorization-token in the *UATokens* list and removes it. Therefore, if a user subsequently requests an access to a resource via deleted ID_{ut}, this request will be denied. RAM forwards an encrypted and signed termination request M_8 to ACM in order to update the current user's authorizations. The M_8 request has payload $M_{rs} = (ID_{rs}, ID_u, ID_{ut}, Close, Ts_{rs})$ as well as the digital signature $S_{rs}\{M^*_{rs}\}$ signed by RAM private key. At the authentication server, after authenticating the sender of M_8, the keyword "Close" indicates ACM to revoke the current user's active role and authorized permissions associated with user's authorization-token ID_{ut}.

A Resource Access Suspension Protocol

Since in our GSTRBAC users are mobile, a user might momentarily depart the authorized location from which he/she currently accessing some resources and come back after a small period of time. To maintain our design efficiency and ease of use, the user's privileges should not be permanently revoked, instead, these privileges must be frozen or suspended for the short period of time the user is off-site and returned back when the user is on-site.

We achieve this design goal by modifying the communication steps of the ongoing resource usage protocol in order to define the suspending resource usage protocol. In this protocol, RCM suspends user access and sends a resource access deferral package $M_7^{'}$ at the time a user unexpectedly leaves a current valid position. This message has a quite identical format to M_7 except that the keyword "Close" is changed to "Freeze" to indicate that the user's privileges should be frozen for a certain time period.

A time window *Tw* is appended to M_7' package to determine the freezing time throughout a user cannot exercise previously granted access rights. Once a user crosses back the boundary of the user's valid location and prior to the *Tw* expiration , he/she can automatically practices earlier activated roles with no need to resend an activation request. At the resource server side, RAM reads the key word "Freeze" and instead of deleting user's AT, it prevents the user from exercising rights for the duration defined in the constant *Tw*.

If RAM gets a resource access request from the user within the time window *Tw*, it handles the requested resource to the user, and deletes *Tw* from the user's record. Nonetheless, in case RAM does not hear anything from the user throughout *Tw*, it forwards package M_8 to ACM in order to revoke the current user's active role and the authorized permissions combined with the user's authorization-token ID_{ut}.

5.3 Securing Against Some Common Attacks

Since our primary concern is protecting applications' resources from improper access, it is important to guarantee that our architecture design is indeed secure from many security threats. In this section, therefore, we describe a threat model that is capable to categorize many attack methods in which an adversary would use to exploit vulnerabilities in our architecture design. This treat model also informally assess the protective measures supported by our design that can alleviate those common threats.

The proposed threat model examines some portions of the threat modeling process proposed in these studies [93, 94]. This model is a process that we conducted to identify the attack methods that an adversary would use to uncover vulnerabilities in our design. In order to identify these attack methods, we need to analyze the architecture components as well as the flow of data between those components. Furthermore, to create a useful threat model, we should look at each architecture component through an adversary's eye and consider both passive and active attacks. In the analysis of our architecture design, we propose three threat models: the first threat model studies the software implementing the architecture components, the second model examines attacks on the messages exchanged between those components, and the third threat model analyzes vulnerabilities in the protocol designs that interconnect those components.

Attacks on Software Components

Since the attack methods in the first threat model are not directly related to the architecture design, we assume that RAM and ACM components are implemented in computing machines equipped with appropriate software protection. Moreover, the RCM component is installed in a phone's secure element, which prevents unauthorized access. For example, Android platform provides a number of access control mechanisms, based on a Linux kernel, that protect access to shared data and functionality across multiple applications [95]. RAM and ACM servers should employ some preventative measures to tackle common attack methods, including, a strong cryptographic technique, a proper firewall configuration, a malware prevention, a proper anonymous protocol, etc. With a strong cryptography, no matter how much computational power is available for the attacker, the cryptosystem keys cannot be broken. It is important to for the preventive techniques to have the latest security updates.

For the denial-of-service attack (DoS), RAM and ACM are protected from such attack because our architecture supports the distributed resource paradigm. In practice, the exchanges in step 1 and step 2 play as countermeasures against DoS with distributed RAM and ACM servers throughout a network. A user client RCM submits requests to a number of distributed RAM servers, similarly, RAM requests authorization-tokens through a number of distributed ACM servers. As such, in case of a compromised user successfully saturates a RAM sever using some methods of attack (e.g., ICMP flood, SYN flood, and Degradation-of-service), other RAM servers will be available to provide services to other users.

However, with the distributed denial-of-service attack (DDoS attack) or other DoS attack classes, a client attacker might succeed to bottleneck some or all servers. Such attack success is inherently a result of the vulnerabilities in the network design, but not because of our protocol design. To mitigate the DDoS attack, service nodes in Intranet or in the software architecture should utilize appropriate DDoS detection and prevention mechanisms (e.g., Firewalls, Blackholing and Sinkholing, Clean pipes, and blacklisting), which block illegitimate traffic and only allow traffic that is diagnosed as a legitimate traffic.

Attacks on Protocol Messages

For the second threat model, the message confidentiality, message integrity, message negation, and message identity are important to protect communication messages. These features are provided in our architecture.

The message confidentiality is guaranteed invulnerable to eavesdropping by a proper public key encryption. The message integrity is maintained by concatenating each message with the message digest, and as such, if an intruder intercepts and alters a communication message, the receiver can detect that. The message negation is protected by associating digital signatures with messages; the digital signature provides a non-repudiation proof of the message origin. Message identity is provided by using public key certificates and digital signatures. Using these two features, the sender and the receiver are able to mutually authenticate their identities. We assume that private keys used to sign the message digests are only possessed by the signers, and the public key certificates have one-to-one mapping between the public key and the owner. Furthermore, passwords also provide a proof of identity.

Attacks by Malicious Entities

The third threat model investigates attacks by malicious entities that impersonate as legitimate protocol's endpoints. In particular, this treat model examines attack methods posed by a malicious RCM client, a malicious RAM server, or a malicious ACM on the protocol design. In the following, we show how our resources are protected from improper access by malicious protocol endpoints.

Malicious RCM: The goal of a malicious RCM client (a user) is having an unauthorized access to application resources via the RAM servers. The followings discuss a number of countermeasures in our protocols to deter common attack methods that authorized/unauthorized users can carry out to gain an access to protected resources. We first consider the attacks that unauthorized users might perform:

- **Eavesdropping:** This attack allows an adversary unit to make a copy of communication exchanges; then discloses sensitive information such as passwords, identifiers, and learn procedures for making access. Nevertheless, the passes between RCM and RAM are encrypted using public keys and passwords are transformed (e.g., one-time password). Therefore, eavesdropping attacks cannot cause a harm to our architecture.

- **Modifications:** A modification attack occurs when an attacker intercepts and modifies the contents of communication passes in order to impersonates others to gain an access. This attack is not applicable to our protocol because paths between RCM and RAM servers are encrypted and combined with signed checksums; therefore, RAM receivers validate those checksums before accepting incoming traffics.

- **Replay:** In this attack, an eavesdropper intercepts authentication exchanges coming from a legitimate RCM and later replays that exchange as it comes from a lawful RCM client resulting in harmful consequences such as redundant orders of the same item. Since the communication messages sent from RCM clients are encrypted and associated with timestamp as a non-repeated value, replay attacks will be detected. In addition, one-time passwords are non-repeatable value preventing an adversary from reusing passwords in the replayed passes. Therefore, our authentication protocol is not subject to the replay attack.

- **Reflection**: The reflection attack on the same target occurs when a malicious user intercepts a message coming from the target RCM and reuse it as a response coming from a legitimate recipient RAM. The reflection attack is of a great concern to our authentication protocols that relay on the similar encryption/decryption keys. Since passes are encrypted by the recipient public key, the sender cannot decrypt the reflected messages with its private key. The protocols' exchanges are also associated with signed message digests as well as senders' identifiers which help a sender to become aware of the messages reflection. Besides of these features, the formats of the request messages and response messages are different. Consequently, the protocols are not vulnerable to the reflection attack.

- **Man-in-the-middle (MITM)**: In MITM attack, an intermediate intruder eavesdrops the communication between the victims and intercepts the exchanges coming from one sender victim and retransmits them to the intended recipient as they are directly coming from the victim; the entire connection is managed by the intruder. This attack is of a great concern to communication parties that lack the mutual authentication scheme. In our protocol is designed with assumption that entities authenticate each other identities before accepting the incoming traffic and the traffic is encrypted, digitally signed, and appended with senders' identifiers, one-time passwords, and timestamps. As a result, these features preclude the chances of MITM attacks [96].

- **Compromised-Key:** The goal of a compromised-key attack is for an eavesdropper to obtain a public key of a RAM server by some means and use it to create a fake access request package. In order for this attacker to succeed, it should have intercepted the exchanges between some lawful RCM and RAM endpoints and learn information such as user's identifier ID_u, device identifier ID_s, and layout of the request package. This attack is of our great concern to our protocol design because public keys are typically not kept secret. From the first glance at the protocol design, we might uncertainly judge that this attack is possible because the ID_u and ID_s identifiers are sent in clear. However, our protocol is solid in front of the compromised-key attack. The encrypted user's password, user's identifier, and signed checksum in the request package are significant preventive measures against such attack. In addition, the one-time password cannot be reused in the false request.

- **Illegitimate Use**: There are two variations of this attack. The first attack might occur when a participant tries to gain access to resources using a cell-phone of another legitimate participant assuming the second party is oblivious about this onslaught. Our design is secure from such attack by virtue of user identifier ID_u and device identifier ID_s the request package, which are pertained to a distinct user certificate at the RAM servers. Therefore, the user access is declined in case those parameters do not match in the user's certificate. The second practicability of this attack happens when a malicious user tries to make an access via lost or stolen cell-phones. However, such assailant cannot breach our authentication protocol since a user access is controlled by a strong password assuming that user's password is securely stored in its device.

Strictly speaking, the proposed protocols are enhanced with a number of security measures such as the public key cryptography, digital certificates, checksums, one-time passwords, digital signatures, timestamps, and encrypted senders'/receivers' identifiers. In most attack scenarios, these techniques have been used for a long time and have proven practices for preventing, detecting, deterring many of those attacks. In particular, it is mathematically not possible to discover a private key from its pair public key. Moreover, the strength of the cryptography or hashing algorithm is implementation details that differ based on the criticality of the application data and services. We next consider the attacks that might be carried out by authorized users.

- **Reusing Authorization-Tokens:** A user may want to access a resource using his past authorization-token. However, when the STZone expires, the authorization-token gets deleted from RCM and cannot be used.

- **Modifying Authorization-Tokens:** A user may modify the authorization-tokens stored at his site. However, authorization-tokens cannot be modified as they are stored in a secure tamper-proof storage at a user's mobile device.

- **Users Collusion**: With collusion attack, two or more users collude in order to commit a fraud. That is, user u_1 gets a valid authorization-token for a role and sends it by some means to user u_2 in order to allow u_2 to violate conflicting constraints. However, this attack is not applicable to our protocols because each authorization-token's identifier adheres to a particular user's device.

- **Distributed Resources:** In this attack, a disgruntled RCM client exploits services from several distributed RAM servers. We discuss the possibility of this attack bearing in mind that the data consistency at all servers is preserved. In particular, a disgruntled RCM client requests a role activation from a designated RAM server and later attempts to activate a conflicting role from another RAM server with the purpose of committing a fraud. In this regard, the second RAM forwards user's request of the conflicting role to one of the ACM servers. Since the policy state is consistent at all ACM servers, that ACM server rejects the user's request because of activation conflict constraints. Therefore, this attack is not possible in our protocols.

Malicious RAM: The primary goal of a disgruntled RAM is to compromise a security policy. Such attack might disrupt the continuity of business operations for a company by either deleting or altering policy rules, or allow a user to commit a fraud or a theft. In order to achieve this goal, the disgruntled RAM eavesdrops the communication channel between RAM and ACM severs and initiates hostile connections with the ACM servers. For instance, an attacker fabricates a number of authentication-token requests to disclose the underlying authorization rules in an application policy. There are number of methods of performing this attack such as replay, reflection, man-in-the-middle, compromised-key, eavesdropping, and modification attacks.

However, our protocol is enhanced with a number of protective techniques to alleviate these attacks. The traffic between RAM and ACM servers is protected by public keys, digital certificates, checksums, one-time passwords, digital signatures, and timestamps. These techniques are widely accepted through proven practices for preventing, detecting, deterring many of those security threats.

Malicious ACM: In a simple form of this attack, a corrupted ACM sever issues a false authorization-token tolerating unauthorized access to critical resources. Additionally, since RAM servers are oblivious about the underlying policies, the false authorization-token may allow users to play unauthorized roles, access conflicting roles, or acquire conflicting permissions. Our protocol is designed with ACM servers acting in good faith, they are well-developed and behaving properly most of the time. That is, the reported authorization-tokens from truthful ACM servers are always considered correct.

In all other forms of false reporting attacks, RAM servers only accept authorization-tokens from a set of certified ACM servers that have pre-deployed public key certificates. With the false reporting attack, an ACM attacker may surreptitiously subvert a system through impersonation of a lawful ACM server. Such ACM intercepts a number of authorization-token responses from the lawful ACM server and interprets the responses' contents, then reroutes harmful authorization-tokens to requesting RAM servers. Evidently, our protocol design ensures effective protection measures preventing corrupted ACM servers to subvert users' privileges.

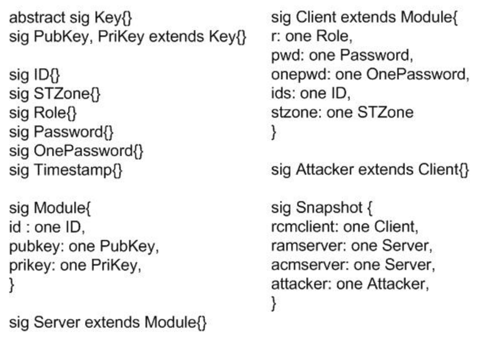
5.4 Formal Protocol Analysis

In the preceding section we have given informal guarantees that certain types of attacks leading to access control breaches do not occur in our architecture. However, our security analysis is by no means complete because of lack of formality. Often times, formal analysis may reveal problems that are not apparent in informal analysis. Our second goal in the analysis context includes formal verification of the protocol to provide further assurance that the attacks mentioned do not occur.

In this section we use Alloy [15] to check whether the resource usage protocol described in the paper is vulnerable to the Man-In-The-Middle (MITM) attack. Alloy is a formal specification language that has been used to rigorously analyze security policies (e.g., see [16, 97, 98]). It has very good tool support in the form of the Alloy Analyzer that translates an Alloy specification into a boolean formula that is evaluated by embedded SAT-solvers.

An Alloy model consists of signature declarations, fields, facts and predicates. A signature defines a set of objects. A field belongs to a signature and represents a relation between two or more signatures. In Alloy invariants can be defined as facts. Facts are statements that define constraints on the elements of the model. Operations can be specified as predicates. Predicates are parameterized constraints that can be invoked from within facts or other predicates. Properties to be checked can be expressed as predicates. A property holds for an Alloy model if the Alloy Analyzer finds an instance satisfying the predicate that specifies the property.

Figure 5.3 shows an fragment of an Alloy model for the resource usage protocol (see Figure 6) described in the paper. For the details on the entire Alloy specification, we refer to Appendix C. Signature *Module* in the Alloy model represents network participants, and it has a unique identifier



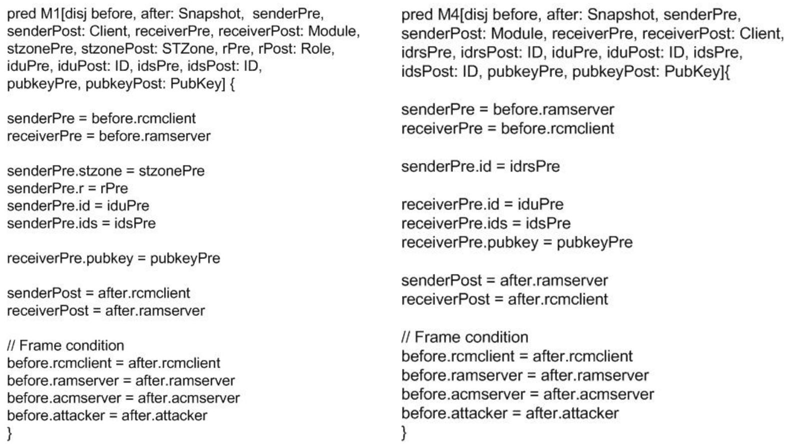
```
abstract sig Key{}
sig PubKey, PriKey extends Key{}

sig ID{}
sig STZone{}
sig Role{}
sig Password{}
sig OnePassword{}
sig Timestamp{}

sig Module{
id : one ID,
pubkey: one PubKey,
prikey: one PriKey,
}

sig Server extends Module{}

sig Client extends Module{
r: one Role,
pwd: one Password,
onepwd: one OnePassword,
ids: one ID,
stzone: one STZone
}

sig Attacker extends Client{}

sig Snapshot {
rcmclient: one Client,
ramserver: one Server,
acmserver: one Server,
attacker: one Attacker,
}
```

Figure 5.3: An Fragment of an Alloy Model for the Resource Usage Protocol

(i.e., an instance of *ID*), a public key (i.e., an instance of *PubKey*) and a private key (i.e., an instance of *PriKey*). The basic entities involved in the network communication are represented by sub-signatures of *Module* (e.g., *Client*, *Sever*, and *Attacker*). Thus a module can be a server, a client, or an attacker.

An instance of *Client* can be a *RCM* client that has a requested role (i.e., an instance of *Role*), a password (i.e., an instance of *Password*), one-time password (i.e., an instance of *OnePassword*), a device identifier (i.e., an instance of *ID*), and an instance of *STZone* that records the client's current zone.

An instance of *Server* can be either a *RAM* server or an *ACM* server. An instance of *Snapshot* represents a system state consisting object configurations at a particular time. It typically consists of a *RCM* client, a *RAM* server, an *ACM* server, and an attacker.



```
pred M1[disj before, after: Snapshot,  senderPre,
senderPost: Client, receiverPre, receiverPost: Module,
stzonePre, stzonePost: STZone, rPre, rPost: Role,
iduPre, iduPost: ID, idsPre, idsPost: ID,
pubkeyPre, pubkeyPost: PubKey] {

senderPre = before.rcmclient
receiverPre = before.ramserver

senderPre.stzone = stzonePre
senderPre.r = rPre
senderPre.id = iduPre
senderPre.ids = idsPre

receiverPre.pubkey = pubkeyPre

senderPost = after.rcmclient
receiverPost = after.ramserver

// Frame condition
before.rcmclient = after.rcmclient
before.ramserver = after.ramserver
before.acmserver = after.acmserver
before.attacker = after.attacker
}
```

(a) Message M_1

```
pred M4[disj before, after: Snapshot, senderPre,
senderPost: Module, receiverPre, receiverPost: Client,
idrsPre, idrsPost: ID, iduPre, iduPost: ID, idsPre,
idsPost: ID, pubkeyPre, pubkeyPost: PubKey]{

senderPre = before.ramserver
receiverPre = before.rcmclient

senderPre.id = idrsPre

receiverPre.id = iduPre
receiverPre.ids = idsPre
receiverPre.pubkey = pubkeyPre

senderPost = after.ramserver
receiverPost = after.rcmclient

// Frame condition
before.rcmclient = after.rcmclient
before.ramserver = after.ramserver
before.acmserver = after.acmserver
before.attacker = after.attacker
}
```

(b) Message M_4

Figure 5.4: Alloy Message Predicates

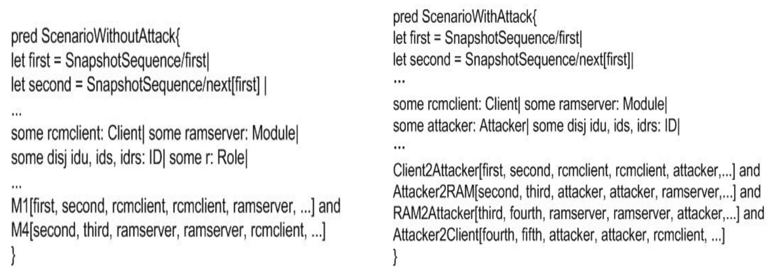
Figure 5.4(a) shows an example of an Alloy predicate that specifies the message $M1$ shown in the resource usage protocol (see Figure 5.2). The $M1$ predicate takes as input a *before* snapshot and an *after* snapshot, where the *before* snapshot describes a system state before the message is sent, and the *after* snapshot describes a system state after the message has been sent. Parameter *senderPre* represents a message sender in the *before* snapshot while *senderPost* represents a message sender in the *after* snapshot. Similarly, *receiverPre*, *stzonePre*, *rPre*, *iduPre*, *idsPre*, and *pubkeyPre* represent states of objects accessed by the predicate in the *before* snapshot while *receiverPost*, *stzonePost*, *rPost*, *iduPost*, *idsPost*, and *pubkeyPost* represent states of objects in the *after* snapshot.

The predicate specifies that the message sender must be a *RCM* client and the message receiver must be a *RAM* server. The sender's attributes must have the same value with the parameters of the predicate. For example, the value of the sender's current zone must be equal to the value of the *stzonePre* parameter. The frame condition in the predicate specifies that the participants of the protocol must remain the same after the message has been sent.

Since it is assumed that *RAM* and *ACM* are trusted in the protocol, only the network communication (e.g., $M1$ and $M4$) between a *RCM* client and a *RAM* server is analyzed using the Alloy Analyzer. Figure 5.4(b) shows an Alloy predicate that specifies the $M4$ message from a *RAM* server to a *RCM* client. The $M4$ message includes the client's identity information (*senderPre.id = idsPre*) that ensures the protocol described in the paper is immune from an MITM attack.

Figure 5.5(a) shows an Alloy predicate that specifies a scenario without the MITM attack in which a *RCM* client sends $M1$ to an *RAM* server and then receives $M4$ from the *RAM* server. *SnapshotSequence* in the predicate represents a sequence of snapshots, while *first* is the first snapshot in the sequence, *second* is the second snapshot and etc. *first* represents a system state before $M1$ has been sent, *second* represents a system state after $M1$ has been sent and before $M4$ has been sent, and *third* represents a system state after $M4$ has been sent.

The Alloy Analyzer uses the predicate in Figure 5.5(a) to query whether there exists an instance that satisfies the predicate. For this scenario, the Analyzer did return an instance, indicating that the communication protocol between the *RCM* client and the *RAM* server was successfully simulated.



```
pred ScenarioWithoutAttack{
let first = SnapshotSequence/first|
let second = SnapshotSequence/next[first] |
...
some rcmclient: Client| some ramserver: Module|
some disj idu, ids, idrs: ID| some r: Role|
...
M1[first, second, rcmclient, rcmclient, ramserver, ...] and
M4[second, third, ramserver, ramserver, rcmclient, ...]
}
```

(a) Regular *RCM* and *RAM* Passes

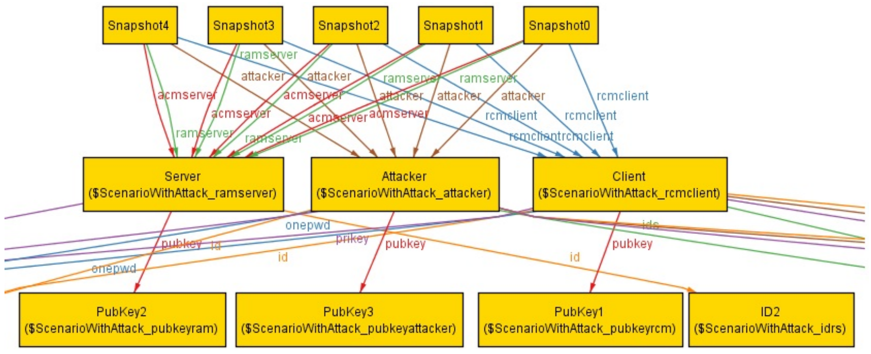
```
pred ScenarioWithAttack{
let first = SnapshotSequence/first|
let second = SnapshotSequence/next[first]|
...
some rcmclient: Client| some ramserver: Module|
some attacker: Attacker| some disj idu, ids, idrs: ID|
...
Client2Attacker[first, second, rcmclient, rcmclient, attacker,...] and
Attacker2RAM[second, third, attacker, attacker, ramserver,...] and
RAM2Attacker[third, fourth, ramserver, ramserver, attacker,...] and
Attacker2Client[fourth, fifth, attacker, attacker, rcmclient, ...]
}
```

(b) MITM *RCM* and *RAM* Passes

Figure 5.5: Alloy Simulation Predicates

To verify whether the protocol is immune from an MITM attack, we introduce an attacker between a *RCM Client* and a *RAM Server*. The attacker makes independent connections with the *RCM Client* and the *RAM Server* respectively, and relays messages between them. In an MITM attack scenario, message $M1$ is replaced by two messages, $Client2Attacker$ and $Attacker2RAM$, where $Client2Attacker$ is a message from the client to the attacker and $Attacker2RAM$ is a message from the attacker to the server. Similarly, $M4$ is replaced by $RAM2Attacker$ and $Attacker2Client$, where $RAM2Attacker$ is a message from the server to the attacker and $Attacker2Client$ is a message from the attacker to the client.

Figure 5.5(b) shows an Alloy predicate that simulates an MITM attack scenario. The Alloy Analyzer uses the predicate in Figure 5.5(b) to query whether there exists an instance that satisfies the predicate. For this MITM attack scenario, the Analyzer returned no instance, indicating that the MITM attack was not successfully simulated under the protocol described in the paper. Thus, the proposed protocol is immune from an MITM attack.



Snapshot4
Snapshot3
Snapshot2
Snapshot1
Snapshot0
ramserver
attacker
acmserver
rcmclient
Server ($ScenarioWithAttack_ramserver)
Attacker ($ScenarioWithAttack_attacker)
Client ($ScenarioWithAttack_rcmclient)
onepwd
pubkey
prikey
id
ids
PubKey2 ($ScenarioWithAttack_pubkeyram)
PubKey3 ($ScenarioWithAttack_pubkeyattacker)
PubKey1 ($ScenarioWithAttack_pubkeyrcm)
ID2 ($ScenarioWithAttack_idrs)

Figure 5.6: A Partial Alloy Instance for the *ScenarioWithAttack* Predicate shown in Figure 5.5(b)

In order to test our protocol, we eliminated `senderPre.id = idrsPre` from

*Attacker*2*Client*. Without this statement, we can no longer verify the identity of the sender. The attacker can pose as the *RAM Server* causing the MITM attack to succeed. The Alloy Analyzer in this case would return an instance (see Figure 5.6) that demonstrates the MITM attack scenario. The Alloy instance shown in Figure 5.6 consists of five snapshots, each of which is associated with a server, a client and an attacker. *Snapshot*0 corresponds to *first* in the *ScenarioWithAttack* predicate, representing the first snapshot in the sequence, *Snapshot*1 corresponds to *second* etc.

The *first* snapshot (*Snapshot*0) specifies an initial state before the *Client*2*Attacker* message has been sent, the *second* (*Snapshot*1) represents the state after the *Client*2*Attacker* message has been sent, the *third* snapshot (*Snapshot*2) specifies the state after the *Attacker*2*RAM* message has been sent, the fourth snapshot (*Snapshot*3) represents a system state after the *RAM*2*Attacker* message has been sent, and the fifth snasphot (*Snapshot*4) represents a system state after the *Attacker*2*Client* message has been sent. Other attacks such as replay, reflection, etc can also be simulated using the approach described in the paper. The complete version of the Alloy model of the successful MITM Attack is shown in Appendix D.

More details on the MITM attack and its fix can be found in [96]. Other attacks such as replay, reflection, etc can also be simulated using the approach described in the paper. We only provided the details of the simulation approach on the MITM attack since the MITM attack is one of the most sophisticated attacks.

5.5 Experimental Evaluation

In this section, we describe an empirical evaluation of our architecture performance. We develop a proof-of-concept prototype for our architecture enforcing GSTRBAC in an Android mobile application. The prototype implements our design in a distributed system with multiple autonomous virtual servers. Our Android client and servers are written using Java programming language. The RCM client is implemented as an Android “mobile app” component runs in a user’s cell-phone [99]. RAM and ACM components are executed in traditional Java server programs.

The primary reasons motivated our choice of using the Google Android software stack to develop our prototype are its free and open source nature. Additionally, Google Android is built on the top of a Linux distribution that includes a Java Virtual Machine (JVM) designed to run mobile devices [99]. Google Android platform has also many features for incorporating a GPS receiver package into mobile applications code. Android has a powerful Software Development Kit (SDK) [100] that is easy-to-use in building mobile applications run on a variety of smartphones' technologies. The Android SDK provides a debugger, libraries, and a handset emulator that are necessary to write and test Android platform applications. The Android emulator allows us to simulate mobile applications before actual use. The development environment of our prototype is Eclipse IDE integrating the Android Development Tools (ADT) plug-in. For the first run of the Android application, we utilize the AVD manager in Eclipse to create a new Android device enhanced with a GPS receiver.

To implement the basic resource usage protocol, we have adopted source code from the *FlexiProvider* Toolkit [101] that has Java based cryptographic modules, including public key, digital signature, and *MD5* message digest. The toolkit applies fast and secure cryptographic algorithms that are written in Java language. Prior to running the experiment, each entity should have its private key and stores the public key of the communicated endpoints in a local file. We used the *KeyPairGenerator* class to produce a pair of public and private keys for each entity. The public keys are securely distributed using *AES* symmetric-key encryption. For the message authentication code, RCM, RAM, and ACM components employ the *MD5* method supported by *FlexiProvider*.

The relational database is used to represent the application and policy data. This relational database is realized in the open source database MySQL server [102]. The MySQL server has two tables, the first table is the application table which stores the users' login information and the second table is the policy table that stores GSTRBAC policy rules.

In order to incorporate the location capabilities of Android into our prototype, we used the *LocationManager* package [100] to track the current user position and capture the local time at which the location is retrieved. The *LocationManager.requestLocationUpdates* method revises the user's device's location every fixed period of time through an enabled location provider GPS. A class implementing the *LocationListener* interface handles changes in the device location. The *LocationManager.getLastKnownLocation* method in the *LocationListener* implementer gets the last known location object which has the altitude, latitude, and longitude information. When the Android emulator starts for the first time, it reports the current location *Null* because there is no last known location to be obtained. Thus, we used the *DDMS* view of Eclipse to manually feed mock location data into the Android emulator. Once the emulator's GPS device has the dummy data, the *LocationListener* implementer is triggered to retrieve the current location. Google Maps Application Programming Interface (API) is utilized to display the location's coordinates on the screen. We then manually record the logical locations in a local file.

The correctness of the mobile application components are tested via the Android emulator and the test showed that our prototype works as expected. Our Android handset emulator in Figure 5.7 displays the Android app component runs in a user's mobile. This handset emulator prompts a user to select his/her role and enters the user's identifier and password. Once the user enters these information and hits the connect button, the Android application software retrieves the last known user location and the local time. Then, it composes an access request package and sends it to one of the available virtual RAM servers in a secure manner.

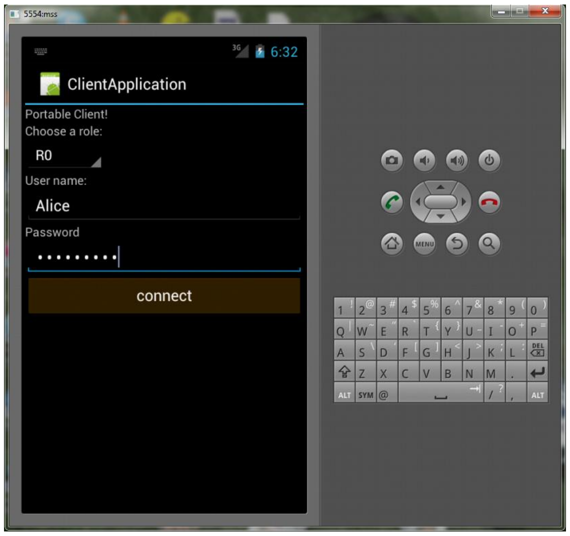


Figure 5.7: Android Handset Emulator

In the experiment setup, only one machine is used to measurer the back-end servers' overhead for processing access requests. The experiment is performed in one machine in order to eliminate the network delay from the local computations of the architecture components. Furthermore, the time needed for the GPS receiver to get the location coordinates from the base station varies from milliseconds to double digit minutes, or it might not get the coordinates at all in some blind areas. However, this is not as a result of our design. The experiment is carried out in a machine with a Windows 7 platform running on Intel(R) Core(TM) 2 Duo CPU at 2.20 GH with 4.00 GB RAM.

The experiment evaluates the architecture performance on three virtual handset Android emulators, three RAM virtual servers, and three ACM virtual servers. These virtual machines communicate via traditional sockets. A virtual server running a local centralized MySQL database is also instantiated on the same machine. The database server is accessible by the RAM and ACM servers in order to process users' requests. For each request, the handset emulator opens a new connection with one of the virtual RAM servers and closes the connection at the time it receives a response. The RAM server in turn opens a new connection with one of the ACM servers if the user login information is correct. The RAM server closes the connection when it gets a response from the endpoint ACM server.

To evaluate different spatio-temporal access scenarios, we have stored the logical locations and role names in two local files. Thus, for each request, the handset emulator randomly selects a location name and a role name from these files and sends them along with other information in the request package. This approach allows us to test whether our application works as anticipated and validates the policy correctness. We measure the total time needed to issue and send a new access request until the time a response is received. The response delay is evaluated using 150 requests sent simultaneously from three Android handset emulators. Each emulator sends 50 requests. The responses vary based on the information in the request packages. For example, a request is approved if and only if the login information is correct, the requested role can be authorized, and the current user's zone is acceptable. Otherwise, the request is rejected.

Table 5.2: Back-end Average Response Delay

Response	**Average Delay**
0-Approved	*73.66 ms*
1-Rejected (Improper login data)	*29.56 ms*
2-Rejected (Improper role)	*67.50 ms*
3-Rejected (Improper zone)	*81.43 ms*
Total Average Delay:	*63.04 ms*

The results in Table 5.2 show the average response delay for each response type as well as the total average delay in milliseconds. The overall delay yielded by the basic resource usage protocol is *63.04 ms*. Consequently, implementing our architecture for enforcing GSTRBAC is indeed viable. The rejected requests due to invalid login information yields *29.56 ms*, which is the smallest response delay because RAM servers send these responses immediately without consultation with ACM servers. This implies that the design goal of reducing the response time by eliminating unnecessary steps is fulfilled. Additionally, the prototype demonstrates that there are no bottleneck points apparently degrading the performance.

Chapter 6

Conclusions and Future Work

Access control is the most important techniques to ensure the security of systems. There has been a lot of work done by security researchers in developing flexible and semantically rich access control models. Currently, the Role Based Access Control Model (RBAC) is the de facto standard. RBAC supports many features which make it widely accepted and used in various applications. However, notwithstanding its popularity, RBAC has been found lacking in many applications as they keep emerging with the growing utilization of wireless networks and mobile devices.

Such applications necessitate sophisticated authorization models where access to a resource depends on the credentials of the user and also on the location and time of access. Consequently, researchers have extended the traditional access control RBAC to provide spatio-temporal access control. Unfortunately, we found in this work that there are some types of mobile applications requirements that are not completely satisfied by any of the proposed RBAC extensions. These models either lack the specification flexibility or cannot be used to specify some novel requirements that are presented in this dissertation. Therefore, the development of new access control models are required to secure access in the mobile applications such as to those presented in this dissertation.

Mere development of RBAC models does not guarantee the safety of systems. A potential flaw in the policy specification results in inconsistent state or erratic system behavior. The verification and validation of underlying access control policies have been performed informally by manual inspection or formally by applying formal analysis techniques. The informal verification technique requires the involvement of human and as a result the manual inspection is tedious and prone to ambiguity. Researchers have advocated the use of formal analysis techniques for analyzing RBAC such as the one in [87].

Towards this end, researchers have proposed different security analysis methods to ensure the consistency of the access control specification and it is free from ambiguities. Among these methods, researchers have advocated the use of existing formal specification languages. Each of these studies exhibits some shortcomings and advantages. This work investigated the current approaches based on some properties in mobile applications that must be rigorously checked to prevent security beaches. This research emphasised that the current formal models are only relevant for verifying particular category of RBAC models, and they are insufficient for verifying some features interactions in presence of spatio-temporal conditions with role hierarchy and separation of duties. Furthermore, most of the existing verification methods verifies a limited number of security constraints due to the state-space limitation, and also they cannot be used to check complex temporal properties in the existence of location constraints. As a result, researchers need to propose analysis techniques that reduce the state-space size and thereby increase the versification time.

Furthermore, in this research we argued that an architecture model for implementing new access control models is needed to naturally investigate the practical viability of these models and analyze the system requirements to adopt such policy models. However, we found that a very limited studies have considered the real-world implementation issues of RBAC. A major issue is that the proposed extensions of RBAC look very complex to use. For example, the implication in terms of configuring a system based on this approach is mostly left over.

Furthermore, we do not know the cost of keeping it up to date and the impact on other existing access control solutions. We should also be able to predict that does a system require major changes in order to apply newly developed models. The developer of new models need to provide more evidence about the value provided by their models: a case study and trial should have been carried out in a real-world scenario. This is actually a key step that should have been done along the introduction of new models to assess the proposed approach against real-world constraints. Having, a full prototype being implemented provides more information about how an access control model is integrated with real applications/services to provide the required level of access control.

Broadly speaking, the primary focus of this dissertation is to provide an access control framework for mobile applications that is motivated by the aforementioned issues of the existing access control solutions. We have introduced a consolidated access control model which improves upon the expressiveness of the existing models and flexibly supports various mobile applications requirements. For conflict detection and correction in a security policy, we have also developed an automated analysis approaches which can religiously perform qualitative and quantitative analysis of many spatio-temporal properties. Furthermore, we have defined an architecture model for designing a system enforcing our model in either a centralized or a distributed system.

In Section 6.1 we disuses the contributions of this dissertation in some more details and outline some pointers of future research works in Section 6.2.

6.1 Summary of the Contributions

In Chapter 3 we proposed a generalized spatio-temporal RBAC model (GSTRBAC) that incorporates spatio-temporal components to all RBAC entities and relationships. We introduced a formal representation of (physical and logical) locations, time, and their combination, which we called a spatio-temporal zone (STZone). The notion of STZone is an abstraction to express in a compact and natural way time- and location-based policies. This avoids some of the problems in previous extensions of RBAC where location and time information were handled separately. Using the zone concept, the number of entities was reduced, the model was easy to be customized, many security properties were controlled flexibly, and the model has a great potential to be integrated with various applications. With different types of zones, multi-dimensional security requirements were easily expressed with no need for employing different models or altering the formal semantic of the model.

We defined how the standard components of the RBAC model are parametrized by the notion of STZone in order to build the GSTRBAC model. The proposed model supports many important security features and it expressively defines granular mobile applications polices. In addition to traditional role hierarchy and SoD constraints, we showed that the model flexibly specifies different forms of pre-requisite, post-requisite, and triggers. The model is formalized in UML with OCL constraints. and it has been demonstrated by the specification of a policy in a military context. Furthermore, the extended version of GSTRBAC model is formalized by first-order logic and we demonstrated how the extended model can be adapted to the complicated real-world healthcare application DDSS. Following our model, a security designer simply defines a set of zones where each model entity is accessible. Consequently, the access decision is contingent upon evaluating the STZone associated with a user making an access request and the STZones associated with the requested entity that define where and when the entity is available.

This model addresses new and challenging problems raised by today's internet environment with widely-used mobile devices. In particular, it copes with the challenges posed by mobile applications in which policies crucially depend on location and time. The model should have a limited complexity when adding new coordinates for location and time information. Unlike other works, this work introduced the novel "STZone" to model such additional information. Such design achieves the required complexity and functionality at the same time. Additionally, the mobile environment policies could change also very dynamically. Thus such a well-designed model should be able to handle dynamic policy changes in a clean and efficient way. We naturally showed that the design of the "STZone" structure fully focuses on such requirements and handles the problem well in many of access control scenarios.

When considering location and time as additional supporting factor, the real-time access validation, such as on a per-request, post-requisite, and triggers levels, becomes more sensitive, and efficiency is also critical. We demonstrated that the "STZone" is designed to be associated with not only roles, but also permissions and objects. Such design decision supports the real-time access validation and improves the efficiency of such validation, especially on such levels. Furthermore, we have presented some real-world mobile applications in which our access model was able to achieve the required validation efficiency, at different levels, for multiple requests from a moving user.

GSTRBAC model has many features that may interact with each other causing conflicts, inconsistencies, and security breaches. Additionally, constraints could be specified stronger than needed resulting in some roles, permissions, or objects being inaccessible. Consequently, it is important to analyze GSTRBAC policies before applying them. As such, in Chapter 3 and 4, we have developed two novel analysis approaches of spatio-temporal policies specified by our proposed model.

We first proposed an automated verification approach based on the USE constraint solver tool. A user can manually create some model instances and then verifying their correctness against the class diagram model. This approach also supports the automatic generation of instances that represent some policy changes performed by administrative or user actions. When the policy instance does not conform to the GSTRBAC class diagram or it violates some property, the tool promptly shows how the property has been violated. Specifically, it illustrates the relevant entities and the relationships and how their interactions have caused for the property violation. For example, if the security designer finds that some pre-requisite constraint has been violated via an assignment operation, he can either change the pre-requisite relation or change the constraint depending on the application requirement. To illustrate how our verification approach is used, we presented an example illustrating how the USE tool can be exploited to investigate some scenarios of an access control policy in a military application.

In the second analysis approach, we provided an approach for checking strict temporal constraints like duration, liveness, execution time, and bounded response time constraints while considering mobility. We explained how access control policies was translated to a timed-automata model, and a set of security properties were feasibly specified in TCTL queries, and UPPAAL toolbox was used to check whether the timed-automata model satisfies those properties. This approach was seen capable to check strict temporal constraints and interactions between different features in the DDSS system. A number of algorithms with their computation complexity for mapping applications' polices to corresponding timed-automata were also presented. To get an efficient analysis result, a number of techniques for eliminating the state-space explosion problem were provided based on eliminating entities from the model which have no impact on the property of our interest.

In Chapter 5 we introduced an enforcement mechanism of our spatio-temporal RBAC model in mobile applications, to demonstrate that our model is viable in real-world applications. We proposed a platform-independent architecture for a mobile system enforcing our spatio-temporal model. We made the design decision of separating policy from the point of use in order to integrate our design in many applications. We developed a number of protocols that consider spatio-temporal information for initiating and maintaining access under different circumstances.

The proposed access control protocols might be subject to security breaches. Towards this end, we developed threat models and discussed the safeguard techniques that we take to protect against some important threats to our design. We also formally analyzed our protocols using Alloy to provide assurance that they are indeed free from attacks. Our analysis approach is capable of expressing complex structural constraints and behavior of our protocol. It is supported by an automated constraint solver Alloy Analyzer that searches instances of the protocol model to check for satisfaction of system properties. We provided Alloy code that illustrates how to model some protocol attacks and rigorously verify them.

It is not sufficient to propose an architecture and a set of protocols for accessing resources. In order to completely demonstrate the feasibility of our approach, we developed a proof-of-concept prototype and showed how spatio-temporal policies can be implemented for a mobile application. We described the implementation details. We also conducted some experiments that give the delays incurred for performing authorization checks using our model. The overall results indicated that the proposed architecture model is feasible to adopt in either centralized or distributed systems.

6.2 Future Research Work

The work presented in this dissertation has a great potential to be extended a long many dimensions. The following outlines some promising directions of future works in the context of proposing new models and analysis approaches.

In the first context of defining novel models to cope with the growing access control requirements, we plan the following model extensions. The models discussed in Chapter 3 and 4 define a police in an off-line manner and a system executes the police at run time. That is, the STZone constraints are defined in the policy are static in nature, they do not change during the execution of the model. We plan to develop an adoptive model that supports the dynamic nature of mobile applications. The extended model should consider the effect of the dynamic association of STZones with RBAC components at the model run time. It proposes some mechanisms of a run time generation of the authorization STZones. We refer of such type of access control models to models at run time.

Workflow is a fast evolving technology that is recently deployed by many mobile applications. Workflow enables the automatic execution of a business process by an information system usually referred to a Workflow Management System (WfMS) [103]. Each business process in a workflow is a set of activities (or tasks) that are performed by multiple collaborating actors (human or machine agents) to outcome a particular product. A workflow defines two main aspects of a business process: the control flow and the authorization rules. The control flow defines the dependencies between the business process tasks that are typically executed in a particular order in a workflow. Authorization rules controls the execution of tasks by legitimate actors to ensure the data security in a workflow. These rules may also define the separation of duties between tasks to prevent frauds and errors. Therefore, an access control model must be provided to formally define the authorization rules of a workflow [104]. It should define the assignment of tasks to roles/users and the association of permissions with tasks.

We plan to define a mobile business process that refers to a set of tasks that are performed by nomadic devices (e.g., smartphone). A typical example of mobile business process is the mobile DDSS system. The primary security requirement in such system is that users are allowed to perform tasks on-site and at certain time intervals. For example, a vector control team receives a request to perform a spry task for some infected houses, a team member reads the request details from their handheld devices, then makes a journey to patients' premisses and collects the data in electronic format and sends it to the main office via a mobile device. We refer to these activities by mobile activities. As such, the next phase of the work is to develop a spatio-temporal access control model for business processes performed with mobile computers. We would like to extend our spatio-temporal access control model for workflows which consists of a set of tasks that are coordinated by control-flow, data-flow and dependencies. It would be interesting to see how these various dependencies interact with the spatio-temporal constraints of the workflow.

Another extension is to develop an access control model for cross-domain interoperability systems operating in insecure environments. An example of such inter-policy systems is an access control model for Web-services. Web Services are coupled applications that communicate in order to provide services to users. Such applications are more dynamic and distributed in comparison with traditional client-server applications. In such systems users are service requesters access a web service system to get local and global services. The local services do not involve services from other system providers since a system completes its tasks locally. For global services, a system collaborates with other providers in order to fulfil the requited services. The global services might be composited from local and global services from other system provider too. Therefore, the services in such system must be uniquely identified to be either local services or global services. For example, different service providers exchange messages to make the booking of a plain, hotel, and car for a user trip. Access control for interoperable systems are considerably complex due to dynamic, distributed, and heterogeneous nature of systems. The challenge in this kind of systems is to develop an effective access control model that deals with such aspects. Contextual information such as time and location conditions, especially for mobile applications, must be incorporated into the access control model to authorize an access. Additionally, the dynamic change of context information should be captured by the access control model while users on-the-move. We believe that our model can be extended to serve these objectives.

Delegation of access rights is one of the most important aspect of access control that our model, at the current state, does not support. The delegation of authorities is a business rule related to the access control policies for many organizations. A delegation operation allows an entity to pass all or part of its privileges to another entity under certain conditions defined by organizations' policies. The entity that transfers rights is often referred to as a delegator, and the entity that use that rights is called the delegatee. In many organizations, a delegations is needed in the situations like backup of a job function, decentralization of authorities, and collaboration of work. The first and third business rules require a temporary delegation while the second rule needs a durable delegation. An access control model is needed to cover these requirements in which delegation can only take place in a secure manner. In mobile applications such as ones presented in this dissertation, the delegation must depend on the spatio-temporal information. In many situations the transfer of rights happens for a short period of time to perform a task in a specific place. For example, a physician may transfer part of his privileges to a nurse for a given period of time and in a particular ward, when he is in an emergency. We plan to extend our model to incorporate delegations in mobile applications. In such model, the delegation of authorization is contingent based on the STZones conditions associated the model entities. Furthermore, our delegation model should deal with complex delegation operations through multi-level of role hierarchies and separation of duties in presence of spatio-temporal constraints.

We also plan to provide a more flexible spatio-temporal access control that is able to make authorization decisions in the presence of uncertainty, which is possible if the user's location cannot be accurately determined. For example, we considered in our model that the locations of objects are captured at any single point in time line. For continuously moving objects, such as users who are riding a train or boarding a plain, we should extend the definition of the location information in our model to support such requirement. We also plan to extend our model that is not only traditionally provides a yes/no answer to an access decision, it might ask for some actions to be taken in order to allow an access. In addition, this new model may also partially disable some access rights when a user moves to an unauthorized zone as an alternative of entirely disabling the user access. Penalties in case of violating a policy, such as repeatability moving to invalid zones after access, are another scheme that would be useful to incorporate into our prospective model. Last, but not least, our future work in this aspect also includes deploying our model for a real-world healthcare dengue decision support system (DDSS). This will allows one to naturally reason about the scalability in terms of the number of entities and relationships.

In the second context of developing rigours analysis approaches, we plan to introduce the following extension of our approaches. A short term of a research direction is to extend the UML/OCL analysis approach in Chapter 3 to verify temporal properties. In our approach, a UML model specifying an access control policy is analyzed against a set of temporal properties expressed in object-oriented temporal logic. We would like to check temporal properties such as something happens in the next step, always happen, eventually will happen, and always true until something else happens in the future. We intend to express such temporal properties in temporal OCL (TOCL), which is a temporal logic extension to OCL [105]. From a high-level perspective, it takes an access control policies specified by UML class diagram and a temporal property, then it checks if there is a scenario supported by the class diagram violating that property. This analysis approach should meet these objectives: it allows the validation of behavioral and temporal properties without the need of transformation, it is lightweight and henceforth cost-effective since it does not have sophisticated notations or require mathematical maturity, and temporal properties in our access control model should be specified in a form that is amenable to analyze.

Access control is an important mechanism to ensure the data security in workflow systems. An access control model handles the assignment of tasks and roles and defines SoD conflicts between tasks and roles. Particularly, in mobile application domain, workflow management systems are influenced by location and time information making them very sensitive to use. An analysis approach of such model is needed to uncover any violations at model and application levels. As such, another direction of work is developing a verification approach for spatio-temporal workflows. This analysis approach aims to uncover conflicts between tasks, improper execution of tasks by unauthorized users, violation of dependencies, deadlocks, etc, in the context of spatio-temporal domain. Deadlocks may take place in a workflow in case no participant is available to perform a task while some tasks are waiting the completion of that task at the first place. This problem arises due to an error in the workflow specification. In our workflow model, a user access to tasks is controlled by the GSTRBAC model. That is, tasks are associated with STZones that defines where and when a user can perform those tasks. We are interesting in the use of the CPN language for modeling the tasks and control flows dependencies, and employ appropriate toolbox to analyze the correct execution of the workflow model with regard to spatio-temporal constraints.

References

[1] S. Ahson and M. Ilyas, *Location-Based Services Handbook: Applications, Technologies, and Security*. CRC Press, Boca Raton, FL, USA, 2011, ISBN-10: 1420071963.

[2] W. Arensman, J. Whipple, and M. Boler, "A Public Safety Application of GPS-Enabled Smartphones and the Android Operating System," in *SMC*, 2009, pp. 2059–2061.

[3] C. Doukas, T. Pliakas, and I. Maglogiannis, "Mobile Healthcare Information Management Utilizing Cloud Computing and Android OS," in *EMBC*. IEEE, 2010, pp. 1037–1040.

[4] R. Salomon, M. Lüder, and G. Bieber, "iFall: A New Embedded System for The Detection of Unexpected Falls," in *PerCom Workshops*, 2010, pp. 286–291.

[5] A. Maji, A. Mukhoty, A. Majumdar, J. Mukhopadhyay, S. Sural, S. Paul, and B. Majumdar, "Security Analysis and Implementation of Web-Based Telemedicine Services with a Four-Tier Architecture," in *PCTHEALTH*, 2008, pp. 46–54.

[6] R. Abdunabi and I. Ray, "Extensions to the Role Based Access Control Model for Newer Computing Paradigms," in *CCWIC*, 2010.

[7] R. Abdunabi and I. Ray, "A Comparison of Security Analysis Techniques for RBAC Models," in *CCWIC*, 2010.

[8] M. Toahchoodee, R. Abdunabi, I. Ray, and I. Ray, "A Trust-Based Access Control Model for Pervasive Computing Applications," in *DBSec*, 2009, pp. 307–314.

[9] I. Ray and M. Toahchoodee, "A Spatio-temporal Role-Based Access Control Model," in *DBSec*, 2007, pp. 211–226.

[10] S. Aich, S. Sural, and A. Majumdar, "STARBAC: Spatio Temporal Role Based Access Control," in *OTM*, 2007, pp. 1567–1582.

[11] M. Toahchoodee and I. Ray, "On the Formalization and Analysis of a Spatio-Temporal Role-Based Access Control Model," *Journal of Computer Security*, vol. 19, no. 3, pp. 399–452, 2011.

[12] S. Aich, S. Mondal, S. Sural, and A. Majumdar, "Role Based Access Control with Spatiotemporal Context for Mobile Applications," *Transactions on Computational Science*, vol. 4, pp. 177–199, 2009.

[13] L. Chen and J. Crampton, "On Spatio-Temporal Constraints and Inheritance in Role-Based Access Control," in *ASIACCS*, 2008, pp. 205–216.

[14] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, 2nd ed. Addison-Wesley Professional, 2005, ISBN-10: 0321267974.

[15] D. Jackson, "Alloy: A Lightweight Object Modelling Notation," *ACM Transactions on Software Engineering Methodology*, vol. 11, no. 2, pp. 256–290, 2002.

[16] A. Samuel, A. Ghafoor, and E. Bertino, "A Framework for Specification and Verification of Generalized Spatio-Temporal Role Based Access Control Model," Purdue University, Tech. Rep. CERIAS 2007-08, February 2007.

[17] M. Toahchoodee and I. Ray, "Using Alloy to Analyse a Spatio-Temporal Access Control Model Supporting Delegation," *IET Information Security*, vol. 3, no. 3, pp. 75–113, 2009.

[18] B. Shafiq, A. Masood, J. Joshi, and A. Ghafoor, "A Role-Based Access Control Policy Verification Framework for Real-Time Systems," in *WORDS*, 2005, pp. 13–20.

[19] A. Ratzer, L. Wells, H. Lassen, M. Laursen, J. Qvortrup, M. Stissing, M. Westergaard, S. Christensen, and K. Jensen, "CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets," in *ICATPN*, 2003, pp. 450–462.

[20] R. Sandhu, K. Ranganathan, and X. Zhang, "Secure Information Sharing Enabled by Trusted Computing and PEI Models," in *ASIACCS*, 2006, pp. 2–12.

[21] S. Osborn, R. Sandhu, and Q. Munawer, "Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access Control Policies," *ACM Transactions on Information and System Security*, vol. 3, no. 2, pp. 85–106, 2000.

[22] L. Craig, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, 3rd ed. Prentice Hall, 2004, ISBN: 0131489062.

[23] M. Gogolla, F. Büttner, and M. Richters, "USE: A UML-Based Specification Environment for Validating UML and OCL," *Science of Computer Programming*, vol. 69, no. 1-3, pp. 27–34, 2007.

[24] R. Alur and D. Dill, "A Theory of Timed Automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994.

[25] P. Argenio, J. Katoen, T. Ruys, and J. Tretmans, "The Bounded Retransmission Protocol Must Be on Time!" in *TACAS*, 1997, pp. 416–431.

[26] K. Havelund, K. Larsen, and A. Skou, "Formal Verification of a Power Controller Using the Real-Time Model Checker UPPAAL," in *ARTS*, 1999, pp. 277–298.

[27] M. Bozga, J. Hou, O. Maler, and S. Yovine, "Verification of Asynchronous Circuits using Timed Automata," *ENTCS*, vol. 65, no. 6, pp. 47–59, 2002.

[28] K. Havelund, A. Skou, K. Larsen, and K. Lund, "Formal Modeling and Analysis of an Audio/Video Protocol: an Industrial Case Study using UPPAAL," in *RTSS*, 1997, pp. 2–13.

[29] R. Alur and R. Kurshan, "Timing Analysis in COSPAN," in *Hybrid Systems*, 1995, pp. 220–231.

[30] C. Daws, A. Olivero, S. Tripakis, and S. Yovine, "The Tool KRONOS," in *Hybrid Systems*, 1995, pp. 208–219.

[31] G. Behrmann, A. David, and K. Larsen, "A Tutorial on Uppaal," in *SFM*, 2004, pp. 200–236.

[32] R. Alur, C. Courcoubetis, and D. Dill, "Model-Checking for Real-Time Systems," in *LICS*, 1990, pp. 414–425.

[33] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, "Symbolic Model Checking for Real-Time Systems," *Information Science*, vol. 111, no. 2, pp. 193–244, 1994.

[34] B. Lampson, "Protection," *Operating Systems Review*, vol. 8, no. 1, pp. 18–24, 1974.

[35] D. Denning, "A Lattice Model of Secure Information Flow," *Communications of the ACM*, vol. 19, no. 5, pp. 236–243, 1976.

[36] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-Based Access Control Models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[37] USA Department of Defense Standard, *Department of Defense Trusted Computer Systems Evaluation Criteria (TCSEC)*. Department of Defense, 1985, ISBN-10: 0788143255.

[38] D. Bell and L. Lapadula, "Secure Computer System: Unified Exposition and Multics Interpretation," DTIC Document, Tech. Rep., March 1976.

[39] K. Biba, "Integrity Considerations for Secure Computer Systems," DTIC Document, Tech. Rep., 1977.

[40] R. Sandhu, "Lattice-Based Access Control Models," *IEEE Computer*, vol. 26, no. 11, pp. 9–19, 1993.

[41] R. Anderson, *Security Engineering - A Guide to Building Dependable Distributed Systems* , 2nd ed. Wiley, 2008, ISBN: 978-0-470-06852-6.

[42] J. Joshi, W. Aref, A. Ghafoor, and E. Spafford, "Security Models for Web-Based Applications," *Communications of the ACM*, vol. 44, no. 2, pp. 38–44, 2001.

[43] J. Joshi, E. Bertino, U. Latif, and A. Ghafoor, "A Generalized Temporal Role-Based Access Control Model," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 1, pp. 4–23, 2005.

[44] R. Simon and M. Zurko, “Separation of Duty in Role-Based Environments,” in *CSFW*, 1997, pp. 183–194.

[45] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati, “An Access Control Model Supporting Periodicity Constraints and Temporal Reasoning,” *ACM Transactions on Database Systems*, vol. 23, no. 3, pp. 231–285, 1998.

[46] A. Gal and V. Atluri, “An Authorization Model for Temporal Data,” in *CCS*, 2000, pp. 144–153.

[47] E. Bertino, P. Bonatti, and E. Ferrari, “TRBAC: A Temporal Role-Based Access Control Model,” *ACM Transactions on Information and System Security*, vol. 4, no. 3, pp. 191–233, 2001.

[48] M. Covington, W. Long, S. Srinivasan, A. Dey, M. Ahamad, and G. Abowd, “Securing Context-Aware Applications Using Environment Roles,” in *SACMAT*, 2001, pp. 10–20.

[49] F. Hansen and V. Oleshchuk, “SRBAC: A Spatial Role-Based Access Control Model for Mobile Systems,” in *NORDSEC*, 2003, pp. 129–141.

[50] E. Bertino, B. Catania, M. Damiani, and P. Perlasca, “GEO-RBAC: a Spatially Aware RBAC,” in *SACMAT*, 2005, pp. 29–37.

[51] I. Ray, M. Kumar, and L. Yu, “LRBAC: A Location-Aware Role-Based Access Control Model,” in *ICISS*, 2006, pp. 147–161.

[52] S. Chandran and J. Joshi, “*LoT-RBAC*: A Location and Time-Based RBAC Model,” in *WISE*, 2005, pp. 361–375.

[53] S. Jajodia, M. Kudo, and V. S. Subrahmanian, “Provisional Authorizations,” in *E-Commerce Security and Privacy*, 2001, pp. 133–159.

[54] C. Bettini, S. Jajodia, X. Wang, and D. Wijesekera, “Provisions and Obligations in Policy Rule Management,” *Journal of Network and Systems Management*, vol. 11, no. 3, pp. 351–372, 2003.

[55] M. Hilty, D. Basin, and A. Pretschner, "On Obligations," in *ESORICS*, 2005, pp. 98–117.

[56] J. Park and R. Sandhu, "The UCON_{ABC} Usage Control Model," *ACM Transactions on Information and System Security*, vol. 7, no. 1, pp. 128–174, 2004.

[57] C. Yuan, Y. He, J. He, and Z. Zhou, "A Verifiable Formal Specification for RBAC Model with Constraints of Separation of Duty," in *Inscrypt*, 2006, pp. 196–210.

[58] I. Ray, N. Li, R. France, and D.-K. Kim, "Using UML to Visualize Role-Based Access Control Constraints," in *SACMAT*, 2004, pp. 115–124.

[59] K. Sohr, G. Ahn, M. Gogolla, and L. Migge, "Specification and Validation of Authorisation Constraints Using UML and OCL," in *ESORICS*, 2005, pp. 64–79.

[60] M. Richters and M. Gogolla, "Validating UML Models and OCL Constraints," in *UML*, 2000, pp. 265–277.

[61] M. Toahchoodee and I. Ray, "On the Formal Analysis of a Spatio-temporal Role-Based Access Control Model," in *DBSec*, 2008, pp. 17–32.

[62] M. Toahchoodee, I. Ray, K. Anastasakis, G. Georg, and B. Bordbar, "Ensuring Spatio-Temporal Access Control for Real-World Applications," in *SACMAT*, 2009, pp. 13–22.

[63] K. Anastasakis, B. Bordbar, G. Georg, and I. Ray, "UML2Alloy: A Challenging Model Transformation," in *MoDELS*, 2007, pp. 436–450.

[64] K. Jensen, L. Kristensen, and L. Wells, "Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems," *International Journal on Software Tools for Technology Transfer*, vol. 9, no. 3-4, pp. 213–254, 2007.

[65] R. Laborde, B. Nasser, F. Grasset, F. Barrère, and A. Benzekri, "A Formal Approach for the Evaluation of Network Security Mechanisms Based on RBAC Policies," *Electronic Notes in Theoretical Computer Science*, vol. 121, pp. 117–142, 2005.

[66] L. Kristensen and L. Petrucci, “An Approach to Distributed State Space Exploration for Coloured Petri Nets,” in *ICATPN*, 2004, pp. 474–483.

[67] S. Mondal, S. Sural, and V. Atluri, “Towards Formal Security Analysis of GTRBAC Using Timed Automata,” in *SACMAT*, 2009, pp. 33–42.

[68] K. Godary, I. Augé-Blum, and A. Mignotte, “SDL and Timed Petri Nets versus UPPAAL for the Validation of Embedded Architecture in Automotive,” in *FDL*, 2004, pp. 672–684.

[69] B. Bérard, F. Cassez, S. Haddad, D. Lime, and O. Roux, “Comparison of the Expressiveness of Timed Automata and Time Petri Nets,” in *FORMATS*, 2005, pp. 211–225.

[70] F. Cassez and O. Roux, “Structural Translation from Time Petri Nets to Timed Automata,” *Journal of Systems and Software*, vol. 79, no. 10, pp. 1456–1468, 2006.

[71] D. Aprile, S. Donatelli, A. Sangnier, and J. Sproston, “From Time Petri Nets to Timed Automata: An Untimed Approach,” in *TACAS*, 2007, pp. 216–230.

[72] J. Srba, “Timed-Arc Petri Nets vs. Networks of Timed Automata,” in *ICATPN*, 2005, pp. 385–402.

[73] E. Uzun, V. Atluri, S. Sural, J. Vaidya, G. Parlato, A. L. Ferrara, and P. Madhusudan, “Analyzing Temporal Role Based Access Control models,” in *SACMAT*, 2012, pp. 177–186.

[74] S. Mondal and S. Sural, “Security Analysis of Temporal-RBAC Using Timed Automata,” in *IAS*, 2008, pp. 37–40.

[75] S. Mondal and S. Sural, “A Verification Framework for Temporal RBAC with Role Hierarchy (Short Paper),” in *ICISS*, 2008, pp. 140–147.

[76] M. Kirkpatrick and E. Bertino, “Enforcing Spatial Constraints for Mobile RBAC Systems,” in *SACMAT*, 2010, pp. 99–108.

[77] R. Abdunabi, M. Al-Lail, I. Ray, and R. France, “Specification, Validation, and Enforcement of a Generalized Spatio-Temporal Role-Based Access Control Model,” *IEEE Systems Journal*, 2013, To appear.

[78] "The Organization for the Advancement of Structured Information Standards (OASIS). eXtensible Access Control Markup Language (XACML)," 2012. [Online]. Available: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml/

[79] A. Anderson, "XACML Profile for Role-Based Access Control (RBAC)," *OASIS Access Control TC Committee Draft*, vol. 1, p. 13, 2004.

[80] M. Xu and D. Wijesekera, "A Role-Based XACML Administration and Delegation Profile and its Enforcement Architecture," in *SWS*, 2009, pp. 53–60.

[81] R. Sandhu, V. Bhamidipati, and Q. Munawer, "The ARBAC97 Model for Role-Based Administration of Roles," *ACM Transactions on Information Systems Security*, vol. 2, no. 1, pp. 105–135, 1999.

[82] V. Atluri and A. Gal, "An Authorization Model for Temporal and Derived Data: Securing Information Portals," *ACM Transactions on Information and System Security*, vol. 5, no. 1, pp. 62–94, 2002.

[83] R. Milner, *Communication and concurrency*, ser. PHI Series in Computer Science. Prentice Hall, 1989, ISBN: 978-0-13-115007-2.

[84] M. Niézette and J. Stévenne, "An Efficient Symbolic Representation of Periodic Time," in *CIKM*, 1992, pp. 161–168.

[85] R. Sandhu, D. Ferraiolo, and R. Kuhn, "The NIST Model for Role-Based Access Control: Towards a Unified Standard," in *ACM Workshop on Role-Based Access Control*, 2000, pp. 47–63.

[86] G. Ahn and R. Sandhu, "Role-based Authorization Constraints Specification," *ACM Transactions on Information and System Security*, vol. 3, no. 4, pp. 207–226, 2000.

[87] S. Jha, N. Li, M. Tripunitara, Q. Wang, and W. Winsborough, "Towards Formal Verification of Role-Based Access Control Policies," *IEEE Transactions on Dependable and Secure Computing*, vol. 5, no. 4, pp. 242–255, 2008.

[88] "US Government. Global Positioning System," March 15, 2012. [Online]. Available: http://gps.gov/

[89] D. Menascé, G. Popek, and R. Muntz, "A Locking Protocol for Resource Coordination in Distributed Databases," *ACM Transactions on Database Systems*, vol. 5, no. 2, pp. 103–138, 1980.

[90] B. Bose and S. Sane, "DTCOT: Distributed Timeout based Transaction Commit Protocol for Mobile Database Systems," in *ICWET*, 2010, pp. 518–523.

[91] M. T. Özsu and P. Valduriez, *Principles of Distributed Database Systems*, Second ed. Prentice-Hall, 1999, ISBN-10: 1441988335.

[92] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston, MA, USA: Addison-Wesley, 1995, ISBN-0: 201-63361-2.

[93] M. Howard and D. Leblanc, *Writing Secure Code*, Second ed. Redmond, WA, USA: Microsoft Press, 2002, ISBN = 0735617228.

[94] F. Swiderski and W. Snyder, *Threat Modeling (Microsoft Professional)*, Second ed. Redmond, WA, USA: Microsoft Press, 2004, ISBN: 0735619913.

[95] A. Chaudhuri, "Language-Based Security on Android," in *PLAS*, 2009, pp. 1–7.

[96] G. Lowe, "An Attack on the Needham-Schroeder Public-Key Authentication Protocol," *Information Processing Letters*, vol. 56, no. 3, pp. 131–133, 1995.

[97] A. Schaad and J. Moffett, "A Lightweight Approach to Specification and Analysis of Role-Based Access Control Extensions," in *SACMAT*, 2002, pp. 13–22.

[98] W. Sun, R. France, and I. Ray, "Rigorous Analysis of UML Access Control Policy Models," in *POLICY*, 2011, pp. 9–16.

[99] "Google Inc. The Android Mobile (OS)," 2012. [Online]. Available: http://www.android.com/

[100] "Google Inc. Android Developers," 2012. [Online]. Available: http://www.developer.android.com/

[101] "Technische Universität Darmstadt. FlexiProvider Research Group," 2012. [Online]. Available: http://www.flexiprovider.de/overview.html/

[102] "MySQL Inc. The World's Most Popular Open Source Database," 2012. [Online]. Available: http://www.mysql.com/

[103] D. Hollingsworth, "Workflow management coalition: The workflow reference model," *Document Number TC00-1003*, no. 1.1, 1995.

[104] S. Li, A. Kittel, D. Jia, and G. Zhuang, "Security Considerations for Workflow Systems," in *NOMS*, 2000, pp. 655–668.

[105] P. Ziemann and M. Gogolla, "OCL Extended with Temporal Logic," in *Ershov Memorial Conference*, 2003, pp. 351–357.

Appendix A

USE Specification of The UML/OCL GSTR-BAC Model

```
model GSTRBAC
--***************************Classes ****************
-- classes

class User
attributes
  name : String
operations
assignRole(r: Role, z:STZone): UserRoleAssignment
deassignRole(r:Role, z:STZone)
activateRole(r:Role, z:STZone): UserRoleActivation
deactivateRole(r:Role,z:STZone)
getAssignedRoles(z:STZone): Set(Role)=
self.relations->select(r |
r.oclIsTypeOf(UserRoleAssignment)and r.zone=z)->
collect( r| r.role)->asSet()
getActivatedRoles(z:STZone): Set(Role)=
self.relations->select(r |
r.oclIsTypeOf(UserRoleActivation)and r.zone=z)->
collect( r| r.role)->asSet()
getAuthorizedRoles(z:STZone):
Set(Role)= self.getAssignedRoles(z)->
union(self.getAssignedRoles(z)->collect(r|
r.getAllAHInheritedRoles(z))->asSet())
checkAccess(o:Object,a:Activity,z:STZone):Boolean =
getActivatedRoles(z)->collect( r |
r.getAuthorizedPermissions(z))->
asSet()->exists( p | p.object=o and p.activity=a)
end


class STZone
end

class Role
operations
addAHJuniorRole(r:Role,z:STZone): A_Hierarchy
  deleteAHJuniorRole(r:Role,z:STZone)
  addIHJuniorRole(r:Role,z:STZone): I_Hierarchy
  deleteIHJuniorRole(r:Role,z:STZone)
  addSSoDRole(r:Role,z:STZone): RSSOD
  deleteSSoDRole(r:Role,z:STZone)
  addDSoDRole(r:Role,z:STZone): DSOD
  deleteDSoDRole(r:Role,z:STZone)
  assignPermission(p:Permission,z:STZone): PermissionAssignment
  deassignPermission(p:Permission,z:STZone)
  getSSoDRoles(z:STZone): Set(Role)= self.sod->select( s |
  s.zone=z and s.oclIsTypeOf(RSSOD))->collect(s |
  s.getInvolvedRoles())->union(self.SOD->select( s |
  s.zone=z and s.oclIsTypeOf(RSSOD))->collect(s |
  s.getInvolvedRoles()))->excluding(self)->asSet()
getDSoDRoles(z:STZone): Set(Role)= self.sod->select( s |
s.zone=z and s.oclIsTypeOf(DSOD))->collect(s |
s.getInvolvedRoles())->union(self.SOD->select( s |
s.zone=z and s.oclIsTypeOf(DSOD))->collect(s |
s.getInvolvedRoles()))->excluding(self)->asSet()
getJuniorRoles(z:STZone): Set(Role) = RoleHierarchy.allInstances->
select(h| h.seniorRole=self and h.zone=z)->
collect( rh | rh.juniorRole)->asSet()
getAHJuniorRoles(z:STZone): Set(Role)=
A_Hierarchy.allInstances->select(ah |
ah.seniorRole=self and ah.zone=z)->
collect(ah1 | ah1.juniorRole)->asSet()
getIHJuniorRoles(z:STZone): Set(Role)=
I_Hierarchy.allInstances->select(ah |
ah.seniorRole=self and ah.zone=z)->
collect(ah1 | ah1.juniorRole)->asSet()
getPrerequisiteRoles(): Set(Role) =
self.prerequisiteRole->asSet()
inherits(r:Role,z:STZone): Boolean = if
(self.getJuniorRoles(z)->includes(r))
then true else self.getJuniorRoles(z)->
exists(j | j.inherits(r,z))endif

inheritsAH(r:Role,z:STZone): Boolean = if
(self.getAHJuniorRoles(z)->includes(r))
then true else self.getAHJuniorRoles(z)->
exists(j | j.inheritsAH(r,z))endif

inheritsIH(r:Role,z:STZone): Boolean = if
(self.getIHJuniorRoles(z)->includes(r))
then true else self.getIHJuniorRoles(z)->
exists(j | j.inheritsIH(r,z))endif

getAllAHInheritedRoles(z:STZone):
Set(Role)= Role.allInstances->
select(r | self.inheritsAH(r,z))->asSet()

getAllIHInheritedRoles(z:STZone): Set(Role)=
Role.allInstances->select(r | self.inheritsIH(r,z))->asSet()

getAssignedPermissions(z:STZone): Set(Permission)=
self.permAssig->select( pa| pa.zone=z )->
collect( pa1| pa1.permission)->asSet()

getAuthorizedPermissions(z:STZone): Set(Permission)=
self.getAssignedPermissions(z)->
union(self.getAllIHInheritedRoles(z)->collect(r |
r.getAssignedPermissions(z)))->asSet()

end

class Permission
operations
   addSoDPermission(p:Permission,z:STZone): PSSOD
   deleteSoDPermission(p:Permission,z:STZone)
   getSoDPermissions(z:STZone): Set(Permission)=
   self.pssod->select( s | s.zone=z)->collect(s |
   s.getInvolvedPermissions())->union(self.PSSOD->
   select(s | s.zone=z)->collect(s |
   s.getInvolvedPermissions()))->excluding(self)->asSet()
   getPrerequisitePermissions(): Set(Permission) =
   self.prerequisitePermission->asSet()
end

class Object
end

class Activity
end

class Location
end

class TimeInterval
end

abstract class UserRoleRlation
end

class UserRoleAssignment < UserRoleRlation
end

class UserRoleActivation < UserRoleRlation
end

abstract class RoleHierarchy
end

class A_Hierarchy < RoleHierarchy
end

class I_Hierarchy < RoleHierarchy
end

abstract class SOD
  operations
   getInvolvedRoles(): Set(Role) =
   self.firstRole->including(self.secondRole)
end
```

```
class RSSOD < SOD
end

class DSOD < SOD
end

class PermissionAssignment
end

class PSSOD
operations
   getInvolvedPermissions(): Set(Permission) =
   self.firstPermission->including(self.secondPermission)
end




--************************Associations *************
-- associations

association URRUser between
  User[1] role user
  UserRoleRlation[*] role relations
end

association URRRole between
  Role[1] role role
  UserRoleRlation[*] role relations
end

association URRZone between
  STZone[1] role zone
  UserRoleRlation[*] role relations
end

aggregation ZoneLocation between
  STZone [1..*] role include
  Location [1] role location
end

aggregation ZoneTimeInterval between
  STZone [1..*] role include
  TimeInterval [1] role interval
end

association UserZone between
  User [*] role users
  STZone [1..*] role currentzones
end


association RoleZone between
  Role [1..*] role roles
  STZone [1..*] role allowedzones
```

```
end

association PermissionZone between
  Permission [1..*] role permissions
  STZone [1..*] role zones
end

association ObjectZone between
  Object [*] role objects
  STZone [1..*] role zones
end

aggregation PermissionObject between
  Permission[*] role permission
  Object[1] role object
end

aggregation PermissionActivity between
  Permission[*] role permission
  Activity[1] role activity
end


association RolePermZone between
  PermissionAssignment [*] role permAssig
  STZone[1] role zone
end

association SODZone between
  SOD [*] role sod
  STZone[1] role zone
end

association PSSODZone between
  PSSOD [*] role pssod
  STZone [1] role zone
end

association RHZone between
  RoleHierarchy [*] role rh
  STZone[1] role zone
end

association RH1Role between
  Role [1] role juniorRole
  RoleHierarchy[*] role RH
end

association RH2Role between
  Role [1] role seniorRole
  RoleHierarchy[*] role rh
end

association SOD1Role between
  Role[1] role firstRole
  SOD[*] role sod
end

association SOD2Role between
  Role[1] role secondRole
  SOD[*] role SOD
end

association PSSOD1Permission between
  Permission [1] role firstPermission
  PSSOD [*] role pssod
end

association PSSOD2Permission between
  Permission [1] role secondPermission
  PSSOD [*] role PSSOD
end

association PerAssToRole between
  Role[1] role role
  PermissionAssignment[*] role permAssig
end

association PerAssiToPermission between
  Permission[1] role permission
  PermissionAssignment[*] role PermAssig
end

association PrerequisiteRole between
  Role[*] role prerequisiteRole
  Role[*] role requistorRole
end

association PrerequisitePermission between
  Permission[*] role prerequisitePermission
  Permission[*] role requistorPermission
end


--*************Constraints and Invariants********

constraints

context RSSOD
inv SSOD_Constraint: not UserRoleAssignment.allInstances->
exists(ura1,ura2 | ura1.user=ura2.user and
ura1.role=self.firstRole and ura2.role=secondRole and
ura1.zone=self.zone and ura2.zone=self.zone)

context User
inv SSOD_With_RH_Constraint: STZone.allInstances->
forAll( z | not self.getAuthorizedRoles(z)->
exists(r1,r2 | r1.getSSoDRoles(z)->includes(r2)))

context User
```

```
inv Activation_Constraint_with_RH: self.currentzones->
forAll(z| self.getAuthorizedRoles(z)->
includesAll(self.getActivatedRoles(z)))


context Role
inv Permission_Inheritance_Constraint1:
STZone.allInstances->forAll(z |
self.getAuthorizedPermissions(z)->
includesAll(self.getAllIHInheritedRoles(z)->
collect(r | r.getAssignedPermissions(z))->asSet()))
inv Permission_Inheritance_Constraint2:
STZone.allInstances->forAll(z |
self.getAllIHInheritedRoles(z)->forAll( r |
r.getAssignedPermissions(z)->
intersection(self.getAuthorizedPermissions(z))=
r.getAssignedPermissions(z)))


context User
inv Activation_Constraint: self.currentzones->
forAll(z| self.getAssignedRoles(z)->
includesAll(self.getActivatedRoles(z)))

context Role
 inv Hierarchy_Cycle_Constraint: not STZone.allInstances->
 exists(z| self.inherits(self,z))

context DSOD
inv DSOD_Constraint1: not UserRoleActivation.allInstances->
exists(ura1,ura2 | ura1.user=ura2.user and
ura1.role=self.firstRole and ura2.role=secondRole
and ura1.zone=self.zone and ura2.zone=self.zone)

context User
  inv DSOD_Constaint2: STZone.allInstances->
  forAll( z | not self.getActivatedRoles(z)->
  exists(r1,r2 | r1.getDSoDRoles(z)->includes(r2)))

context PSSOD
inv PSOD_Constraint1: not PermissionAssignment.allInstances->
exists(pa1,pa2 | pa1.role=pa2.role and
pa1.permission=self.firstPermission and
pa2.permission=self.secondPermission and
pa1.zone=self.zone and pa2.zone=self.zone)

context Role
  inv PSOD_RH_Constaint: STZone.allInstances->
  forAll( z | not self.getAuthorizedPermissions(z)->
  exists(p1,p2 | p1.getSoDPermissions(z)->includes(p2)))

context User
inv Prerequist_URAssign: STZone.allInstances->
forAll(z | Role.allInstances->forAll(r1 |
(self.getAssignedRoles(z)->includes(r1))
implies (self.getAssignedRoles(z)->
includesAll(r1.getPrerequisiteRoles())))))

context User
inv Prerequist_URActiv: STZone.allInstances->
forAll(z | Role.allInstances->forAll(r1 |
(self.getActivatedRoles(z)->includes(r1)) implies
(self.getActivatedRoles(z)->
includesAll(r1.getPrerequisiteRoles())))))

--****Operations Specifications***********


--***********User Operations***********

context User::assignRole(r: Role, z:STZone):
UserRoleAssignment
   pre assignRolePreCond1_definedObjects:
   r.isDefined and z.isDefined
   pre assignRolePreCond2_ZoneIncluded:
   self.currentzones->includes(z) and
   r.allowedzones->includes(z)
  pre assignRolePreCond3_RoleNotAssigned:
  self.getAssignedRoles(z)->excludes(r)
  pre assignRolePreCond4_RoleNotSSoD:
  self.getAssignedRoles(z)->collect(r |
  r.getSSoDRoles(z))->excludes(r)
  post AssignSTRolePostCond1_NewUserRoleRelation:
  (self.relations - self.relations@pre)->size()=1
post AssignSTRolePostCond2_NewRoleAssignment:
(self.relations - self.relations@pre)->
forAll( rl | rl.oclIsNew() and
rl.oclIsTypeOf(UserRoleAssignment) and rl.zone=z
and rl.role->includes(r))
post AssignSTRolePostCond3_RoleIsAssigned:
self.getAssignedRoles(z)->includes(r)


context User::deassignRole(r:Role, z:STZone)
pre deassignRolePreCond1_RoleIsAssigned:
self.getAssignedRoles(z)->includes(r)
post deassignRolePostCond1_RoleDeassigned:
self.getAssignedRoles(z)->excludes(r)
post deassignRolePostCond2_RoleAssignmentObjectDeleted:
(self.relations@pre - self.relations)->size()=1 and
(UserRoleAssignment.allInstances@pre -
UserRoleAssignment.allInstances)->size()=1

context User::activateRole(r: Role, z:STZone): UserRoleActivation
   pre activateRolePreCond1_denfinedObject:
   r.isDefined and z.isDefined
pre activateRolePreCond2_ZoneIncluded:
self.currentzones->includes(z) and r.allowedzones->includes(z)
  pre activateRolePreCond3_RoleNot:
  self.getActivatedRoles(z)->excludes(r)
```

```
pre activateRolePreCond4_RoleIsAssigned:
getAssignedRoles(z)->includes(r)
post activateRolePostCond1_NewUserRoleRelation:
(self.relations - self.relations@pre)->size()=1
post activateRolePostCond2_NewRoleActivation:
(self.relations - self.relations@pre)->
forAll( rl | rl.oclIsNew() and rl.oclIsTypeOf(UserRoleActivation)
and rl.zone=z and rl.role->includes(r))
post activateRolePostCond3_RoleIsAssigned:
self.getActivatedRoles(z)->includes(r)

context User::deactivateRole(r:Role,z:STZone)
pre deactivateRolePreCond1_RoleIsActivated:
self.getActivatedRoles(z)->includes(r)
post deactivateRolePostCond1_RoleDeactivated:
self.getActivatedRoles(z)->excludes(z)
post deactivateRolePostCond2_RoleActivationDeleted:
(self.relations@pre - self.relations)->size()=1 and
(UserRoleActivation.allInstances@pre -
UserRoleActivation.allInstances)->size()=1


--***********Role Operations***********

context Role::addAHJuniorRole(r:Role,z:STZone): A_Hierarchy
pre addAHJuniorRolePreCond1_definedObjects:
r.isDefined and z.isDefined
pre addAHJuniorRolePreCond2_ZoneIncluded:
self.allowedzones->includes(z) and
r.allowedzones->includes(z)
pre addAHJuniorRolePreCond3_NotAHJuniorRole:
getAHJuniorRoles(z)->excludes(r)
post addAhJuniorRolePostCond1_NewRoleHierarchy:
(self.rh - self.rh@pre)->size()=1 and
(self.RH - self.RH@pre)->size()=1
post addAhJuniorRolePostCond2_NewRoleA_Hierarchy:
(self.rh - self.rh@pre)->forAll( rh | rh.oclIsNew()
and rh.oclIsTypeOf(A_Hierarchy) and rh.zone=z
and rh.juniorRole=r and rh.seniorRole=self)
post addAhJuniorRolePostCond3_RoleIsAdded:
self.getAHJuniorRoles(z)->includes(r)

context Role::deleteAHJuniorRole(r:Role, z:STZone)
pre deleteAHJuniorRolePreCond1_RoleIsJuniorRole:
self.getAHJuniorRoles(z)->includes(r)
post deleteAHJuniorRolePostCond1_RoleDeleated:
self.getAHJuniorRoles(z)->excludes(r)
post deleteAHJuniorRolePostCond2_AHierarchyObjectDeleted:
(self.rh@pre - self.rh)->size()=1 and
(self.RH@pre - self.RH)->size()=1 and
(A_Hierarchy.allInstances@pre - A_Hierarchy.allInstances)->size()=1

context Role::addIHJuniorRole(r:Role,z:STZone): I_Hierarchy
pre addIHJuniorRolePreCond1_definedObjects:
r.isDefined and z.isDefined
```

```
pre addIHJuniorRolePreCond2_ZoneIncluded:
self.allowedzones->includes(z) and r.allowedzones->includes(z)
pre addIHJuniorRolePreCond3_NotIHJuniorRole:
getIHJuniorRoles(z)->excludes(r)
post addIHJuniorRolePostCond1_NewRoleHierarchy:
(self.rh - self.rh@pre)->size()=1 and
(self.RH - self.RH@pre)->size()=1
post addIHJuniorRolePostCond2_NewRoleI_Hierarchy:
(self.rh - self.rh@pre)->forAll( rh | rh.oclIsNew()
and rh.oclIsTypeOf(I_Hierarchy) and rh.zone=z
and rh.juniorRole=r and rh.seniorRole=self)
post addIHJuniorRolePostCond3_RoleIsAdded:
self.getIHJuniorRoles(z)->includes(r)


context Role::deleteIHJuniorRole(r:Role, z:STZone)
pre deleteIHJuniorRolePreCond1_RoleIsJuniorRole:
self.getIHJuniorRoles(z)->includes(r)
post deleteIHJuniorRolePostCond1_RoleDeleated:
self.getIHJuniorRoles(z)->excludes(r)
post deleteIHJuniorRolePostCond2_IHierarchyObjectDeleted:
(self.rh@pre - self.rh)->size()=1 and
(self.RH@pre - self.RH)->size()=1 and
(I_Hierarchy.allInstances@pre - I_Hierarchy.allInstances)->size()=1

context Role::addSSoDRole(r:Role,z:STZone): RSSOD
pre addSSoDRolePreCond1_definedObjects:
r.isDefined and z.isDefined
pre addSSoDRolePreCond2_ZoneIncluded:
self.allowedzones->includes(z) and r.allowedzones->includes(z)
pre addSSoDRolePreCond3_NotSSODRole: getSSoDRoles(z)->excludes(r)
post addSSoDRoleRolePostCond1_NewSOD:
(self.sod - self.sod@pre)->size()=1 and
(self.SOD - self.SOD@pre)->size()=1
post addSSoDRoleRolePostCond2_NewSSOD:
(self.sod - self.sod@pre)->forAll( sod | sod.oclIsNew()
and sod.oclIsTypeOf(RSSOD) and sod.zone=z
and sod.firstRole=r and sod.secondRole=self)
post addSSoDRoleRolePostCond3_RoleIsAdded:
self.getSSoDRoles(z)->includes(r)

context Role::deleteSSoDRole(r:Role,z:STZone)
pre deleteSSoDRolePreCond1_RoleIsSSoDRole:
self.getSSoDRoles(z)->includes(r)
post deleteSSoDRolePostCond1_RoleDeleated:
self.getSSoDRoles(z)->excludes(r)
post deleteSSoDRolePostCond2_SSODObjectDeleted:
(self.sod@pre - self.sod)->size()=1 and
(self.SOD@pre - self.SOD)->size()=1  and
(RSSOD.allInstances@pre - RSSOD.allInstances)->size()=1

context Role::addDSoDRole(r:Role,z:STZone): DSOD
pre addDSoDRolePreCond1_definedObjects:
r.isDefined and z.isDefined
pre addDSoDRolePreCond2_ZoneIncluded:
```

```
self.allowedzones->includes(z) and
r.allowedzones->includes(z)
pre addDSoDRolePreCond3_NotDSODRole:
getDSoDRoles(z)->excludes(r)
post addDSoDRoleRolePostCond1_NewSOD:
(self.sod - self.sod@pre)->size()=1 and
(self.SOD - self.SOD@pre)->size()=1
post addDSoDRoleRolePostCond2_NewDSOD:
(self.sod - self.sod@pre)->forAll( sod |
sod.oclIsNew() and sod.oclIsTypeOf(DSOD)
and sod.zone=z and sod.firstRole=r and
sod.secondRole=self)
post addDSoDRoleRolePostCond3_RoleIsAdded:
self.getDSoDRoles(z)->includes(r)


context Role::deleteDSoDRole(r:Role,z:STZone)
pre deleteDSoDRolePreCond1_RoleIsSSoDRole:
self.getDSoDRoles(z)->includes(r)
post deleteDSoDRolePostCond1_RoleDeleated:
self.getDSoDRoles(z)->excludes(r)
post deleteDSoDRolePostCond2_DSODObjectDeleted:
(self.sod@pre - self.sod)->size()=1 and
(self.SOD@pre - self.SOD)->size()=1  and
(DSOD.allInstances@pre - DSOD.allInstances)->size()=1

context Role::assignPermission(p:Permission,z:STZone):
PermissionAssignment
pre assignPermissionPreCond1_definedObjects:
p.isDefined and z.isDefined
pre assignPermissionCond2_ZoneIncluded:
p.zones->includes(z) and self.allowedzones->includes(z)
pre assignPermissionPreCond3_PermissionNotAssigned:
self.getAssignedPermissions(z)->excludes(p)
pre assignPermissionPreCond4_PermissionNotSSoD:
self.getAssignedPermissions(z)->collect(per |
per.getSoDPermissions(z))->excludes(p)
post assignPermissionPostCond1_NewPermissionAssignment:
(self.permAssig - self.permAssig@pre)->size()=1
post assignPermissionPostCond2_NewRoleAssignment:
(self.permAssig - self.permAssig@pre)->forAll( pa |
pa.oclIsNew() and pa.zone=z and pa.permission->includes(p))
post assignPermissionPostCond3_PermissionIsAssigned:
self.getAssignedPermissions(z)->includes(p)


context Role::deassignPermission(p:Permission,z:STZone)
pre deassignPermissionPreCond1_PermissionIsAssigned:
self.getAssignedPermissions(z)->includes(p)
post deassignPermissionPostCond1_PermissionDeassigned:
self.getAssignedPermissions(z)->excludes(p)
post deassignPermissionPostCond2_PermissionAssignmentObjectDeleted:
(self.permAssig@pre - self.permAssig)->size()=1
and (PermissionAssignment.allInstances@pre -
PermissionAssignment.allInstances)->size()=1
```

```
--****Permission Operations*****************

context Permission::addSoDPermission(p:Permission,z:STZone): PSSOD
pre addSoDPermissionPreCond1_definedObjects:
p.isDefined and z.isDefined
pre addSoDPermissionPreCond2_ZoneIncluded:
self.zones->includes(z) and p.zones->includes(z)
pre addSoDPermissionCond3_NotSSODPermission:
getSoDPermissions(z)->excludes(p)
post addSoDPermissionPostCond1_NewSOD:
(self.pssod - self.pssod@pre)->size()=1 and
(self.PSSOD - self.PSSOD@pre)->size()=1
post addSoDPermissionPostCond2_NewPSOD:
(self.pssod - self.pssod@pre)->forAll( pssod |
pssod.oclIsNew() and pssod.zone=z and
pssod.firstPermission=p and pssod.secondPermission=self)
post addSoDPermissionPostCond3_PermissionIsAdded:
self.getSoDPermissions(z)->includes(p)

context Permission::deleteSoDPermission(p:Permission,z:STZone)
pre deleteSoDPermissionPreCond1_PermissionIsSSoDRole:
self.getSoDPermissions(z)->includes(p)
post deleteSoDPermissionPostCond1_RoleDeleated:
self.getSoDPermissions(z)->excludes(p)
post deleteSoDPermissionPostCond2_PSSODObjectDeleted:
(self.pssod@pre - self.pssod)->size()=1 and
(self.PSSOD@pre - self.PSSOD)->size()=1  and
(PSSOD.allInstances@pre - PSSOD.allInstances)->size()=1
```

Appendix B

Partial UPPAAL Timed-Automata Code of The DDSS Policy Model

```
<?xml version="1.0" encoding="utf-8"?><!DOCTYPE
nta PUBLIC '-//Uppaal Team//DTD
Flat System 1.1//EN'
'http://www.it.uu.se/research/group/darts/uppaal/
flat-1_1.dtd'><nta><declaration>//
Place global declarations here.
clock x;

const int N = 12;
//int[0,7] uid;
//const int L=5;

const int mainwarehouse = 0, stateclinic = 1,
cityclinic = 2, statepo=3, cityepo=4,
mainoffice = 5, citywarehouse=6, vmainoffice = 7,
vcityoffice = 8, city = 9; // locations

const int shcid = 0, chcid = 1, seid=2, ceid=3,
pmid = 4, cmmid=6, cvmid=8, vctid = 10,
vstid = 10; // locations

chan activate_SHC[N], activate_CHC[N],
activate_SE[N], activate_CE[N],
activate_PM[N], activate_CMM[N],
      activate_CVM[N], activate_VCT[N], activate_VST[N];

chan Ap1[N], Ap2[N], Ap3[N], Ap4[N], Ap12[N],
Ap13[N], Ap7[N], Ap10[N], Ap16[N], Ap17[N];
chan Ep1[N], Ep2[N], Ep3[N], Ep4[N], Ep7[N],
Ep12[N], Ep13[N], Ep10[N], Ep17[N], Ep16[N];

chan Aobj2[N], Aobj1[N], Aobj4[N], Aobj5[N], Aobj6[N];
chan Eobj2[N], Eobj1[N], Eobj4[N], Eobj5[N], Eobj6[N];

urgent chan  deactivate_VCT[N], deactivate_VST[N],
deactivate_SHC[N], deactivate_CHC[N],
deactivate_PM[N], deactivate_SE[N],
deactivate_CE[N], deactivate_CMM[N],
deactivate_CVM[N];

// SHC Roles
int SHC_L[N]; int SHC_u[7]; int SHC_count[N];
```

```
// CHC role
int CHC_L[N]; int CHC_u[7]; int CHC_count[N];

// SE role
int SE_L[N]; int SE_u[7]; int SE_count[N];

// CE role
int CE_L[N]; int CE_u[7]; int CE_count[N];

// PM role
int PM_L[N]; int PM_u[7]; int PM_count[N];

// CVM role
int CVM_L[N]; int CVM_u[7]; int CVM_count[N];

// CMM role
int CMM_L[N]; int CMM_u[7]; int CMM_count[N];

// VCT role
int VCT_L[N]; int VCT_u[7]; int VCT_count[N];

// VST role
int VST_L[N]; int VST_u[7]; int VST_count[N];

// user location indicator
int u_L[N];

////////////////////////////////////////////
// Check that a user can leave a place

bool out(int uid){

if (u_L[uid] == 0) return true; else return false;

}

///////////////////////////////////////////////////

bool acheck(int uid, int rid, int l) {

if (rid == shcid &amp;&amp; SHC_u[uid] == 0
&amp;&amp; SHC_L[l] == 0 ) {return true;}
else
if (rid == chcid &amp;&amp; CHC_u[uid] == 0
&amp;&amp; SHC_u[uid] == 1 &amp;&amp;
CHC_L[l] == 0) {return true;}
else
if (rid == seid &amp;&amp; SE_u[uid] == 0
&amp;&amp; SE_L[l] == 0 ) {return true;}
else
if (rid == pmid &amp;&amp; PM_u[uid] == 0
&amp;&amp; PM_L[l] == 0) {return true;}
else
if (rid == ceid &amp;&amp; CE_u[uid] == 0
&amp;&amp; SE_u[uid] == 1 &amp;&amp;
CE_L[l] == 0) {return true;}
else
if (rid == cmmid &amp;&amp; CMM_u[uid] == 0
&amp;&amp; CMM_L[l] == 0) {return true;}
else
if (rid == cvmid &amp;&amp; CVM_u[uid] == 0
&amp;&amp; CVM_L[l] == 0) {return true;}
else
if (rid == vctid &amp;&amp; VCT_u[uid] == 0
&amp;&amp; VCT_L[l] == 0) {return true;}
else
if (rid == vstid &amp;&amp; VST_u[uid] == 0
&amp;&amp; VST_L[l] == 0) {return true;}
return false;
}

bool dcheck(int uid, int rid, int l) {

if (rid == shcid &amp;&amp; SHC_u[uid] == 1
&amp;&amp; SHC_L[l] == 1 &amp;&amp; CHC_u[uid] == 0)
{return true;}
else
if (rid == chcid &amp;&amp; CHC_u[uid] == 1
&amp;&amp; CHC_L[l] == 1) {return true;}
else
if (rid == seid &amp;&amp; SE_u[uid] == 1
&amp;&amp; SE_L[l] == 1 &amp;&amp; CE_u[uid] == 0)
{return true;}
else
if (rid == pmid &amp;&amp; PM_u[uid] == 1
&amp;&amp; PM_L[l] == 1) {return true;}
else
if (rid == ceid &amp;&amp; CE_u[uid] == 1
&amp;&amp; CE_L[l] == 1) {return true;}
else
if (rid == cvmid &amp;&amp; CVM_u[uid] == 1
&amp;&amp; CVM_L[l] == 1) {return true;}
else
if (rid == vctid &amp;&amp; VCT_u[uid] == 1
&amp;&amp; VCT_L[l] == 1) {return true;}
else
if (rid == vstid &amp;&amp; VST_u[uid] == 1
&amp;&amp; VST_L[l] == 1) {return true;}

return false;
}

void aupdate(int uid, int rid, int l)
{

  if (rid == shcid) {SHC_L[l] = 1; SHC_u[uid] = 1;
  SHC_count[l]++; u_L[uid]++;}
  else
  if (rid == chcid) {CHC_L[l] = 1; CHC_u[uid] = 1;
  CHC_count[l]++; u_L[uid]++;}
  else
  if (rid == seid) {SE_L[l] = 1; SE_u[uid] = 1;
  SE_count[l]++; u_L[uid]++;}
  else
  if (rid == pmid) {PM_L[l] = 1; PM_u[uid] = 1;
  PM_count[l]++; u_L[uid]=1;}
  else
  if (rid == ceid) {CE_L[l] = 1; CE_u[uid] = 1;
  CE_count[l]++; u_L[uid]++;}
  else
  if (rid == cmmid) {CMM_L[l] = 1; CMM_u[uid] = 1;
  CMM_count[l]++; u_L[uid]++;}
  else
  if (rid == cvmid) {CVM_L[l] = 1; CVM_u[uid] = 1;
  CVM_count[l]++; u_L[uid]++;}
  else
  if (rid == vctid) {VCT_L[l] = 1; VCT_u[uid] = 1;
  VCT_count[l]++; u_L[uid]++;}
  else
  if (rid == vstid) {VST_L[l] = 1; VST_u[uid] = 1;
  VST_count[l]++; u_L[uid]++;}

}

void dupdate(int uid, int rid, int l)
{
  if (rid == shcid) {SHC_L[l] = 0; SHC_u[uid] = 0;
  u_L[uid]--; if (SHC_count[l] == 1) { SHC_count[l] = 0;}
  else { SHC_count[l]--;} }
  else
  if (rid == chcid) {CHC_L[l] = 0; CHC_u[uid] = 0;
  u_L[uid]--; if (CHC_count[l] == 1) { CHC_count[l] = 0;}
  else { CHC_count[l]--;} }
  else
  if (rid == seid) {SE_L[l] = 0; SE_u[uid] = 0;
  u_L[uid]--; if (SE_count[l] == 1) { SE_count[l] = 0;}
  else { SE_count[l]--;} }
  else
  if (rid == pmid) {PM_L[l] = 0; PM_u[uid] = 0;
  u_L[uid]=0; if (PM_count[l] == 1) {PM_count[l] = 0;}
  else { PM_count[l]--;} }
  else
  if (rid == ceid) {CE_L[l] = 0; CE_u[uid] = 0;
  u_L[uid]--; if (CE_count[l] == 1) { CE_count[l] = 0;}
  else { CE_count[l]--;} }
  else
  if (rid == cmmid) {CMM_L[l] = 0; CMM_u[uid] = 0;
  u_L[uid]--; if (CMM_count[l] == 1) { CMM_count[l] = 0;}
  else { CMM_count[l]--;} }
  else
  if (rid == cvmid) {CVM_L[l] = 0; CVM_u[uid] = 0;
  u_L[uid]--; if (CVM_count[l] == 1) { CVM_count[l] = 0;}
  else { CVM_count[l]--;} }
  else
  if (rid == vctid) {VCT_L[l] = 0; VCT_u[uid] = 0;
  u_L[uid]--; if (VCT_count[l] == 1) { VCT_count[l] = 0;}
  else { VCT_count[l]--;} }
  else
  if (rid == vstid) {VST_L[l] = 0; VST_u[uid] = 0;
  u_L[uid]--; if (VST_count[l] == 1) { VST_count[l] = 0;}
  else { VST_count[l]--;} }

}
```

Appendix C

Alloy Specification of The Access Control Protocol

```
module SecurityProtocol

open util/ordering[Snapshot] as SnapshotSequence

abstract sig Key{}
sig PubKey, PriKey extends Key{}

sig ID{}
sig STZone{}
sig Role{}
sig Password{}
sig OnePassword{}
sig Timestamp{}

sig Snapshot {

rcmclient: one User,
ramserver: one Module,
acmserver: one Module,

attacker: one Module,
}

sig Module{
id : one ID,
pubkey: one PubKey,
prikey: one PriKey,
}

sig User extends Module{
stzone: one STZone,
r: one Role,
pwd: one Password,
onepwd: one OnePassword,
ids: one ID
}

sig Attacker extends User{}

pred M1[disj before, after: Snapshot,
senderPre, senderPost: User,
receiverPre, receiverPost: Module,
stzonePre, stzonePost: STZone, rPre, rPost: Role,
iduPre, iduPost: ID, idsPre, idsPost: ID,
pubkeyPre, pubkeyPost: PubKey] {

senderPre = before.rcmclient
receiverPre = before.ramserver

senderPre.stzone = stzonePre
senderPre.r = rPre
senderPre.id = iduPre
senderPre.ids = idsPre

receiverPre.pubkey = pubkeyPre

senderPost = after.rcmclient
receiverPost = after.ramserver

before.rcmclient = after.rcmclient
before.ramserver = after.ramserver
before.acmserver = after.acmserver
before.attacker = after.attacker
}

pred RCM2Attacker[disj before, after: Snapshot,
senderPre, senderPost: User, receiverPre, receiverPost: Attacker,
stzonePre, stzonePost: STZone, rPre, rPost: Role,
iduPre, iduPost: ID, idsPre, idsPost: ID,
pubkeyPre, pubkeyPost: PubKey]{

senderPre = before.rcmclient
receiverPre = before.attacker

senderPre.stzone = stzonePre
senderPre.r = rPre
senderPre.id = iduPre
senderPre.ids = idsPre

receiverPre.pubkey = pubkeyPre

senderPost = after.rcmclient
receiverPost = after.attacker

before.rcmclient = after.rcmclient
before.ramserver = after.ramserver
before.acmserver = after.acmserver
before.attacker = after.attacker
}

pred Attacker2RAM[disj before, after: Snapshot,
senderPre, senderPost: Attacker,
receiverPre, receiverPost: Module, stzonePre, stzonePost:
STZone, rPre, rPost: Role, iduPre, iduPost: ID,
idsPre, idsPost: ID, pubkeyPre, pubkeyPost: PubKey]{

senderPre = before.attacker
receiverPre = before.ramserver
```

```
senderPre.stzone = stzonePre
senderPre.r = rPre
senderPre.id = iduPre
senderPre.ids = idsPre

receiverPre.pubkey = pubkeyPre

senderPost = after.attacker
receiverPost = after.ramserver

before.rcmclient = after.rcmclient
before.ramserver = after.ramserver
before.acmserver = after.acmserver
before.attacker = after.attacker
}

pred M4[disj before, after: Snapshot,
senderPre, senderPost: Module, receiverPre,
receiverPost: User, idrsPre, idrsPost: ID, iduPre,
iduPost: ID, idsPre, idsPost: ID, pubkeyPre,
pubkeyPost: PubKey]{

senderPre = before.ramserver
receiverPre = before.rcmclient

senderPre.id = idrsPre

receiverPre.id = iduPre
receiverPre.ids = idsPre

receiverPre.pubkey = pubkeyPre

senderPost = after.ramserver
receiverPost = after.rcmclient

before.rcmclient = after.rcmclient
before.ramserver = after.ramserver
before.acmserver = after.acmserver
before.attacker = after.attacker
}


pred RAM2Attacker[disj before, after:
Snapshot, senderPre, senderPost: Module, receiverPre,
receiverPost: Attacker, idrsPre, idrsPost:
ID, iduPre, iduPost: ID, idsPre, idsPost: ID,
pubkeyPre, pubkeyPost: PubKey]{

senderPre = before.ramserver
receiverPre = before.attacker

senderPre.id = idrsPre

receiverPre.id = iduPre
receiverPre.ids = idsPre

receiverPre.pubkey = pubkeyPre

senderPost = after.ramserver
receiverPost = after.attacker

before.rcmclient = after.rcmclient
before.ramserver = after.ramserver
before.acmserver = after.acmserver
before.attacker = after.attacker
}

pred Attacker2RCM[disj before, after: Snapshot,
senderPre, senderPost: User, receiverPre,
receiverPost: Attacker, idrsPre, idrsPost:
ID, iduPre, iduPost: ID, idsPre, idsPost: ID, pubkeyPre,
pubkeyPost: PubKey]{

senderPre = before.attacker
receiverPre = before.rcmclient

senderPre.id = idrsPre

receiverPre.id = iduPre
receiverPre.ids = idsPre

receiverPre.pubkey = pubkeyPre

senderPost = after.attacker
receiverPost = after.rcmclient

before.rcmclient = after.rcmclient
before.ramserver = after.ramserver
before.acmserver = after.acmserver
before.attacker = after.attacker
}

pred Scenario1{
let first = SnapshotSequence/first|let second =
SnapshotSequence/next[first] | let third =
SnapshotSequence/next[second]|
some rcmclient: User| some ramserver: Module|
some disj idu, ids, idrs: ID| some r: Role|
some stzone: STZone| some disj pubkeyrcm, pubkeyram: PubKey|
M1[first, second, rcmclient, rcmclient, ramserver,
ramserver, stzone, stzone, r, r, idu, idu, ids, ids,
pubkeyram, pubkeyram] and M4[second, third, ramserver,
ramserver, rcmclient, rcmclient, idrs, idrs, idu, idu, ids, ids,
pubkeyrcm, pubkeyrcm]
}

run Scenario1 for 3

pred Scenario2{
```

```
let first = SnapshotSequence/first|let second =
SnapshotSequence/next[first]|
let third = SnapshotSequence/next[second]|
let fourth = SnapshotSequence/next[third]|
let fifth = SnapshotSequence/next[fourth]|
some rcmclient: User|
some ramserver: Module| some attacker: Attacker|
some disj idu, ids, idrs: ID| some r: Role|
some stzone: STZone| some disj pubkeyrcm, pubkeyram,
pubkeyattacker: PubKey| RCM2Attacker[first, second,
rcmclient, rcmclient, attacker, attacker, stzone, stzone, r, r,
idu, idu, ids, ids, pubkeyattacker, pubkeyattacker] and
Attacker2RAM[second, third, attacker, attacker,
ramserver, ramserver, stzone, stzone,
r, r, idu, idu, ids, ids, pubkeyram, pubkeyram] and
RAM2Attacker[third, fourth, ramserver, ramserver,
attacker, attacker, idrs, idrs, idu, idu, ids, ids,
pubkeyattacker, pubkeyattacker] and Attacker2RCM[fourth, fifth,
attacker, attacker, rcmclient, rcmclient, idrs, idrs,
idu, idu, ids, ids, pubkeyrcm, pubkeyrcm]
}

run Scenario2 for 5
run Attacker2RCM
```

Appendix D

Alloy Specification of Successful MITM Attack

```
module SecurityProtocol

open util/ordering[Snapshot] as SnapshotSequence

abstract sig Key{}
sig PubKey, PriKey extends Key{}

sig ID{}
sig STZone{}
sig Role{}
sig Password{}
sig OnePassword{}
sig Timestamp{}

sig Module{
id : one ID,
pubkey: one PubKey,
prikey: one PriKey,
}

sig Server extends Module{}

sig Client extends Module{
r: one Role,
pwd: one Password,
onepwd: one OnePassword,
ids: one ID,
stzone: one STZone
}

sig Attacker extends Client{}

sig Snapshot {
rcmclient: one Client,
ramserver: one Server,
acmserver: one Server,
attacker: one Attacker,
}

pred Client2Attacker[disj before, after: Snapshot,
senderPre, senderPost: Client, receiverPre,
receiverPost: Attacker, stzonePre, stzonePost: STZone, rPre,
rPost: Role, iduPre, iduPost: ID, idsPre, idsPost: ID,
pubkeyPre, pubkeyPost: PubKey]{

senderPre = before.rcmclient
receiverPre = before.attacker

senderPre.stzone = stzonePre
senderPre.r = rPre
senderPre.id = iduPre
senderPre.ids = idsPre

receiverPre.pubkey = pubkeyPre

senderPost = after.rcmclient
receiverPost = after.attacker

before.rcmclient = after.rcmclient
before.ramserver = after.ramserver
before.acmserver = after.acmserver
before.attacker = after.attacker
}

pred Attacker2RAM[disj before, after: Snapshot,
senderPre, senderPost: Attacker, receiverPre,
receiverPost: Module, stzonePre, stzonePost: STZone, rPre,
rPost: Role, iduPre, iduPost: ID, idsPre, idsPost: ID,
pubkeyPre, pubkeyPost: PubKey]{

senderPre = before.attacker
receiverPre = before.ramserver

senderPre.stzone = stzonePre
senderPre.r = rPre
senderPre.id = iduPre
senderPre.ids = idsPre

receiverPre.pubkey = pubkeyPre

senderPost = after.attacker
receiverPost = after.ramserver

before.rcmclient = after.rcmclient
before.ramserver = after.ramserver
before.acmserver = after.acmserver
before.attacker = after.attacker
}

pred RAM2Attacker[disj before, after: Snapshot,
senderPre, senderPost: Module, receiverPre,
receiverPost: Attacker, idrsPre, idrsPost: ID, iduPre,
iduPost: ID, idsPre, idsPost: ID, pubkeyPre, pubkeyPost: PubKey]{

senderPre = before.ramserver
receiverPre = before.attacker

senderPre.id = idrsPre
```

```
receiverPre.id = iduPre
receiverPre.ids = idsPre

receiverPre.pubkey = pubkeyPre

senderPost = after.ramserver
receiverPost = after.attacker

before.rcmclient = after.rcmclient
before.ramserver = after.ramserver
before.acmserver = after.acmserver
before.attacker = after.attacker
}

pred Attacker2Client[disj before, after: Snapshot,
senderPre, senderPost: Client, receiverPre, receiverPost:
Attacker,
idrsPre, idrsPost: ID, iduPre, iduPost: ID,
idsPre, idsPost: ID, pubkeyPre, pubkeyPost: PubKey]{

senderPre = before.attacker
receiverPre = before.rcmclient

// This is a key to fix the MITM attack because the
// client expects a message containing
//the server’s identity, while
// senderPre.id = idrsPre in the predicate ensures that
// the client will receive an identity from the sender, which is
// the attacker in the MITM attack scenario.
// senderPre.id = idrsPre

receiverPre.id = iduPre
receiverPre.ids = idsPre

receiverPre.pubkey = pubkeyPre

senderPost = after.attacker
receiverPost = after.rcmclient

before.rcmclient = after.rcmclient
before.ramserver = after.ramserver
before.acmserver = after.acmserver
before.attacker = after.attacker
}


pred ScenarioWithAttack{
let first = SnapshotSequence/first|let second =
SnapshotSequence/next[first]| let third = SnapshotSequence/next[second]|
let fourth = SnapshotSequence/next[third]|
let fifth = SnapshotSequence/next[fourth]|
some rcmclient: Client| some ramserver: Module| some attacker: Attacker|
some disj idu, ids, idrs: ID| some r: Role| some stzone: STZone|
some disj pubkeyrcm, pubkeyram, pubkeyattacker: PubKey|
Client2Attacker[first, second, rcmclient, rcmclient,
attacker, attacker, stzone, stzone, r, r, idu, idu, ids,
ids, pubkeyattacker, pubkeyattacker] and Attacker2RAM[second,
third, attacker, attacker, ramserver, ramserver, stzone,
stzone, r, r, idu, idu, ids, ids,  pubkeyram, pubkeyram] and
RAM2Attacker[third, fourth, ramserver, ramserver,
attacker, attacker, idrs, idrs, idu, idu, ids, ids,
pubkeyattacker, pubkeyattacker] and Attacker2Client[fourth, fifth,
attacker, attacker, rcmclient, rcmclient, idrs, idrs, idu, idu, ids, ids,
pubkeyrcm, pubkeyrcm]
}

run ScenarioWithAttack for 5
```