An Automated Approach to Detect Inconsistency and Semi-consistency Spatio-Temporal Role Based Access Control Specification

Emsaieb Geepalla, Behzad Bordbar
University of Birmingham
UK
{E.M.E.Geepalla, B.Bordbar}@cs.bham.ac.uk

ABSTRACT: *In the last decade, researchers have developed a number of RBAC extensions such as Spatio-Temporal Role Based Access Control (STRBAC) that takes into consideration the location of users and the time of access. These models that incorporate time and location information into the basic model RBAC are mainly proposed to support many mobile applications and wireless networks. However by doing so, it further increases the complexity of an already complex Access Control model. As a result, this increases the possibility of having contradictory statements in the Access Control specification. Such statements are commonly known as inconsistencies. In this study, we provide a formal definition of inconsistency in Spatio-Temporal Role Based Access Control (STRBAC) and then define several examples of inconsistencies in STRBAC specification. To achieve this, we shall first present formal algebraic notations of STRBAC model. In addition, the paper introduces the concept of "semi-consistency" in STRBAC and presents several scenarios that are involving semi-consistencies. A semi-consistency is a special case where the inconsistency can be avoided if the assignment of user to role is controlled. Finally, the paper presents an eclipse application called AC2Alloy that transforms STRBAC specification into Alloy. The produced Alloy models can be analysed using Alloy Analyser in order to detect inconsistencies and semi-consistencies in the STRBAC specification. With the help of an example, we show how AC2Alloy converts the STRBAC model to the Alloy model and verifies the resulting model using the Alloy analyser to identify inconsistencies and semi-consistencies.*

Keywords: Spatial Temporal Access Control, Alloy, Automated Analysis

Received: 17 June 2012, Revised 31 July 2012, Accepted 5 August 2012

1. Introduction

The recent advances in mobile computing, wireless networks and other technologies involved in remote accessing of resources has prompted an urgent need for the creation of Access Control systems which takes into consideration the location of the user and the time of access. Such information is essential for controlling various spatio-temporal sensitive applications, which rely on the Access Control mechanism, in organizations. For example, the access to some resources at an organisation could be permissible only at a specific time and a specific location. In order to be adaptable to the requirements of such applications and technologies with both spatial constraint and temporal constraint, several Access Control models have been proposed [3, 7, 9, 10, 13, 20, 21, 22]. The STRBAC model is one of the Access Control models that has been presented to cater for the need of context information [10]. The STRBAC model incorporates various rules, such as, Role Hierarchy, Separation of Duties constraints, and Cardinality constraints.

It is possible that STRBAC rules conflict with each other, in particular when various rules are combined to form a new specification. A typical situation is when User Role Assignment and Role Hierarchy cause a violation of Separation of Duty constraints. For example, an organisation may require that the same user should not be assigned to the two conflicted roles Teller and Loan Officer, at the same time and same location, whereas it requires that a user u be the Branch Manager and the role Branch Manager is a senior role to the Teller and Loan Officer at any time and any location. Such situations are often referred to as inconsistencies in Access Control specification. This example is inconsistent because the user u can assign to the two conflicted roles; Teller and Loan Officer, at the same time and same location because he/she is assigned to the role Branch Manager which is a senior role to the two conflicted roles Teller and Loan Officer.

Sometimes the STRBAC specification is consistent, however a minor change to the specification by choosing an unsuitable allocation of the users to roles makes the specification inconsistent. We shall refer to these as "*semi-consistency*" in the specification. To the best of our knowledge we are the first to introduce the concept of "*semi-consistency*" in STRBAC specification. If we can identify such scenarios, then we can adopt pre-emptive action by putting in constraints such as Java assertion to avoid such changes to the specification. These scenarios could pose dangerous security issues that could even cause the downfall of the organization [2]. It is therefore essential to perform an analysis of STRBAC models to identify inconsistencies and semi-consistencies in the specification.

This paper gives a Relational Semantics of the STRBAC model to provide a formal definition of inconsistency in STRBAC model. Using the Relational Semantics we shall present several scenarios which we have come across during our studies that may cause inconsistencies in the STRBAC specification. In addition, the paper introduces the concept of semi-consistency and presents several scenarios that may cause semi-consistencies in the STRBAC specification. The paper also proposes an automated method that allows users to create STRBAC models and transforms them into the required Alloy code automatic, thus allowing for powerful analysis to identify inconsistencies and semi-consistencies in STRBAC specification to take place. The suggested method has been implemented as an Eclipse Plug-in called AC2Alloy [15].

The rest of the paper is organised as follows. Section II briefly presents related works that have motivated this research. In section III we introduce our relational semantics of the STRBAC model. Section IV provides definitions of inconsistency and semi-consistency and presents several scenarios that may cause them. In section V we introduce an example, which will be used to describe our approach. Section VI presents our method to detect inconsistencies and semi-consistencies in STRBAC. Section VII provides evaluation of our approach. The paper ends with a conclusion.

2. Related Work

There is a plethora of works using automated techniques for analysis of Access Control policies such as [1, 3, 4]. Some of the approaches do not particularly address STRBAC, like RBAC [11]. Huang et al. [1] propose an algorithm to detect inconsistencies in the RBAC policy, which is based on the Tarjan's SCC algorithm [7]. The authors provide formal definitions for the inconsistencies in the RBAC model. However this work has not considered the environment information such as time and location that is needed in many applications today. Additionally, some of the scenarios that they defined may not cause inconsistencies, unless a new user to role assignment is added to the specification. We think it is important to distinguish between scenarios which have inconsistent specification and cases of "*potential*" inconsistencies.

Researchers have also used Alloy [5, 16, 17, 18] for analysis of Access Control specification. Alloy is a language used for modelling and specification of object-oriented systems. Alloy is supported by a tool called The Alloy Analyser. The analyser is an automated constraint solver that transforms the Alloy code into Boolean expressions, providing the analysis by its embedded SAT solvers.

Jackson et al. [8] demonstrate how to use Alloy to verify the internal consistency of RBAC-96 Schema. This work has demonstrated that using Alloy has sufficient expressive power to prescribe implementation independent specification of Access Control systems. However this work has not considered the environment information such as time and location that is needed in many applications today. Additionally, the transformation between RBAC and Alloy was carried out manually. For a small system the creation of model transformation between RBAC model and Alloy could be mange manually, however due to the complexity and size of modern systems an automated transformation are required.

Another effort to verify the consistency of Access Control specification using Alloy was proposed by Samuel et al. [3]. Their

work illustrates how GST-RBAC can be analysed using Alloy. In their work the transformation between GST-RBAC and Alloy was carried out manually. Although the transformation presented seems to be correct, in general manual transformation is tedious and might pose errors, especially when the Access Control specification is very large. This paper presents an automated transformation which eliminates the bugs which may be caused by a manually creation of Alloy.

The analysis of STRBAC models with Alloy has been extensively studied by Indrakshi et al. [4]. Indrakshi et al. present formalization for STRBAC model and they used Alloy for checking Access Control models [4]. In this paper we further the research of [4] by sharing our experience of dealing with inconsistencies identified by automated checkers such as Alloy as well as introducing the concept of semi-consistency. Identifying semi-consistencies helps the system designer to avoid changes that might cause inconsistencies. Moreover, this paper proposes an automated method that converts STRBAC specification into Alloy model, and then makes use of Alloy analyser to detect inconsistencies and semi-consistencies.

3. Formal Algebraic Notations of STRBAC

This section introduces formal algebraic notations of the Spatio-Temporal Role Based Access control (STRBAC) to specify the components of the STRBAC model. Our metaphor of Spatial Temporal Access Control is based upon recent work of Inderakshi et al [10]. In this paper, we restrict ourselves to the STRBAC model without some its features such sessions and delegation. These features can be modelled in a similar way and they can be added to our approach. If there is no chance of confusion, we sometimes use the phrase "*Access Control* " instead of STRBAC in the rest of the paper.

Definition 3.1. Suppose that *U, R, P, T, L* are finite and non-empty sets of Users, Roles, Permissions, Times and Locations, respectively. Then an Access Control specification A can be seen as a subset of the Cartesian product $U \times R \times P \times T \times L$. We write $(u, r, p, t, l) \in A$ meaning that the user u is assigned to the role r and has the permission p at the location l and the time t. The five tuples (u, r, p, t, l) will be generated from the interaction between the Access Control rules which are defined next. These rules specify permissible coordinates for the five tuples.

3.1 User Role Assignment (URA)

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

3.2 Permission Role Assignment (PRA)

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

3.3 Role Hierarchy (RH)

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

• Unrestricted Role Hierarchy *URH* Let r_i and r_j be roles such that $r_i \succeq r_j$, that is, the role r_i is a senior role to the role r_j at any time and any location.

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

$\forall u \in U, \forall t \in T, \forall l \in L\ URA(u, r_i, t, l) \Rightarrow URA(u, r_j, t, l) \wedge$

$\forall p \in P, \forall t \in T, \forall l \in L\ PRA(r_j, p, t, l) \Rightarrow PRA(r_i, p, t, l)$

• Time Dependent Role Hierarchy *TRH* Let ri and r_j be roles such that $r_i \succeq_{t'} r_j$, that is, the role r_i is a senior role to the role r_j only at the time t' and at any location.

$\forall r_i, r_j \in R, \forall t' \in l\ (r_i \succeq_t r_j)$, then

$\forall u \in U, \forall l \in L\ URA(u, r_i, t', l) \Rightarrow URA(u, r_j, t', l) \wedge$

$\forall p \in P, \forall l \in L, \forall l \in L\ PRA(r_j, p, t', l) \Rightarrow PRA(r_i, p, t', l)$

• Location Dependent Role Hierarchy *LRH* Let ri and rj be roles such that $r_i \succeq_{l'} r_j$, that is, the role r_i is a senior role to the role r_j only at the location l' and at any time.
$\forall r_i, r_j \in R, \forall l' \in L\ (r_i \succeq_{l'} r_j)$, then
$\forall u \in U, \forall t \in T\ URA\ (u, r_i, t, l') \Rightarrow URA\ (u, r_j, t, l') \wedge$
$\forall p \in P, \forall t \in T\ PRA\ (r_j, p, t, l') \Rightarrow PRA\ (r_i, p, t, l')$

• Time and Location Dependent Role Hierarchy *TLRH* Let r_i and r_j be roles such that $r_i \succeq_{t', l'} r_j$, that is, the role r_i is a senior role to the role r_j only at the time t' and at location l'.
$\forall r_i, r_j \in R, \forall t' \in T, \forall l' \in L\ (r_i \succeq_{t', l'} r_j)$, then
$\forall u \in U\ URA\ (u, r_i, t', l') \Rightarrow URA\ (u, r_j, t', l') \wedge$
$\forall p \in P\ PRA\ (r_j, p, t', l') \Rightarrow PRA\ (r_i, p, t', l')$

3.4 Location Hierarchy (LH)

It is a partial order on the set locations that specifies which location contains another location, $LH \subseteq L \times L$. We write $l_i \succeq l_j$ meaning that the location l_i is outer-location to the inner-location l_j. This means that if there is a user who is assigned to the role r at the outer-location l_i, then he/she can also assign the role r at the inner-location l_j and if the role r the has a permission p at the outer-location l_i, then the role r should also has the same permission p at the inner-location l_j.
$\forall l_i, l_j \in L\ (l_i \succeq l_j)$, then
$\forall u \in U, \forall r \in R, \forall t \in T\ URA\ (u, r, t, l_i) \Rightarrow URA\ (u, r, t, l_j) \wedge$
$\forall p \in P, \forall r \in R, \forall t \in T\ PRA\ (r, p, t, l_i) \Rightarrow PRA\ (r, p, t, l_j)$

3.5 Separation of Duty between Role (SoDR)

It is a constraint that is used to specify that two mutually exclusive roles cannot be simultaneously assigned to the same user at the same time and the same location. We write $sodr\ (r_i, r_j, t', l')$ meaning that the two exclusive roles r_i and r_j should not be assigned by the same user at time t' and location l'. SoDR can be unrestricted $sodr\ (r_i, r_j)$, time dependent $sodr\ (r_i, r_j, t')$, location dependent $sodr\ (r_i, r_j, l')$, or time and location dependent $sodr\ (r_i, r_j, t', l')$.

• Unrestricted Separation of Duty over Roles SoDR Let r_i and rj be exclusive roles such that $sodr\ (r_i, r_j)$, that is, the role r_i and the role rj should not be assigned by the same user at any location and at any time.
$\forall r_i, r_j \in R\ sodr\ (r_i, r_j)$, then
$\forall t \in T, \forall l \in L, \neg \exists u \in U\ URA\ (u, r_i, t, l) \wedge URA\ (u, r_j, t, l)$

• Time Dependent Separation of Duty over Roles (TSoDR) Let r_i and r_j be exclusive roles such that $sodr\ (r_i, r_j, t')$, that is, the role r_i and the role r_j should not be assigned by the same user at the time t' and at any location.
$\forall r_i, r_j \in R, \forall t \in T\ sodr\ (r_i, r_j, t')$, then
$\forall l \in L, \neg \exists u \in U\ URA\ (u, r_i, t', l) \wedge URA\ (u, r_j, t', l)$

• Location Dependent Separation of Duty over Roles (LSoDR) Let ri and r_j be exclusive roles such that $sodr\ (r_i, r_j, l')$, that is, the role r_i and the role r_j should not be assigned by the same user at the location l' and at any time.
$\forall r_i, r_j \in R, \forall l' \in T\ sodr\ (r_i, r_j, l')$, then
$\forall t \in T, \neg \exists u \in U\ URA\ (u, r_i, t, l') \wedge URA\ (u, r_j, t, l')$

• Time and Location Dependent Separation of Duty over Roles (TLSoDR)
Let r_i and r_j be exclusive roles such that $sodr\ (r_i, r_j, t', l')$, that is, the role r_i and the role r_j should not be assigned to the same user at the time t' and the location l'.
$\forall r_i, r_j \in R, \forall t' \in T, \forall l \in L\ sodr\ (r_i, r_j, t', l')$, then
$\neg \exists u \in U\ URA\ (u, r_i, t', l') \wedge URA\ (u, r_j, t', l')$

3.6 Separation of Duty between Permissions (SoDP)

It is a constraint that is used to specify that two mutually exclusive permissions cannot be simultaneously assigned to the same role at the same time and the same location. We write *sodo* (p_i, p_j, t', l') meaning that the two exclusive permissions p_i and p_j should not be assigned by the same role at time t0 and location l0. SoDP can be unrestricted *sodo* (p_i, p_j), time dependent *sodo* (p_i, p_j, t'), location dependent *sodo* (p_i, p_j, l'), or time and location dependent *sodo* (p_i, p_j, t', l').

• Unrestricted Separation of Duty between Permissions (*USoDP*)

Let p_i and p_j be exclusive permissions such that *sodo* (p_i, p_j), that is, the permission p_i and the permission p_j should not be assigned by the same role at any location and at any time.

$\forall p_i, p_j \in P$ *sodr* (p_i, p_j), then

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

• Time Dependent Separation of Duty between Permissions (*TSoDP*)

Let p_i and p_j be exclusive permissions such that *sodr* (p_i, p_j, t'), that is, the permission p_i and the permission p_j should not be assigned by the same role at the time t' and at any location.

$\forall p_i, p_j \in P, \forall t' \in T$ *sodr* (p_i, p_j, t'), then

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

• Location Dependent Separation of Duty between Permissions (*LSoDP*)

Let p_i and p_j be exclusive permissions such that *sodr* (p_i, p_j, l'), that is, the permission p_i and the permission p_j should not be assigned by the same role at the location l' and at any time.

$\forall p_i, p_j \in P, \forall l' \in L$ *sodr* (p_i, p_j, l'), then

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

• Time and Location Dependent Separation of Duty between Permissions (*STSoDP*)

Let p_i and p_j be exclusive permissions such that *sodr* (p_i, p_j, t', l'), that is, the permission p_i and the permission p_j should not be assigned to the same role at the location l' and at the time t'.

$\forall p_i, p_j \in P, \forall t' \in T, \forall l' \in L$, *sodr* (p_i, p_j, t', l'), then

$\neg \exists r \in R\, PRA(r, p_i, t', l') \wedge PRA(r, P_j, t', l')$

3.7 Cardinality Constraints over Roles (CCR)

It is a constraint that restrict the number of users that can have a role at the same time and the same location. We write *ccr* (r_i, t', l', n), meaning that the role r_i has restriction, so that it should not be assigned to more than n users at time t' and location l'. *CCR* could be Unrestricted *ccr* (r_i, n), Time dependent *ccr* (r_i, t', n), Location dependent *ccr* (r_i, l', n), or Time and Location dependent *ccr* (r_i, t', l', n).

• Unrestricted Cardinality Constraints over Roles (*UCCR*)

Let the role r_i be a restricted role, such that *ccr* (r_i, n), that is the role r_i should not be assigned by more than n users at any time and any location. This should satisfy the following constraint:

$\forall r_i \in R\,(r_i, n)$, then

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

• Time Dependent Cardinality Constraints over Roles (*TCCR*)

Let the role r_i be a restricted role, such that *ccr* (r_i, t', n), that is the role r_i should not be assigned by more than n users at time t' and at any location. This should satisfy the following constraint:

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

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

• Location Dependent Cardinality Constraints over Roles (*LCCR*)

Let the role r_i be a restricted role, such that $ccr(r_i, l', n)$, that is the role r_i should not be assigned by more than n users at location l' and at any time. This should satisfy the following constraint:
$\forall r_i \in R, \forall l' \in L\ (r_i, l', n)$, then
$\forall u \in U, \forall t \in T \# (URA(u, r_i, t, l') \leq n$

• Time and Location Dependent Cardinality Constraints over Roles (TLCCR)
Let the role r_i be a restricted role, such that $ccr(r_i, t', l', n)$, that is the role ri should not be assigned to more than n users at time t' and location l'.
This should satisfy the following constraint.
$\forall r_i \in R, \forall t' \in L, \forall l' \in L\ (r_i, t', l', n)$, then
$\forall u \in U \# (URA(u, r_i, t', l') \leq n$

3.8 Cardinality Constraints over Permissions (CCP)
It is a constraint that restrict the number of roles that can have a permission at the same time and the same location. We write $ccp(p_i, t', l', m)$, meaning that the permission pi has restriction, so that it should not be assigned by more than m roles at time t' and location l'. CCP could be Unrestricted $ccp(p_i, m)$, Time dependent $ccp(p_i, t', m)$, Location dependent $ccp(p_i, l', m)$, or Time and Location dependent $ccp(p_i, t', l', m)$.

• Unrestricted Cardinality Constraints over Permissions (*UCCP*)
Let the permission p_i be a restricted permission, such that $ccp(p_i, m)$, that is the permission p_i should not be assigned by more than m roles at any time and any location. This should satisfy the following constraint:
$\forall p_i \in P\ ccp(p_i, m)$, then
$\forall r \in R, \forall t \in T, \forall l \in L \# (PRA(r, p_i, t, l) \leq m$

• Time Dependent Cardinality Constraints over Permissions (*TCCP*)
Let the permission p_i be a restricted permission, such that $ccp(p_i, t', m)$, that is the permission p_i should not be assigned by more than m roles at time t' and any location. This should satisfy the following constraint:
$\forall p_i \in P, \forall t' \in T\ ccp(p_i, t', m)$, then
$\forall r \in R, \forall l \in L \# (PRA(r, p_i, t', l) \leq m$

• Location Dependent Cardinality Constraints over Permissions (*LCCP*)
Let the permission p_i be a restricted permission, such that $ccp(p_i, l', m)$, that is the permission p_i should not be assigned by more than m roles at location l' and any time. This should satisfy the following constraint:
$\forall p_i \in P, \forall l' \in L\ ccp(p_i, l', m)$, then
$\forall r \in R, \forall t \in T \# (PRA(r, p_i, t, l') \leq m$

• Time and Location Dependent Cardinality Constraints over Permissions (*STCCP*)
Let the permission p_i be a restricted permission, such that $ccp(p_i, t', l', m)$, that is the permission p_i should not be assigned by more than m roles at time t' and location l'. This should satisfy the following constraint:
$\forall p_i \in P, \forall t' \in T, \forall l' \in L\ ccp(p_i, t', l', m)$, then
$\forall r \in R \# (PRA(p_i, t', l', m) \leq m$

4. Inconsistency and Semi-consistency in STRBAC Specification

In this section we provide a formal definition of inconsistency in the STRBAC specification and then we present several examples of inconsistencies in the STRBAC specification with the help of the formal algebraic notations presented in the previous section. In addition to that, this section introduces a new concept called semi-consistency in the STRBAC specification. Semi-consistency is a special case in which the inconsistency can be avoided if the assignment of user to role is controlled. Finally, this section presents different scenarios that may cause semi-consistencies in the STRBAC specification.

4.1 Inconsistency in STRBAC Specification

In the STRBAC model, having various rules makes the design of Access Control system simpler by breaking a complex specification to various rules such as User Role Assignment and Separation of Duty between Roles are considered separately. However having multiple rules may result in inconsistency when various rules interact with each other. If this is the case, then there will be no a subset A of the Cartesian product $U \times R \times P \times T \times L$ that satisfies all the STRBAC rules. As a result, an inconsistent specification means that it is not possible to find a five tuples (u, r, p, t, l), so that the coordinates satisfies all rules (User Role Assignment, Permission Role Assignment, Role Hierarchy, Location Hierarchy, Separation of Duty constraints and Cardinality constraints).

Definition 4.1. Suppose that *URA, PRA, RH, LH, SoDR, SoDP, CCR* and *CCP* is a set of Access Control rules. An Access Control specification is termed inconsistent if there is no non-empty set $A \subseteq U \times R \times P \times T \times L$ such that A satisfies all Access Control rules.

In [1] Chao Huang et al. present examples of inconsistencies that might occur during the interaction between different RBAC rules. Following their examples we are also listing here some examples of the inconsistencies in STRBAC specification that we have come across during our studies.

• Inconsistency between User Role Assignment (*URA*) and Separation of Duty over Roles (*SoDR*). For example,
$\exists u \in U, \exists t \in T, \exists l \in L, \exists\, t \in T, \exists\, r_i, r_j \in R\, Sodr\,(r_i, r_j) \wedge URA\,(u, r_i, t, l) \wedge URA\,(u, r_j, t, l)$.
Then there is inconsistency caused by the interaction between the *URA* and the *SoDR*, because the user u access to the two conflicted roles r_i and r_j which is not permissible according to *SoDR*.

• Inconsistency between User Role Assignment (*URA*) and Cardinality Constraint over Roles (*CCR*). For example,
$\exists u \in U, \exists t \in T, \exists l \in L, \exists t \in T, \exists r_i \in R\, ccr\,(r_i, n) \wedge \# \mid URA\,(u, r_i, t, l) \mid > n.$
Then there is inconsistency in the specification caused by the interaction between the *URA* and the *CCR*, because the number of User Role Assignment for the restricted role r_i is larger than *n*, which violates the cardinality constraint.

• Inconsistency between User Role Assignment (*URA*) and Role Hierarchy (*RH*). For example,
$\exists\, r_i, r_j, r_k \in R, \exists u \in U, \exists t \in T, \exists l \in L\,(r_i \succeq r_j) \wedge (r_j \succeq r_k) \wedge (r_k \succeq r_i) \wedge (URA\,(u, r_i, t, l) \vee URA\,(u, r_j, t, l) \vee URA\,(u, r_k, t, l)$.
Then there is inconsistency in the specification caused by the interaction between *URA* and *RH*, because the cycle of *RH* allows the user, who is assigned to the junior role to get, the permissions of the senior roles.

• Inconsistency between User Role Assignment (*URA*), Permission Role Acquire (*PRA*) and Separation of Duty between Permissions (*SoDP*). For example,
$\exists u \in U, \exists t \in T, \exists l \in L, \exists t \in T, \exists\, p_i, p_j \in P\, Soda\,(p_i, p_j) \wedge URA\,(u, r, t, l) \wedge PRA\,(r, p_i, t, l) \wedge PRA\,(r, p_j, t, l)$.
Then there is inconsistency caused by the interaction between the *URA*, *PRA* and the *SoDP*, because *URA* allows a user *u* to access the role *r* which is assigned to the two conflicted permissions p_i and p_j, which is not permissible according to (*SoDP*).

• Inconsistency between User Role Assignment (*URA*), Role Hierarchy (*RH*) and Separation of Duty between Roles (*SoDR*). For example, $\exists u \in U, \exists t \in T, \exists l \in L, \exists\, r_i, r_j \in R\,(r_i \succeq r_j)\, Soda\,(r_i, r_j) \wedge URA\,(u, r_i, t, l)$.
Then there is inconsistency between *URA*, *RH* and *SoDR*, because the user *u* can access to the two conflicted roles r_i based on *URA* and r_j based on *URA* and *RH*, which is not permissible according to *SoDR*.

• Inconsistency between User Role Assignment (*URA*), Role Hierarchy (*RH*), Permission Role Acquire (*PRA*) and Separation of Duty between Permissions (*SoDP*). For example, $\exists u \in U, \exists\, r_i, r_j \in R, \exists t \in T, \exists l \in L, \exists t \in T, \exists\, p_i, p_j \in P, (r_i \succeq r_j) \wedge Soda\,(p_i, p_j) \wedge URA\,(u, r_i, t, l) \wedge PRA\,(r, p_i, t, l) \wedge PRA\,(r, p_j, t, l)$.
Then there is inconsistency between the *URA*, PRA, RH and the SoDP, because a user u can access the role r_i based on *URA*, and, based on *RH*, the same user *u* can access the role r_j, which is assigned to the two conflicted permissions p_i and p_j, which is not permissible according to *SoDP*.

The above list is not exhaustive as we have only considered the unrestricted *RH, SoDR, SoDP, CCR* and *CCP* to define the

above scenarios. There could be more complicated scenarios composed from the above scenarios when the time dependent, location dependent and time and location dependent constraints are considered.

4.2 Semi-consistency in STRBAC Specification

In this section we introduce the term semi-consistency in the STRBAC specification. Semi-consistency is a weak form of consistency that the system is not inconsistent but it can be potentially inconsistent with a minor change to the specification. Consider an Access Control specification that contains the following statements: a Separation of Duty between Permissions *SoDP* statement claims that the two permissions p_2 and p_2 should not be assigned by the same role at time *t* and location *l*, while a Permission Role Assignment *PRA* statement claims that a role *r* can have the two permissions p_2 and p_2 at any time and any location. Although the above two statements are contradictory, there will be no inconsistency in the specification if there is no user assigned to the role *r* at the time *t* and the location l. A minor change for example assigning a user to the role *r* will result in inconsistency. We refer to such scenarios as semi-consistency.

Definition 4.2. We call an Access Control specification *S* semi-consistent if: 1) It is consistent (see Definition 2 in section IV). 2) There are a finite number of users $u_1, u_2, ..., u_n \in U$ and roles $r_1, r_2, ..., r_n \in R$, so that if a new specification S' consisting of *S* in addition to $ura(u_1, r_1, t, l)$, $ura(u_1, r_1, t, l)$, ..., $ura(u_n, r_n, t, l)$, so that S0 is inconsistent. In case of semi-consistency we may try to either modify the specification to remove that semi-consistency or we may add logical constraint (Java assertions) to avoid inappropriate User Role Assignment. In effect, the idea is to identify "*bad* " User Role Assignment which may cause inconsistencies, and prevent them. Next we present some examples involving semi-consistencies.

• Semi-consistency because of the Interaction between Permission Role Acquire (*PRA*) and Separation of Duty between Permissions (*SoDP*). For example, $\exists r \in R, \exists t \in T, \exists l \in L, \exists p_i, p_j \in P \wedge PRA(r, p_i, t, l) \wedge PRA(r, p_j, t, l) \wedge Soda(p_i, p_j)$.
The above specification is semi-consistent because a minor change for example assign a user to the role r, at the time t and the location l allows that user to have the two conflicted permissions p_1 and p_j, which are assigned to the role *r* at time *t* and location *l*. Such changes should be prevented.

• Semi-consistency because of Role Hierarchy (*RH*). For example,
$\exists r_i, r_j, r_k \in R, \exists u \in U, \exists t \in T, \exists l \in L (r_i \succeq r_j) \wedge (r_j \succeq r_k) \wedge (r_k \succeq r_i)$.
The above specification semi-consistent because a minor change for example assign a user to one of the roles r_i, r_j or r_k, at time *t* and location *l* will result in an inconsistency. Such changes should be prevented.

• Semi-consistency because of the interaction between User Role Assignment (*URA*) and Cardinality Constraints over Roles (*CCR*). For example, $\exists u_1 \in U, \exists r \in R, \exists t \in T, \exists l \in L\ ura(u_1, r, t, l) \wedge ccr(r, t, l, 2)$.
The above specification is semi-consistent because a minor change for example include $ura(u_2, r, t, l)$ and $ura(u_3, r, t, l)$ will result in inconsistency. Such changes should be prevented.

5. Running Example

This section introduce the outline of a running example which will be used to demonstrate our approach. This example is based on a typical Banking system taken from [5] consisting of several security policies. The application is used by various bank officers to perform several transactions (i.e. transaction on customer loan accounts) at different locations and during specific periods of time.

5.1 Security Rules

5.1.1 User Role Assignment

Users are assigned to roles in the Banking system as illustrated in Table 1.

5.1.2 Permission Role Assignment

Roles are assigned to permissions in the Banking system as illustrated in Table 2.

5.1.3 Role Hierarchy

Some roles in the Banking system are related using unrestricted Role Hierarchy as illustrated in Figure 1:

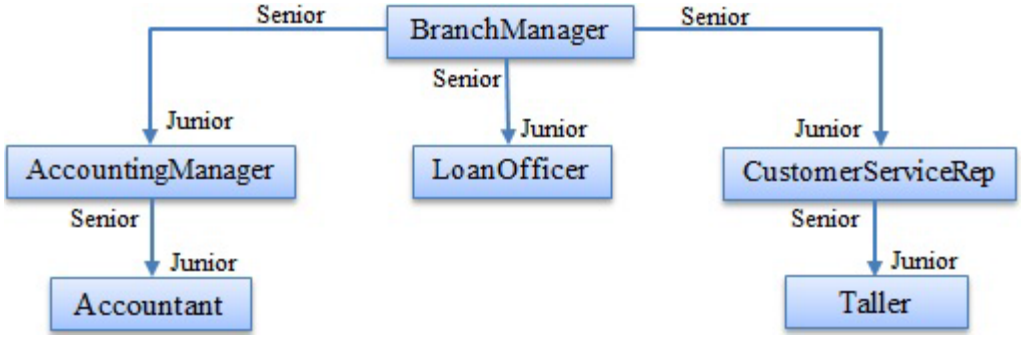

Figure 1. Role Hierarchy

Users	Roles	Times	Locations
Dave	Branch Manager	DayTime	office2
Mark	Accounting Manager	NightTime	office1
Sarah	Accountant	NightTime	office1

Table 1. User Role Assignment

Roles	Permissions	Times	Locations
Teller	POne: Input and Modify transactions against Customer Deposit Accounts	DayTime	office2
Loan Officer	PTwo: Create and Modify status of Loan Accounts	DayTime	office2
Accountant	PThree: Generate General Ledger Reports	DayTime/ NightTime	office1
Accounting Manager	PFour: Modify Ledger Posting Rules	DayTime	office1

Table 2. Permission Role Assignment

5.1.4 Separation of Duty between Roles
The roles Loan Officer and Accounting Manager cannot be assigned to the same user at any time and any location.

5.1.5 Separation of Duty between Permissions
The permissions PThree and PFour should not be assigned by the same role during the DayTime and at office1.

5.1.6 Cardinality Constraints over Roles
The maximum number of users that can be assigned to Accountant role during the NightTime and at office1 is One.

6. Inconsistencies and Semi-consistencies detection method

To identify inconsistencies and semi-consistencies we shall make use of SAT-solvers. To do so, we have developed an Eclipse Plug-in called AC2Alloy. Figure 2 shows a screen shot of the tool.

6.1 Description of AC2Alloy
AC2Alloy is a model transformation that makes use of Model Driven Architecture (MDA) [6] methodology to transform a STRBAC model to Alloy. Model transformation is used to convert STRBAC specification into Alloy automatically. To conduct the model transformation, a set of transformation rules have been defined. The rules map various elements of STRBAC metamodel into the elements of the Alloy metamodel. The rules are executed via SiTra [12]. The transformation rules have been implemented

into an Eclipse Plug-in application called AC2Alloy. Figure 3 depicted the architecture of AC2Alloy. It shows that if a user provides STRBAC model as an input to AC2Alloy, an XML representation for that specification will be generated automatically at the GUI level, and then an Alloy model will be created. To analyse the STRBAC specification, the tool will automatically generate several Alloy checks, so that the user can execute them using Alloy Analyser.

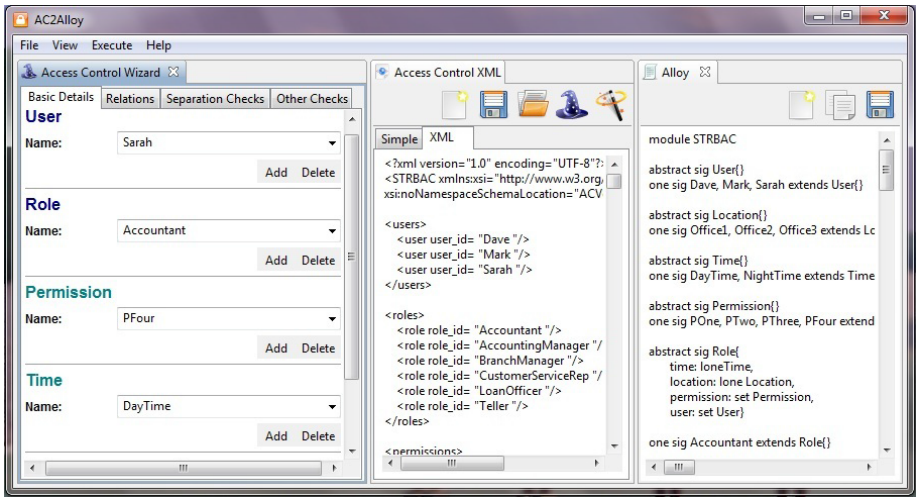

Figure 2. Screen Shot of AC2Alloy

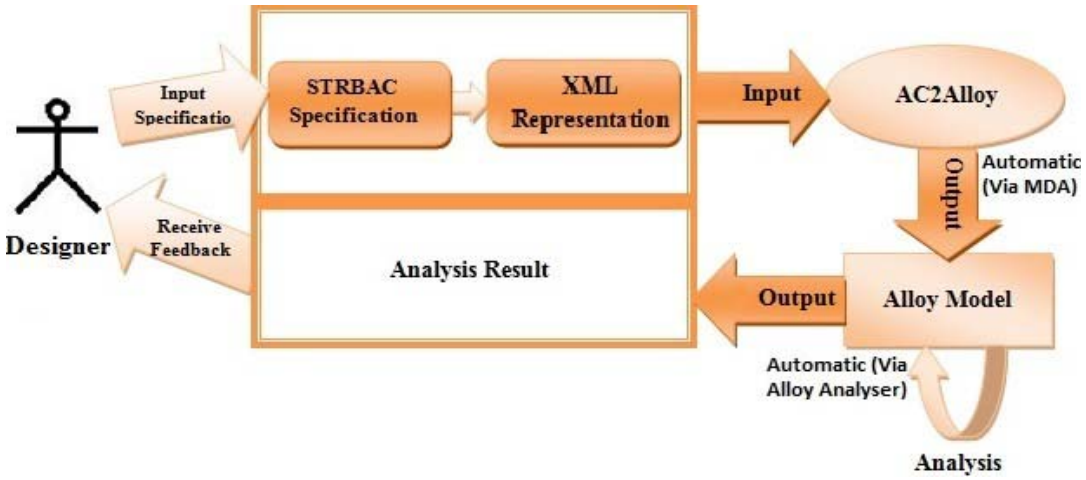

Figure 3. AC2Alloy Architecture

6.2 Transformation between STRBAC model and Alloy

AC2Alloy is used to generate an Alloy model from the STRBAC specification in the context of the Bank system. The basic components of STRBAC such as sets of Users, Permissions, Times and Locations will be transformed into *signatures* in Alloy as shown in Figure 4.

```
abstract sig User{}
one sig Dave, Mark, Sarah extends User{}
abstract sig Time{}
one sig DayTime, NightTime extends Time{}
abstract sig Location{}
one sig office1, office2 extends Location{}
abstract sig Permission{}
one sig POne, PTwo, PThree, PFour extends Permission{}
```

Figure 4. Transformation of Users, Times, Locations and Permissions

The set of Roles will be transformed into *signatures, facts* and *predicates*. The *facts* and *predicates* are used to represent the relationships between roles, users, permissions, times and locations. Such relationships are expressed by User Role Assignment (*URA*) and Permission Role Acquire (*PRA*). For example AC2Alloy will transform the role Accounting Manager and the user assignment of the role Accounting to the following Alloy code.

```
abstract sig Role {time: lone Time,
    location: lone Location,
    user: set User,
    permission: set Permission}
one sig AccountingManager extends Role {}
fact AccMan_fact{all self:AccountingManager | AccManCondition[self]}
pred AccManCondition[self:AccountingManager]{((self.permission=none)&&
    (self.location=office1)&&(self.time=NightTime)&&(self.user=Mark))}
```

Figure 5. Transformation of the role Accounting Manager and its User Role Assignment

The Permission Role Assignment injects the predicate within Alloy code for the Roles with new assignment information. For example the transformation of the Permission Role Assignment of the role Accounting will inject the *predicate AccManCondition* within the role Accounting Manager with new assignment information as illustrated in Figure 6.

```
pred AccManCondition[self:AccountingManager]{((self.permission=none)&&
    (self.location=office1)&&(self.time=NightTime)&&(self.user=Mark)) ||
    ((self.permission=PFour)&&(self.location=office1)&&(self.time=DayTime)
    &&(self.user=none))}
```

Figure 6. Example of the Transformation of Permissions Role Assignment

The Role Hierarchy also injects the *predicates* within the Alloy code for the Roles with new assignment information. For example the effect of the transformation of the hierarchy between the roles Accounting Manager and Accountant will be transformed into new assignment information which will be injected to the *predicates* within the Alloy code for the roles Accounting Manager and Accountant. For example the senior role Accounting Manager can inherit all the permissions assigned to the junior role Accounting, then the *predicate AccManCondition* within the role Accounting Manager will be injected with new assignment information as shown in Figure 7.

```
pred AccManCondition[self:AccountingManager]{((self.permission=PThree)&&
    (self.location=office1)&&(self.time=NightTime)&&(self.user=Mark)) ||
    ((self.permission=PThree+PFour)&&(self.location=office1)&&
    (self.time=DayTime)&&(self.user=none))}
```

Figure 7. Example of the transformation of Role Hierarchy

6.3 Model Analysis

Alloy checks must be formulated prior to analysis by Alloy Analyser. The generated Alloy model for the Bank system contains several *checks* to detect inconsistencies and semi-consistencies in the specification. Due to space consideration we only explain two of the Alloy *checks* that have been generated to detect semi-consistencies and inconsistencies.

6.3.1 Example of AC2Alloy Identifying Semi-consistency

The Separation of Duty constraints will be transformed to *predicates* and *checks*. For example the Separation of Duty between the two permissions PThree and PFour during the DayTime and at office1 will be transformed to the following Alloy code:

```
pred SODP[p1, p2:Permission, l:Location, t:Time]{all r1:Role |
     ((p1 in r1.permissions)&&(t in r1.time)&&(l in r1.location))=>
     ((p2 not in r1.permissions)&&(t in r1.time)&&(l in r1.location))}
SoDP1 :check {SODP[PThree, PFour, DayTime, office1]}
```

Figure 8. Example of the Transformation of Separation of Duty between Permissions

The execution of the Alloy shows that Alloy Analyser picked up a counterexample as depicted in Figure 9. This means the specification is semi-consistent, because that the two conflicted permissions PThree and PFour are assigned by the Branch Manager at the DayTime and Location office1 due to the Role Hierarchy, and the role Branch Manager is not assigned by any user during the DayTime and at office1. However a minor change to the Bank system by assigning a user to the Branch Manager during the time DayTime and at the Location office1 will result in inconsistency. Such changes should be prevented.

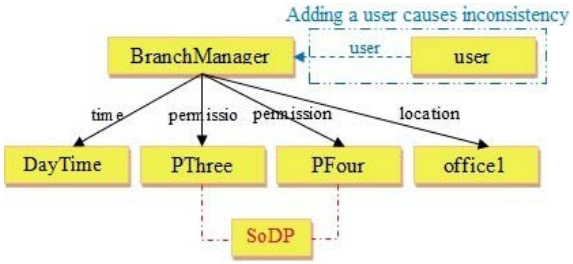

Figure 9. Counterexample for SoDP1 check: Semi-consistency Detection

6.3.2 Example of AC2Alloy Identifying Inconsistency

The Cardinality constraints will be transformed to checks. For example the Cardinality constraints over the role Accountant during the NightTime and at office1 will be transformed to *checks* in Alloy as shown in Figure 10.

```
CC1 :check {((office1 in Accountant.location)&&
(NightTime in Accountant.time)=>(#Accountant.user <2))}
```

Figure 10. Example of the Transformation of Cardinality Constraint over Roles

The execution of the Alloy shows that Alloy Analyser picked up a counterexample as depicted in Figure 11. This means the specification is inconsistent because there is more than one user (Mark and Sarah) are assigned to the role Accountant, which is not permissible according to the Cardinality constraint over the role Accountant. The role Accountant is assigned to the user Sarah because of the direct assignment using User Role assignment, while it is assigned to the user Mark, because Mark is assigned to the role Accounting Manager during the NightTime at the location office1 and the role Accounting Manager is a senior role to the role Accountant.

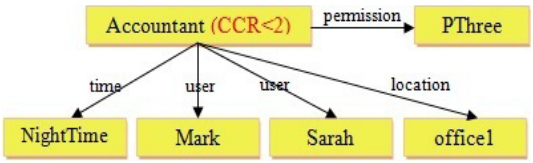

Figure 11. Counterexample for *CC*1 *check*: Inconsistency Detection

7. Performance

To evaluate the performance of AC2Alloy tool, a set of case studies including the case study discussed in section 4 were used. The results of these case studies have shown that AC2Alloy could successfully convert these case studies to Alloy in order to be analysed using Alloy Analyser. To evaluate the scalability of AC2Alloy we have started testing our approach using a medium scale access control system. This has shown us that the tool could transform a mediumscale access control very quick. Then we have increased the size of the system several time and it has been found that the speed of the transformation start decrease slightly as the size of the system increase. In all cases however, the correct transformation was produced. These experiments were performed on a laptop with a AMD Athlon(tm)x2 dual-core ql-64 2.10 GHZ and 3GB RAM under Windows 7. The unit for time spent is ms (millisecond). Table 3 presents the results for the several testing cases.

We have also tested the Alloy code produced by AC2Alloy. To do so, we have created 6 scenarios and computed the time spent on analysis of every scenario as illustrated in table 4. In every scenario, we have increased the number of one component of the STRBAC model. For example, in the scenario 2 we have increased the number of Users and in the scenario 3 we have increased the number of Roles. Table 4 shows that in all scenarios a counterexample has been found. This means Alloy analyser has found

at lest one instance which is conflicted with the Access Control specification in the all scenarios.

Scenario	User	Role	Permission	Time	Location	URA	PRA	RH	LH	SoD	CC	No of all Elements	Speed of Transformation
Scenario 1	50	10	10	5	5	50	25	5	2	10	5	177	150ms
Scenario 2	100	20	20	10	10	100	50	10	4	20	10	354	220ms
Scenario 3	150	30	30	15	15	150	75	15	6	30	15	531	500ms
Scenario 4	200	40	40	20	20	200	100	20	8	40	20	708	2800ms
Scenario 5	250	50	50	25	25	250	125	25	10	50	25	885	1300ms
Scenario 6	300	60	60	30	30	300	200	30	12	60	30	1062	2000ms

Table 3. AC2Alloy Evaluation Result

We have also tested the Alloy code produced by AC2Alloy. To do so, we have created 6 scenarios and computed the time spent on analysis of every scenario as illustrated in table 4. In every scenario, we have increased the number of one component of the STRBAC model. For example, in the scenario 2 we have increased the number of Users and in the scenario 3 we have increased the number of Roles. Table 4 shows that in all scenarios a counterexample has been found. This means Alloy analyser has found at lest one instance which is conflicted with the Access Control specification in the all scenarios.

Another observation from the table 4 is that the time spent on analysis of Alloy code increases slightly as the number of all elements of STRBAC increase. This is because Alloy Analyser is a SAT-solver based and SAT-solving time may vary enormously depending on factors such as clause ordering, number of variables and average length of clauses [14]. It can also be noted that some elements of the STRBAC model such as Roles effect the speed of analysis more than the other elements. This is because in our model transformation every role is transformed into a Signature, Fact and Predicate in Alloy. This means the number of variables and clauses which will be created by the SAT-solver for every role is larger than the number of variables and clauses which will be created by the SAT-solver for any other elements such as user or permission. As consequence, when the number of roles increased by N number, the time spent on analysis will be larger than the time spent when any other elements of the STRBAC increased by the same number *N*.

Scenario	User	Role	Permission	Time	Location	URA	PRA	RH	LH	SoD	No of all Elements	Counterexample	Time Spent on Analysis
Scenario 1	30	10	10	2	5	30	10	2	2	1	102	Found	12.6ms
Scenario 2	300	10	10	2	5	300	10	2	2	1	642	Found	31.4ms
Scenario 3	30	100	10	2	5	300	10	2	2	1	462	Found	39.2ms
Scenario 4	30	10	100	2	5	30	100	2	2	1	282	Found	19.7ms
Scenario 5	30	10	10	2	5	30	10	25	2	1	125	Found	13.1ms
Scenario 6	30	10	10	2	5	30	10	2	25	1	125	Found	13.2ms

Table 4. Alloy code Test Result

8. Conclusion

In this paper we have provided formal algebraic notations of STRBAC model. With the help of the formal algebraic notations we have formalised the definition of inconsistency in the STRBAC specification. The paper has also introduced the new concept semi-consistency in the STRBAC specification. Additionally, the paper has proposed an automated method that transform the STRBAC specification to Alloy model and then make the use of Alloy analyser to analyse the produced Alloy model to detect inconsistencies and semi-consistencies in the STRBAC specification. The suggested method and the involving transformation is described with the help of an example.

References

[1] Huang, C., Sun, J., Wang, X., Si, Y. J. (2009). Security Policy Management for Systems Employing Role Based Access Control Model, *Information Technology Journal*, 8 (5) 726-734.

[2] Indrakshi Ray, Manachai Toahchoodee. (2008). A Spatio-Temporal Access Control Model Supporting Delegation for Pervasive Computing Applications. *In*: Proceedings of the 5th International Conference on Trust, *Privacy and Security in Digital Business*, p. 48.58, Turin, Italy, September .

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

[4] Manachai Toahchoodee, Indrakshi Ray. (2008). On the Formal Analysis of a Spatio-Temporal Role-Based Access Control Model. *In*: Proceedings of the 22nd Annual IFIPWG11.3Working Conference on Data and Applications Security, p. 17.32, London, U.K., July.

[5] Jackson.Daniel. (2006), Software Abstractions Logic, Language, and Analysis, Cambridge: The MIT Press.

[6] MDA: Model Driven Architecture, Object Management Group, (2005), www.omg.org/mda/.

[7] Liang Chen, Jason Crampton. (2008). On Spatio-Temporal Constraints and Inheritance in Role-Based Access Control. *In*: Proceedings of the 2008 ACM Symposium on Information, *Computer and Communications Security*, p. 205.216, Tokyo, Japan, March.

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

[9] Hsing-Chung Chen, Shiuh-Jeng Wang, Jyh-Horng Wen, Yung-Fa Huang, Chung-Wei Chen. (2010). A Generalized Temporal and Spatial Role-Based Access Control Model. JNW 5 (8) 912-920. *Journal of Multimedia Processing and Technologies*, 14.

[10] Indrakshi Ray, Manachai Toahchoodee. (2007). A Spatio-temporal Role-Based Access Control Model. *In*: Proceedings of the 21st Annual IFIPWG11.3Working *Conference on Data and Applications Security*, p. 211.226, Redondo Beach, CA, July.

[11] David F. Ferraiolo, Richard Kuhn, D., Ramaswamy Chandramouli. (2007). Role Based Access Control. 2nd Edition.

[12] Akehurst, D. H., Bordbar.Behzad, Evans, M., Howells, W. G. J., McDonald Maier, K. D. (2006). SiTra: Simple Transformations in Java, ACM/IEE 9th International *Conference on Model Driven Engineering Languages and Systems*, LNCS, 4199, p. 351-364.

[13] Samrat Mondal, Shamik Sural: XML-based policy specification framework for spatiotemporal access control. SIN 2009: 98-103

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

[15] Transformation of Spatio-Temporal Role Based Access Control Specification to Alloy,Geepalla, E., Bordbar, B., Last, J. (2012). 2nd International *Conference on Model and Data Engineering* (MEDI-2012), LNCS, 7602, 67-78

[16] Daniel Jackson. (2000). Automating First-order relational logic. *In*: Proceedings of the 8th ACM SIGSOFT International symposium on Foundations of Software Engineering, p. 130-139, San Diego, CA, USA, November.

[17] Daniel Jackson. (2002). Micromodels of Software: Lightweight Modelling and Analysis with Alloy. At http://alloy.mit.edu/alloy2website/reference-manual.pdf.

[18] Daniel Jackson. (2004). Alloy 3.0 reference manual. At http://alloy.mit.edu/reference-manual.pdf.

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

[20] Chen, L., Crampton, J. (2008). On spatio-temporal constraints and inheritance in role-based access control. *In*: Proceedings of the 2008 ACM Symposium on Information, *Computer and Communications Security*, p. 356-369.

[21] LE Thi Kim Tuyen, DANG Tran Khanh, KOUNEN Pierre, HOUDA Chabbi Drissi, (2012). STRoBAC-Spatial Temporal Role Based Access Control. *In*: Proceedings of the 4th International *Conference on Computational Collective Intelligence Technologies and Applications* (ICCCI), Part II, LNAI 7654, Springer-Verlag Heidelberg, p. 201- 211, 28-30 November, Ho Chi Minh city, Vietnam.

[22] Kumar, M., Newman, R. E. (2006). STRBAC-An Approach Towards Spatio-Temporal Role-Based Access Control. *In*: Communication, *Network and Information Security*, USA, p. 150-155.