Chapter 1

RBAC/MAC SECURITY FOR UML

T. Doan, S. Demurjian, T.C. Ting, and C. Phillips
Department of Computer Science and Engineering, University of Connecticut and
Department of Electrical Engineering and Computer Science, U.S. Military Academy
[thuong,steve,ting]@engr.uconn.edu and charles.phillips@usma.edu

Abstract In software construction, analysis investigates system requirements and
design captures system functionality. To facilitate analysis and design,
one popular technique is the unified modeling language, UML. In UML,
there are use-case diagrams for the interaction of users with system com-
ponents, class diagrams for the static classes and relations among them,
and sequence diagrams for the dynamic behavior of objects. However,
analyzing and designing security requirements in UML is not directly
supported. In this chapter, we incorporate role-based access control
(RBAC) and mandatory access control (MAC) into UML use-case and
class diagrams. In addition, we provide analysis across the UML dia-
grams, as actors, use cases and classes are defined, to support a degree
of security assurance (with mutual exclusion), thereby realizing secure
software design in UML. We briefly report on our RBAC/MAC enhance-
ments into Borland’s UML tool Together Control Center.

Keywords: RBAC, MAC, UML, Secure Software Design.

1. Introduction

In today’s web-based, distributed, and security conscious world, the
construction of large-scale software systems is a complex task, requiring
the comprehensive specification of requirements. However, the inclusion
of security analysis and design has often been an afterthought, relegated
to latter stages of the development life cycle or placed in the hands
of a database administrator. With a security scope that ranges from
security policy definition (what needs to be protected) to authorization
(grant/revoke privileges to users) to authentication (verify the users),
the availability of analysis (security assurance) during software design
will greatly improve the ability to attain and enforce security. In support of security analysis and design, this chapter reports on our research on incorporating role-based access control (RBAC) and mandatory access control (MAC) into the unified modeling language, UML [6].

UML combined the approaches of [5, 11, 18] and others into a standard, bringing stability to the object-oriented design market. In UML, nine different types of diagrams are available to model an application’s behavior and interactions. However, direct support for security (i.e., RBAC and MAC) in UML [14] is not provided. While there have been other efforts on security and UML [9, 12, 17, 19], we believe that there is a need for an approach for integrated security analysis and design in UML. Specifically, we report on the incorporation of RBAC and MAC into UML use-case diagrams (interaction of users with system components), class diagrams (the static classes and relations among them), and sequence diagrams (methods calls among objects). We detail extensions that capture RBAC and MAC in an integrated fashion with UML modeling elements. Using this as a basis, we propose secure relation constraints, SRCs, to enforce MAC and RBAC for a UML design. SRCs insure that the defined security is consistent with UML constructs and elements, and are the basis for more complex security analysis for UML that can attain mutual exclusion.

In the remainder of this chapter: Section 2 provides background information on MAC, RBAC, and UML; Section 3 details a formal model for RBAC/MAC extensions of UML; Section 4 examines SRCs among UML elements, available security analysis, and the integration of our work into the UML tool Borland’s Together Control Center (TCC); Section 5 reviews related research, and Section 6 concludes this chapter.

2. Background Concepts

In MAC, security levels (typically unclassified (U), confidential (C), secret (S), and top secret (T) forming a partial order \( U < C < S < T \)) are assigned to each subject (clearance - CLR) and each object (classification - CLS). The permission of the subject to perform some operation on the object depends on the relation between CLR and CLS as dictated by: Simple Security Property (“read down - no read up”) [3]; Simple Integrity Property (“write down - no write up”) [4]; Strict *-Property (“write equal”) [15]; and Liberal *-Property (“write up - no write down”) [3]. In RBAC [7, 8, 20, 10], roles are assigned to users to specify named functions or assignments that those users need to perform in the organization. Each role is then authorized to perform some operations on certain objects.
The UML [6, 14] has a wide range of capabilities and features; for the enhancement of UML with MAC and RBAC, we concentrate on:

- **A use case diagram** is a collection of use cases and actors. A use case represents an encapsulation of behavior for a specific portion of an application. Use cases can be related by: **B generalizes** means that A inherits behaviors of B; **C includes** D specifies that C contains the behaviors defined in D; and **E extends** F specifies that F may be augmented with some behaviors defined in E. An actor is an external entity that interacts with use cases to represent events (business processes) in the system, with a generalization relation from a child actor A to a parent actor B indicating that A inherits properties from B for specialization, and an association relation to a use case to indicate the actor utilizing the use case.

- **A class diagram**, composed of classes, is for the static structure of the conceptual model. A class is an abstraction for a set of objects that have the same attributes and methods of their behaviors. In implementation, an operation of the class is called message.

- **A sequence diagram** indicates the dynamic object interactions over time to accomplish a specific task. The purpose of a sequence diagram is to model flow of control, to illustrate a typical scenario or processing, thereby providing perspective on usage and flow of messages across the various objects that comprise an application.

In this chapter, we adopt only the Simple Security Property and Simple Integrity Property for MAC, and consider only inheritance for specialization (the child inherits the parent with properties that may be extended) for actors and use cases as specified in [14]. We assume there is no cycle in actor and class inheritance, use case inclusion and extension relations.

To illustrate the above concepts, consider a Survey Institution that performs and manages public surveys. After the raw data is collected, the senior staff person adds a survey header into the database. Then, a senior or junior staff member adds questions into that survey, and may categorize questions or add a new question category if needed. However, there are special questions that only senior staff are allowed to perform, specifically, data entry and classify. Figure 1.1a depicts a use-case diagram for creating a new survey entry in the “Survey Management.” The actor for the Staff role has two child actors for Junior Staff and Senior Staff roles, inherited by specialization with extension. Generally, the Staff actor can perform the use case Add Question which includes the use case Categorize Question, and can be extended to the use case Add Question Category if a new category must be added to the database. But, only the Senior Staff actor can perform the use case...
Add Survey Header (for a new survey header entry) and the use case Add Special Question (special questions) which includes the use case Classify Question Sensitivity (to define the questions). Figure 1.1b illustrates the sequence diagram for the use case Add Survey Header with only the main flow of events shown. To create a new survey header “Internet Usage”, the Senior Staff person enters data and submits in the Survey_Header>Add_Page, which will search for the survey title in the Survey_Repository object (of class Survey_List) and then send new header data to Survey_Repository via the Add_Survey_Header message. The Survey_Repository object creates a new survey header object Internet_Usage of class Survey_Header and updates itself by adding a new item in its list of surveys.

Figure 1.1. (a) Use Case Diagram for Create New Survey and (b) Sequence Diagram for Add Survey Header.

3. A Model for MAC/RBAC in UML

In this section, we describe our RBAC/MAC model integrated into UML use case and class diagrams. To begin, we define the set of element symbols \( \Sigma_{Elt} = \Sigma_{SL} \cup \Sigma_{LT} \cup \Sigma_{UC} \cup \Sigma_{Role} \cup \Sigma_{Actor} \cup \Sigma_{Class} \cup \Sigma_{Meth} \) where the \( \Sigma_j \) are disjoint sets of MAC security levels, life times, use cases, roles, actors, classes, and methods, respectively. First, we define MAC security levels and lifetimes [16]:

**Def. 1** The set of MAC Security Levels \( \Sigma_{SL} = \{ \sigma_{SL}^1 = \perp, \sigma_{SL}^2, \ldots \} \) is a linearly ordered set where \( \sigma_{SL}^i < \sigma_{SL}^j \) with \( i < j \) means the level \( j \) has a higher security concern than that of \( i \). We use \( \leq \) for less than or equal and \( \geq \) (also called “dominate” relation) for not <.

For example, a security manager may define \( \Sigma_{SL} = \{U, C, S, TS\} \) with the security order relation \( \perp = U < C < S < TS \).
Def. 2 A lifetime $lt \in \Sigma^{LT}$ is a time interval with start time ($st$) and end time ($et$), $[st, et], et \geq st$, where $st$ and $et$ is of the form "year-month-day [hour:minute:second]". We denote $lt = \emptyset$ if $st = et$. Relations between $lt$s $X$ and $Y$ are:

$X \triangleright Y$ means $Y.st \geq X.st$ and $Y.et \leq X.et$; $X \vartriangleleft Y \equiv Y \triangleright X$; If $ST = \max\{X.st, Y.st\}$ and $ET = \min\{X.et, Y.et\}$, then $Y \cap X$ is $\emptyset$ if $ET \leq ST$ or $[ST, ET]$ if $ET > ST$; and $lt = [ct, \infty]$ is current time (ct) onward.

To accommodate the security features for a use case, we define:

Def. 3 A use case $uc$ of the use case set $\Sigma^{UC}$ is a structure with properties: name $uc.Name$; lifetime $uc.LT$; classification $uc.CLS \in \Sigma^{SL}$; and, body $uc.Body$.

In Figure 1.1a, $uc_1$ has $uc_1.Name=Add Survey Header$, $uc_1.LT = ["2004-01-01", \infty]$ (Jan. 01, 2004 onward), and $uc_1.CLS = S$. To define MAC and RBAC in our model, we structure a role as follows:

Def. 4 A role $r$ of the role set $\Sigma^{Role}$ is a structure with properties: name $r.Name$; LT $r.LT$; and, clearance $r.CLR \in \Sigma^{SL}$.

In our approach, an actor represents one organizational role defined by the security officer, which differs from actor-use case roles in UML [14], which are used by actors to communicate with each specific use case.

Def. 5 An actor $a$ of the actor set $\Sigma^{Actor}$ is a structure with properties: name $a.Name$, and $a.R$, the role that $a$ represents.

In Figure 1.1a, $a_1 = Staff$ has $a_1.R = "Staff"$ where $a_1.R.LT = ["2004-01-01", \infty]$ and $a_1.R.CLR = C$. Next, we define extensions for UML classes and methods.

Def. 6 A class $c$ of the class set $\Sigma^{Class}$ is a structure with properties: name $c.Name$; LT $c.LT$; maximum/minimum classifications, $c.CLS_{max}$ and $c.CLS_{min}$ (both in $\Sigma^{SL}$) with $c.CLS_{max} \geq c.CLS_{min}$; and, attribute set $c.AttrS$ which each of the form attribute\_name: type which is a basic type (e.g., Integer, Real, etc.) or a class name.

Def. 7 A method $m$ of the method set $\Sigma^{Meth}$ is a structure with the properties: name $m.Name$; LT of $m$; classification $m.CLS \in \Sigma^{SL}$; $m.Cl$, the class that defines $m$; input parameters $m.InputS$; return type $m.Output$ (may return void); and, body $m.Body$.

For relations, we define a set of relation symbols $\Psi = \{\psi^1, \psi^2, \ldots\}$ where $\nu_i$ is the relation name. Next, for use case inheritance, inclusion, and extension, we define $\psi_i$:
Def. 8 Use Case Inheritance relation \( \psi^{Uth} \subseteq \Sigma^{UC} \times \Sigma^{UC} \): \( \psi^{Uth}(uc_i, uc_j) \) iff \( uc_i \) inherits (directly specializes) \( uc_j \).

We denote \( uc.Uc(UthP) = \{ uc_k | \psi^{Uth}(uc, uc_k), uc_k \neq uc \} \) as the (direct) parents of \( uc \) (\( uc.Uc(UthP) = \emptyset \) if \( \neg \exists uc_k : \psi^{Uth}(uc, uc_k) \)) and recursively, \( uc.Uc(UthP*) = uc.Uc(UthP) \cup \bigcup_{uc_k \in uc.Uc(UthP)} uc.k.Uc(UthP*) \) as the ancestors of \( uc \).

Def. 9 Use Case Inclusion relation \( \psi^{Uic} \subseteq \Sigma^{UC} \times \Sigma^{UC} \): \( \psi^{Uic}(uc_i, uc_j) \) iff \( uc_i \) includes (\( \ll include \gg \)) \( uc_j \).

We denote \( uc.Uc(UicC) \) and \( uc.Uc(UicC*) \) as the use cases that \( uc \) directly includes and transitively includes, respectively.

Def. 10 Use Case Extension relation \( \psi^{Uex} \subseteq \Sigma^{UC} \times \Sigma^{UC} \): \( \psi^{Uex}(uc_i, uc_j) \) iff \( uc_j \) extends (\( \ll extend \gg \)) the base \( uc_i \).

Next, we define \( \psi^r \) among roles, actors or actors with use cases:

Def. 11 Role Inheritance relation \( \psi^{Rih} \subseteq \Sigma^{Role} \times \Sigma^{Role} \): \( \psi^{Rih}(r_i, r_j) \) iff \( r_i \) inherits (directly specializes) \( r_j \).

We denote \( a.Ac(AihP) \) as \( a \)'s parent and \( a.Ac(AihP*) \) \( a \)'s ancestors.

Def. 12 Actor Inheritance relation \( \psi^{Aih} \subseteq \Sigma^{Actor} \times \Sigma^{Actor} \): \( \psi^{Aih}(a_i, a_j) \) iff \( a_i \) inherits (directly specializes) \( a_j \).

We denote \( a.Cl(AihP) \) as \( a \)'s parent and \( a.Cl(AihP*) \) \( a \)'s ancestors.

Def. 13 Actor-Use Case Association relation \( \psi^{AuA} \subseteq \Sigma^{Actor} \times \Sigma^{UC} \): \( \psi^{AuA}(a, uc) \) iff \( a \) interacts with \( uc \) by association.

Let \( a.Uc(AuA) \) be the use cases \( a \) communicates with, and \( uc.Ac(AuA) \) the actors communicating with \( uc \). In Figure 1.1a, we have \( \psi^{AuA}(a_1, uc_2) \), \( a_1.Uc(AuA) = \{ uc_2 \} \), and \( uc_2.Ac(AuA) = \{ a_1 \} \). Now, we specify relations between classes, use cases, and/or actors.

Def. 14 Class Inheritance relation \( \psi^{Cih} \subseteq \Sigma^{Class} \times \Sigma^{Class} \): \( \psi^{Cih}(c_i, c_j) \) iff \( c_i \) inherits \( c_j \) (\( c_i \) a child of \( c_j \)).

We denote \( c.Cl(CihP) \) as the parent of \( c \) and \( c.Cl(CihP*) \) as the ancestors of \( c \). (Note that at this time we have not considered the Class Association relation.)

Def. 15 Use Case-Class Utilization relation \( \psi^{UCUz} \subseteq \Sigma^{UC} \times \Sigma^{Class} \): \( \psi^{UCUz}(uc, c) \) iff \( uc \) utilizes \( c \).
We denote \( uc.Cl(UC^{Uz}) = \{c|\psi^{UC^{Uz}}(uc,c)\} \) as the classes utilized by \( uc \), and \( c.Uc(UC^{Uz}) = \{uc|\psi^{UC^{Uz}}(uc,c)\} \) as the use cases utilizing \( c \). Now, we consider various relations of methods.

**Def. 16** Method-Method Calling relation \( \psi^{MCa} \subseteq \Sigma^{Meth} \times \Sigma^{Meth} \):

\[
\psi^{MCa}(m_i, m_j) \text{ iff } m_i \text{ calls } m_j.
\]

We denote \( m.Mt(M^{CaC}) = \{m_k|\psi^{MCa}(m, m_k), m_k \neq m\} \) (with the condition \( m_k \neq m \) to exclude \( m \) out of the set \( m.Mt(M^{CaC}) \) in the case of recursive call) as the methods that \( m \) directly calls and recursively, \( m.Mt(M^{CaC^*}) = m.Mt(M^{CaC}) \cup \left[ \bigcup_{m_k \in m.Mt(M^{CaC})} m_k.Mt(M^{CaC^*}) \right] \) as the methods that \( m \) calls directly or indirectly (via some methods).

**Def. 17** Class-Method Defining relation \( \psi^{CMDef} \subseteq \Sigma^{Class} \times \Sigma^{Meth} \):

\[
\psi^{CMDef}(c, m) \text{ iff } c \text{ defines } m \text{ (i.e. } m.Cl = c).\]

**Def. 18** Use Case-Method Utilization relation \( \psi^{UMUz} \subseteq \Sigma^{UC} \times \Sigma^{Meth} \):

\[
\psi^{UMUz}(uc, m) \text{ iff } uc \text{ directly utilizes } m.
\]

We denote \( uc.Mt(UM^{Uz}) = \{m|\psi^{UMUz}(uc, m)\} \) as the methods directly utilized by \( uc \), and \( m.Uc(UM^{Uz}) = \{uc|\psi^{UMUz}(uc, m)\} \) as use cases that directly utilize \( m \).

### 4. Security Assurance for UML

In this section, we focus on security assurance for UML, concentrating on the security analysis that can occur as relations are defined between various UML elements. To address this, in Section 4.1, we utilize the model as given in Section 3 to define secure relation constrains, SRC, which enforce MAC, RBAC, and mutual exclusion requirements on different UML modeling elements. Using this as a basis, Section 4.2 explores the process of security analysis as a UML design is defined and evolved over time. Finally, Section 4.3 details our prototyping effort for the SRCs using the UML tool Together Control Center.

#### 4.1 Secure Relation Constraints (SRC)

A secure relation constraint, SRC, represents the actions that must be checked whenever relations are added in a UML diagram, and can involve a MAC Constraint (MACC), a RBAC Constraint (RBACC), or both. The first group of SRCs are MACCs for use case inheritance (child’s CLS dominates the parent’s CLS), use case inclusion (base’s CLS dominates the included use case’s CLS), use case extension (extended use case’s
CLS dominates the extending use case’s CLS), class inheritance (child’s CLS dominates parent’s CLS), and use-case/class relations (use case’s CLS must dominate the CLS_{min} of the class).

**SRC U^{Ih}, U^{Ic} and U^{Ex} (Use Case Inheritance, Inclusion, and Extension SRC):** Given the relation ψ^{U^{η}}(uc_{i}, uc_{j})
MACC: ψ^{U^{η}}(uc_{i}, uc_{j}).SC_{MAC} = uc_{i}.CLS \geq uc_{j}.CLS, for η = Ih, Ic, and Ex, for use case inheritance, inclusion, and extension.

**SRC C^{Ih} (Class Inheritance SRC):** Given the relation ψ^{C^{Ih}}(c_{i}, c_{j})
(c_{i} child of c_{j})
MACC: ψ^{C^{Ih}}(c_{i}, c_{j}).SC_{MAC} = c_{i}.CLS_{min} \geq c_{j}.CLS_{min}.

**SRC UC^{Uz} (Use Case-Class Utilization SRC):** Given the relation ψ^{UC^{Uz}}(uc, c) (uc utilizes c)
MACC: ψ^{UC^{Uz}}(uc, c).SC_{MAC} = uc.CLS \geq c.CLS_{min}.

The second group of SRCs are MACCs for method invocations (calling method’s CLS must dominate the CLS of the called method), class/method dependencies (method’s CLS must be between the class’s CLS_{min} and CLS_{max}), and use-case/method interactions (use case’s CLS must dominate the method’s CLS).

**SRC M^{Ca} (Method Calling SRC):** Given the relation ψ^{M^{Ca}}(m_{i}, m_{j})
MACC: ψ^{M^{Ca}}(m_{i}, m_{j}).SC_{MAC} = m_{i}.CLS \geq m_{j}.CLS.

**SRC CM^{Def} (Class-Method Utilization SRC):** Given the relation ψ^{CM^{Def}}(c, m)
MACC: ψ^{CM^{Def}}(c, m).SC_{MAC} = (c.CLS_{max} \geq m.CLS) \land (m.CLS \geq c.CLS_{min}).

**SRC UM^{Uz} (Use Case-Method Utilization SRC):** Given the relation ψ^{UM^{Uz}}(uc, m)
MACC: ψ^{UM^{Uz}}(uc, m).SC_{MAC} = uc.CLS \geq m.CLS.

The third group of SRCs are MACCs and RBACCs for actor/use case associations and actor inheritance. For actor/use case associations, a employs uc. For MACC, the actor’s role CLR must dominate the uc.CLS. For RBACC, a.R must have privileges and valid LT to utilize uc, the ancestor classes utilized by uc, direct/indirect methods utilized by uc and its parent and included use cases, recursively. To represent these recursive sets, we denote uc.Cl(UC^{Uz,IhP*}) = uc.Cl(UC^{Uz}) ∪ \left[ \bigcup_{c_{i} \in uc.Cl(UC^{Uz})} c_{i}.Cl(C^{IhP*}) \right] as the set of all classes utilized by uc and
uc.Mt(UM^{Uz,CaC*}) = uc.Mt(UM^{Uz}) \cup \left( \bigcup_{m_i \in uc.Mt(UM^{Uz})} m_i.Mt(M^{CaC*}) \right)
as the set of all methods utilized by uc which includes direct and indirect method calls. Next, we denote uc.Uc = \{ uc_i | \psi^{U^h}(uc, uc_i) \lor \psi^{U^c}(uc, uc_i) \} as the set of use cases that uc inherits or includes, and recursively, uc.Uc* = uc.Uc \cup \left( \bigcup_{uc_i \in uc.Uc} uc_i.Uc \right) as the set of all directly and transitively ancestor or included use cases of uc. Then, uc.Cl* = uc.Cl(UC^{Uz,IhP*}) \cup \left( \bigcup_{uc_i \in uc.Uc*} uc_i.Cl(UC^{Uz,IhP*}) \right) as the set of all classes utilized by uc and all direct and transitive ancestors or included use cases of uc, and the ancestors of these classes which may not be directly utilized by these use cases. Lastly, uc.Mt* = uc.Mt(UM^{Uz,M^{CaC*}}) \cup \left( \bigcup_{uc_i \in uc.Uc*} uc_i.Mt(UM^{Uz,CaC*}) \right) is the set of all methods utilized by uc and direct and transitive ancestors or included use cases of uc and direct and indirect method calls.

Now, we define privileges of roles for use cases, classes, and methods, to establish the conditions under which privileges can be granted.

**Def. 19** A Role-Use case Privilege relation is defined as $
\psi^{RU} \subseteq \Sigma_{Role} \times \Sigma_{UC} \times \Sigma_{LT}$: $\psi^{RU}(r, uc, lt)$ iff the role r is authorized to utilize use case uc during the lifetime lt.

**Def. 20** A Role-Class Privilege relation is defined as $
\psi^{RC} \subseteq \Sigma_{Role} \times \Sigma_{Class} \times \Sigma_{LT}$: $\psi^{RC}(r, c, lt)$ iff the role r is authorized to utilize class c during the lifetime lt.

**Def. 21** A Role-Method Privilege relation is defined as $
\psi^{RM} \subseteq \Sigma_{Role} \times \Sigma_{Meth} \times \Sigma_{LT}$: $\psi^{RM}(r, m, lt)$ iff the role r is authorized to utilize method m during the lifetime lt.

Notationally, $\psi^{RU}(r, uc, \emptyset)$, $\psi^{RC}(r, c, \emptyset)$, and $\psi^{RM}(r, m, \emptyset)$ mean that role r is prohibited from utilizing uc, c, and m, respectively. We denote $SC^{AU}_{RBAC}(a, uc) = \psi^{RU}(a.R, uc, lt^{RU}) \land [\emptyset \neq lt^{RU} \land (a.R.LT \cap uc.LT)]$ as the direct RBACC between actor a associated with use case uc where the role a.R can utilize uc and the lifetime of this privilege is the non-empty interval within the intersection of the a.R.LT and uc.LT. Similarly, we have $SC^{AC}_{RBAC}(a, c) = \psi^{RC}(a.R, c, lt^{RC}) \land [\emptyset \neq lt^{RC} \land (a.R.LT \cap c.LT)]$ and $SC^{AM}_{RBAC}(a, m) = \psi^{RM}(a.R, m, lt^{RM}) \land [\emptyset \neq lt^{RM} \land (a.R.LT \cap m.LT)]$. Given this background, MACC and RBACC for actor-use case association is:

**SRC $AT^A$ (Actor-Use Case Association SRC):** Given the relation $\psi^{AU}(a, uc)$
MACC: $\psi^{AU}(a, uc).SC_{MAC} = a.R.CLR \geq uc.CLS$ (role's CLR dominates uc's CLS).

RBACC: $\psi^{AU}(a, uc).SC_{RBAC} = SC^{AU}_{RBAC}(a, uc) \land SC^{AU}_{RBAC}(a, uc)$, where

$SC^{AU}_{RBAC}(a, uc) = SC^{AU}_{RBAC}(a, uc) \land \left[ \bigcup_{uc_k \in uc.Uc} \bigcup_{a.Uc} SC^{AU}_{RBAC}(a, uc_k) \right]$

$SC^{AU}_{RBAC}(a, uc) = \bigwedge_{uc_k \in uc.CI} SC^{AC}_{RBAC}(a, uc_k)$,

$SC^{AU}_{RBAC}(a, uc) = \bigwedge_{uc_k \in uc.Mt} SC^{AC}_{RBAC}(a, uc_k)$

$SC^{AU}_{RBAC}(a, uc)$, $SC^{AU}_{RBAC}(a, uc)$, and $SC^{AU}_{RBAC}(a, uc)$ are RBACCs that allow role $a.R$ to utilize affected use cases, classes and methods by the association from $a$ to $uc$. Generally, let $a.Uc = a.Uc(\text{AU}^A) \cup \bigcup_{uc_k \in a.Uc(\text{AU}^A)} uc_k.Uc *$ be the set of all use cases that role $a.R$ can utilize by the associations from $a$ to directly connected use cases, and $a.Uc* = a.Uc \bigcup \bigcup_{uc_k \in a.Uc(\text{AU}^A)} a.Uc$ the set of all use cases that role $a.R$ can utilize by the associations of $a$ and its ancestors with connected use cases. Similarly, let $a.Cl = \bigcup_{uc_k \in a.Uc(\text{AU}^A)} uc_k.Cl*$ be the set of all classes that role $a.R$ can utilize via the associations from $a$ to directly connected use cases, $a.Cl* = a.Cl \bigcup \bigcup_{uc_k \in a.Uc(\text{AU}^A)} a.Cl$ the set of all classes that role $a.R$ can utilize by the associations from $a$ and its ancestors with connected use cases. Finally, let $a.Mt = \bigcup_{uc_k \in a.Uc(\text{AU}^A)} uc_k.Mt*$, we denote $a.Mt* = a.Mt \bigcup \bigcup_{uc_k \in a.Uc(\text{AU}^A)} a.Mt$ as the set of all methods that role $a.R$ can utilize by the associations from $a$ and its ancestors with connected use cases. Now we specify MACC and RBACC for an actor inheritance relation, where actor $a_i$ inherits actor $a_j$. For MACC, the child actor’s $R.CLR$ must dominate its parent’s $R.CLR$. For RBACC, the child’s role inherits the parent’s role, and is able to utilize all of the use cases, classes and methods of the parent’s role.

SRC $A^{Th}$ (Actor-Inheritance SRC): Given the relation $\psi^{A^{Th}}(a_i, a_j)$

MACC: $\psi^{A^{Th}}(a_i, a_j).SC_{MAC} = a_i.R.CLR \geq a_j.R.CLR$.

RBACC: $\psi^{A^{Th}}(a_i, a_j).SC_{RBAC} = \psi^{R^{Th}}(a_i, a_R, a_j, R) \land SC^{A^{Th}U}_{RBAC}(a_i, a_j) \land SC^{A^{Th}C}_{RBAC}(a_i, a_j) \land SC^{A^{Th}M}_{RBAC}(a_i, a_j)$, where

$SC^{A^{Th}U}_{RBAC}(a_i, a_j) = \bigwedge_{uc_k \in a.Uc} SC^{AU}_{RBAC}(a_i, uc_k)$,
RBAC/MAC Security for UML

Finally, we can establish mutual exclusion (ME) conditions between two roles with respect to a particular use case, which provides us with the ability to dictate exclusionary conditions.

**Def. 22** Two roles \( r_i \) and \( r_j \) have a mutual exclusion relation on use case \( uc \) during \( lt \) denoted \( ME^{RU}(r_i, r_j, uc, lt) \), iff \( r_i \) and \( r_j \) cannot utilize \( uc \) at the same time during \( lt \neq \emptyset \). \((ME^{RU}(r_i, r_j, uc, lt) \) is specified explicitly by the security policy.\)

This leads to an SRC for mutual exclusion between roles on a use case.

**SRC \( MEC^{RU} \) (Role-Use Case ME SRC):** Given the relation \( ME^{RU}(r_i, r_j, uc, lt) \), \( lt \neq \emptyset \), let \( a_k . R \) be \( r_k \) and \( lt_k \) lifetime in \( \psi^{RU}_{a_k . R, uc, lt} \) \( k \in \{i, j\} \),

\[ ME^{RU}(r_i, r_j, uc, lt) = \text{if } ME^{RU}(r_i, r_j, uc, lt) \land (lt \neq \emptyset) \text{ then } \neg [(uc \in a_i . UC \cap a_j . UC) \land SC_{RBAC}(a_i, uc) \land SC_{RBAC}(a_j, uc) \land (lt \cap lt_i \cap lt_j \neq \emptyset)]. \]

ME/MEC for roles on class \( ME^{RC}(r_i, r_j, c, lt)/MEC^{RC}(r_i, r_j, c, lt) \) and method \( ME^{RM}(r_i, r_j, m, lt)/MEC^{RM}(r_i, r_j, m, lt) \) are defined similarly.

### 4.2 Security Analysis

Security analysis for UML utilizes the SRCs for relations and mutual exclusion types (see Section 4.1 again), to maintain RBACCs and MACCs as a UML design is defined and modified. The MACCs are straightforward (domination), with the RBACCs requiring additional checks due to the indirect dependencies that can arise as connections occur throughout the different UML diagrams. Specifically, we must check \( SC_{RBAC}^\eta(a, x) \) where \( \eta \) is U,C, and M and \( x \in \Sigma^{UC}, \Sigma^{Class} \) and \( \Sigma^{Meth} \) is for use case, class, or method, respectively.

A classic approach to RBAC may set up tables for \( \psi^{RU}_{a, R, x, lt^x} \)'s where each tuple explicitly stipulates that role \( a.R \) be allowed to use \( x \) during \( lt^x \). However, since a UML design is heavily interconnected via relations with numerous actors, use cases, classes, and methods, the construction of flat tables would be problematic. Consequently, our approach for RBACC checking advocates a three step methodology:

**Step 1 : Establish Positive Privileges based on Connections in UML Diagrams.**
When connections are made in UML diagrams between actor $a$ and (directly or indirectly) $a$-affected elements $x$’s (i.e., use cases, classes, or methods), the tuples $\psi_{R \eta}(a.R, x, lt^x)$’s are implicitly set to be true with the lifetime $lt^x = a.R.LT \cap x.LT$. Note that by $a$-affected element $y$ we mean $y \in a.Uc^*$, $a.Cl^*$ or $a.Mt^*$.

**Step 2: Declare Negative Privileges by Security Policy.**

The security officer *explicitly* specifies the use cases, classes, and methods that role $a.R$ cannot use by setting the lifetime $lt^x$ of the corresponding tuples to $\emptyset$.

**Step 3: Discover the Privilege Conflicts.**

For each actor, check the SRC’s $AU^A$ on that actor and $AI^h$ if the actor has a parent. Any $a$-affected element $x$ that $SC_{RBAC}^R(a, x)$ is *false* will be reported as an RBAC error. Methodologically, we suggest one starts with actors without a parent, and then proceed with descendant actors using a depth-first search strategy.

Our approach also allows role-element mutual exclusion $ME^R_{\eta}$ via an SRC $MEC^R_{\eta}$ check on every $ME^R_{\eta}(r_i, r_j, x, lt) = false$ indicates a role-$\eta$ mutual exclusion error. To organize this process, consider Table 1.1. During initial design (i.e., use cases and actors with only actors’ role.CLS concerned), the checks can be limited to MACC for actors and use cases. As a design starts to evolve with classes and methods, MACC checks can be expanded to the entire MACC column in Table 1.1. Once relations and role privileges are defined, the checks in the RBACC column are activated. Finally, as mutual exclusions among roles are defined, the checks in the Role MEC column are enabled. By considering the iterations of a UML design, the checks at early stages can be limited to MACC, which can be performed using a trigger, e.g., on the event of *Draw a connection between elements $x$ and $y$*, if MACC of $x$ and $y = true$ then *Establish the connection.*

<table>
<thead>
<tr>
<th>Actor</th>
<th>$\nu = AI^h$</th>
<th>$\psi_{R^h}$</th>
<th>$\psi_{R^\eta}$</th>
<th>$SC_{MAC}^AI^h, SC_{RBAC}^{AI^h}$</th>
<th>$SC_{MAC}^{AI^h}, SC_{RBAC}^{AI^h}$</th>
<th>$MEC^{AI^h}$</th>
<th>$SC_{RBAC}^{AI^h}, SC_{RBAC}^{AI^h}$</th>
<th>$MEC^{AI^h}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case</td>
<td>$\nu \in {U^{Ih}, U^{Ie}, U^{Ex}, AU^A}$</td>
<td>$SC_{RBAC}^{AI^h}, SC_{RBAC}^{AI^h}$</td>
<td>$MEC^{AI^h}$</td>
<td>$SC_{RBAC}^{AI^h}, SC_{RBAC}^{AI^h}$</td>
<td>$MEC^{AI^h}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class</td>
<td>$\nu \in {C^{Ih}, UC^{Iu}}$</td>
<td>$SC_{RBAC}^{AI^h}, SC_{RBAC}^{AI^h}$</td>
<td>$MEC^{AI^h}$</td>
<td>$SC_{RBAC}^{AI^h}, SC_{RBAC}^{AI^h}$</td>
<td>$MEC^{AI^h}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>$\nu \in {M^{Ca}, CM^{DxJ}, UM^{Us}}$</td>
<td>$SC_{RBAC}^{AI^h}, SC_{RBAC}^{AI^h}$</td>
<td>$MEC^{AI^h}$</td>
<td>$SC_{RBAC}^{AI^h}, SC_{RBAC}^{AI^h}$</td>
<td>$MEC^{AI^h}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Table 1.1.* MEC Checks for Actors, Use Cases, Classes, and Methods.
4.3 Prototyping Effort

During the last six months, we have been transitioning RBAC, MAC, and the SRCs into Borland’s UML tool Together Control Center (TCC). TCC provides Open APIs and a plug-in structure, which has allowed us to incorporate our security properties (e.g., CLS and CLR) into UML across the different diagrams, and to include custom Java code that realizes the SCRs (Section 4.2), to dynamically analyze the security as a UML design is constructed. To illustrate, in Figure 1.2a, actor $a_3$’s security property contains role Senior Staff with its $LT=\left[\text{"2004-01-01", \text{infinity}}\right]$ and $CLR = S$ (Secret), where UML’s Security property display has been altered with our own custom code. For SCRs, consider Figure 1.2b; if we attempt to connect actor $a_1$:Staff to use case $uc_1$:Add Survey Header, an error dialog is displayed, since the $Staff.CLR = C < uc_1.CLR = S$ violates the MACC of SRC $AU^A$.

![Figure 1.2](image)

Figure 1.2. (a) Assigning the Security Property for an Actor and (b) Violating MAC Constraint of Actor-Use Case Association.

5. Related Research

There have been a number of other research efforts in security for UML. In [9], a proposed Framework for Network Enterprise utilizes UML to describe a RBAC model for representing RBAC requirements without separation of duty [10]. Similarly, there have been techniques proposed to utilize UML to describe RBAC [19] and MAC/RBAC [17] modeling and processing. Both of these efforts have focused on utilizing UML elements to model (MAC/)RBAC systems rather than taking a view of secure software analysis and design (A&D), which is as our focus. Analogously to our work, [12] has proposed extended UML features to accommodate security requirements via a Abstract State Machine model to formalize UML elements (use cases omitted) and extend several stereotypes to accommodate their proposed security framework.
towards theoretical security verification with UML. It contrasts with our approach to extend properties of essential UML elements in order to directly apply MAC/RBAC security models for secure A&D. [13] introduced SecureUML as a UML-based modeling language for model-driven security with extended meta-model elements for RBAC representation, which does not include MAC. This approach requires the designer to explicitly assign extended stereotypes “Permission”/“ActionType” for “Role” and application elements whereas, in our work, it is done implicitly by drawing connections in use case and sequence diagrams. The RBAC constraint conflicts were not considered in their work. Closer to our work, [1, 2] proposed a framework to incorporate security into UML design. However, this work only deals with use case modeling on RBAC without considering include/extend relations and sequence diagrams.

6. Concluding Remarks and Ongoing Research

In this chapter, we have extended the UML to model RBAC and MAC for actors, use cases, classes, and methods, across use-case, class, and sequence diagrams. Our modeling effort is bolstered by analysis via security relation constraints, SRCs, to analyze UML designs as relations between the different modeling elements are defined. Such security analysis can be facilitated by a methodology and in practice via the UML tool Together Control Center. Practically, our ongoing work is focusing on completing the prototype. From a research perspective, our objective is to utilize the model and associated analysis (Sections 3 and 4.1) as a basis for formal security specification and analysis, towards security policy consistency, availability, and redundancy.

Acknowledgement: Thanks to A. Ketterl for his prototyping efforts.

References


REFERENCES


