

**İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF INFORMATICS**

**A UML PROFILE FOR ROLE-BASED ACCESS CONTROL**

**M.Sc. Thesis by  
Çağdaş CİRİT**

**Department : Advanced Technologies in Engineering**

**Programme : Computer Science**

**JUNE 2009**



**A UML PROFILE FOR ROLE-BASED ACCESS CONTROL**

**M.Sc. Thesis by  
Çağdaş CİRİT  
(704061005)**

**Date of submission : 04 May 2009  
Date of defence examination: 08 June 2009**

**Supervisor (Chairman) : Assist. Prof. Dr. Feza BUZLUCA (ITU)  
Members of the Examining Committee : Prof. Dr. Nadia ERDOĞAN (ITU)  
Prof. Dr. Oya KALIPSIZ (YTU)**

**JUNE 2009**



**İSTANBUL TEKNİK ÜNİVERSİTESİ ★ BİLİŞİM ENSTİTÜSÜ**

**ROL-TABANLI ERİŞİM DENETİMİ İÇİN BİR UML PROFİLİ**

**YÜKSEK LİSANS TEZİ  
Çağdaş CİRİT  
(704061005)**

**Tezin Enstitüye Verildiği Tarih : 04 Mayıs 2009**

**Tezin Savunulduğu Tarih : 08 Haziran 2009**

**Tez Danışmanı : Yrd. Doç. Dr. Feza BUZLUCA (İTÜ)  
Diğer Jüri Üyeleri : Prof. Dr. Nadia ERDOĞAN (İTÜ)  
Prof. Dr. Oya KALIPSIZ (YTÜ)**

**HAZİRAN 2009**



## **FOREWORD**

I would like to express my deep appreciation and thanks for my advisor, Assist. Prof. Dr. Feza BUZLUCA.

I also want to thank my parents, Ümmü and Mehmet; my brother, Ümit; my sister Özlem; and my wife, Şule, for their understanding and support.

May 2009

Çağdaş Cirit  
Computer Engineer





## TABLE OF CONTENTS

	<u>Page</u>
<b>ABBREVIATIONS</b> .....	<b>ix</b>
<b>LIST OF TABLES</b> .....	<b>x</b>
<b>LIST OF FIGURES</b> .....	<b>xi</b>
<b>SUMMARY</b> .....	<b>xiii</b>
<b>ÖZET</b> .....	<b>xv</b>
<b>1. INTRODUCTION</b> .....	<b>1</b>
1.1 Motivation .....	1
1.2 Suggested Approach.....	2
1.3 Contributions .....	3
1.4 Thesis Outline .....	4
<b>2. BACKGROUND</b> .....	<b>5</b>
2.1 RBAC .....	5
2.1.1 Core RBAC .....	6
2.1.2 Hierarchical RBAC .....	7
2.1.3 SSD Relations .....	7
2.1.4 DSD Relations.....	8
2.2 MDA.....	9
2.3 UML .....	9
2.3.1 Class Diagram .....	10
2.3.2 UML Profile .....	10
2.4 OCL.....	11
2.5 Related Works .....	12
<b>3. RBAC UML PROFILE</b> .....	<b>15</b>
3.1 Conceptual Model .....	15
3.2 Proposed UML Profile for RBAC.....	16
3.2.1 User Stereotype .....	18
3.2.2 Role Stereotype .....	18
3.2.3 Resource Stereotype.....	20
3.2.4 Operation Stereotype.....	20
3.2.5 Permission Stereotype.....	21
3.2.6 Session Stereotype .....	22
3.2.7 ResourceAssignment Stereotype.....	23
3.2.8 UserAssignment Stereotype.....	24
3.2.9 PermissionAssignment Stereotype.....	25
3.2.10 RoleInheritance Stereotype .....	25
3.2.11 SoD Stereotype .....	26
3.2.12 SSD Stereotype .....	27
3.2.13 DSD Stereotype.....	27
3.2.14 CriticalPermission Stereotype.....	27
3.2.15 TimeConstraint Stereotype .....	28
3.3 OCL Expressions for Profile Constraints.....	29

<b>4. EXAMPLE DESIGN PROBLEM .....</b>	<b>31</b>
4.1 Problem Domain Requirements .....	31
4.2 Access Control Requirements .....	31
4.3 System Design Model.....	32
4.4 Security Model .....	32
4.5 Platform Independent Model.....	39
4.6 Ill-formed Security Model.....	39
<b>5. CONCLUSION AND RECOMMENDATIONS .....</b>	<b>41</b>
<b>REFERENCES .....</b>	<b>43</b>
<b>APPENDICES .....</b>	<b>45</b>
<b>CURRICULUM VITA.....</b>	<b>59</b>

## **ABBREVIATIONS**

<b>ANSI</b>	: American National Standards Institute
<b>CASE</b>	: Computer-Aided Software Engineering
<b>DSD</b>	: Dynamic Separation of Duty
<b>INCITS</b>	: International Committee for Information Technology
<b>MDA</b>	: Model Driven Architecture
<b>MDS</b>	: Model Driven Security
<b>MOF</b>	: Meta-Object Facility
<b>OCL</b>	: Object Constraint Language
<b>OMG</b>	: Object Management Group
<b>PIM</b>	: Platform Independent Model
<b>PSM</b>	: Platform Specific Model
<b>RBAC</b>	: Role-Based Access Control
<b>SoD</b>	: Separation of Duty
<b>SSD</b>	: Static Separation of Duty
<b>UML</b>	: Unified Modeling Language
<b>XMI</b>	: XML Metadata Interchange
<b>XACML</b>	: eXtensible Access Control Markup Language

## LIST OF TABLES

	<b><u>Page</u></b>
Table 2.1 : OCL Constraints Types.....	12
Table A.1 : Global OCL Definitions .....	46
Table B.1 : Error Messages .....	51
Table C.1 : Errors of the ill-formed security model.....	54

## LIST OF FIGURES

	<u>Page</u>
Figure 2.1 : Core RBAC .....	6
Figure 2.2 : Hierarchical RBAC .....	7
Figure 2.3 : SSD Relations.....	8
Figure 2.4 : DSD Relations .....	8
Figure 2.5 : Model Transformation in MDA Approach.....	10
Figure 3.1 : Conceptual model of RBAC elements and constraints .....	16
Figure 3.2 : RBAC UML Profile .....	17
Figure 4.1 : Hospital Automation System Class Diagram.....	32
Figure 4.2 : RBAC UML Profile Core Components applied to problem domain .....	34
Figure 4.3 : RBAC UML Profile Hierarchical RBAC applied to problem domain...	34
Figure 4.4 : RBAC UML Profile Constrained RBAC applied to problem domain ...	37
Figure 4.5 : Platform Independent Model of the Hospital Automation System .....	38
Figure 4.6 : Ill-formed Security Model of the Hospital Automation System .....	40



## **A UML PROFILE FOR ROLE-BASED ACCESS CONTROL**

### **SUMMARY**

When building an access control aware system, domain specifications are designed typically separate from security specifications. Main reason of this separation is representing security design models as structured text like policy files on the other hand visualizing domain specifications by graphical models like Unified Modeling Language (UML) models. This causes a gap between security modeling and system design modeling. Even if security modeling is structured at the early phases of development, security mechanisms are placed in to the system at the final phases, this causes another gap in the middle. These gaps affect security and maintainability of the resulting system in a bad way.

This study presents a solution that uses Model Driven Architecture (MDA) approach for bridging these gaps. A UML Profile for Role-Based Access Control (RBAC) is proposed. With this UML Profile, access control specifications can be modeled graphically together with problem domain specifications from the beginning of the design phase, making it possible to extend security integration over the entire development process.

Major contribution of this study is introducing a UML Profile for RBAC, to integrate security specifications of access control into the development process from the beginning; to form a well-defined Platform Independent Model (PIM) that can be used to automatically generate the corresponding Platform Specific Model (PSM) or generate code directly by transformation functions; to maintain technology independence and reusability, transformation functions handle technology-specific details; to simplify the work of developers; to benefit from the advantage of wide-range of commercial and non-commercial Computer-Aided Software Engineering (CASE) tools support by using easily interchangeable and lightweight UML extension mechanism. Additional contributions are employing significant RBAC constraints like Static Separation of Duty (SSD) and Dynamic Separation of Duty (DSD) into the profile, and introducing how Object Constraint Language (OCL) is used to validate well-formedness (syntax) and meaning (semantics) of information models against the RBAC.





## ROL-TABANLI ERİŞİM DENETİMİ İÇİN BİR UML PROFİLİ

### ÖZET

Erişim denetiminin yapılacağı bir sistem oluşturulurken domen isterleri genellikle güvenlik isterlerinden ayrı olarak tasarlanır. Bu ayrımın başlıca sebebi güvenlik tasarım modelleri, ilke dosyaları gibi yapılandırılmış metin olarak ifade edilirken domen isterlerinin birleştirilmiş modelleme dili (UML) modelleri gibi grafiksel modellerle görselleştirilmeleridir. Bu ayrım güvenlik modellemesi ve sistem tasarım modellemesi arasında bir boşluk oluşmasına sebep olur. Güvenlik modellemesi, geliştirmenin erken safhalarında biçimlendirilse bile güvenlik mekanizmalarının sisteme dâhil edilmesi geliştirmenin son safhalarında yapılır. Buda geliştirmenin ara safhalarında başka bir boşluğun oluşmasına sebep olur. Bahsedilen bu boşluklar ortaya çıkan sistemin güvenliğini ve bakım kolaylığını kötü yönde etkiler.

Bu çalışmada, bahsedilen boşlukların doldurulabilmesi için model güdümlü mimari (MDA) yaklaşımıyla, Rol-Tabanlı Erişim Denetimi (RBAC) için bir UML Profili geliştirilerek çözüm önerisi getirilmiştir. Bu UML Profili, tasarım aşamasının başında erişim denetim isterlerinin domen isterleriyle birlikte grafiksel olarak modellenmesini sağlayarak güvenlik entegrasyonunun geliştirme sürecinin tamamına yayılacak şekilde yapılabilmesine olanak sağlar.

Bu çalışmanın başlıca katkısı, erişim denetimi isterlerini geliştirme sürecine en başından itibaren dâhil edebilmek; dönüşüm fonksiyonlarının otomatik olarak ilgili platforma özel model (PSM) veya direkt kod üretebilmesi için iyi tanımlanmış bir platform bağımsız model (PIM) oluşturabilmek; dönüşüm fonksiyonlarının teknolojiye özel detayları kotarabilmesi sayesinde teknoloji bağımsızlığı ve tekrar kullanılabilirliği sağlayabilmek; geliştiricilerin işlerini kolaylaştırabilmek; kolaylıkla değiş tokuş edilebilir ve hafif sıklet bir UML genişletme mekanizması kullanarak çok sayıda ticari ve ticari olmayan bilgisayar destekli sistem mühendisliği (CASE) aracının desteğinden faydalanabilmek amacıyla RBAC için bir UML Profili ortaya çıkarmaktır. Ayrıca statik görevler ayrılığı (SSD) ve dinamik görevler ayrılığı (DSD) gibi önemli RBAC kısıtlarını profile dâhil etmek ve RBAC'a dayalı modellerin biçimsel (sentaktik) ve anlamsal (semantik) olarak iyi durumda olup olmadığının denetiminin yapılabilmesi için nesne kısıt dilinin (OCL) nasıl kullanıldığını tanıtmak diğer katkılarıdır.



## **1. INTRODUCTION**

This thesis describes and illustrates the proposed UML Profile for RBAC, which is used for bridging security modeling and system design modeling. With this UML Profile, access control specifications and problem domain specifications can be designed together with graphical models and UML notations from the beginning of the design phase. The model to that the proposed UML Profile applied, forms the PIM that can be used by transformation functions to generate corresponding code or PSM. SSD, DSD, prerequisite, cardinality and time-based constraints, which are significant RBAC constraints, are employed in the UML Profile. OCL is used to express UML Profile constraints that are used to validate well-formedness and check RBAC constraint rules of the PIM.

### **1.1 Motivation**

Model building is a standard software engineering practice. Model construction during the initial phases of development process, like requirement analysis and system design, provides a foundation for early analysis of the problems and fault detection. As a result, it improves the quality of the resulting system. If the model is formal enough, it can be used to generate the corresponding code. Model building is also used for security requirements but its integration into the overall development process is problematic and suffers from two “gaps” [1]. First gap is the separation between system design modeling and security modeling. Main reason of this separation is representing security design models as structured text like policy files on the other hand visualizing domain specifications by graphical models like UML models. Even if security modeling is structured at the early phases of development, security mechanisms are placed into the system at the final phases, this causes second gap in the middle. These gaps affect security and maintainability of the resulting system in a bad way.

Access control is a security technology that is applied extensively to protect system resources against inappropriate or undesired user access. Many models have been

developed and studied to construct and manage access control systems [2][3][4]. For the last two decades, RBAC [5] has been very popular among access control systems and has been widely accepted because of its ability to reduce the complexity and cost of security administration. Under RBAC, security administration is greatly simplified by using roles, hierarchies, and constraints to organize privileges [6]. In recent years, vendors have begun implementing RBAC features in their products like database management systems, security management and network operating system products, without general agreement on the definition of RBAC features. ANSI INCITS RBAC standard [7] aims to resolve uncertainty and confusion about utility and meaning of RBAC by using a reference model to define RBAC features and then describing the functional specifications for those features. Currently, in most companies, a security administrator manually creates and manages the specification policies for RBAC systems as an independent procedure during the deployment stage after the software design and development. It is very difficult and time consuming to create these policies because of their complex syntax. A UML Profile for RBAC can solve the problem of late integration of security into the entire system development process and simplify creation of the complex policies as well.

Access control specifications could be designed graphically in a language like UML [8]. Resulting security model could be merged with system design model. UML Profiles [8][9] can be used to mark system design model elements as domain specific, here is RBAC, elements and add new building blocks for domain specific concepts. Just modeling may not be enough to make it formal for generating security infrastructure components. It may be supported by a constraint language for syntactic and semantic checking of the model, and generating access control checking mechanism of the resulting system. OCL [10] is the expression language for the UML and appropriate for this kind of support.

## **1.2 Suggested Approach**

This study proposes to use a UML Profile for RBAC to solve the issues raised in Section 1.1. Key components of the proposed UML Profile are; stereotypes, tagged values and OCL constraints. Stereotypes represent basic elements and constraints of the RBAC. Tagged values are used for defining additional attributes for constraints and making relations between profile elements. OCL constraints are used for

validating well-formedness of the system model to that this profile is applied and for checking SSD, DSD, prerequisite, cardinality and time-based constraint violations.

### 1.3 Contributions

The main contributions of this thesis are as follows:

- Introducing a UML Profile for RBAC
  - to integrate security specifications of access control into the development process from the beginning.
  - to form a well-defined PIM in MDA approach [11]. The PIM can be used by transformation functions to automatically generate the corresponding PSM or directly generate code.
  - to maintain technology independence and reusability. Access control technology will evolve or change in time but models and specifications will remain. Transformation functions handle technology-specific details. Using new or updated transformation functions will be enough to adapt to the new technology.
  - to simplify the work of developers. Developers who take part directly in the application design process can easily add security specifications to the model without expertise on security issues, and security administrators who may not understand the software structure and details of problem domain well enough do not need to define complex security policies rather they may focus on transformation functions. Transformation functions for well-known access control infrastructures may be ready to use so developers will just use them to generate code, policies or whatever needed.
  - to benefit from the advantage of wide-range of CASE-tools support. UML Profiles are lightweight and easily interchangeable UML extensions. They have a wide-range of commercial and non-commercial CASE-tools support. Therefore, users can easily deploy these extensions to their already using CASE-tools that support UML

Profile. XML Metadata Interchange (XMI) [12] format is used for the proposed UML Profile to be interchanged.

- Employing significant RBAC constraints like SSD, DSD, prerequisite, cardinality and time-based constraints into the profile.
- Introducing how OCL is used to validate well-formedness and meaning of information models against the RBAC.

#### **1.4 Thesis Outline**

The rest of this thesis is organized as follows. Section 2 reviews the background of the RBAC, the MDA approach, the UML, the UML Profile and the OCL. It also compares this proposed approach with related works. Section 3 defines the proposed UML Profile for RBAC. Section 4 introduces an example design problem and shows how the proposed UML Profile for RBAC is applied to the example design model. Section 5 presents conclusions and future work.

## 2. BACKGROUND

This section gives general background information about the RBAC, the UML, the OCL and the MDA approach. Furthermore, the proposed approach is compared with related works.

### 2.1 RBAC

Using roles to separate access control privileges was first introduced in 1992 [5]. A key feature of this study is that all access is through roles. A role is essentially a collection of permissions, and all users receive permissions only through the roles to which they are assigned. In 1996, Sandhu and colleagues [13] introduced a framework of RBAC models, *RBAC96*. The RBAC model has been widely discussed and further developed since then. In 2000, NIST initiated an effort to establish an international consensus standard for RBAC, publishing a proposal [14] in the ACM RBAC workshop. In 2004, the standard was approved as INCITS 359-2004 [7] by the InterNational Committee for Information Technology (INCITS) standards, which is accredited by the American National Standards Institute (ANSI) to develop industry consensus standards for IT.

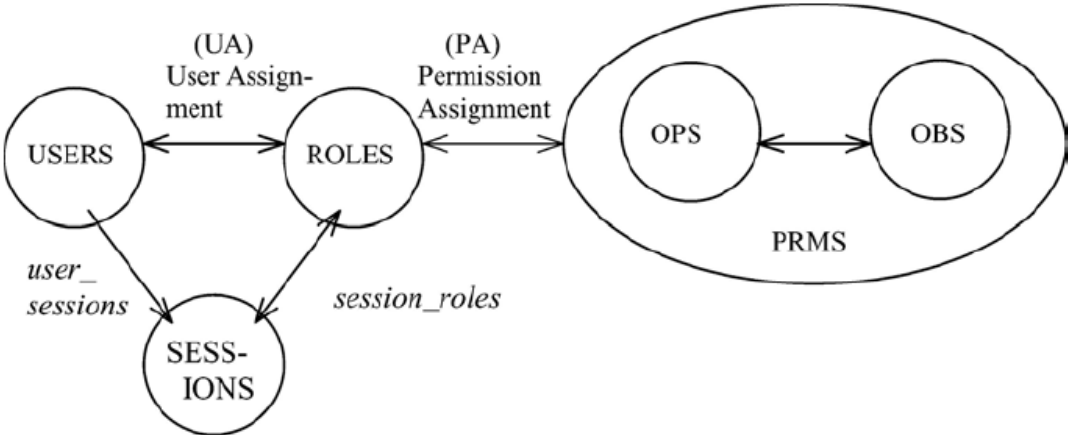
The RBAC can be stated formally using the notions of *users*, *roles*, *permissions*, *operations*, *resources* and *sessions*, and the relationships between these entities. An *operation* is an active process invoked by a *user* who wants to access protected system *resources*. *Permissions* are authorizations to perform operations on the resources. A *Role* is a job function or job title within the organization and associated with some permissions. Users grant permissions by being member of appropriate roles. This greatly simplifies management of permissions. Within an organization, roles are relatively stable, while users and permissions are both numerous and may change rapidly. If a user's responsibilities or qualifications are changed, he can be easily reassigned from one role to another. The access of users to the information is regulated based on their assigned roles. RBAC also includes the notion of user *sessions*. A user establishes a session during which he activates a subset of the roles

assigned to him. Each user can activate multiple sessions; however, each session is associated with only one user. The operations that a user can perform in a session depend on the roles activated in that session and the permissions associated with those roles. The session concept, which is a critical part of the RBAC, distinguishes RBAC from traditional group mechanisms [15]. Without sessions, all roles that are assigned to users, are always activated. This can potentially violate least privilege rule.

This study is based on the RBAC model defined in the ANSI INCITS 359-2004 standard [7]. In this RBAC standard, the RBAC model is defined in terms of four model components; Core RBAC, Hierarchical RBAC, Static Separation of Duty (SSD) Relations, and Dynamic Separation of Duty (DSD) Relations.

**2.1.1 Core RBAC**

Core RBAC defines a minimum collection of RBAC elements as defined above; users, roles, permissions, operations, resources (objects) and sessions, element sets and relations like user-role assignment and permission-role assignment, considered fundamental in any RBAC system. In addition, Core RBAC introduces the concept of role activation as part of user’s session within a computer system. Core RBAC is required in any RBAC system. Core RBAC includes sets of six basic data elements, which are defined in Figure 2.1 [7], called users (USERS), roles (ROLES), objects (OBS), operations (OPS), permissions (PRMS) and sessions (SESSIONS).

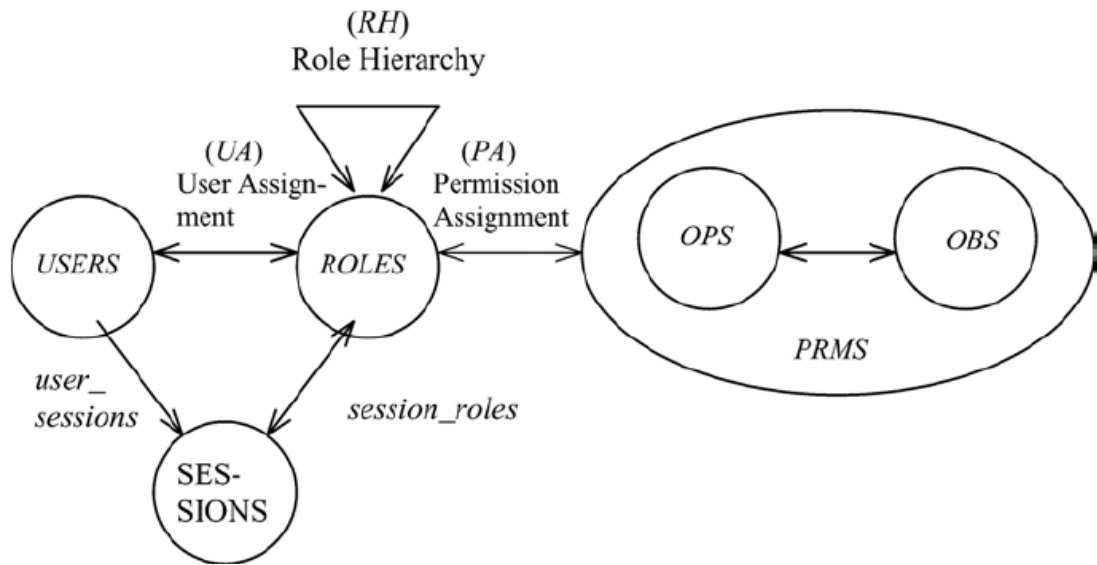


**Figure 2.1 : Core RBAC**



### 2.1.2 Hierarchical RBAC

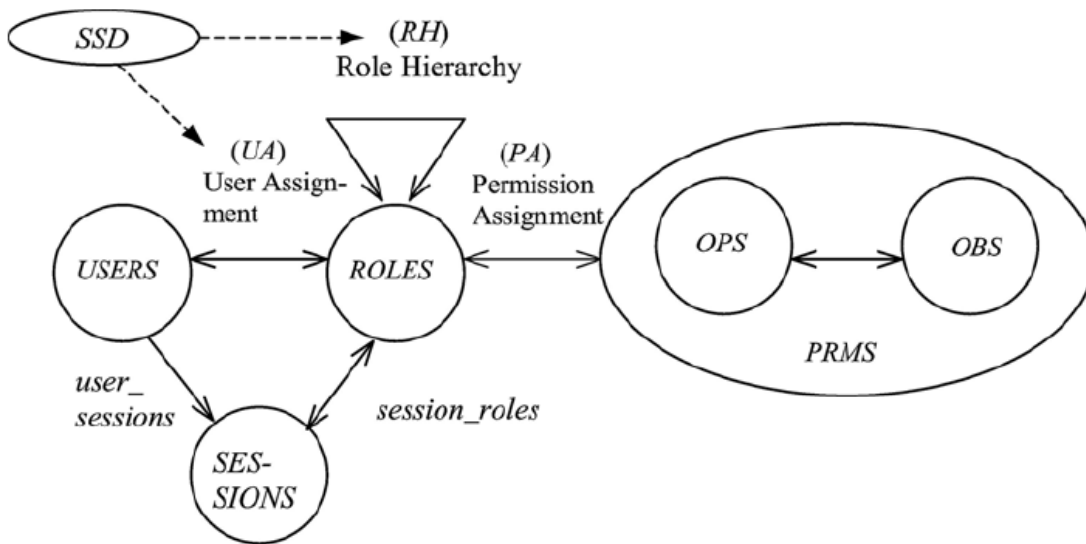
The Hierarchical RBAC component, which is indicated in Figure 2.2 [7], adds relations for supporting role hierarchies. A hierarchy is mathematically a partial order defining seniority relation between roles, whereby senior roles acquire the permissions of their juniors. In the absence of role hierarchies, it is inefficient and administratively cumbersome to specify general permissions repeatedly for a large number of roles, or to assign large numbers of users to general roles [6]. Authorized roles of a user include all assigned roles and their direct and indirect junior roles.



**Figure 2.2 : Hierarchical RBAC**

### 2.1.3 SSD Relations

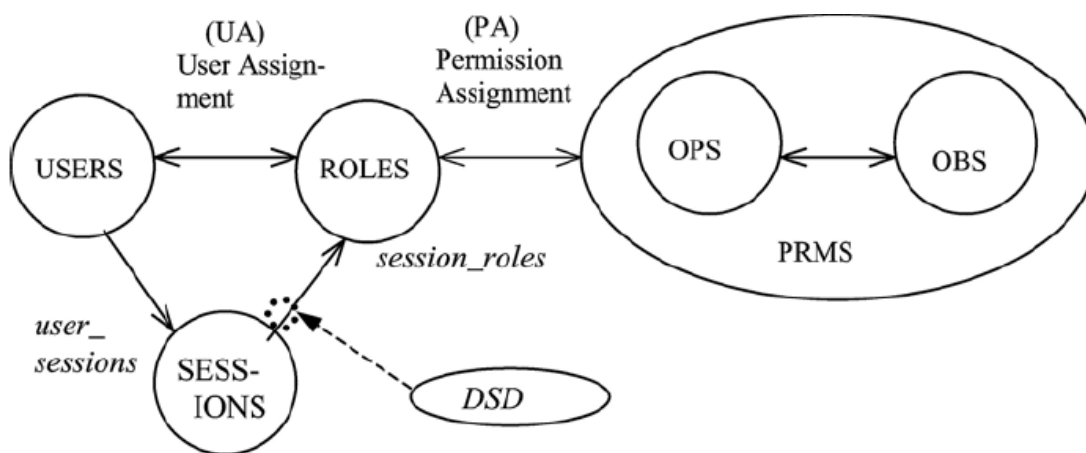
SSD Relations adds exclusivity relations among roles with respect to user assignment. Conflict of interest in a role-based system may arise because of a user gaining authorization for permissions associated with conflicting roles. One means of preventing this form of conflict of interest is through SSD, that is, to enforce constraints on the assignment of users to roles. This means that if a user is assigned to one of the conflicting roles, the user is prohibited from being member of another conflicting role. Because of the potential inconsistencies with respect to SSD relations and inheritance relations of a role hierarchy, the SSD relations model component defines relations in both the presence, as illustrated in Figure 2.3 [7], and absence of role hierarchies.



**Figure 2.3 : SSD Relations**

### 2.1.4 DSD Relations

DSD relations, as illustrated in Figure 2.4 [7], define exclusivity relations with respect to the roles that are activated as a part of a user's session. DSD relations, like SSD relations, are intended to limit the permissions that are available to a user. However, DSD relations differ from SSD relations by the context in which these limitations are imposed. SSD relations define and place constraints on a user's total permission space but DSD relations limit the availability of the permissions over a user's permission space by placing constraints on the roles that can be activated within or across a user's sessions.



**Figure 2.4 : DSD Relations**

## 2.2 MDA

The MDA is an approach to separate the specification of the operation of a system from the details of the way that system uses the capabilities of its platform [11].

MDA provides an approach for, and enables tools to be provided for:

- specifying a system independently of the platform that supports it,
- specifying platforms,
- choosing a particular platform for the system, and
- transforming the system specification into one for a particular platform.

The three primary goals of MDA are portability, interoperability and reusability through architectural separation of concerns. The keystones in MDA are the models and model elements; hence, it is important to use a well-defined modeling language, such as UML, to describe each model precisely. The aim of MDA is that a PIM (high-level model) can be transformed into a PSM (low-level model), as illustrated in Figure 2.5. Therefore, to develop software system it is only have to be designed its conceptual schema with all constraints using UML and OCL respectively.

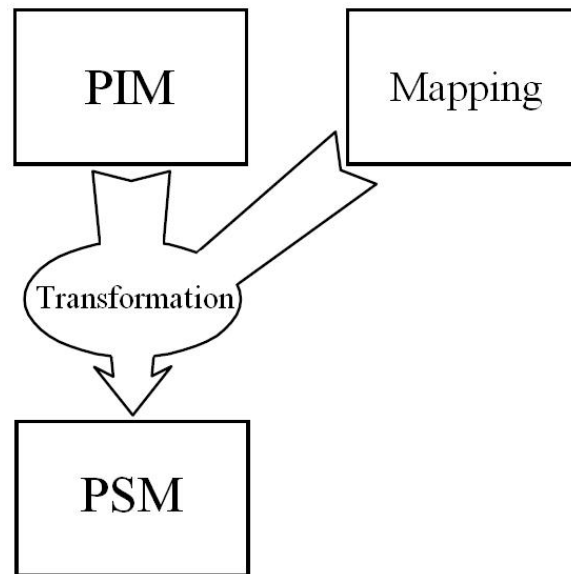
The MDA process is divided into three steps:

1. Build a PIM, that is, a conceptual model of the desired system, which is independent of any implementation technology.
2. Transform the PIM into a PSM that is based on elements and concepts of the implementation in a specific technology.
3. Transform the PSM into code. A tool might transform a PIM directly into deployable code, without producing a PSM, which means Step 2 might be skipped.

## 2.3 UML

The Unified Modeling Language (UML) [8] is a widely used graphical language for modeling object-oriented systems. It helps users to specify, visualize, construct and document the components of software systems during the design and development phase. UML supports the description of the structure and behavior of systems using different model element types and corresponding diagram types. The class diagram,

which is focused in this study, is one of these defined diagram types to provide a structural view of information in a system.



**Figure 2.5 :** Model Transformation in MDA Approach

### 2.3.1 Class Diagram

The structural aspects of systems are defined using classes; each class represents a group of things that have common services, properties, and behavior. Services are described by functions, and properties are described by attributes and associations. Every class participating in an association is connected to the association by an association end, which may also specify the role name of the class and its cardinality in the association. Classes and their relations are depicted in class diagrams.

### 2.3.2 UML Profile

UML Profile [8][9] is a kind of UML extension mechanism. It specializes some of the language's elements, imposes new restrictions on them while respecting the UML metamodel and leaving the original semantics of the UML elements unchanged. Icons and symbols can be specified for these specialized elements. The Object Management Group (OMG) maintains some common and widely accepted profiles, such as UML Profile for CORBA [16] and UML Testing Profile [17].

UML Profiles are defined in terms of three basic mechanisms: *stereotypes*, *tagged values*, and *constraints* [9] that allow tailoring it to fit the needs of a specific domain. A stereotype defines how an existing metaclass may be extended. It can be used to create platform or domain specific terminology or notation in addition to, or in place

of, the ones used for the extended metaclass. A tagged value is an additional meta-attribute that is attached to a metaclass extended by a stereotype. A tagged value has a name and a type, and is member of a specific stereotype. Constraints are expressed in OCL or natural language and can be associated with stereotypes. They impose restrictions on the corresponding metamodel elements. In this way, the properties of a well-formed model can be defined.

## 2.4 OCL

Object Constraint Language (OCL) [10], which is part of the UML, is used to express constraints and properties of model elements in a formal way. OCL, which is based on first-order logic, is a textual language that describes constraints on the UML model with expressions. These expressions typically specify invariant conditions that must hold for the system being modeled or queries over objects described in a model. Note that when the OCL expressions are evaluated, they do not have side effects, which means their evaluation cannot alter the state of the corresponding executing system. Expressions can be used in a number of places in a UML model:

- to specify the initial value of an attribute or association end
- to specify the derivation rule for an attribute or association end
- to specify the body of an operation
- to indicate an instance in a dynamic diagram
- to indicate a condition in a dynamic diagram
- to indicate actual parameter values in a dynamic diagram

There are four types of constraints defined in OCL, shown in Table 2.1.

In OCL expressions, there can be used

- basic types like Integer, Real, String and Boolean;
- basic operations that can be used with the basic types, like mathematical operations, string operations and Boolean operations;

- collections that are structured data types that allow encapsulating more than one element of a same type inside, like Set, Bag, OrderedSet and Sequence;
- operations on collections like select, reject, collect, forAll, exists, iterate and any.

**Table 2.1 : OCL Constraints Types**

<b>OCL Constraints Type</b>	<b>Description</b>
Invariant	An invariant object normally attaches with a class diagram. It is a constraint that states additional rules that must always be obeyed by all objects of the class, type or interface that are defined in the class diagram.
Precondition	A precondition is used to restrict a condition that must be true before an operation executes.
Post-condition	A post-condition is used to restrict a condition that must be true after an operation executes.
Guard	A guard is used to restrict a condition that must be true before a transition in a state machine happens.

A Set is a container where each element inside appears only one time. Therefore, it does not contain duplicate elements. A Bag is like a Set but with duplications allowed. Moreover, OrderedSet and Sequence are the same as Set and Bag in which the elements are ordered.

## 2.5 Related Works

In literature, there are some studies about visualizing RBAC elements and constraints in UML [18][19] and OCL representation of RBAC constraints [20][21]. None of these studies points out a UML Profile that can be used for the both purposes. Basin D. et al. [1] propose an approach, Model Driven Security (MDS), to build secure systems. This approach is very close to the approach that is used in this study, in bridging “gaps” between security models and system design models but they define a new modeling language directly using Meta-Object Facility (MOF) [22]. In this approach, developers should use an extra tool besides their modeling tool, or leave it at all. In other words, a new modeling language requires a new CASE-tool. In contrast, UML Profile is a lightweight UML extension and does not require an extra

tool. Developers can easily deploy a UML Profile to their CASE-tools. Moreover, in MDS, it is not mentioned how to employ RBAC constraints [7][20] into the model. Another close approach to this subject was proposed by Jin X. [23]. She proposes a framework to provide support for modeling the RBAC system in eXtensible Access Control Markup Language (XACML) [24] architecture and automatic generation of policy specification in XACML format. This study uses UML Profile mechanism to integrate security to system development cycle but the profile contains both RBAC elements and XACML elements like Rule and Policy, and only contains SSD RBAC constraint. In contrast to this study, the proposed approach is for just RBAC elements and includes critical RBAC constraints like SSD, DSD, prerequisite, cardinality and time-based constraints. XACML could be a PSM and developers may not know about XACML elements. Developers can design just PIM with the proposed UML Profile for their problem domain. Transformation functions can handle this kind of platform specific details if it is needed.





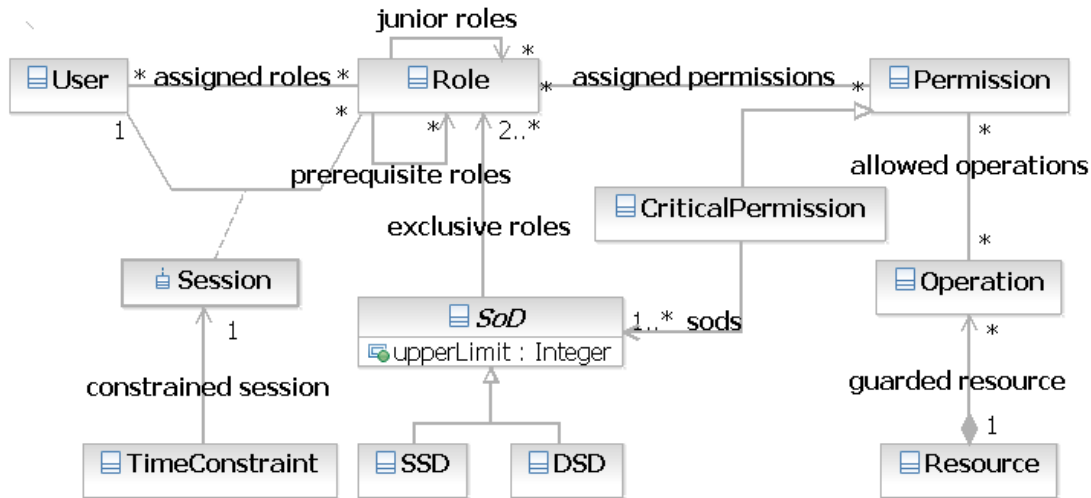
### 3. RBAC UML PROFILE

The proposed UML Profile for RBAC has stereotypes, tagged values and constraints to define a way of modeling RBAC elements and constraints.

#### 3.1 Conceptual Model

The proposed UML Profile includes all elements of the four model components of RBAC, which is mentioned in Section 2.1. Figure 3.1 shows conceptual model of the standard RBAC elements, constraints and their relations, and some additional elements; *prerequisite roles*, *time-based constraint* and *critical permission*. A role can have one or more prerequisite roles means a user can be assigned to this role only if the user is already authorized for those prerequisite roles. Time-based constraints are used to restrict sessions to be established in only allowed time intervals. Operations of a critical task are divided over roles, exclusive roles, by assigning these operations to critical permissions. Every exclusive role in a Separation of Duty (SoD) relation should be assigned to at least one critical permission for that critical task. A critical permission can be assigned to only one role. Critical permissions could not be shared among roles so it guarantees consistency of user-exclusive role assignment and exclusive role-permission assignment, for more information look at [6] p. 107-117. As shown in Figure 3.1, the relations between users and roles, permissions and roles, and operations and permissions are many-to-many. A resource can have one or more operations. A user can establish multiple sessions and can activate one or more authorized roles in a session. A role can have one or more junior roles for the role inheritance relations. A role can have one or more prerequisite roles for the prerequisite constraint. An *SoD* has at least two exclusive roles. SSD and DSD generalize SoD. Upper limit property of the *SoD* kind element is a natural number  $\geq 2$  that no user is assigned to this much or more roles (for SSD), no user can activate in a session this much or more roles (for DSD), included in the exclusive roles set. A *CriticalPermission*, which is a

Permission, has at least one *SoD* element shows for which critical task it is created. A TimeConstraint has a property for session it constraints.



**Figure 3.1 :** Conceptual model of RBAC elements and constraints

### 3.2 Proposed UML Profile for RBAC

The proposed UML Profile can be expressed in three parts; RBAC Core Components, Hierarchical RBAC and Constrained RBAC, to represent four model components of the RBAC standard and additional RBAC constraints. All three parts of the UML Profile has stereotypes, tagged values and constraints. All these stereotypes and their relations are shown in Figure 3.2.

RBAC Core Components part of the proposed UML Profile contains User, Role, Resource, Operation, Permission, Session, ResourceAssignment, UserAssignment and PermissionAssignment stereotypes that are described in Section 3.2.1-3.2.9. Hierarchical RBAC part contains RoleInheritance stereotype that is described in Section 3.2.10. Constrained RBAC part contains SoD, SSD, DSD, CriticalPermission and TimeConstraint stereotypes that are described in Section 3.2.11-3.2.15.

There are OCL expressions that are embedded as owned rules in its constrained stereotype in the UML Profile, in the tables of stereotype definitions. Those expressions use some general OCL definitions that are described in Section 3.3.

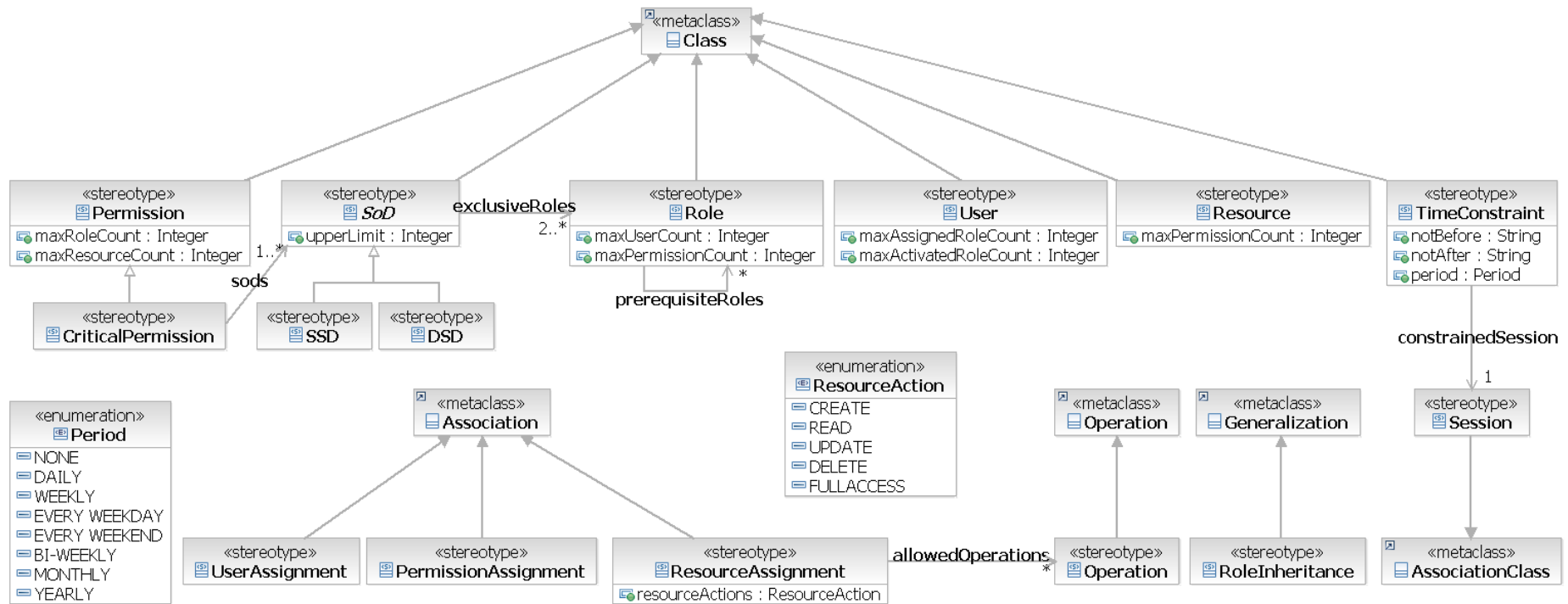


Figure 3.2 : RBAC UML Profile

### 3.2.1 User Stereotype

It represents a human being, machine, network or anything that want to access system resources.

**Icon :** 

**Base Class :** UML::Class

**Parent :** N/A

**Tagged Values :**

- `maxAssignedRoleCount` : Integer [1] = -1: maximum number of roles that can be assigned to this user.
- `maxActivatedRoleCount` : Integer [1] = -1: maximum number of roles that can be activated by this user in a session.

**Constraints :**

[1] Number of the assigned roles of a user should not exceed its `maxAssignedRoleCount` value.

**inv maxAssignedRoleCount :**

`self.isUser` implies `self.asUser.maxAssignedRoleCount > -1` implies  
`self.assignedUserRoles(self)->size() <= self.asUser.maxAssignedRoleCount`

[2] Number of the activated roles in a session should not exceed session owner's `maxActivatedRoleCount` value.

**inv maxActivatedRoleCount :**

`self.isUser` implies `self.asUser.maxActivatedRoleCount > -1` implies  
`self.establishedUserSessions(self)->forAll(endType->select(isRole)->size() <= self.maxActivatedRoleCount)`

### 3.2.2 Role Stereotype

It represents a job function or job title within the context of an organization.

**Icon :** 

**Base Class :** UML::Class

**Parent :** N/A

### Tagged Values :

- `maxPermissionCount` : Integer [1] = -1: maximum number of permissions that can be assigned to this role.
- `maxUserCount` : Integer [1] = -1: maximum number of users that can be assigned to this role.
- `prerequisiteRoles` : Role [\*]: prerequisite roles for this role.

### Constraints :

[1] Number of the assigned permissions of a role should not exceed its `maxPermissionCount` value.

#### **inv maxRolePermissionCount :**

`self.isRole` implies `self.asRole.maxPermissionCount > -1` implies  
`self.assignedRolePermissions(self)->size() <= self.asRole.maxPermissionCount`

[2] Number of the assigned users of a role should not exceed its `maxUserCount` value.

#### **inv maxUserCount :**

`self.isRole` implies `self.asRole.maxUserCount > -1` implies  
`self.assignedRoleUsers(self)->size() <= self.asRole.maxUserCount`

[3] All generalizations of a Role should be RoleInheritance stereotyped generalizations.

#### **inv inheritanceShouldBeRoleInheritance:**

`self.isRole` implies `self.generalization->forall(isRoleInheritance)`

[4] `PrerequisiteRoles` tagged value should not include the owner Role stereotyped class.

#### **inv prerequisiteSelfContain :**

`self.isRole` implies `not self.asRole.prerequisiteRoles->iterate(r; res: Set(Class)=Set{} | res->including(r.base_Class))->includes(self)`

[5] Roles that are included in `prerequisiteRoles` tagged value, and their all parents which means its direct and indirect ancestors, and the tagged value's owner role and its all parents should not form a role set that violates the SSD constraint.

#### **inv prerequisiteSSDConsistency :**

`self.isRole` implies `self.asRole.prerequisiteRoles->notEmpty()` implies let  
`authorisedRequiredRoles : Set(Class) = allFamily(prerequisites(self)->union(Set{self.base_Class}))` in `self.allSSDs->forall(ssd | sodRoles(ssd)->intersection(authorisedRequiredRoles)->size() < ssd.extension_SoD.upperLimit)`

[6] If a CriticalPermission is assigned to a role, this role should be included in separatedRoles tagged value of all SoD elements that are specified in sods tagged value of that CriticalPermission.

**inv shouldBelInSoD :**

```
self.isRole implies assignedRoleCriticalPermissions(self)->forall(cp |
cp.asCriticalPermission.sods->forall(cps : RBAC::SoD |
sodRoles(cps.base_Class)->includes(self)))
```

### 3.2.3 Resource Stereotype

It represents an object that must be protected against inappropriate or undesired access.

**Icon :** 

**Base Class :** UML::Class

**Parent :** N/A

**Tagged Values :**

- maxPermissionCount : Integer [1] = -1: maximum number of permissions that can be assigned to this resource.

**Constraints :**


[1] Number of the assigned permissions of a resource should not exceed its maxPermissionCount value.

**inv maxResourcePermissionCount :**

```
self.isResource implies self.asResource.maxPermissionCount>-1 implies
self.assignedResourcePermissions(self)->
size()<=self.asResource.maxPermissionCount
```

### 3.2.4 Operation Stereotype

It is used to mark functions of the Resource stereotyped classes as protected. Functions are used to access information of the class or make changes on the state of the system, which makes them ideal to represent Operations on Resources in RBAC context.

**Icon :** 

**Base Class :** UML::Operation

**Parent :** N/A

**Tagged Values :** N/A

**Constraints :**

[1] Owner class of the Operation stereotyped function should be Resource stereotyped class.

**inv operationEncloser :**

self.isOperation implies self.owner.isResource

### 3.2.5 Permission Stereotype

It represents an approval to perform operations on protected resources.

**Icon :** 

**Base Class :** UML::Class

**Parent :** N/A

**Tagged Values :**

- maxResourceCount : Integer [1] = -1: maximum number of resources that can be assigned to this permission.
- maxRoleCount : Integer [1] = -1: maximum number of roles that can be assigned to this permission.

**Constraints :**

[1] Number of the assigned resources of a permission should not exceed its maxResourceCount value.

**inv maxResourceCount :**

self.isPermission implies  
self.asPermission.maxResourceCount > -1 implies  
self.assignedPermissionResources(self)->  
size() <= self.asPermission.maxResourceCount

[2] Number of the assigned roles of a permission should not exceed its maxRoleCount value.

**inv maxRoleCount :**

self.isPermission implies self.asPermission.maxRoleCount>-1 implies  
self.assignedPermissionRoles(self)->size()<=self.asPermission.maxRoleCount

### 3.2.6 Session Stereotype

It is established by a user to activate his one or more authorized roles.

**Icon :** 

**Base Class :** UML::AssociationClass

**Parent :** N/A

**Tagged Values :** N/A

**Constraints :**

[1] It should associate a User stereotyped class with one or more Role stereotyped classes.

**inv user\_session\_roles :**

self.isSession implies self.endType->one(isUser) and self.endType->  
exists(isRole) and self.endType->forall(isUser or isRole)

[2] Activated roles and their all parents should not form a role set that violates the DSD constraint.

**inv dsdRule :**

self.isSession implies let sessionAuthorisedRoles : Set(Class) =  
allFamily(activeSessionRoles(self)) in self.allDSDs->forall(dsd |  
sessionAuthorisedRoles->intersection(sodRoles(dsd))->  
size()<dsd.extension\_SoD.upperLimit)

[3] In order to activate a role in a session, User who establishes the session must be authorized for that role.

**inv userAssignedRolesActivation :**

self.isSession and sessionUser(self)<>null implies  
allFamily(assignedUserRoles(sessionUser(self)))->  
includesAll(allFamily(activeSessionRoles(self)))



### 3.2.7 ResourceAssignment Stereotype

It is used to assign a resource to a permission.

**Icon :** N/A

**Base Class :** UML::Association

**Parent :** N/A

**Tagged Values :**

- allowedOperations : Operation [\*]: includes Operation stereotyped functions belong to Resource end of the association. A user who is authorized for the role that is assigned to Permission end of this association, grants right to execute these functions.
- resourceActions : ResourceAction [\*]: includes one or more ResourceAction enumeration values:
  - READ: execute all Operation stereotyped, getter functions of attributes and association ends, and side-effect free functions that do not change the state of the system when they are executed, of the Resource.
  - UPDATE: execute all Operation stereotyped, setter functions of attributes and association ends, and non-side-effect free functions of the Resource.
  - CREATE: execute constructor function of the Resource.
  - DELETE: execute destructor function of the Resource.
  - FULLACCESS: all CREATE, READ, UPDATE and DELETE rights.

**Constraints :**

[1] It should associate a Resource stereotyped class with a Permission stereotyped class.

**inv permission\_resource :**

self.isResourceAssignment implies self.endType->exists(isPermission) and self.endType->exists(isResource)

[2] allowedOperations tagged value should include Operation stereotyped functions that are owned by Resource end of the association.

**inv allowedOperationsOwner :**

self.isResourceAssignment implies let resource : Type = self.endType->  
 any(isResource) in resource<>null implies self.asResourceAssignment.  
 allowedOperations->forAll(op | op.owner=resource)

[3] Both allowedOperations and resourceActions tagged values could not be empty.

At least one of them must contain some elements.

**inv hasOperations :**

self.isResourceAssignment implies  
 self.asResourceAssignment.allowedOperations->notEmpty() or  
 self.asResourceAssignment.resourceActions->notEmpty()

**3.2.8 UserAssignment Stereotype**

It is used to assign a user to a role.

**Icon :** N/A

**Base Class :** UML::Association

**Parent :** N/A

**Tagged Values :** N/A

**Constraints :**

[1] It should associate a User stereotyped class with a Role stereotyped class.

**inv role\_user :**

self.isUserAssignment implies self.endType->exists(isRole) and self.endType->  
 exists(isUser)

[2] User end of the association should be already authorized for prerequisite roles of the Role end, and prerequisite roles of all parents of this Role end.

**inv prerequisiteRule :**

self.isUserAssignment implies let user : Type = endType->any(isUser), role :  
 Type = endType->any(isRole) in (user<>null and role<>null) implies let allPrs :  
 Set (Class) = allFamily(Set{role})->collect(r | prerequisites(r))->asSet() in allPrs->  
 notEmpty() implies authorisedRoles(user)->includesAll(allPrs)

[3] Assignment of a user to a role should not cause violation of the SSD constraint.

**inv ssdRule :**

self.isUserAssignment implies let user : Type = endType->any(isUser), role :  
 Type = endType->any(isRole) in (user<>null and role<>null) implies self.allSSDs  
 ->forAll(ssd | sodRoles(ssd)->includes(role.oclAsType(Class)) implies  
 assignedUserRoles(user)->intersection(sodRoles(ssd))->  
 size()<ssd.extension\_SoD.upperLimit)

### 3.2.9 PermissionAssignment Stereotype

It is used to assign a permission to a role.

**Icon :** N/A

**Base Class :** UML::Association

**Parent :** N/A

**Tagged Values :** N/A

**Constraints :**

[1] It should associate a Permission stereotyped class with a Role stereotyped class.

**inv role\_permission :**

self.isPermissionAssignment implies self.endType->exists(isRole) and  
self.endType->exists(isPermission)

### 3.2.10 RoleInheritance Stereotype

It represents a hierarchy between two roles. General end is junior role and specific end is senior role. Senior role inherits all assigned permissions of the junior role. Users, who are assigned to this senior role, grant these inherited permissions via the generalization.

**Icon :** N/A

**Base Class :** UML::Generalization

**Parent :** N/A

**Tagged Values :** N/A

**Constraints :**

[1] Both general end and specific end of the generalization should be Role stereotyped classes.

**inv role\_role :**

self.isRoleInheritance implies self.general.isRole and self.specific.isRole

[2] It should not cause an inheritance cycle. All parents of the junior role should not include the senior role.

**inv inheritanceCycle :**

self.isRoleInheritance implies not self.general.allParents->select(isRole)->  
includes(self.specific)

[3] RoleInheritance should not cause violation of the SSD constraint.

**inv roleInheritanceSSDRule :**

```
self.isRoleInheritance implies Class.allInstances()->forall(u | u.isUser implies let
  generalFamily : Set(Class) = allFamily(Set {self.general}) in authorisedRoles(u)
->includesAll( generalFamily->union(Set {self.specific})) implies self.allSSDs->
forall(ssd | sodRoles(ssd)-> intersection(generalFamily)->size())>0 implies
authorisedRoles(u)-> intersection(ssdRoles(ssd))->
size())<ssd.extension_SoD.upperLimit))
```

### 3.2.11 SoD Stereotype

It is an abstract stereotype for Separation of Duties that is a fundamental requirement for critical tasks. A critical task should not be completed by a single user in SSD context. Operations of a critical task should not be performed in the same session in DSD context.

**Icon :** N/A

**Base Class :** UML::Class

**Parent :** N/A

**Tagged Values :**

- separatedRoles : Role [2:\*]: includes roles that permissions of a critical task are divided among them.
- upperLimit : Integer [1] = 2: a natural number  $\geq 2$  with the property that,
  - in SSD context, no user is assigned to
  - in DSD context, no user may activate in the same sessionthis much or more roles included in separatedRoles tagged value.

**Constraints :**

[1] upperLimit tagged value should be a natural number between 2 and the size of the separatedRoles array.

**inv allowedRolesUpperLimit :**

```
self.isSoD implies self.asSoD.upperLimit>=2 and
self.asSoD.upperLimit<=self.asSoD.separatedRoles->size()
```

[2] Each Role that is included in separatedRoles tagged value should be assigned to at least one CriticalPermission that has sods tagged value includes this SoD element.

**inv criticalTaskDividedToRoles :**

```
self.isSoD implies sodRoles(self)->forall(role |
assignedRoleCriticalPermissions(role)->exists(cp |
cp.asCriticalPermission.sods.base_Class->includes(self)))
```

**3.2.12 SSD Stereotype**

It is used to ensure that roles in an SSD relationship have no common user assigned. SSD constraints provide reduced risk and fraud, and increased opportunity for detecting errors, since two or more parties are involved in completing a transaction.

**Icon :** 

**Base Class :** UML::Class

**Parent :** SoD

**Tagged Values :** N/A

**Constraints :** N/A

**3.2.13 DSD Stereotype**

It is used to ensure that roles in a DSD relationship are not activated in the same session.

**Icon :** 

**Base Class :** UML::Class

**Parent :** SoD

**Tagged Values :** N/A

**Constraints :** N/A

**3.2.14 CriticalPermission Stereotype**

It is assigned to some protected operations of one or more critical tasks. It is a kind of permission that can be assigned to only one role.

**Icon :** 

**Base Class :** UML::Class

**Parent :** Permission

**Tagged Values :**

- sods : SoD [1:\*]: includes SoD kind stereotyped classes to specify for which SoD relations, this permission is created as critical.

**Constraints :**

[1] sods tagged value should not be empty.

**inv emptySoDs :**

self.isCriticalPermission implies self.asCriticalPermission.sods->notEmpty()


[2] CriticalPermission could be assigned to only one role. It should not be shared among roles.

**inv onlyOneRole :**

self.isCriticalPermission implies self.assignedPermissionRoles(self)->size()<=1

### 3.2.15 TimeConstraint Stereotype

TimeConstraint is used to restrict a Session to be established in only allowed time intervals. If a Session is restricted by more than one TimeConstraint, it can be established when at least one of these constraints is valid at the establishment time.

**Icon :** 

**Base Class :** UML::Class

**Parent :** N/A

**Tagged Values :**

- constrainedSession : Session [1]: a Session stereotyped class that is wanted to be restricted.
- notBefore : String [0:1]: a String value that represents a time. Before this time, constrainedSession cannot be established.
- notAfter : String [0:1]: a String value that represents a time. After this time, constrainedSession cannot be established.
- period : Period [1] = NONE: includes one of the Period enumeration values; NONE, DAILY, WEEKLY, EVERY WEEKDAY, EVERY WEEKEND,

BI-WEEKLY, MONTHLY and YEARLY. At these periods within notBefore and notAfter time interval, constrainedSession can be established.

**Constraints :** N/A

### **3.3 OCL Expressions for Profile Constraints**

OCL invariant expressions are used for defining profile constraints. Each constraint, which is defined in natural language and in OCL Expression on stereotype elements of the RBAC UML Profile (see Section 3.2 stereotype definitions), corresponds to an OCL expression that is embedded in its constrained stereotype element as owned rule in the profile. Each stereotype element is the context for the OCL Expressions of its owned rules. These OCL expressions are used to validate models if the model is well-formed and does not violate any RBAC constraints. OCL expressions are created by considering the role inheritance. All defined OCL expressions can be found in the XMI format of the proposed UML Profile for RBAC [25]. APPENDIX A.1 shows global OCL definitions that are used in OCL expressions for the UML Profile constraints.





## **4. EXAMPLE DESIGN PROBLEM**

In this section, it is introduced a design problem along with its access control requirements, as an example for how the proposed UML Profile for RBAC can be applied to the design model.

### **4.1 Problem Domain Requirements**

It is considered developing a subset of the requirements of a system for hospital automation. In this system, there are doctors, nurses, patients and an external system, pharmacy system as actors. Each Patient is assigned to one Doctor and one or more Nurses. A Doctor can be assigned to more than one Patient. Doctor diagnoses diseases of the Patient and creates a medicine order. Nurse can list the medicine orders. She selects an order from the list and then picks medicines that are listed in that order, from the medicine dispenser. Nurse gives the Patient medications at the times specified in the order. External Pharmacy System is responsible for listing the medicine orders and loading medicines to the medicine dispenser if it is necessary.

### **4.2 Access Control Requirements**

As the thesis proceeds, it will be seen how to formalize a design model for this system along with the following access control demands.

1. Loading medicines to medicine dispenser, creating medicine order and picking medicines from the medicine dispenser are operations forming a critical task. This critical task should not be performed by a single user.
2. A user should already grant diagnosing right to create medicine orders.
3. Only one user, here is the Pharmacy System, can load medicines to the medicine dispenser.
4. Pharmacy System may get reports and status information of the medicine dispenser. However, it should not perform these operations when it is loading medicines to the medicine dispenser.

5. Pharmacy System can load medicines to the dispenser only at a time interval 12:00-13:00 in a day.
6. Only Doctors and Nurses can read patient records. Pharmacy System should not read patient records even if it can read patient medicine orders.

Five of these access control demands are samples of RBAC constraints that are mentioned in the standard [7]; static (1) and dynamic (4) separation of duties, and constraints that are mentioned in [20]; prerequisite (2), cardinality (3) and time-based (5) constraints. Remaining one (6) is sample of the user assignment to a role.

### 4.3 System Design Model

While the UML already provides standards for the design of this system in general, it does not provide everything necessary for the design of access control specifications. Classes [8] can be used for defining the structural aspects of this system. Each class formalizes a set of objects with common services, properties and behavior. Figure 4.1 represents the structural aspect of the problem domain but does not include the security aspect of the system for access control specifications. Lack of the security elements in this class diagram is the problem. A UML Profile can solve this problem.

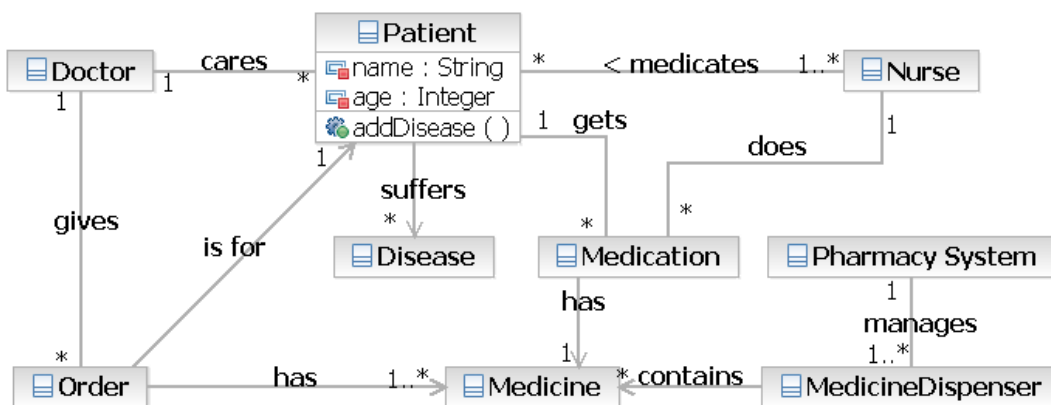


Figure 4.1 : Hospital Automation System Class Diagram

### 4.4 Security Model

Core RBAC elements and their relations for the problem domain can be defined by analyzing problem domain requirements. Core component stereotypes of the proposed UML Profile for RBAC can be used to mark design elements of the problem domain as RBAC core elements.

Doctor, Nurse and PharmacySystem are *Users* of the system. Order, Patient and MedicineDispenser are *Resources* that are required to be protected against inappropriate or undesired user access. There are some *Operations* on the protected resources, adding disease, applying medicines and reading on the Patient; reading, creating and deleting on the Order; dispensing medicines, loading medicines and getting reports and status information on the MedicineDispenser. These operations should be assigned to some *Permissions* so they can be performed by users via *Roles*. Therefore, some permissions and roles should be defined.

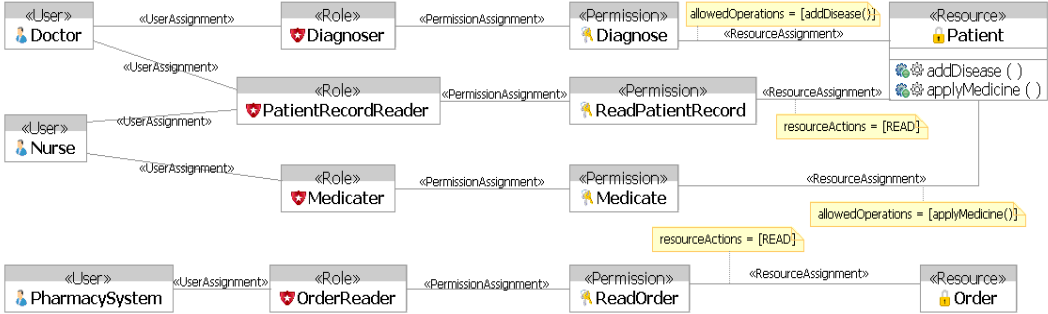
Adding disease can be assigned to Diagnose permission that can be assigned to Diagnoser role. Applying medicines can be assigned to Medicate permission that can be assigned to Medicater role. Reading patient records can be assigned to ReadPatientRecord permission that can be assigned to PatientRecordReader role. Reading order can be assigned to ReadOrder permission that can be assigned to OrderReader role. Creating and deleting order can be assigned to CreateOrder permission that can be assigned to OrderCreator role. Dispensing medicines can be assigned to Dispense permission that can be assigned to the Medicater role. Loading medicines can be assigned to LoadMedicine permission that can be assigned to MedicineLoader role. Getting reports and status information can be assigned to ManageDispense permission that can be assigned to DispenserManager role.

All operation-permission assignments are done by creating relations between corresponding Permissions and Resources that enclose the Operations, with ResourceAssignment. The Operations that are functions of the Resource stereotyped classes, are Operation stereotyped elements. AllowedOperations tagged value of the ResourceAssignments should contain the required Operations. Reading, creating and deleting are Resource actions so appropriate ResourceAction enumeration values should be included in the resourceActions tagged value of the corresponding ResourceAssignments.

Doctor as a user, should be assigned to Diagnoser, PatientRecordReader and OrderCreator roles to fulfill his job. Nurse should be assigned to Medicater and PatientRecordReader roles. PharmacySystem should be assigned to MedicineLoader, DispenserManager and OrderReader roles.

All assignments that are mentioned above are done with appropriate association stereotypes of core components of the proposed UML Profile. Figure 4.2 illustrates

how the core components of the UML Profile for RBAC can be applied to domain model of the hospital automation system. As the thesis proceeds, missing parts will be added to the model.



**Figure 4.2 :** RBAC UML Profile Core Components applied to problem domain

In this example system, it is desired that OrderCreator and Medicater roles should have ReadOrder permission. Instead of assigning this permission to both roles, they may inherit this permission over OrderReader role. This inheritance relation, which is depicted in Figure 4.3, could be created with RoleInheritance Stereotype.



**Figure 4.3 :** RBAC UML Profile Hierarchical RBAC applied to problem domain

In order to model access control requirements of the example system, constrained RBAC elements of the proposed UML Profile can be used.

Access control requirement (1) mentions a critical task that is formed by loading medicine, creating medicine order and dispensing medicine operations. Permissions that are assigned to these operations, should be created as CriticalPermissions and roles that are assigned to these permissions, should be in an SSD relation. Therefore,

LoadMedicine, Dispense and CreateOrder should be CriticalPermissions and MedicineLoader, Medicater, and OrderCreator roles should be included in separatedRoles tagged value of an SSD stereotyped element. For this critical task, MedicineSSD is created and OrderCreator, Medicater and MedicineLoader roles are added to the separatedRoles tagged value of this element. It is assumed that, all roles in this SSD relation are required to be assigned to different users so upperLimit tagged value of MedicineSSD should be 2 which means no user can be assigned to 2 or more roles that are included in the separatedRoles. CriticalPermissions that are mentioned above should contain MedicineSSD in their sods tagged value.

Access control requirement (2) points out that diagnosing is a prerequisite for creating medicine orders. Prerequisite relations are for Roles in RBAC context. Consequently, prerequisiteRoles tagged value of the OrderCreator role that grants right to perform creating medicine orders operation over CreateOrder permission, should include the Diagnoser role that grants right to perform diagnosing operation over Diagnose permission.

Access control requirement (3) is about cardinality constraint for loading medicines to the medicine dispenser operation. MedicineLoader role that grants right to perform this operation over LoadMedicine permission, should have maxUserCount tagged value is set to 1 to cover this requirement. By this way, this role cannot be assigned to another user while it is already assigned to a user, PharmacySystem.

Access control requirement (4) also mentions a critical task that is formed by loading medicine and, getting reports and status information of the MedicineDispenser. This constraint is about DSD, means these operations should not be performed in the same session. MedicineLoader and DispenserManager roles that grant rights to perform operations forming this critical task, should be in a DSD relation. Therefore, LoadMedicine and ManageDispense should be CriticalPermissions and MedicineLoader and DispenserManager roles should be included in separatedRoles tagged value of a DSD stereotyped element. For this critical task, PharmacyDSD is created and MedicineLoader and DispenserManager roles are added to the separatedRoles tagged value of this element. It is assumed that, the roles in this DSD relation are required to be activated in different sessions so upperLimit tagged value of PharmacyDSD should be 2 which means no user can activate 2 or more roles that are included in the separatedRoles in the same session. LoadMedicine and

ManageDispense CriticalPermissions should contain PharmacyDSD in their sods tagged value.

Access control requirement (5) refers a time constraint on loading medicines operation so a TimeConstraint stereotype should be applied to an element of the model. MedicineLoadConstraint is created for this purpose. It should constraint a session in that MedicineLoader role that grants right to perform the loading medicines operation, is activated. Therefore, MedicineLoadSession association class element to that Session stereotype is applied, is created. It associates PharmacySystem user with MedicineLoader role. This session should be established only at a time interval 12:00-13:00 in a day so tagged values of the MedicineLoadConstraint should be set like that constrainedSession is set to the MedicineLoadSession, notBefore tagged value is set to 12:00, notAfter tagged value is set to 13:00 and period tagged value is set to DAILY period enumeration value.

Access control requirement (6) is satisfied by assigning Doctor and Nurse users, not PharmacySystem user, to the PatientRecordReader role that grants right to reading patient records over ReadPatientRecord permission.

Figure 4.4 illustrates how the Constrained RBAC elements of the UML Profile for RBAC can be applied to domain model of the hospital automation system to fulfill the access control requirements.

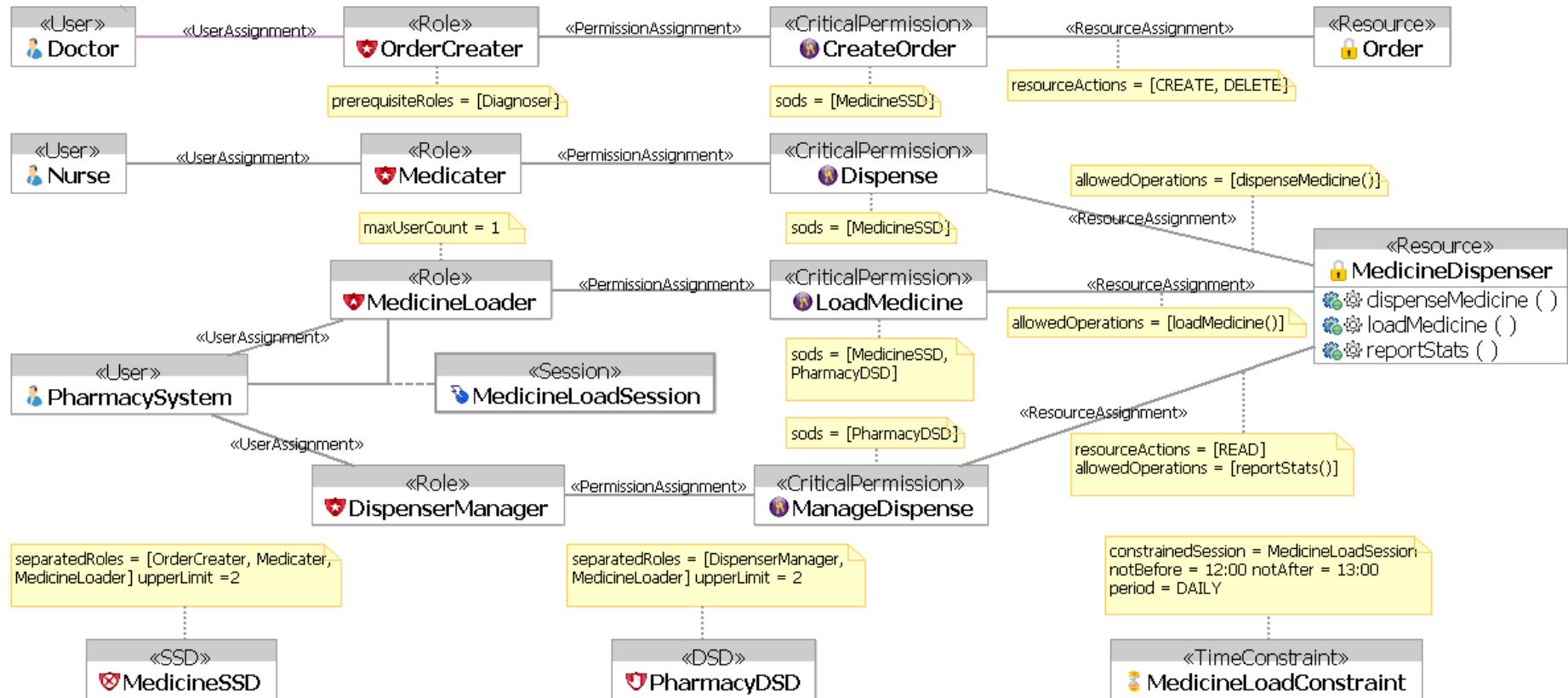


Figure 4.4 : RBAC UML Profile Constrained RBAC applied to problem domain

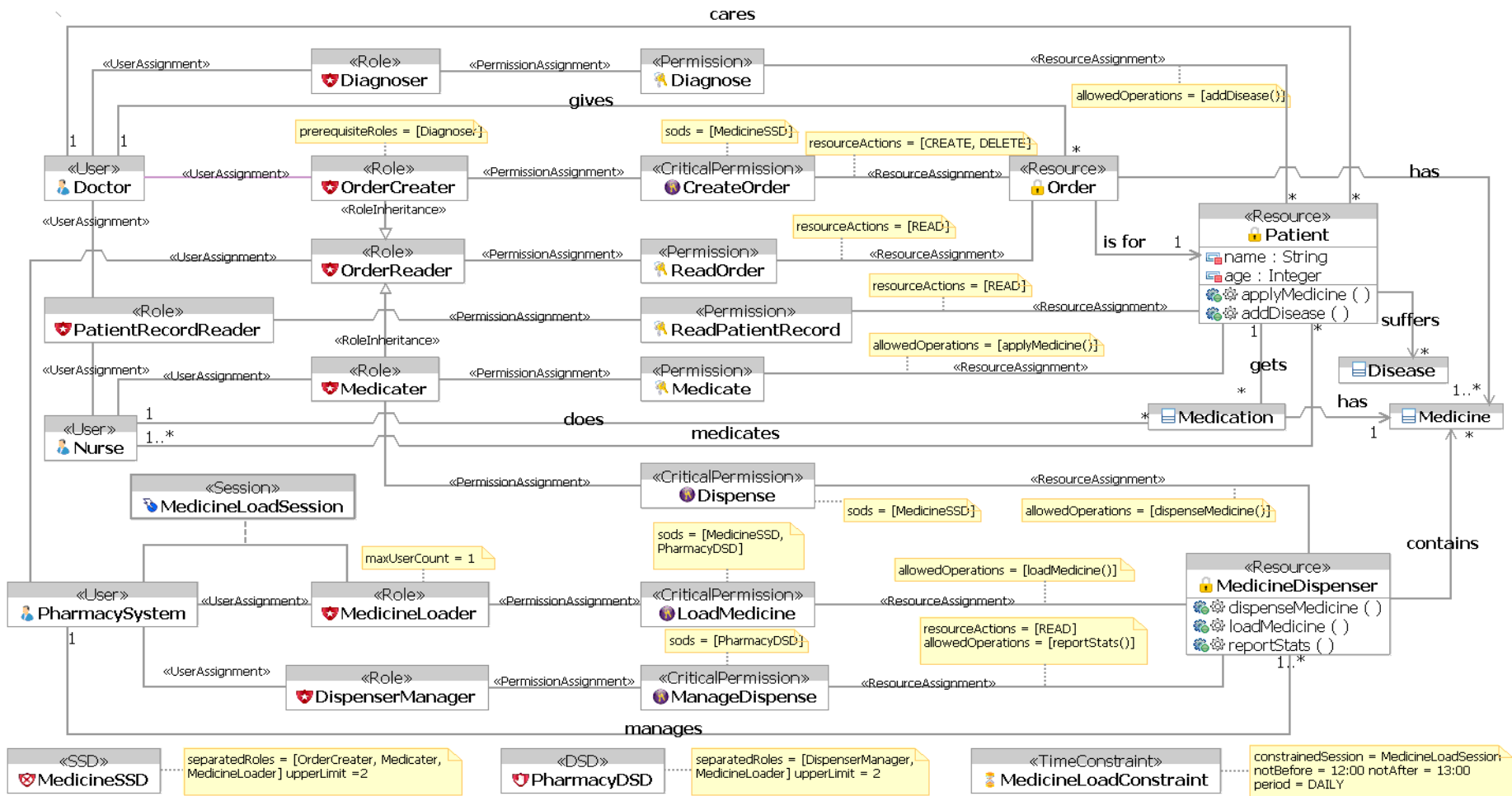


Figure 4.5 : Platform Independent Model of the Hospital Automation System



#### **4.5 Platform Independent Model**

The PIM of the hospital automation system, which is depicted in Figure 4.5, is constructed by modeling domain requirements and access control requirements together by virtue of the proposed UML Profile for RBAC. This model is validated successfully which means it is well-formed and does not violate any RBAC constraint.

#### **4.6 Ill-formed Security Model**

Ill-formed security model of the hospital automation system as illustrated in Figure 4.6 is constructed to constitute design error and constraint violation samples. Whole security model elements are not employed in this model to keep the model simple to show only elements that could not pass validation check. When the model is validated, there are some errors on some elements. All found errors and their reasons are described in APPENDIX C.1. Some CASE-tools like IBM Rational Software Modeler allow defining custom error and warning messages for violated OCL expressions. These messages can be internationalized. Defined error messages for the proposed UML Profile for RBAC are in APPENDIX B.1.

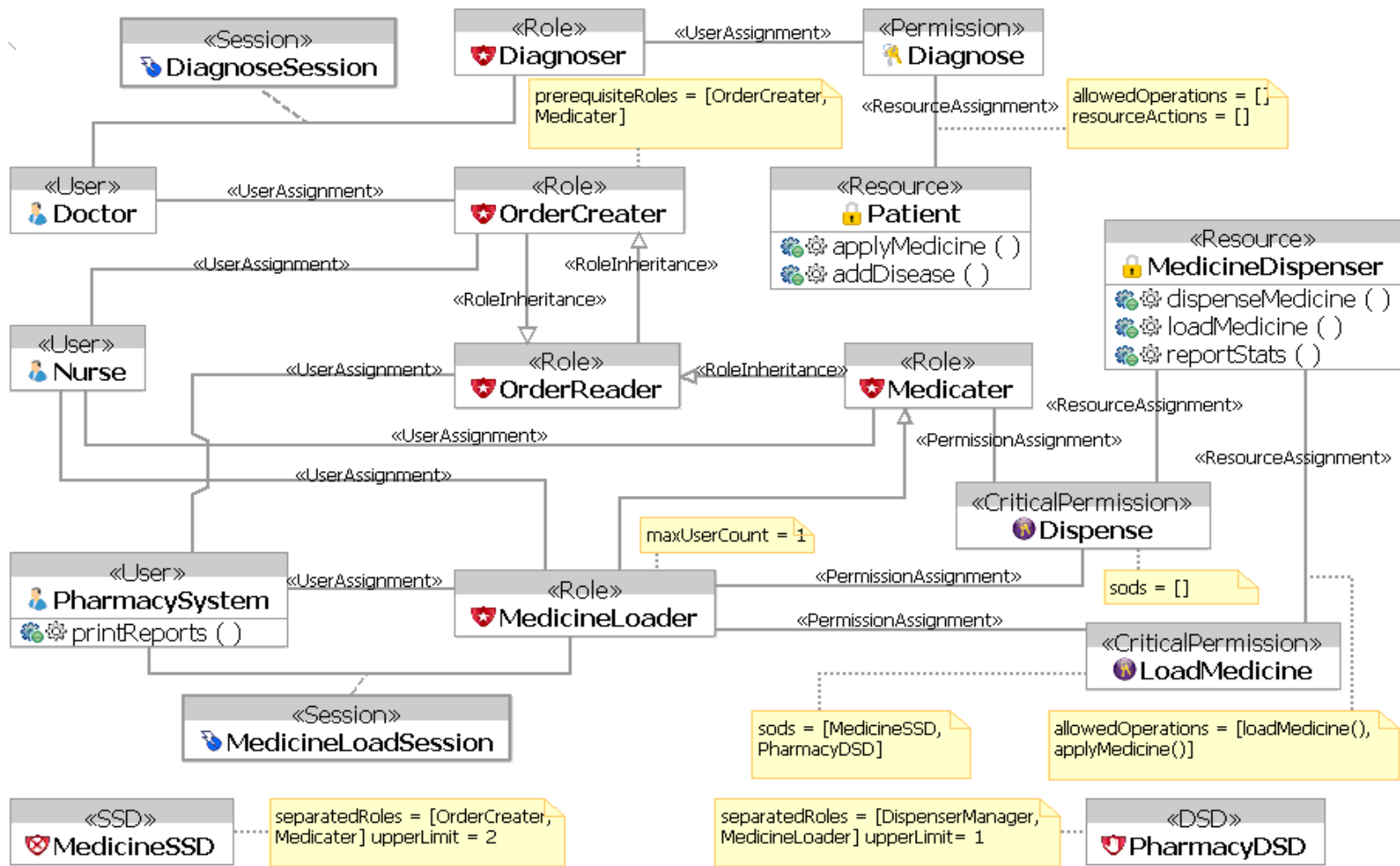


Figure 4.6 : Ill-formed Security Model of the Hospital Automation System

## 5. CONCLUSION AND RECOMMENDATIONS

This thesis has proposed a UML Profile for RBAC, which provides early integration of access control specifications to the entire development process. This study employed RBAC constraints to the UML Profile in order to get use of the strengths of RBAC, such as separation of duties and cardinality constraints. The models to that this profile applied, can be validated against ill-formedness and security constraint violations. By this way, design problems can be realized and fixed earlier. A formal language; OCL is used for validation. The proposed UML Profile is lightweight, easily interchangeable and deployable, and has a wide-range of CASE-tools support.

In this study, the proposed UML Profile is designed to be used only in UML class diagrams, for the structural and static aspects of the system. It can be designed for other popular diagram types of UML like sequence diagram and state diagram, for the dynamic aspects of the system. If the profile can be used for both static and dynamic aspects of the system, it will be more flexible and usable. For example, the profile will be applied to state diagrams for the controller-based systems. Session, which is one of the core elements of the RBAC, is created at run time in the system. It is a dynamic system element so it will be more appropriate to show it in a UML diagram that is for dynamic view of the system. In this thesis, it is provided a way to put Session element in a class diagram. It will be useful if users cannot drop or add a role in an established session, if it is defined at design time which roles will be activated in which sessions, or if a session is required to be restricted by a time-based constraint. One will create transformation functions for a well-known access control infrastructure to examine how the PIM can be used to generate the PSM or generate code directly. Employing role delegation feature and temporal constraints [26] into the profile will enrich it.



## REFERENCES

- [1] **Basin, D., Doser, J., and Lodderstedt, T.**, 2006. Model Driven Security: From UML Models to Access Control Infrastructures. *ACM Transactions on Software Engineering and Methodology*, Vol. **15**, No. 1, 39-91.
- [2] **Bell, D.E., and LaPadula, L.J.**, 1976. Secure Computer System: Unified Exposition and Multics Interpretation. *MTR-2997 Rev. 1*, Bedford, MA: The Mitre Corporation, March.
- [3] **Brewer, D., and Nash, M.** 1989. The Chinese Wall Security Policy. *In Proceedings of the 1989 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 206–214.
- [4] **Clark, D. D., and Wilson, D. R.**, 1987. A Comparison of Commercial and Military Computer Security Policies. *In Proceedings of the 1987 IEEE Symposium on Research in Security and Privacy*. IEEE Press, 184-194.
- [5] **Ferraiolo, D., and Kuhn, R.**, 1992. Role-Based Access Control. *In Proceedings of the 15th NIST-NSA National Computer Security Conference*, 554-563.
- [6] **Ferraiolo, D., Kuhn, R., and Chandramouli, R.**, 2007. Role-Based Access Control, Second Edition. *Artech House, Information Security and Privacy Series*.
- [7] **American National Standard for Information Technology**, 2004. Role Based Access Control, *ANSI INCITS 359-2004*.
- [8] **Object Management Group**, 2009. OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.2 (Feb. 2009).   
<<http://www.omg.org/cgi-bin/doc?formal/09-02-02>>
- [9] **Fuentes-Fernández, L., and Vallecillo-Moreno, A.**, 2004. An Introduction to UML Profiles. *UPGRADE, European Journal for the Informatics Professional*, Vol. **5**, No. 2, 5-13.
- [10] **Object Management Group**, 2006. Object Constraint Language, OMG Available Specification, Version 2.0 (May. 2006).   
<<http://www.omg.org/spec/OCL/2.0/>>
- [11] **Object Management Group**, 2003. MDA Guide Version 1.0.1 (Jun. 2003).   
<<http://www.omg.org/cgi-bin/doc?omg/03-06-01>>
- [12] **Object Management Group**, 2007. MOF 2.0/XMI Mapping, OMG Available Specification, Version 2.1.1. (Dec. 2007)   
<<http://www.omg.org/cgi-bin/doc?formal/2007-12-01>>
- [13] **Sandhu, R., et. al.**, 1996. Role-Based Access Control Models. *IEEE Computer*, Vol. **29**, No.2, 1996, 38-47.

- [14] **Sandhu, R., Ferraiolo, D., and Kuhn, R.**, 2000. The NIST Model for Role-Based Access Control: Towards a Unified Standard. *In Proceedings of the 5<sup>th</sup> ACM Workshop on Role-Based Access Control*, 2000, 47-63.
- [15] **Ferraiolo, D., Kuhn, R., and Sandhu, R.**, 2007. RBAC Standard Rationale: Comments on "A Critique of the ANSI Standard on Role-Based Access Control". *IEEE Security and Privacy*, Vol. **5**, No. 6, 51-53.
- [16] **Object Management Group**, 2002. UML Profile for CORBA Specification, Version 1.0 (Apr. 2002). <<http://www.omg.org/cgi-bin/doc?formal/02-04-01>>
- [17] **Object Management Group**, 2005. UML Testing Profile, Version 1.0 (Jul. 2005). <<http://www.omg.org/cgi-bin/doc?formal/05-07-07>>
- [18] **Ray, I., Li, N., France, R., and Kim, D.**, 2004. Using UML to Visualize Role-Based Access Control Constraints. *In Proceedings of the 9th ACM Symposium on Access Control Models and Technologies*, 2004, 115-124.
- [19] **Shin, M.E., and Ahn, G.**, 2000. UML-Based Representation of Role-Based Access Control. *In Proceedings of the 9th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2000, 195-200.
- [20] **Ahn, G., and Shin, M.E.**, 2001. Role-Based Authorization Constraints Specification Using Object Constraint Language. *In Proceedings of the 10th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2001, 157-162.
- [21] **Wang, H., Zhang, Y., Cao, J., and Yang, J.**, 2004. Specifying Role-Based Access Constraints with Object Constraint Language. *APWeb 2004, LNCS Vol. 3007, Springer Berlin / Heidelberg*, 687-696.
- [22] **Object Management Group**, 2006. Meta Object Facility (MOF) Core Specification, OMG Available Specification, Version 2.0 (Jan. 2006). <<http://www.omg.org/spec/MOF/2.0/>>
- [23] **Jin, X.**, 2006. Applying Model Driven Architecture Approach to Model Role Based Access Control System. *Thesis (M.Sc.)--University of Ottawa*, 2006.
- [24] **Organization for the Advancement of Structured Information Standards**, 2005. Core: eXtensible Access Control Markup Language (XACML) Version 2.0 (Feb. 2005). <[http://docs.oasis-open.org/xacml/2.0/access\\_control-xacml-2.0-core-spec-os.pdf](http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf)>
- [25] **Cirit, C., and Buzluca, F.** 2009. A UML Profile for RBAC, in XMI format. <<http://www.buzluca.info/rbac/RBAC.profile.xmi>>
- [26] **Joshi, J.B.D., Bertino, A., and Ghafoor A.**, 2002. Temporal Hierarchies and Inheritance Semantics for GTRBAC. *In Proceedings of the 7<sup>th</sup> ACM Symposium on Access Control Models and Technologies*, 2002, 74-83.

## **APPENDICES**

**APPENDIX A.1** : Global OCL Definitions for the proposed UML Profile for RBAC

**APPENDIX B.1** : Error messages of the proposed UML Profile for RBAC

**APPENDIX C.1** : Found errors of ill-formed security model of the hospital automation system

## APPENDIX A.1

**Table A.1** : Global OCL Definitions

### **context Classifier**

--Returns parents of the Classifier

```
def:  
    parents(): Set(Classifier) =  
        self.generalization.general
```

--Returns all parents of the Classifier

```
def:  
    allParents(): Set(Classifier) =  
        self.parents()->union(self.parents()->collect(p | p.allParents()))
```

### **context Element**

--Returns true if the element is a User

```
def:  
    isUser : Boolean =  
        self.oclAsType(Class).extension_User<>null
```

--Returns User as a Stereotype

```
def:  
    asUser : Boolean =  
        self.oclAsType(Class).extension_User. oclAsType(RBAC::User)
```

--Returns true if the element is a Role

```
def:  
    isRole : Boolean =  
        self.oclAsType(Class).extension_Role<>null
```

--Returns Role as a Stereotype

```
def:  
    asRole : Role =  
        self.oclAsType(Class).extension_Role. oclAsType(RBAC::Role)
```

--Returns true if the element is a Permission

```
def:  
    isPermission : Boolean =  
        self.oclAsType(Class).extension_Permission<>null
```



**Table A.1 (contd.) : Global OCL Definitions**

```
--Returns Permission as a Stereotype
def:
    asPermission : Permission =
        self.oclAsType(Class).extension_Permission.
        oclAsType(RBAC::Permission)

--Returns true if the element is a Resource
def:
    isResource : Boolean =
        self.oclAsType(Class).extension_Resource<>null

--Returns Resource as a Stereotype
def:
    asResource : Boolean =
        self.oclAsType(Class).extension_Resource.
        oclAsType(RBAC::Resource)

--Returns true if the element is a SoD
def:
    isSoD : Boolean =
        self.oclAsType(Class).extension_SoD<>null

--Returns SoD as a Stereotype
def:
    asSoD : SoD =
        self.oclAsType(Class).extension_SoD. oclAsType(RBAC::SoD)

--Returns true if the element is a SSD
def:
    isSSD : Boolean =
        self.oclAsType(Class).extension_SoD<>null and
        self.oclAsType(Class).extension_SoD. oclIsTypeOf(RBAC::SSD)

--Returns true if the element is a DSD
def:
    isDSD : Boolean =
        self.oclAsType(Class).extension_SoD<>null and
        self.oclAsType(Class).extension_SoD. oclIsTypeOf(RBAC::DSD)

--Returns true if the element is a CriticalPermission
def:
    isCriticalPermission : Boolean =
        self.oclAsType(Class).extension_Permission<>null and
        self.oclAsType(Class).extension_Permission.
        oclIsTypeOf(RBAC::CriticalPermission)
```

**Table A.1 (contd.) : Global OCL Definitions**

```
--Returns CriticalPermission as a Stereotype
def:
    asCriticalPermission : Boolean =
        self.oclAsType(Class).extension_Permission.
        oclAsType(RBAC::CriticalPermission)

--Returns true if the element is a Session
def:
    isSession : Boolean =
        self.oclAsType(AssociationClass). extension_Session<>null

--Returns true if the element is a Operation
def:
    isOperation : Boolean =
        self.oclAsType(Operation). extension_Operation<>null

--Returns true if the element is a ResourceAssignment
def:
    isResourceAssignment : Boolean =
        self.oclAsType(Association). extension_ResourceAssignment<>null

--Returns ResourceAssignment as a Stereotype
def:
    asResourceAssignment : ResourceAssignment =
        self.oclAsType(Association). extension_ResourceAssignment.
        oclAsType(RBAC::ResourceAssignment)

--Returns true if the element is a PermissionAssignment
def:
    isPermissionAssignment : Boolean =
        self.oclAsType(Association). extension_PermissionAssignment<>null

--Returns true if the element is a UserAssignment
def:
    isUserAssignment : Boolean =
        self.oclAsType(Association). extension_UserAssignment<>null

--Returns true if the element is a RoleInheritance
def:
    isRoleInheritance : Boolean =
        self.oclAsType(Generalization). extension_RoleInheritance<>null

--Returns assigned Permission(s) of a Role
def:
    assignedRolePermissions(role : Type) : Set (Type) =
        Association.allInstances()->select(as : Association |
        as.isPermissionAssignment and as.endType->exists(t | t=role))->
        collect(endType)->asSet()->select(isPermission)
```

**Table A.1 (contd.) : Global OCL Definitions**

```
--Returns assigned CriticalPermission(s) of a Role
def:
    assignedRoleCriticalPermissions(role : Type) : Set (Type) =
        assignedRolePermissions(role)->
        select(oclAsType(Class).extension_Permission.
            oclIsTypeOf(RBAC::CriticalPermission))

--Returns assigned Role(s) of a Permission
def:
    assignedPermissionRoles(permission : Type) : Set (Type) =
        Association.allInstances()->select(as : Association |
            as.isPermissionAssignment and as.endType->exists(t | t=permission))->
            collect(endType)->asSet()->select(isRole)

--Returns assigned Resource(s) of a Permission
def:
    assignedPermissionResources(permission : Type) : Set (Type) =
        Association.allInstances()->select(as : Association |
            as.isResourceAssignment and as.endType->exists(t | t=permission))->
            collect(endType)->asSet()->select(isResource)

--Returns assigned Permission(s) of a Resource
def:
    assignedResourcePermissions(resource : Type) : Set (Type) =
        Association.allInstances()->select(as : Association |
            as.isResourceAssignment and as.endType->exists(t | t=resource))->
            collect(endType)->asSet()->select(isPermission)

--Returns assigned User(s) of a Role
def:
    assignedRoleUsers(role : Type) : Set (Type) =
        Association.allInstances()->select(as : Association |
            as.isUserAssignment and as.endType->exists(t | t=role))->
            collect(endType)->asSet()->select(isUser)

--Returns assigned Role(s) of a User
def:
    assignedUserRoles(user : Type) : Set (Type) =
        Association.allInstances()->select(as : Association |
            as.isUserAssignment and as.endType->exists(t | t=user))->
            collect(endType)->asSet()->select(isRole)

--Returns established Session(s) of a User
def:
    establishedUserSessions(user : Type) : Set (Type) =
        AssociationClass.allInstances()->select(asc : AssociationClass |
            asc.isSession and asc.endType->exists(t | t=user))
```

**Table A.1 (contd.) : Global OCL Definitions**

--Returns activated Roles in a Session

def:

```
activeSessionRoles(session : Class) : Set(Type) =  
    session.endType->select(isRole)
```

--Returns owner User of the Session

def:

```
sessionUser(session : Class) : Type =  
    session.endType->any(isUser)
```

--Returns prerequisite roles of a role

def:

```
prerequisites(role : Class) : Set (Class) =  
    role.extension_Role.prerequisiteRoles-> iterate(r;res:Set(Class)=Set{} |  
    res-> including(r.base_Class))
```

--Returns base set union all parents of set members

def:

```
allFamily(baseFamily : Set(Type)) : Set(Class) =  
    baseFamily->union(baseFamily->  
    collect(allParents().oclAsType(Class))->asSet())
```

--Returns all SSD stereotyped classes in the model

def:

```
allSSDs : Set(Class) =  
    Class.allInstances()->select(isSSD)
```

--Returns all DSD stereotyped classes in the model

def:

```
allDSDs : Set(Class) =  
    Class.allInstances()->select(isDSD)
```

--Returns excluded roles in a SoD

def:

```
sodRoles(sod : Class) : Set (Class) =  
    sod.extension_SoD.separatedRoles-> iterate(r;res:Set(Class)=Set{} |  
    res-> including(r.base_Class))
```

--Returns authorized roles of a User

def:

```
authorisedRoles(user : Type) : Set(Class) =  
allFamily(assignedUserRoles(user))
```

## APPENDIX B.1

**Table B.1** : Error Messages

<b>OCL Invariant Name</b>	<b>Error Message</b>
<b>operationEncloser</b>	Operation::operationEncloser   Owner Class of the <<operation>> function should be stereotyped with <<resource>>.
<b>role_permission</b>	PermissionAssignment::role_permission   <<permissionAssignment>> Association should connect <<role>> Class to <<permission>> Class.
<b>role_user</b>	UserAssignment::role_user   <<userAssignment>> Association should connect <<role>> Class to <<user>> Class.
<b>permission_resource</b>	ResourceAssignment::permission_resource   <<resourceAssignment>> Association should connect <<permission>> Class to <<resource>> Class.
<b>allowedOperations Owner</b>	ResourceAssignment::allowedOperationsOwner   allowedOperations have an Operation that does not belong to assigned Resource.
<b>hasOperations</b>	ResourceAssignment::hasOperations   allowedOperations or resourceActions should include some elements.
<b>user_session_roles</b>	Session::user_session_roles   <<session>> AssociationClass should connect <<user>> Class to <<role>> Class(es).
<b>inheritanceShouldBe RoleInheritance</b>	Role::inheritanceShouldBeRoleInheritance   <<role>> Class has a generalization that is not stereotyped with <<roleInheritance>>.
<b>Prerequisite SelfContain</b>	Role::prerequisiteSelfContain   prerequisiteRoles contains owner <<role>> Class.
<b>emptySoDs</b>	CriticalPermission::emptySoDs   sods tagged value of <<criticalPermission>> Class is empty.

**Table B.1 (contd.) : Error Messages**

<b>OCL Invariant Name</b>	<b>Error Message</b>
<b>onlyOneRole</b>	CriticalPermission::onlyOneRole   <<criticalPermission>> Class is assigned to more than one Role.
<b>inheritanceCycle</b>	RoleInheritance::inheritanceCycle   <<roleInheritance>> Generalization caused an Inheritance Cycle.
<b>role_role</b>	RoleInheritance::role_role   general and specific end types should be <<role>> Classes.
<b>maxResourceCount</b>	Permission::maxResourceCount   maxResourceCount of <<permission>> Class is exceeded.
<b>maxRoleCount</b>	Permission::maxRoleCount   maxRoleCount of <<permission>> Class is exceeded.
<b>maxRole PermissionCount</b>	Role::maxRolePermissionCount   maxPermissionCount of <<role>> Class is exceeded.
<b>maxUserCount</b>	Role::maxUserCount   maxUserCount of <<role>> Class is exceeded.
<b>maxAssigned RoleCount</b>	User::maxAssignedRoleCount   maxAssignedRoleCount of <<user>> Class is exceeded.
<b>maxActivated RoleCount</b>	User::maxActivatedRoleCount   maxActivatedRoleCount (in a session) of <<user>> Class is exceeded.
<b>maxResorce PermissionCount</b>	Resource::maxResorcePermissionCount   maxPermissionCount of <<resource>> Class is exceeded.
<b>prerequisiteSSD Consistency</b>	Role::prerequisiteSSDConsistency   prerequisiteRoles violates SSD constraint.
<b>shouldBeInSoD</b>	Role::shouldBeInSoD   <<role>> Class is not included in excludedRoles of SoDs that are included in sods tagged value of assigned <<criticalPermission>> Classes.
<b>allowedRoles UpperLimit</b>	SoD::allowedRolesUpperLimit   upperLimit is not between 2 and size of seperatedRoles.

**Table B.1 (contd.) : Error Messages**

<b>OCL Invariant Name</b>	<b>Error Message</b>
<b>criticalTask DividedToRoles</b>	SoD::criticalTaskDividedToRoles   one or more excludedRoles are not assigned to a CriticalPermission that its sods tagged value includes this SoD.
<b>dsdRule</b>	Session::dsdRule   <<session>> AssociationClass associated DSD role(s) by exceeding upper limit.
<b>userAssigned RolesActivation</b>	Session::userAssignedRolesActivation   a <<role>> Class that is not assigned to session owner <<user>> Class, is associated with <<session>> Class.
<b>prerequisiteRule</b>	UserAssignment::prerequisiteRule   associated <<user>> Class is not already assigned to the prerequisiteRoles of the associated <<role>> Class.
<b>ssdRule</b>	UserAssignment::ssdRule   associated <<user>> Class is assigned to SSD role(s) by exceeding upper limit.
<b>roleInheritance SSDRule</b>	RoleInheritance::roleInheritanceSSDRule   <<roleInheritance>> Generalization violates an SSD constraint.

## APPENDIX C.1

**Table C.1** : Errors of the ill-formed security model

<b>Element</b>	<b>Violated OCL Expression</b>	<b>Reason</b>
<<Role>> MedicineLoader	Role:: maxUserCount	Even though its maxUserCount tagged value is set to 1, the role is assigned to two users, Nurse and PharmacySystem.
<<Role>> MedicineLoader	Role:: inheritance ShouldBeRole Inheritance	It has a generalization that is not stereotyped with <<roleInheritance>>.
<<Role>> OrderCreator	Role:: prerequisiteSelf Contain	Its prerequisiteRoles tagged value contains itself.
<<Role>> OrderCreator	Role:: prerequisiteSSD Consistency	Its prerequisiteRoles tagged value contains Medicater. Medicater and OrderCreator roles are exclusive roles in the MedicineSSD so this prerequisite relation violates the SSD constraint.
<<Role>> MedicineLoader	Role:: shouldBeInSoD	Even though it is assigned to LoadMedicine CriticalPermission, it is not included in excludedRoles tagged value of MedicineSSD that is specified in sods tagged value of LoadMedicine.



**Table C.1 (contd.) : Errors of the ill-formed security model**

Element	Violated OCL Expression	Reason
<p align="center">&lt;&lt;Operation&gt;&gt; printReports()</p>	<p align="center">Operation:: operation Encloser</p>	<p>Owner class of the &lt;&lt;operation&gt;&gt; stereotyped function should be stereotyped with &lt;&lt;resource&gt;&gt;. Its owner class is stereotyped with &lt;&lt;user&gt;&gt;.</p>
<p align="center">&lt;&lt;UserAssignment&gt;&gt; (Diagnoser) (Diagnose)</p>	<p align="center">User Assignment:: role_user</p>	<p>It should connect a &lt;&lt;role&gt;&gt; stereotyped class to a &lt;&lt;user&gt;&gt; stereotyped class but it connects Diagnoser role to Diagnose permission.</p>
<p align="center">&lt;&lt;SSD&gt;&gt; MedicineSSD</p>	<p align="center">SoD:: criticalTask DividedTo Roles</p>	<p>Medicater role that is included in its excludedRoles tagged value, is assigned to Dispense CriticalPermission but sods tagged value of this CriticalPermission does not include MedicineSSD.</p>
<p align="center">&lt;&lt;DSD&gt;&gt; PharmacyDSD</p>	<p align="center">SoD:: allowedRoles UpperLimit</p>	<p>Its upperLimit tagged value is set to 1 but it should be <math>\geq 2</math>.</p>
<p align="center">&lt;&lt;Session&gt;&gt; MedicineLoadSession</p>	<p align="center">Session:: dsdRule</p>	<p>It associates MedicineLoader role but this role is in excludedRoles tagged value of PharmacyDSD and upperLimit is 1 means no role that is included in the excludedRoles can be activated. Activated role count in this session is not small than upperLimit, 1.</p>

**Table C.1 (contd.) : Errors of the ill-formed security model**

Element	Violated OCL Expression	Reason
<p align="center">&lt;&lt;Session&gt;&gt; DiagnoseSession</p>	<p align="center">Session:: userAssigned RolesActivation</p>	<p>Doctor user and Diagnoser role are associated by this session but Diagnoser role is not assigned to Doctor. A User should not activate a role if he is not authorized for that role.</p>
<p align="center">&lt;&lt;ResourceAssignment&gt;&gt; (Medicine Dispenser) (LoadMedicine)</p>	<p align="center">Resource Assignment:: allowed Operations Owner</p>	<p>Its allowedOperations tagged value contains &lt;&lt;operation&gt;&gt; stereotyped applyMedicine() function that does not belong to MedicineDispenser resource.</p>
<p align="center">&lt;&lt;ResourceAssignment&gt;&gt; (Patient) (Diagnose)</p>	<p align="center">Resource Assignment:: hasOperations</p>	<p>Its both allowedOperations and resourceActions tagged values are empty, which is not allowed.</p>
<p align="center">&lt;&lt;UserAssignment&gt;&gt; (Doctor) (OrderCreator)</p>	<p align="center">User Assignment:: prerequisite Rule</p>	<p>OrderCreator role has Medicater role as prerequisite role but Doctor user is not already assigned to the Medicater role.</p>
<p align="center">&lt;&lt;UserAssignment&gt;&gt; (Nurse) (OrderCreator)</p>	<p align="center">User Assignment:: ssdRule</p>	<p>Nurse user is already assigned to Medicater role that has an SSD relation with OrderCreator role. This assignment violates SSD constraint.</p>
<p align="center">&lt;&lt;RoleInheritance&gt;&gt; (senior: OrderReader) (junior: OrderCreator)</p>	<p align="center">Role Inheritance:: inheritance Cycle</p>	<p>OrderReader role is already junior role of the OrderCreator role. This inheritance causes an inheritance cycle, which is not allowed.</p>

**Table C.1 (contd.) : Errors of the ill-formed security model**

Element	Violated OCL Expression	Reason
<p>&lt;&lt;RoleInheritance&gt;&gt; (senior: Medicater) (junior: OrderReader)</p>	<p>Role Inheritance:: roleInheritance SSDRule</p>	<p>OrderReader role inherits permissions from OrderCreator role that means Medicater role indirectly inherits permissions from OrderCreator role via this inheritance but OrderCreator and Medicater roles are in SSD relation. Therefore, this inheritance violates the SSD constraint.</p>
<p>&lt;&lt;CriticalPermission&gt;&gt; Dispense</p>	<p>Critical Permission:: emptySoDs</p>	<p>It has an empty sods tagged value, which is not allowed. Sods tagged value should include at least one SoD element.</p>
<p>&lt;&lt;CriticalPermission&gt;&gt; Dispense</p>	<p>Critical Permission:: onlyOneRole</p>	<p>It is assigned to both Medicater role and MedicineLoader role but it should be assigned to only one role because it is a CriticalPermission that is not sharable.</p>



## CURRICULUM VITA



**Candidate's full name:** Çağdaş CİRİT

**Place and date of birth:** Gülnar/Mersin/Türkiye 01.01.1983

**Permanent Address:** Hürriyet M. Karaca Sk. Ekşioğlu Selvi Sitesi B Blok  
D: 32 Yakacık/Kartal İstanbul Türkiye

**Universities attended:** BSc: Computer Engineering; Izmir Institute of  
Technology; Turkey.

### **Publications:**

- Şaşıoğlu, B., Cirit, Ç., Çakmakkaya, Z., and Örencik, B., 2007: Taktik Sahada Özel Bir Sertifika Doğrulama Problemine Alternatif Yaklaşım. *ISCTurkey 2007 2. Uluslararası Katılımlı Bilgi Güvenliği ve Kriptoloji Konferansı*, Bildiriler Kitabı, p. 270-275.