Anton Naumenko

ANALYSIS AND SEMANTIC DESCRIPTION OF ROLE BASED ACCESS CONTROL MODELS

Master's thesis

Mobile computing

22/03/2005

University of Jyväskylä

Department of Mathematical Information Technology

Author: Anton Naumenko

Contact Information: e-mail: annaumen@cc.jyu.fi

Title: Analysis and Semantic Description of Role Based Access Control Models

Work: Master's Thesis

Number of Pages: 85

Study Line: Mobile Computing

Department: University of Jyväskylä, Department of Mathematical Information Technology

Keywords: RBAC, RDF, OWL, Administrative RBAC, Enterprise Access Control, Enterprise RBAC

Abstract: A lot of research has been done to develop, elaborate and implement RBAC models and their features but little is done towards unifying the vision of different parties except NIST reference model [Ferraiolo2001]. There have been some efforts to develop languages for RBAC data representation [Bacon2002], [OASIS], but they all work on the level of data structure and syntax definitions, and are only of limited use in the conceptual integration of RBAC research field.

The thesis considers the issues of creation and collective use of ontologies to provide a platform-independent language for flexible specification and provisioning of AC policies. Ontology as a semantic description can serve to integrate together different visions and elaborations of existing RBAC models in conceptual way and, what is really important, to facilitate integration of future extensions of RBAC concepts and domain specific adjustments to solid extensible representations. Expressing conceptual semantics of RBAC model enables to start the process to consolidate existing platform-dependent AC models and mechanisms to higher level of abstraction.

Acknowledgements

First of all I want to say thanks to my supervisor Timo Tiihonen, who is vice-rector of University of Jyväskylä and professor of Mathematical Information Technology Department. He provided me with research area for the thesis. Ideas in the thesis are results of our communication, his advises and suggestions. I hope that the thesis is valuable for the University needs.

Special thanks also to Vesa Lappalainen, who is lecturer in University of Jyväskylä, and to his Korppi-team for advises during my work with the master's thesis.

I cannot overestimate the role of two people in my success with studying in University of Jyväskylä. They are Ass. Professor Vagan Terziyan from Mathematical Information Technology Department and Dr. Helen Kaykova. I am grateful to them for their guidance, support and kind attitude in my studying, research and everyday life. Actually Vagan is a person who introduced to me Semantic Web Activity.

Especial thanks to University of Jyväskylä, Kharkov National University of Radioelectronics, and to all people who were involved in establishment of exchange program between these two universities.

Community of Ukrainians provides me real backing, abilities for efficient work and abilities to have fun recreational activities. I appreciate help and concern of all my friends.

Next, I want to say thanks to my parents for their existence and continuous support of all my efforts.

Finally, I am thankful to my wife who encouraged me to go 2000 km from home. She is most significant person for me.

Abbreviations

RBAC – Role-Based Access Control

ARBAC – Administrative RBAC

ERBAC – Enterprise RBAC

SARBAC – Scoped Administrative RBAC

AC – Access Control

EAF – Enterprise Access Control Framework

EAC – Enterprise Access Control

EAM – Enterprise Access Control Model

RCC – Role Control Center

OWL – Web Ontology Language

RDF – Resource Description Framework

RDFS – RDF Schema language

XML – eXtensible Markup Language

Contents

1 Introduction

Thesis has two main goals as can be seen from its title. First one is to make review of Role Based Access Control (RBAC) models. RBAC research area has a lot of different RBAC elaborations. Therefore thesis scopes the RBAC reference model, RBAC data administration models, and Enterprise RBAC model. Second goal is to study the applicability of Semantic Web languages for semantic description of RBAC models and the possibility to create semantically rich language for specification of Access Control (AC) policies. Thus thesis has RDF schemas and OWL ontologies for the reviewed RBAC models and considerations about usability of semantically encoded RBAC data.

1.1 Motivation for RBAC semantic description

A lot of research has been done to develop, elaborate and implement RBAC models and their features but little is done towards unifying the vision of different parties except NIST reference model [Ferraiolo2001]. There is no comprehensive source which describes the general framework of RBAC like guide for developers who want to use this model of AC. There are some efforts to develop languages of RBAC data representation [Bacon2002], [OASIS] but they all work on the level of data structure and syntax definitions, thus providing only few possibilities for conceptual integration of RBAC research field.

A general framework should, among other things, provide tools to integrate different branches of RBAC elaborations because a particular application might lead to a situation when different RBAC elaborations that are contradictory to each other should be applied in one domain. Also as RBAC models are defined in an abstract way, designers need to adjust and enhance models to problem domain requirements for almost all cases.

Creation of a common ontology can simplify work of developers by providing a platform-independent language for flexible specification and provisioning of AC policies. Ontology as a semantic description can serve to integrate together in conceptual way different visions and elaborations of existing RBAC models and, what is really important, to facilitate integrating of future extensions and domain specific adjustments of RBAC concepts to a solid extensible representation.

Expressing conceptual semantics of RBAC models enables the consolidation of existing platform-dependent AC models and mechanisms to higher a level of abstraction on which RBAC model is defined. Also it supports the development of commercial RBAC applications with ontological description and thus unified understanding.

Following advantages are expected to be achieved by RBAC semantic description using Semantic Web languages [SemanticWeb], [RDF], [OWL], and [XML]:

- Possibilities to develop arbitrary model for storage, programming and optimization of platform-dependent models based on RBAC ontology
- Machine readable form of RBAC metadata description allows automation of integration processes of AC mechanisms and models of different environments
- Common understanding of RBAC concepts by humans and applications of different sites by using of RBAC ontology
- Expressive power of Semantic Web languages for capturing all aspects of RBAC abstract models and domain specific features
- Support of arbitrary levels of abstraction to specify business view as AC policies for arbitrary platform dependent model

1.2 Structure of the thesis

This work will rely heavily on three concepts: RBAC, RDF and OWL. These are briefly introduced on a general level as background information for the reader.

Chapter 2 contains NIST RBAC reference model's taxonomy and concepts analysis. Also it depicts design and comparison of corresponding RDF Schema and OWL ontology. Chapter 3 is dedicated for review and semantic description of Administrative RBAC models. In Chapter 4 description of Enterprise RBAC model semantics is given. Chapter 5 and Chapter 6 are written in the context of the University of Jyväskylä as target organization for RBAC implementation. Chapter 5 considers the general aspects of RBAC

models in the context of the University. Chapter 6 contains the specification of functionality of a prototype application for RBAC data administration.

1.3 Background

1.3.1 Role-Based Access Control

RBAC has been subject to active research for almost the whole last decade. As a result number of models and their extensions exists. The scope of the thesis consists of the reference model defined in NIST standard [Ferraiolo2001], Administrative RBAC models and Enterprise RBAC models.

The concept of RBAC has evolved to a technology thanks to a lot of research done in different branches of this field. Starting from theoretical formal descriptions of conceptual and operational features of RBAC models and ending to particular applications and implementations in different domains, business cases, environments, etc. As a technology RBAC is quite attractive for commercial use in cases of distributed and heterogeneous environments because of its potential to reduce costs and complexity of security issues in large organizations [RBAC book].

The main source of simplification in administration is achieved by using roles, hierarchies and constraints to organize privileges. As RBAC roles corresponds to individual's positions, duties and activities, cost reduction is achieved because positions, duties and organizational structures are more stable within enterprises than the positions of the employees. Another great advantage of RBAC is that it is policy neutral. It is possible to configure RBAC to support wide variety of traditional and domain specific AC policies.

Recently RBAC models have been used in a class of products called Enterprise Security Management Systems (ESMS). ESMS products are typically used for centralized management of authorizations for resources resident in several heterogeneous systems (called target systems) distributed throughout the enterprise though they may provide other security administration features such as password synchronization, single sign-on, and PKI as well. [RBAC book]

1.3.2 Resource Description Framework

The Resource Description Framework (RDF) is a framework for representing information in the Web [RDF]. It is intended for integration of a variety of applications using XML for syntax and URIs for naming [SemanticWeb]. The RDF is a structure for describing and interchanging metadata on the Web [Powers2003]. The RDF is expressive and flexible technology to describe arbitrary different domains and thus it is widely applicable. The World Wide Web Consortium (W3C) has been designing RDF as a basis technology to support Semantic Web activity and it gives following statement to describe the RDF:

The RDF is a language designed to support the Semantic Web, in much the same way that HTML is the language that helped initiate the original Web [Powers2003].

RDF is a framework for supporting resource description, or metadata (data about data), for the Web. RDF provides common structures that can be used for interoperable XML data exchange [SemanticWeb]. The RDF gives developers tools to encode meaning by expressing problem domain concepts and relations between them using RDF statements and connecting these statements to a semantic network. RDF, like XML and relational databases, follows object based domain decomposition for data representation but remains more generic and more expressive. There are also variety of software tools to work with RDF including tools for creating RDF, for creating vocabulary for RDF called Schema (RDFS), for querying RDF, for making inference based on RDF defined semantic network, etc.

So RDF brings to XML technology the same functionality as relational algebra to commercial database systems. RDF defines classes of problem domain concepts and their properties to create vocabulary of the domain in the same way like creation of tables and relationships between tables defines a schema of the database. XML can encode the contents of a relational database, XML can encode the contents of an RDF-based model – but XML isn't a replacement because XML is nothing more than syntax. A metadata vocabulary is needed to be able to use XML to record business domain information in such a way that any business can be documented, and RDF provides this capability [Powers2003].

1.3.3 Web Ontology Language

The Web Ontology Language (OWL) is developed to be used when the information contained in documents should be formalized more strictly to enable applications to process it, as opposed to situations where the content only needs to be presented to humans [OWL]. OWL can be used to explicitly express the meaning of concepts in a vocabulary, which is called ontology, and the relations between these concepts.

OWL uses URIs for naming and RDF for description to create the following advantages to ontologies [OWL]:

- Ability to be distributed across many systems
- Scalability to Web needs
- Compatibility with Web standards for accessibility and internationalization
- Openness and extensibility

OWL reuses XML [XML], RDF and RDF Schema [RDF] and introduces more sophisticated vocabulary to express relations between the classes and characteristics of the properties. OWL has more expressing power to encode meaning and semantics than XML, RDF, and RDF-S, and thus OWL goes beyond these languages in its ability to represent data about problem domain in a formal way.

So OWL is a sophisticated technology to precisely describe domain specific knowledge about classes, relationships between classes, properties of classes and constraints on relationships between the classes and properties of the classes.

OWL should be used in cases when issues about knowledge, data and software integration are important. OWL ontology gives an upper level formal description of the problem to unify understanding and interpretation of it by humans and applications.

2 RBAC reference model review and semantic description

2.1 RBAC standard overview

Most comprehensive and reliable RBAC model is defined as NIST standard proposal "RBAC Reference Model and the RBAC System and Administrative Functional Specification" [Ferraiolo2001]. Definition of the RBAC reference model and functional specification is done using theory of sets and first order logic predicates.

The reference model in the standard consists of submodels which form a taxonomy of possible RBAC functionalities [Sandhu1996] as figure 2.1 illustrates.

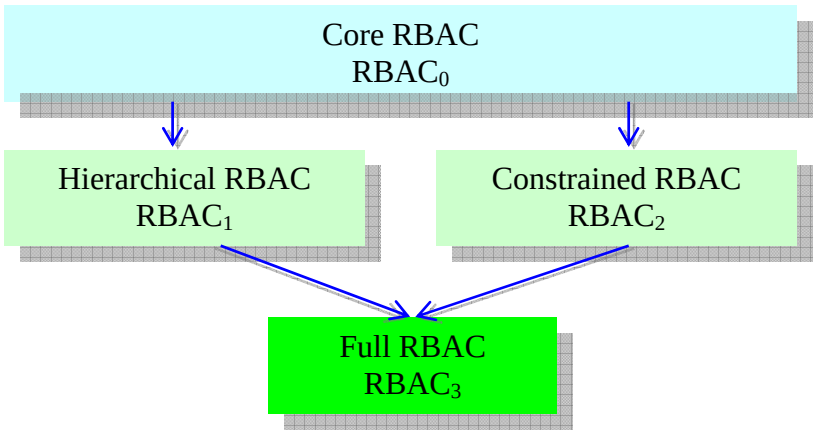

Figure 2.1 – Taxonomy of RBAC reference model submodels

$RBAC_0$ is the base model and is referred to as Core RBAC. Core RBAC defines the base set of essential concepts and functionalities for arbitrary implementation of RBAC. All further submodels include Core RBAC submodel and functionality by default.

$RBAC_1$ or Hierarchical RBAC extends Core RBAC model introducing partial order between roles which defines relationship of inheritance of permissions from role to role. Hierarchical RBAC recognizes limited and general role hierarchy. Limited role hierarchy constrains inheritance relation so that role can have only one ascendant. General role hierarchy supports multiple inheritance. $RBAC_1$ also defines functions to work with role inheritance.

$RBAC_2$ or Constrained RBAC adds the concepts of Separation of Duty constraints which are Static Separation of Duty and Dynamic Separation of Duty constraint.

$RBAC_3$ or Full RBAC consolidates all optional functionalities to the most featured reference model for RBAC implementation and it includes Core RBAC, role hierarchy and separation of duty constraints.

Functional specification of the RBAC standard describes administrative commands, supporting system functions, review functions and advanced review functions for all submodels separately and depicts changes in function definitions in the case of mixed usage of submodels.

2.2 Core RBAC

2.2.1 Description and analysis

Core RBAC incorporates a minimal set of features that can be used to build access control system to implement the RBAC model. Thus core RBAC formalizes the most essential elements, relations and functions of RBAC. More sophisticated RBAC model components include core RBAC as a basis. As stated in the standard, the core RBAC features mainly review traditional group-based control features.

The fundamental RBAC data elements are users, roles and permissions and relation between these elements as shown on figure 2.2. There is a statement in RBAC which distinguishes it from other access control models and paradigms. It states that users can be assigned to permissions only through roles, and never directly. This makes RBAC to implement any policy of access control by formulating it using role as a semantic construct.

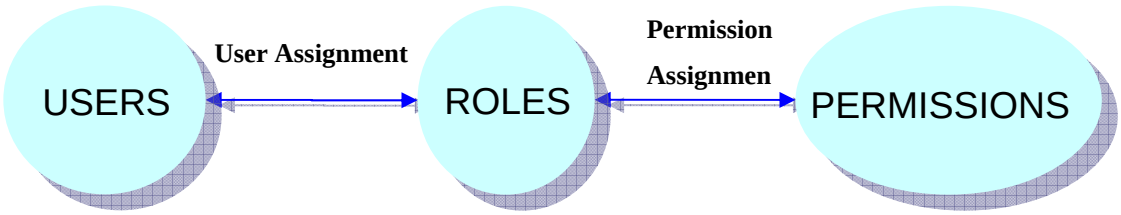

Figure 2.2 – Main elements of RBAC

User can be considered as a human being as it is in RBAC standard, but the definition of user depends on the viewpoint and can be extended to include other subjects of access control, such as machines, software, etc. Role is treated as a “job function within a context of organization” [Ferraiolo2001]. Role has also semantics which describes authority and responsibilities of users assigned to the job function. Roles and assignment of roles to users and of roles to permissions are central points of RBAC. These relations are of many-to-many type and thus give flexibility and granularity in permissions-to-users assignment when formulating organizational policy within RBAC model. That is why RBAC is considered to be policy neutral access control model.

Permission is a concept to formally describe user privileges within an organization. Permission states what operation can be executed on what set of objects protected by RBAC. Generally permission is a named relation between operations and objects as defined on figure 2.3.

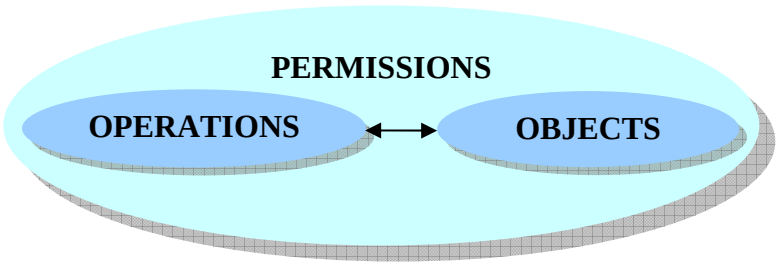

Figure 2.3 – Permissions as named relations between operations and objects

I consider operation as some kind of software activity (function or invocation of a method with given parameters). RBAC standard defines operation as “an executable image of a

program, which upon invocation executes some function for user" [Ferraiolo2001]. So from this definition of operation and from definition of permission we can see that object is something on which operation can be executed. RBAC views objects in two senses as information containers (files, directories, table within database, etc.) and as system resources (printers, disks, etc).

Thus core RBAC model can be presented as its element sets and relations among them as it is shown on figure 2.4. Additional element here is session. Purpose of this element is to model the dynamics of real information systems in which user has to execute some operation using some definite role or subset of roles from whole set of his or her roles at some instant of time. In such case he will create a session with an active role or roles from the set of all user roles. So user can have one or more roles as his active roles within session, but a session can be mapped only to one user. That is why relation user_session has type one-to-many. Relation session_role has type many-to-many because in one session zero or more roles can be activated, and one role can be activated by zero or more sessions.

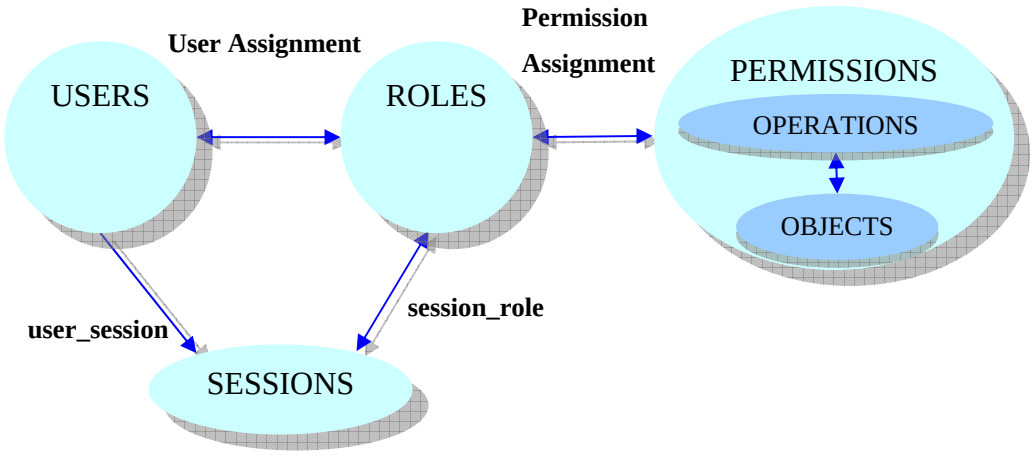

Figure 2.4 – Core RBAC model element sets and relations

2.2.2 RDF schema for Core RBAC

As I want to design ontology using RDF schema language for Core RBAC elements and relations then all classes which represent elements and relations should be specified first.

```
<rdfs:Class rdf:about="User" rdfs:label="User">
            <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
```

This specification above defines a new resource with the name “User” which is a class and is a subclass of Resource class defined in RDFS. Specification for the rest of the classes for core RBAC elements and relations looks analogical and is located in Appendix A.

The next step is to define properties of all the classes to capture the structure of core RBAC. In RDFS the properties are defined by stating the domain and the range of the property. The following specifies the property “name” to instances of classes which are included in the domain. The value of the property is a string of rdfs:Literal datatype.

```
<rdf:Property rdf:about="name" rdfs:label="name">
            <rdfs:domain rdf:resource="Object"/>
            <rdfs:domain rdf:resource="Operation"/>
            <rdfs:domain rdf:resource="Permission"/>
            <rdfs:domain rdf:resource="Role"/>
            <rdfs:domain rdf:resource="User"/>
            <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
```

Figure 2.5 shows RDFS for Core RBAC submodel as a UML class diagram [UML]. The complete Core RBAC specification is located as part of $RBAC_3$ or Full RBAC RDF schema which is in Appendix A.

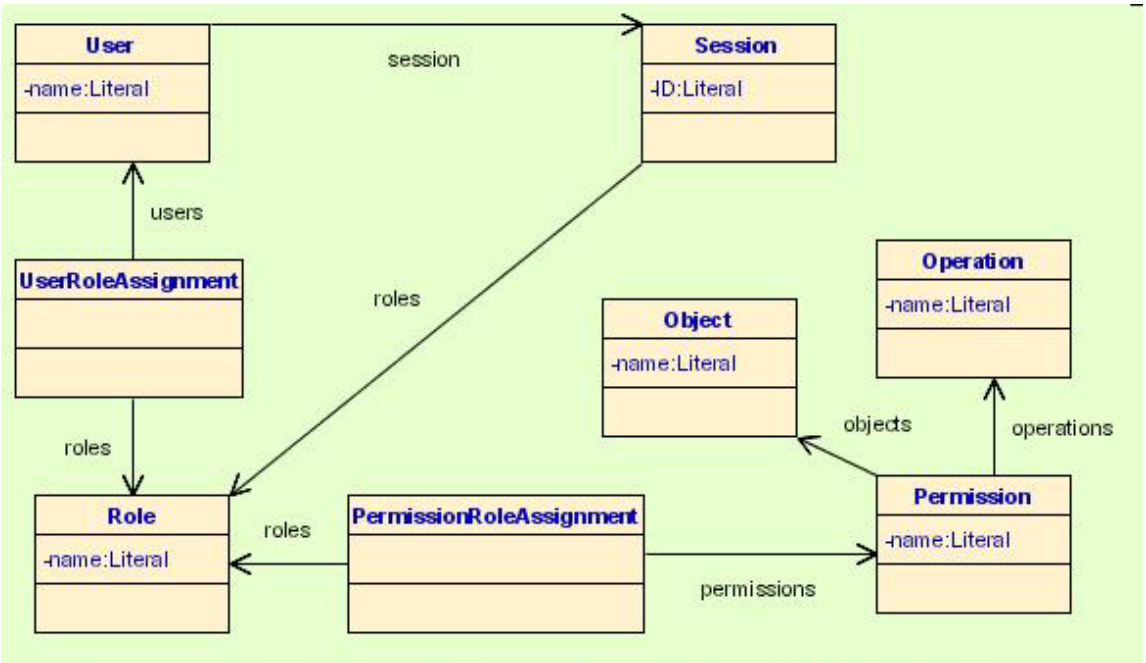

Figure 2.5 – Core RBAC RDF schema classes and properties UML representation

2.2.3 Design of OWL ontology for Core RBAC

OWL ontology captures the semantics of the Core RBAC submodel and is represented in a figure 2.6. It also contains information that is additional to RDFS specification. For instance owl:inverseOf property which can point on relation between properties if association exists between instances of two classes.

Thus the main difference of OWL ontology to RDFS is that Core RBAC relations are specified as classes in RDFS and as associations through class properties and owl:inverseOf property's characteristic in OWL.

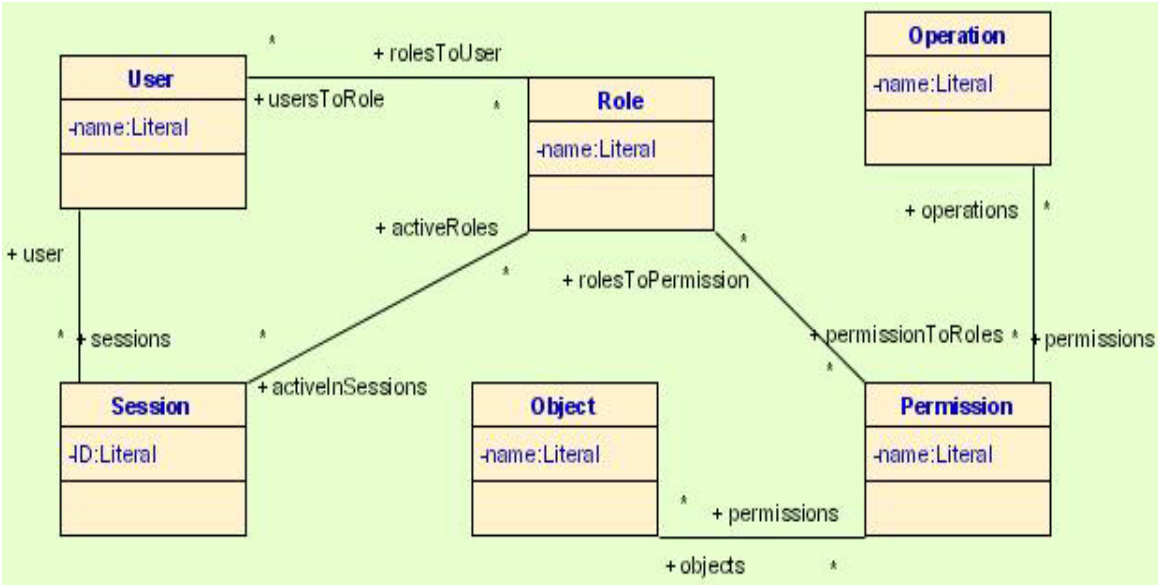

Figure 2.6 – Core RBAC OWL classes and properties UML representation

The following below in the thesis figures for illustration of RDFS and OWL RBAC specifications have the same notation to depict classes, properties and domain, range, inverseOf characteristics of properties as figures 2.5 and 2.6 have.

2.3 Hierarchical RBAC

2.3.1 Description and analysis

In real world applications of RBAC organizations can have users with different job functions but with intersected sets of privileges within some information system. Often some set of permissions should be assigned to large number of users. Features to make possible inheritance of permissions and user membership among roles make the assignments of users and permissions more efficient in RBAC. Therefore, hierarchical RBAC adds role hierarchy to enrich core RBAC model. The notion of role hierarchy is used to describe the natural hierarchy of authority and responsibilities within organization. Mathematically hierarchy is a partial order defining a seniority relation between elements. Changes to RBAC model from adding the hierarchy are shown in figure 2.7. From organizational point of view the traditional hierarchy of roles is built from top to down and higher roles are more authoritative (seniors) then lower ones (juniors).

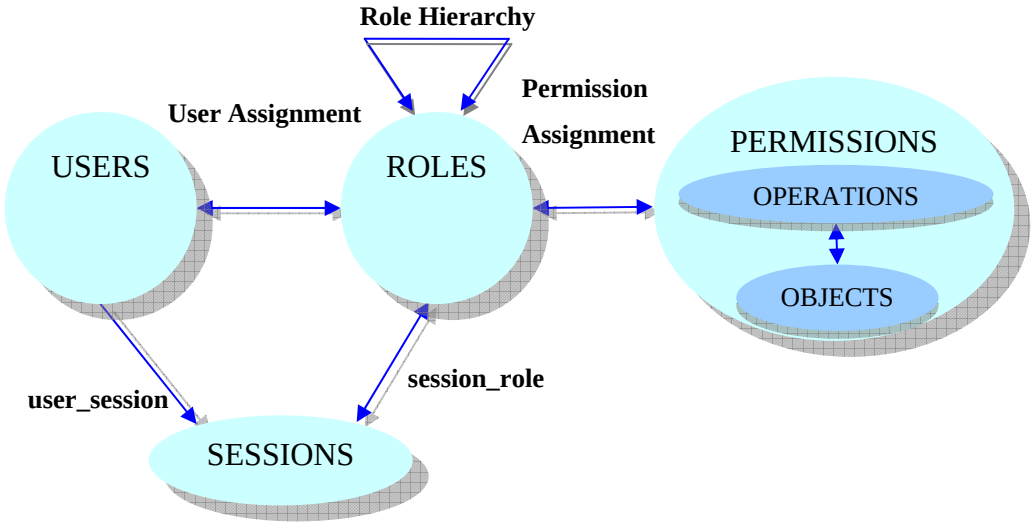

Figure 2.7 – Hierarchical RBAC with role hierarchy

Structuring roles in RBAC defines relations of inheritance within roles. Two approaches exist to describe inheritance relation among roles. First one has been defined as permissions inheritance. A role inherits a second role if all privileges of the second one are also privileges of the first role. Within traditionally built role hierarchy inheritance of permissions among roles occurs from higher roles to lower ones. This kind of inheritance relation is shown by figure 2.8. So role R3 inherits roles R4 and R5. Role R1 inherits roles R2 and R3. Thus R1 inherits all privileges of R2 and R3 role, which inherits all privileges from roles R4 and R5. Arrows denote relation of roles inheritance.

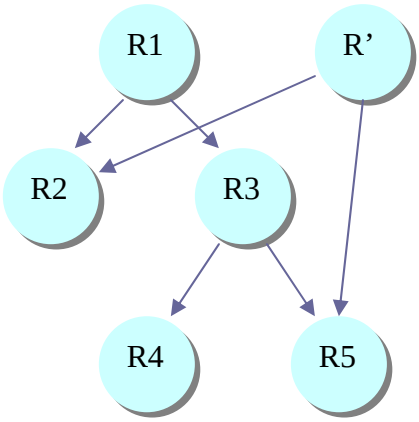

Figure 2.8 – Permissions inheritance within role hierarchy

Second approach to describe relation of inheritance among roles has been defined in terms of user containment. First role contains second role if all users authorized to second role also authorized to first one. In figure 2.9 arrows denotes relation of user containment, this figure describes situation when R2 and R3 roles contain at least all users of role R1, and roles R4 and R5 contain all users of role R3.

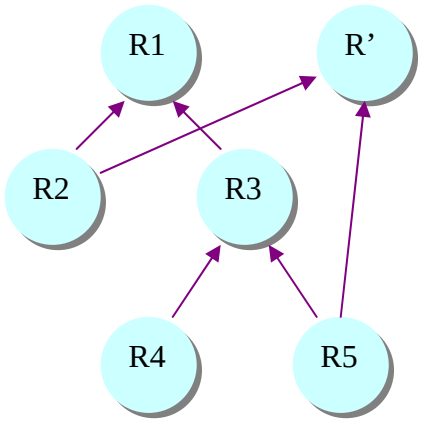

Figure 2.9 – User containment relation within role hierarchy

To reduce the complexity of building role hierarchy, RBAC introduces two types of role hierarchies. First one is general role hierarchy which implements well known concept of multiple inheritances. Thus general role hierarchy can describe multiple inheritances of permissions and user membership among roles. It means that the general role hierarchy supports situations when a role can inherit permissions from two or more roles by permissions inheritance relation, and role can inherit user membership from more than one role. Such support gives great flexibility to general role hierarchy in designing hierarchy. Role as a set of permissions can be derived by inheritance from multiple lower roles which can be viewed as subsets of given role permissions. Note that in RBAC standard lower roles are intended to reflect corresponding organizational and business structures instead of being just named set of permissions. Finally, user containment relation or user inheritance makes it possible to consider user assignment and user inheritance as one relation. In such case user can be assigned to role by user inheritance. Role can either inherit other role or user in sense of user inheritance.

The case when role hierarchy restricts roles to have only one immediate descendant is called limited role hierarchy. Limited role hierarchy implies that the structure of the hierarchy is a tree structure.

Examples of general role hierarchy and limited role hierarchy can be found in figure 2.8 and 2.9 (without considering role R' in figure 2.9). In figure 2.8 each role R1, R3 and R' has relation of permission inheritance with two roles. In figure 2.9 no role except R' has relation of user inheritance (user containment) from more than one role.

2.3.2 Role hierarchy specification in the RDF schema

Role inheritance defines a binary relation between roles and can be defined in RDF schema as a resource RoleInheritance which has properties fromRole and toRole. The following specification can be added to core RBAC RDF schema specification to support role inheritance by the schema.

```
<rdfs:Class rdf:about="RoleInheritance" rdfs:label="RoleInheritance">
            <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdf:Property rdf:about="fromRole" rdfs:label="fromRole">
            <rdfs:domain rdf:resource="RoleInheritance"/>
            <rdfs:range rdf:resource="Role"/>
</rdf:Property>
<rdf:Property rdf:about="toRole" rdfs:label="toRole">
            <rdfs:domain rdf:resource="RoleInheritance"/>
            <rdfs:range rdf:resource="Role"/>
</rdf:Property>
```

This specification can be shown by graph representation as it is done in figure 2.10.

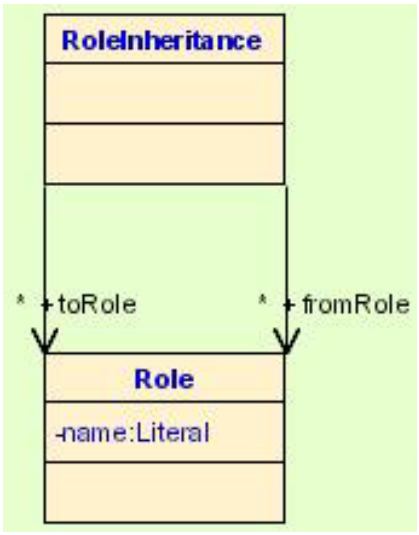

Figure 2.10 – Hierarchical RBAC RDF schema additional classes and properties

2.3.3 Role hierarchy specification in OWL ontology

To define role inheritance relation in OWL it is enough to specify two properties of class "Role" "ascendants" and "descendants" that are inverse to each other as figure 2.11 illustrates. OWL provides also possibilities to encode the additional semantic characteristic of transitivity. Limited hierarchy can be achieved by restricting the cardinality of the "ascendant" property to be not more then one.

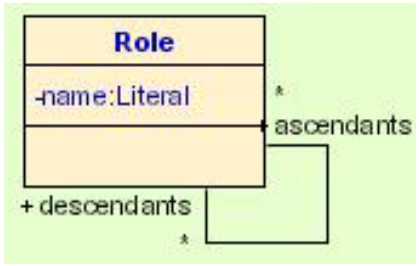

Figure 2.11 – Hierarchical RBAC additional OWL classes and properties

2.4 Constrained RBAC

2.4.1 Description and analysis

Typically administrators perform user to role assignment. User assignment should be performed with awareness of business rules which separate responsibilities and authorities among personnel. Since administrators are not expected to be experienced in sophisticated business rules such as conflict of interest, RBAC model should contain tools to describe this explicitly.

Thus, constrained RBAC extends RBAC model by adding separation of duty relations. Purpose of the separation of duty relation is to express a policy restriction that prohibits granting a user more privileges than reasonable according to organizational status. Constrained RBAC introduces two relations, static and dynamic separation of duty. These constraints restrict indirectly user assignment to roles.

Static separation of duty (SSD) is proposed in RBAC model to define mutually disjoint user assignments with respect to sets of roles. This relation should describe situation where user can not be assigned to roles which are in conflict of interest with already assigned roles. If static separation relations are defined centrally in organization and enforced in RBAC, administrator's attempts to assign a user to conflicting roles will fail and business rules will be respected.

SSD is defined in RBAC as a set of roles and a cardinality greater than one which indicates the number of roles that would violate the SSD relation when assigned to user. Such a definition of SSD can describe wide variety of static separation policies, but in practice it is mainly used to implement a policy where user can not be assigned to some pair of roles simultaneously. For instance when some role has permissions to run a transaction and an other has permissions to acknowledge it. From business point of view these roles are mutually exclusive and should not be assigned to one user simultaneously.

Formalization of SSD depends on the existence of role hierarchy as illustrated in figure 2.12. If role hierarchy is applied, a user can be authorized to a role through user inheritance relation among roles. Generally SSD relation can be defined in two ways:

- The role hierarchy can include constraints on inheritance relation among roles and push hierarchical RBAC to become more complicated and thus more intersected with constrained RBAC. For instance users and permissions inheritance is constrained by mobile and immobile membership concepts which are described as essential part of the Administrative RBAC model in chapter 3.

- Second option is to define SSD constraining authorization of users to roles instead of assignment of users as in the case of flat roles.

RBAC standard follows the second option to maintain granularity of RBAC model components.

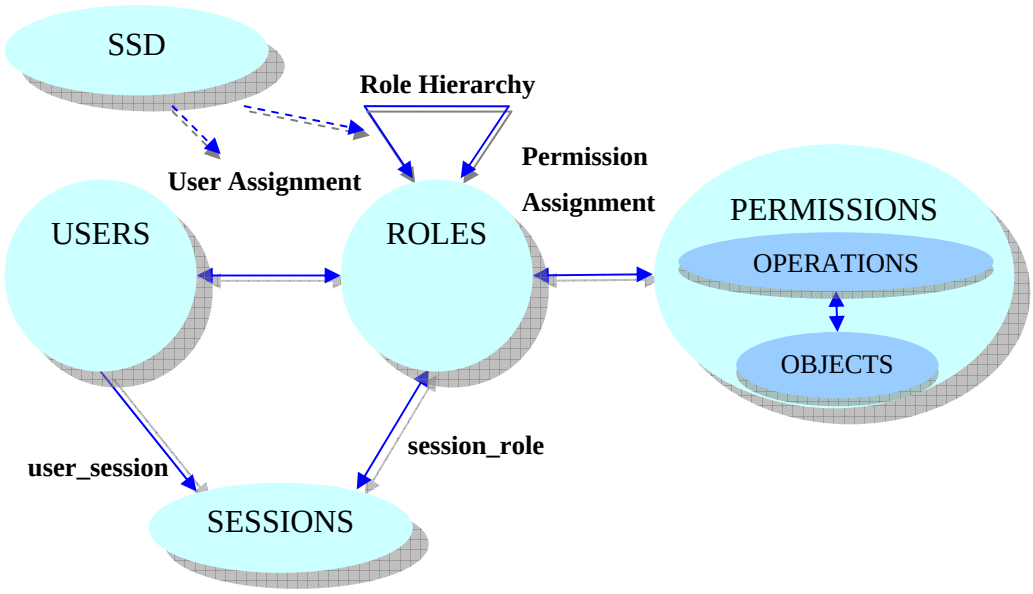

Figure 2.12 – SSD relation in hierarchical RBAC

Dynamic separation of duty (DSD) has almost the same nature as static separation in the sense of restricting the user permissions. DSD differs from SSD in point of time when these policies are applied. As mentioned above the SSD relation is used when user is assigned to a role. DSD is introduced to restrict user from activating the assigned roles within one session. Figure 2.13 illustrates the place of DSD relation and its influence on role activation.

The time when DSD is applied is not the only difference from SSD. SSD relation restricts possible variants of user assignment with respect to whole set of roles. DSD relation restricts user to gain permissions by simultaneously activating roles to which user has been already assigned.

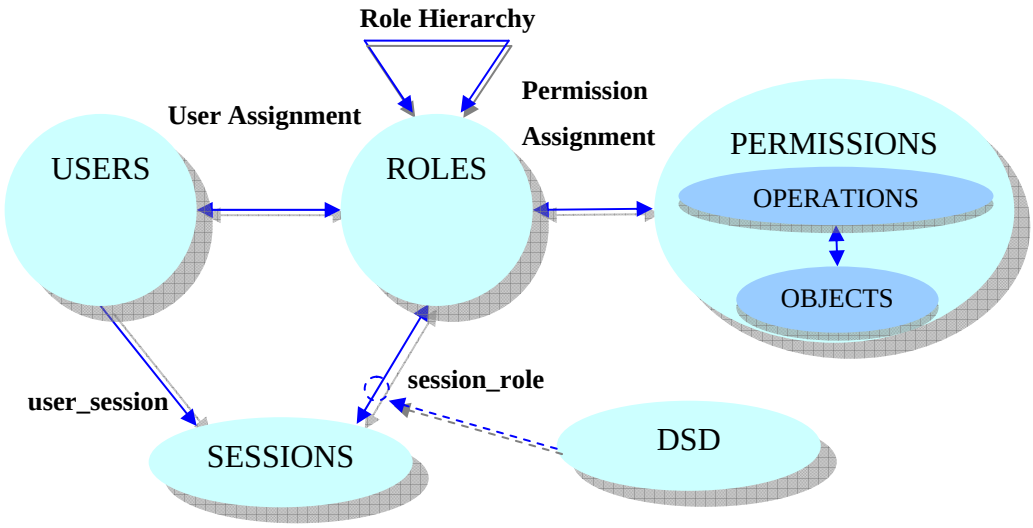

Figure 2.13 – DSD relation in constrained RBAC

Proposed dynamic constraint helps RBAC systems to respect the principle of least privileges dynamically. User has only the permissions to perform the duty related to some job function only in time of performing of the duty and does not have unnecessary permissions.

DSD relation is defined within RBAC model analogically to SSD relation. So it has two elements. First is set of roles that can contradict each other. Second is natural number, greater or equals to two, which denotes that no user can activate such number or more roles from DSD set of roles in one session.

2.4.2 Specification of constraints in the RDF schema

To specify the separation of duty constraints it is enough to add to RBAC RDF schema two classes “SSD” and “DSD”, properties “name”, “cardinality”, and “roles” which describes

the set of mutually exclusive roles. Figure 2.14 shows graph representation of additional classes and their properties.

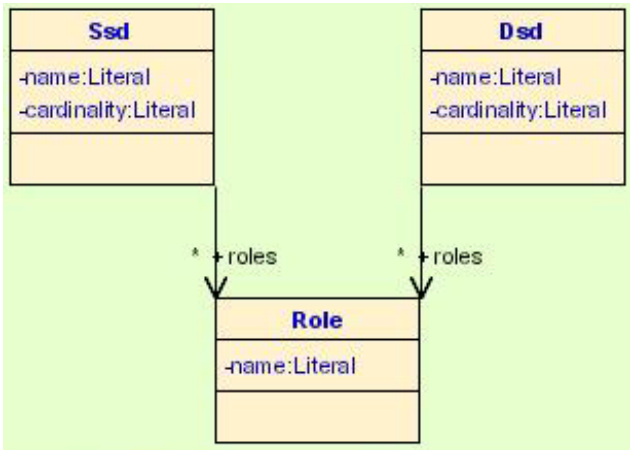

Figure 2.14 – Constrained RBAC RDF schema additional classes and properties

2.4.3 Constrained RBAC specification in OWL ontology

The OWL specifies classes of SSD and DSD as well. It also provides more semantics about the association between constraint's role sets and roles by inverseOf characteristic. Figure 2.15 depicts OWL specification of Constrained RBAC additional features.

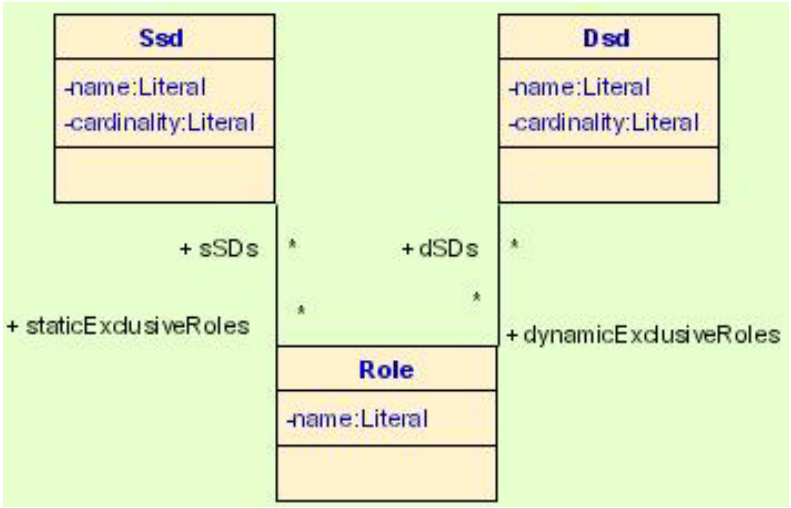

Figure 2.15 – Constrained RBAC OWL additional classes and properties

2.5 Comparison of RBAC model specification by RDF schema and OWL

As it is stated on Semantic Web activity site [SemanticWeb] OWL is more expressive in semantic specification than RDF. For RBAC reference model specification this advantage results in a more sophisticated RBAC model semantic definition. Differences between OWL and RDFS which were used in this chapter are the following:

- Thanks to owl:inverseOf characteristic of property in OWL, RBAC relations are expressed as associations without necessity to define classes like in RDFS. This gives an additional advantage in implementing of the review functionalities of RBAC because of conceptually simpler dependencies between instances of user, role and permission. For instance a particular role as an instance of owl:Class contains itself links to user instances which are assigned to it and user instances have direct links to roles.

- Definition of ascendant and descendant properties for role class embodies the additional semantic of transitivity of inheritance relation in OWL specification.

- The OWL supports cardinality restriction which is useful for the name property and for specification of the limited role hierarchy

2.6 Chapter summary

The conclusion of this chapter is that OWL is more appropriate for RBAC models specification in comparison to RDFS. On the other hand RDF is an older technology for semantic description and thus has more tools to work with it.

Anyway, RDFS and OWL can be used together and supplement each other. But on my opinion, the better the semantics are defined the better will be unification of understanding and the easier will be integration and further extension of RBAC ontology. Thus OWL is to be preferred in case when abstract, flexible and expressive language is needed.

Issues about particular usage of RBAC data encoded according to RDFS and OWL RBAC specification are points of consideration of chapters 4, 5, and 6.

3 RBAC data administration models

For a system with large number of administrators the delegation of administrative authority over the RBAC concepts should be defined. For this purpose it looks very attractive to use the RBAC model itself to administrate RBAC data.

Sketch of properties and requirements of ideal RBAC based model for administration of RBAC data can help in comparison of current research results in this area. Some of the requirements are quite subjective, but at least they give a starting point for discussion.

- Completeness. Ideal model should formalize administration of all RBAC concepts instead of providing guide for administration of some part of RBAC model.
- Simplicity. The administration model should not require new additional concepts.
- Recursive Definition. Ideal model should support possibility to construct multilevel administration using RBAC model for administration of RBAC data on previous level.
- Continuity. This is complex feature which is met if terms of RBAC reference model used; if there is no unproved changes in RBAC reference model concept definitions and functional specification; if RBAC reference model is applied to itself recursively instead of unnecessary introducing variety of new concepts; etc.

3.1 Administrative RBAC

3.1.1 ARBAC97 model analysis and semantic specification

After introducing the RBAC96 model [Sandhu1996], a model of role based administration of RBAC was proposed [Sandhu1997]. It was called Administrative RBAC97. ARBAC97 concentrates on three components of RBAC – assignment of the users and permissions to the roles and building of the role hierarchy. Structure of ARBAC97 is illustrated in figure 3.1.

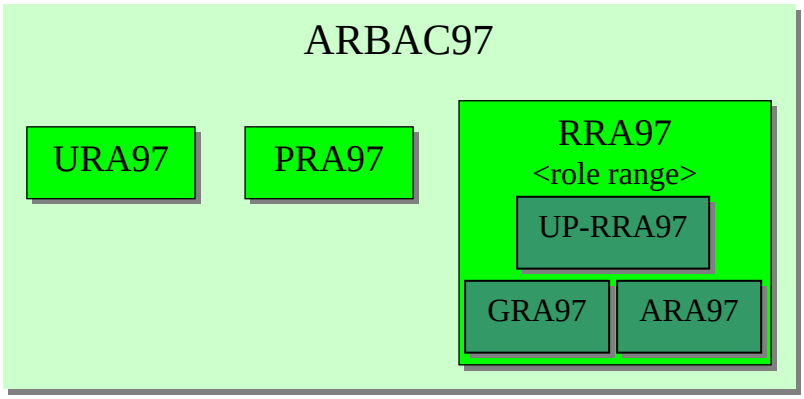

Figure 3.1 – Submodels and concepts of ARBAC97

URA97, PRA97 and RRA97 submodels are intended for administration of user to role, permission to role and role to role assignments respectively as the relations of RBAC96.

URA97 and PRA97 both introduce definitions of prerequisite conditions and relations of assignment and revocation authorization of users and permissions respectively. URA97 and PRA97 are dual in the sense of used formalisms and they differ only in user or permission assignment context.

RRA97 recognizes three types of roles that are abilities, groups and UP-roles with corresponding components for their administration ARA97, GRA97 and UP-RRA97 respectively [Sandhu1998]. Role concept is decomposed to types with criteria of user and permission membership. Abilities can have only permissions and other abilities as members. Groups can have only users and other groups as members. UP-roles do not have restrictions on membership. GRA97 and ARA97 have the same duality as URA97 and PRA97 models. Also GRA97 and ARA97 are derived from URA97 and PRA97 respectively because assigning groups to roles is very similar to assigning users to role and analogically for abilities and permissions to role assignment.

ARBAC97 changes the RBAC96 model and the RDF schema specification and OWL ontology specification for RBAC reference model should be modified to support ARBAC97 vision. Figure 3.2 shows the specification of ARBAC97 elaboration of RBAC reference model by introducing new role types. On this figure an URA is the UserRoleAssignment, a PRA is the PermissionRoleAssignment; Group, Ability and UP-

role are subclasses of Role; UP-RRA, ARA and GRA are concepts used to form the role hierarchy.

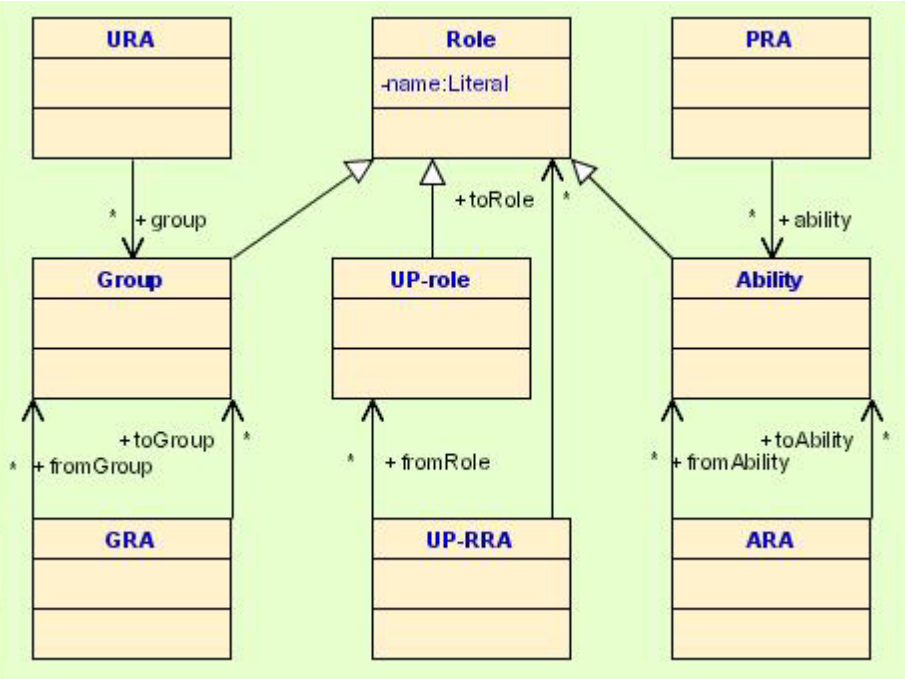

Figure 3.2 – RDFS specification of user role types in ARBAC97 vision on RBAC model

Only operations of assigning or revoking of users and permissions and operations of modification for roles are managed by ARBAC97

ARBAC97 defines a prerequisite condition and a role range which becomes the authority range in RRA97 model to manage the role hierarchy [Sandhu1998]. As a permission points to the operations on the objects, then the prerequisite condition and the role range concepts are needed to define the set of administrative objects to create administrative permissions.

Prerequisite condition is a boolean expression which consists of roles and logical operations. It is used to define the set of users in the case of URA or the set of permissions in the case of PRA on which, the administrative role can operate (assign, revoke).

Role range is a short notation to define the set of roles as objects on which administrative role can perform operations. Administrative role within ARBAC97 can get authority to assign, revoke users, permissions and roles to roles in role range and modify roles of role

range. Administrative role can have multiple role ranges according to its authority and responsibility. Figure 3.3 illustrates ARBAC97 specification using RDFS language.

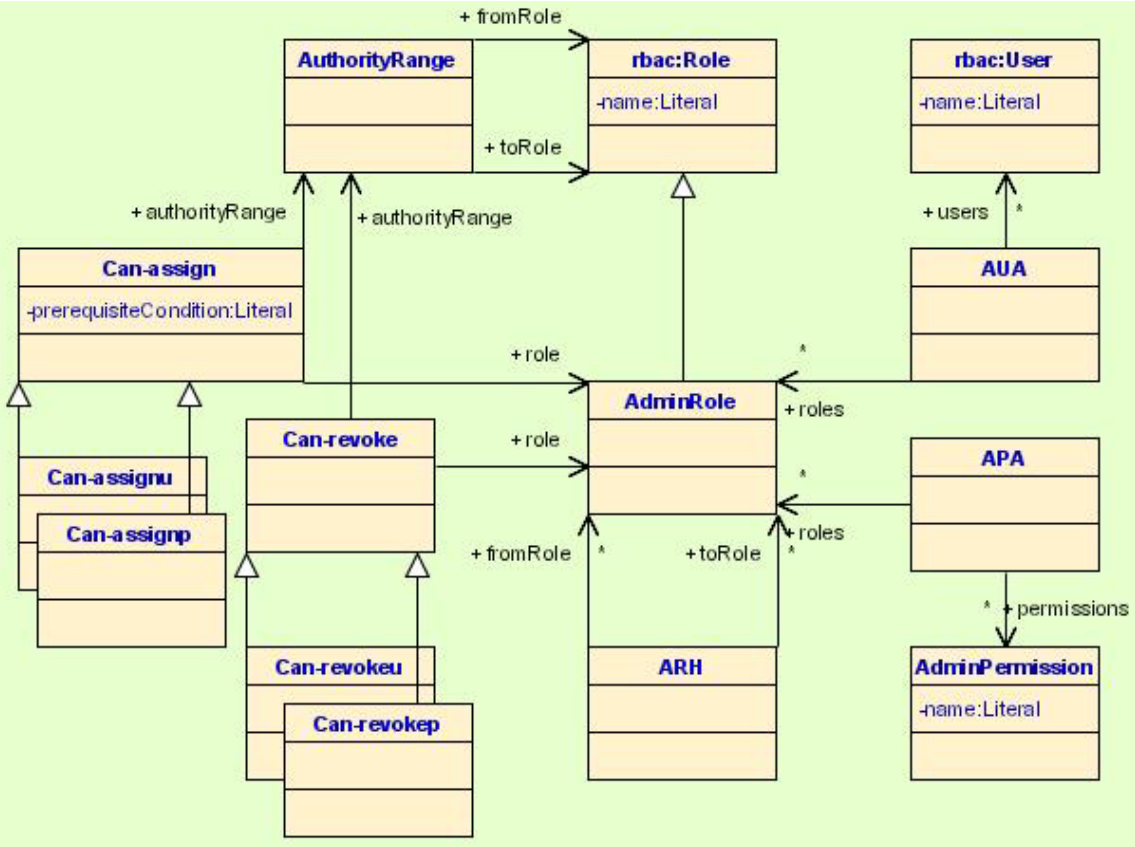

Figure 3.3 – ARBAC97 model specification using RDFS language

On the figure above ARH is a Administrative Role Hierarchy; APA is a Administrative Permission Assignment; AUA is a Administrative User Assignment; rbac:Role is a class of RBAC model roles from “rbac” namespace; rbac:User is a class of RBAC model users from “rbac” namespace; Object is a base class for rbac:Object, rbac:Operation, rbac:Permission, rbac:Ability, rbac:Group, rbac:UP-role, rbac:DSD, rbac:SSD, rbac:PRA, rbac:URA, rbac:User, rbac:ARA, rbac:GRA, rbac:UP-RRA; AuthorityRange can contain four instances of Range class which express open, closed, left-side open and right-side open intervals. Can-assignu and can-revokeu classes depict relations to authorize user assignment and revocation by administrative roles. Can-assignp and can-revokep illustrate similar relations for administration role authority over permissions assignment. Classes exist for groups assignment (can-assigng, can-revokeg), abilities assignment (can-assigna,

can-revokea), and can-modify relation which authorizes manipulation within UP-roles. But they are not shown on this figure because they are identical to can-assignp and can-revokep classes.

Weaknesses and shortcomings of this family of models are:

- Incompleteness. ARBAC97 does not support all RBAC components as objects of access control
- Unrecursive definition. Administrative role hierarchy and administrative permissions, administrative user and permission assignment are defined according to RBAC concepts but as the separate components from RBAC components.
- Complexity. New concepts are defined to create administrative permissions in sense of identifying RBAC objects and ARBAC operations.
- Inefficiency. The task of user and permission assignment brings redundant work for administrators. Instead of assigning user to the most top role according to user authority, he will be assigned several times to satisfy prerequisite conditions.
- Redundancy. Multi-step user assignment, additional classes of roles bring redundancy in RBAC data.

3.1.2 ARBAC99 model analysis and semantic specification

ARBAC99 introduces two kinds of user and permission memberships in a role [Sandhu1999]. First, which is called a mobile membership has the same properties as in ARBAC97. Second, immobile membership restricts users and permissions to be assigned further in spite of prerequisite conditions. So this model inherits all shortcomings of ARBAC97 model, and concentrates on restricting user and permission distribution among roles. Notion of these four kinds of membership can be considered as some type of RBAC constraint which gives more rich and flexible control over administrative authority for user and permission to role assignment. Figure 3.4 depicts ARBAC99 submodels and introduced types of membership.

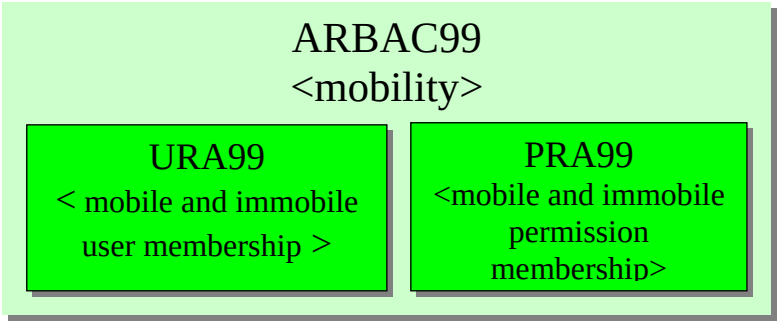

Figure 3.4 – Submodels and concepts of ARBAC99

It is enough to specify additional concept Mobility which has two instances – mobile and immobile membership. Then ARBAC99 model also requires changing RBAC model specification because instances of URA and PRA relations should refer to instances of mobility. Figure 3.5 illustrates additional metadata specification through RDFS class for mobility concept and properties for user and permission assignment relations.

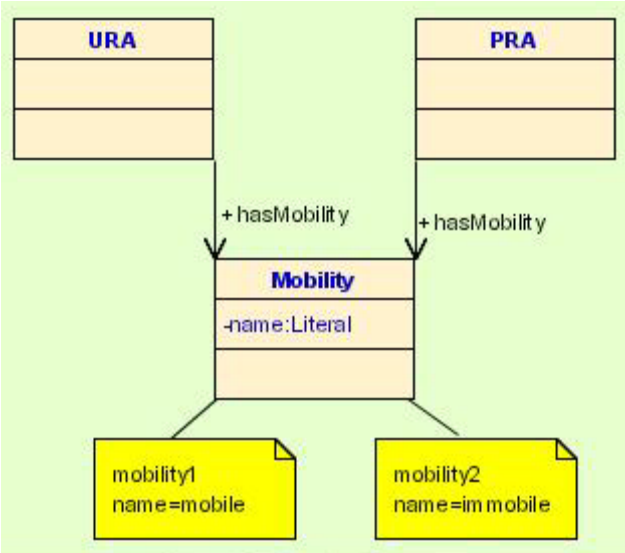

Figure 3.5 – Mobility concept RDFS specification

3.1.3 ARBAC02 model analysis and semantic specification

The ARBAC02 model was proposed in [Oh2002]. It extends prerequisite condition by concepts of user and permission pools. Good overview of URA97 and PRA97

shortcomings related to administration of user and permission membership is given in the article. Figure 3.6 shows submodels and concepts of ARBAC02.

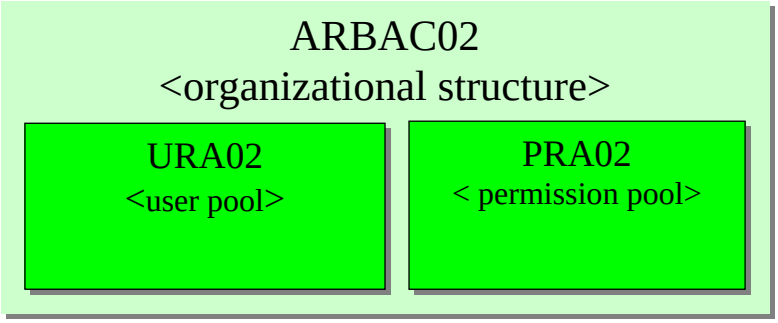

Figure 3.6 – Submodels and concepts of ARBAC02

Organization structure is used as user and permission pools which have the hierarchy of the organization formed by organization units. In ARBAC02 the prerequisite condition uses the organization units to define sets of users or permissions under which administrative role has authority. ARBAC02 solves shortcomings of multi-step user/permission assignment. It is evident that same results can be achieved using ARBAC97 by constructing a hierarchy of groups as a user pool and a hierarchy of abilities as a permission pool in a way to correspond to the organization's structure and by restricting the prerequisite condition to be defined using roles from pools. Figure 3.7 illustrates specification of ARBAC02 model concepts in RDFS language. Appendix C contains the RDFS ontology of all three ARBAC models.

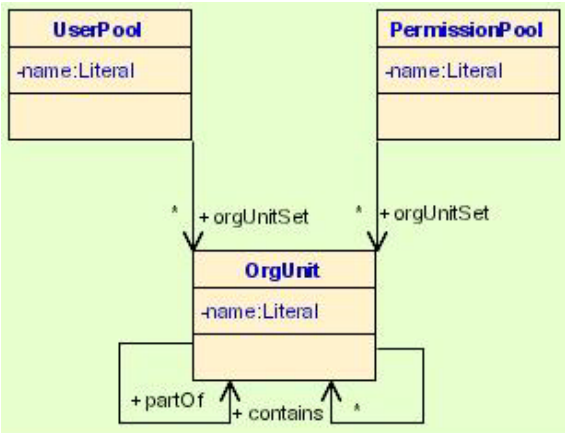

Figure 3.7 – RDFS specification of ARBAC02 model concepts

3.2 Scoped administrative RBAC

The notion of administrative scope was introduced in [Crampton2002] in contrast to the concept of role range. Administrative scope is defined using operations from the theory of sets, operations borrowed from the partial order theory and using the role hierarchy structure. It changes dynamically during modifications of the role hierarchy. Family of four models RHA1, RHA2, RHA3 and RHA4 is proposed in the context of administration of the role hierarchy (operations on roles: addRole, deleteRole, addEdge, deleteEdge). These models are based on the administrative scope as a concept to define set of objects (roles) under which the administrative role has authority.

Later RHA4 model was extended to include user and permission assignment administration to the model. The resulting model was called SARBAC (scoped administration of RBAC) [Crampton2003]. In this article good comparison of SARBAC and ARBAC97 models is given. Figure 3.8 depicts RDFS specification of SARBAC model elements and relations.

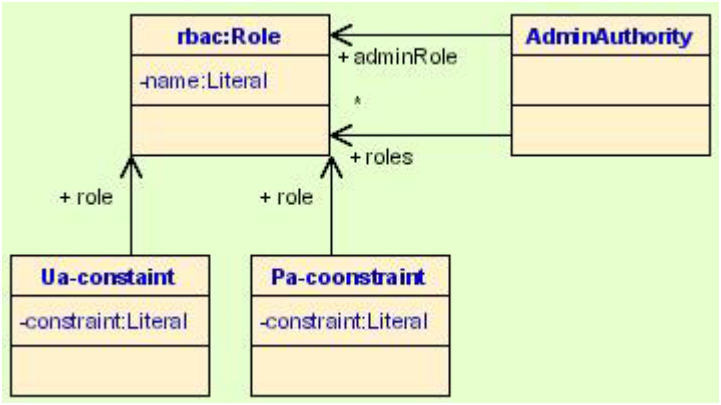

Figure 3.8 – SARBAC elements and relation specification

SARBAC also as ARBAC97 has a problem of multi-step user and permission assignment. Generally SARBAC model is more attractive than the family of ARBAC models, because of

- SARBAC has smaller number of components and the concepts are simpler.

- There are no disjoint sets of RBAC and administrative roles and permissions, so RBAC concept of role is used for specification of administration roles as well as RBAC concept of permission is used for specification of administrative permissions. Thus SARBAC applies RBAC concepts recursively.

- SARBAC concepts for specification of administration authority over RBAC data have dynamic nature

- SARBAC has more applicability in the role hierarchy structures and also in user and permission assignment administration

- It provides support of large amount of role hierarchy operations

- Operations based on administrative scope are more intuitively understandable than ones based on encapsulated role range

3.3 Role Control Center administration concept

So far two concepts to define the sets of objects for administration, particularly the sets of roles have been considered. Those are role range and administrative scope which can be used within arbitrary partially ordered sets. A third possibility was introduced in article [Ferraiolo2003b] and is called role view. This article was dedicated to describing features of Role Control Center (RCC).

RCC proposes to administrate RBAC data using role view and role graph. Role graph represents a hierarchy of roles. Users are included in the graph. Relation of inheritance of permissions used in the role graph of RCC constitutes both relations of RBAC which are user to role assignment and role inheritance. So in the role graph only inheritance relation is used as type of arc. Users can inherit roles and roles can inherit other roles.

In a role graph, role view is defined by a set of roles. This set of roles forms a sub-graph of the role graph. Sub-graph consists of roles in the defining set and all roles which transitively inherit these roles.

Role view concept allows specifying sets of roles and users easily by selecting the most general roles to form a role view for administration. As RCC reuses ARBAC model for administration, only the view concept has to be specified (Figure 3.9).

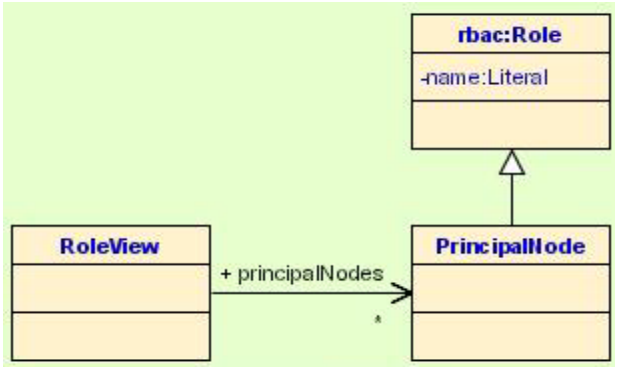

Figure 3.9 – Graph representation of RCC concepts specification

RCC covers only the administration of roles and user memberships. So this system does not implement administration of all RBAC concepts and relations. Also, in RCC a role can delegate permissions only in the case when these permissions are assigned to the role within permission to delegate. Thus RCC does not support the right to delegate without the right to use permission.

3.4 Chapter summary

The ARBAC family of models has a lot of shortcomings and weaknesses. These models are incomplete, complex in the sense of number of new concepts which were introduced. They assume changes in RBAC model. Also ARBAC models bring redundant steps in user and permission assignment process. In spite of the above mentioned ARBAC models provide valuable research in the field of RBAC administration understanding and modeling.

SARBAC scope and RCC role view concepts are more natural and easer to use than the role range in ARBAC. SARBAC can be considered to be more complete than the others. RCC can also serve as an implementation example of the administration of RBAC data using RBAC model. Specification of the elements and relations of SARBAC and ARBAC

models is not as important as their operational semantics. Issues about operational semantics of administration of RBAC data functionalities are considered in the next chapter.

Basically all RBAC administrative models follow the RBAC reference model. Some of the elaborations introduce new concepts and change the RBAC reference model. The main difference between administrative models is the approach to express sets of RBAC concepts to delegate authority among administrative roles. From such a point of view, the role scope and the role view look more attractive and natural to operate with. My opinion is that none of the above mentioned models looks like ideal model for RBAC data administration. Main reason for this conclusion is the unclear definition of the functional part of administrative models. The concepts and relations in the models are defined rigorously but still for each model something remains uncovered with the functional specification or there is incompleteness in support of RBAC standard as a whole. Thus development of ideal administrative RBAC model and functional specification remains as a task for further research. It is attractive to design such model in a level of semantic description to allow high scale of integration possibilities for RBAC elaborations.

Remark: it seems that RDFS language capabilities are not enough to express the semantics of recursive application of RBAC model for RBAC data administration. Thus further I will concentrate on OWL for semantic specification and integration of RBAC models.

4 Enterprise access control using semantic specification

There are a lot of implementations and publications on the use of RBAC in particular systems like database management systems, operating systems, network file systems or web-based business applications. It is more seldom to use RBAC model for enterprise wide solutions. In what follows I will refer to such solutions as Enterprise Security Management System (ESMS) [Ferraiolo2003a] and RCC can be a good example of a component of such system. Heterogeneous types of resources and environments results to high complexity of implementation of ESMS.

4.1 Enterprise access control overview

An enterprise access control (EAC) framework (EAF) [Ferraiolo2003a] defines components and functionality of ESMS.

Any arbitrary access control system can be considered to have two components:

- Access control policy specification component
- Access control enforcement mechanism

Policy specification component embodies some access control model to specify the policy using the terms of this model. Access control model describes all modes of access to resources. Enforcement mechanism implements the policy controlling subject to object access process.

EAC deals with a variety of access control systems which differ from environment (platform, business application, etc.) to environment. I will refer to such systems, their components and environments by the term native.

Native systems (NS) have their own implementation of the policy specification (model) and enforcement mechanism. Thus the main challenge of EAF is to integrate native features and their implementation to allow administration of access control on higher level of abstraction.

EAF approach provides centralized specification and management of access control in enterprise:

- EAC model – formalism for platform independent policy and requirements specification
- EAC data that corresponds to EAC model
- Tool sets to manage EAC model and data, and to map that data to host systems and environments

4.2 Enterprise RBAC model

Enterprise RBAC (ERBAC) is an EAC model based on RBAC reference model. Enterprise role is the main concept of ERBAC [Kern2002a] as usual role is for RBAC.

This role gathers all corresponding roles in native systems and thus collects all their permissions. It is obvious that the definition of enterprise role differs from the RBAC role only semantically, thus constraining the role notion to be on enterprise level, and does not differ in the sense of formal description. ERBAC itself does not authorize users but instead it manages native access control systems to keep EAC consistent. That is why ERBAC does not support the session concept. In the highest level of abstraction ERBAC can be illustrated in figure 4.1 which depicts conceptual vision of ERBAC model with sets of main elements and relations between them. Target system (TS) has almost the same meaning as the native system mentioned above, but with the emphasis that the native system is the target of policy propagation.

Enhanced ERBAC introduces advanced features to ERBAC [Kern2002b]. The multiple possibilities to build role hierarchies based on some criteria of role decomposition lead to creation of different role hierarchies which are connected between each other by multiple inheritances. This makes administration of the role hierarchy complicated. One solution to reduce the complexity is to create parameterized roles. Alex Kern enhanced ERBAC model to parameterize roles by introducing attributes and rules. In such a situation role to

user, user to permission and role-to-role assignments have attributes which specify additional information. User attributes mainly describe personal information, organizational status (unit, position, etc.) and constraints for role assignment. The notion of rules is not defined rigorously but execution of all functions to manipulate with RBAC data should be verified against corresponding rules.

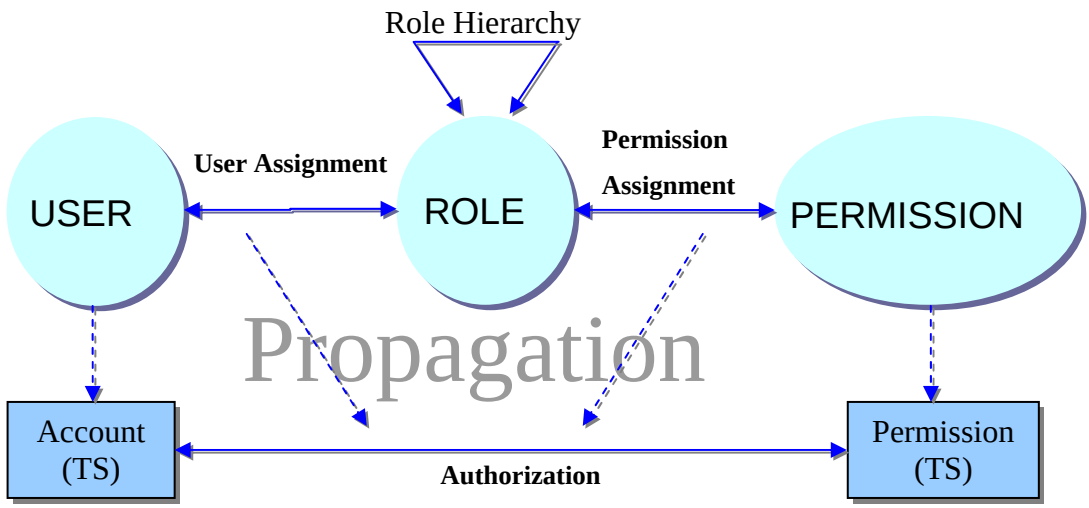

Figure 4.1 – ERBAC model main elements and relations

Later the administrative ERBAC (A-ERBAC) model was proposed [Kern2003]. This model recognizes the problem that all components of RBAC should be some how collected under administration authority, not only roles and membership relations as in ARBAC model family. A-ERBAC considers ERBAC as a target system, reuses concepts of ERBAC model and introduces the concept of scope. Figure 4.2 shows the A-ERBAC model.

Objects here are the ERBAC concepts:

- User
- Role
- Account
- Permission

- User-role assignment (URA)
- Role-role assignment (RRA)
- Permission-role assignment (PRA)
- Static Separation of Duties (SSD)

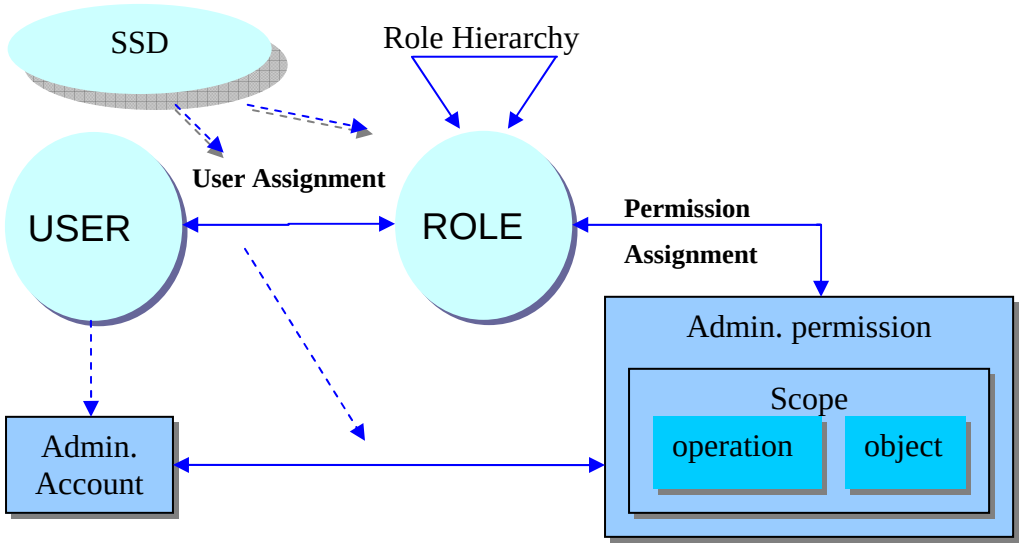

Figure 4.2 – A-ERBAC model vision

Administrative accounts and permissions are considered to be regular ERBAC objects, thus they should be administrated as other accounts and permissions.

A-ERBAC defines a set of four operations to access ERBAC objects. These are view, insert, change and delete. As A-ERBAC incorporates enhanced ERBAC model, then view and change operations are also executable on object's attributes. Object's attributes are always inserted and deleted together with the objects.

So the scope defines the ERBAC objects that are subjects of administrative authority. Scopes can form hierarchies themselves. A particular scope is defined over a hierarchy of scopes using combination of nodes, trees and "exclude" concepts. A node represents an atomic scope, tree defines a sub-tree under some node, "exclude" specifies set of nodes using atomic scopes of nodes and trees that should be excluded from the result scope.

Atomic scope specifies administrative authority over objects explicitly by manual selection of objects or implicitly by using object's attribute information.

Thus administrative permission consists of operations, objects (object types) and scopes which define objects themselves.

Scope concept in A-ERBAC brings redundancy, because it represents the same notion as role and can be replaced by role concept.

4.3 Use of OWL and RDF for implementation of Enterprise AC

4.3.1 EAC policy specification language

The EAF has to provide a language expressive enough for the EAC model, in our case ERBAC model. This subchapter evaluates OWL and RDF languages for expressing semantics as candidates for AC policy specification.

On my opinion ERBAC is a good EAC model but it also has to be richer in the sense that ERBAC needs to support different elaborations of RBAC and their administrative models and concepts. So in such a definition ERBAC model grows to an integrative ERBAC or EAC model which integrates all RBAC elaborations.

Conditions of heterogeneous systems, environments, resources and huge amount of users in Enterprise, for which a consolidated AC management system is appropriate, define some set of requirements for the policy specification language characteristics. The policy specification language has to be expressive enough to support formal model of AC and domain specific restrictions and features. From one side the language is expected to be used by humans with different qualifications. Thus it must provide an abstract view of AC for a regular user who usually makes formal decisions and concrete and detailed views for support security personnel. From other side ESMS supposes to provision all information of the EAC model to target systems. Thus language has to be platform independent and widely supported. Wide support of a specification language means availability of tools for editing, transforming, parsing, validating, transmitting, and etc.

Semantic Web standards [SemanticWeb] are considered as policy specification languages to provide the integration feature to EAC model. These languages also meet all the requirements which are mentioned in the previous paragraph.

4.3.2 Semantic relations of RBAC elaborations in EAC model

EAC model here is the whole set of RBAC elaborations starting from RBAC reference model, ARBAC family, etc and including support of possible future extensions.

Integration of different RBAC models can be achieved by specifying these models as OWL ontologies with corresponding namespaces. Thus integration of models is cross linking of the model concepts and definitions of relation specifications to express all semantic dependencies between models.

Figure 4.3 illustrates that OWL ontology of the EAC model can consolidate in one formal description the RBAC elaborations. ARBAC, RCC, SARBAC, ERBAC, AERBA models are described above; IRBAC is RBAC model for intranet security [Ferraiolo1999]; TMAC is a Team-based Management of Access Control model [Thomas1997]; GTRBAC is Generalized Temporal RBAC model [Bertino2001, Joshi2003, Joshi2001, Joshi2002]; ORBAC is a Object-oriented RBAC model [Chang2001a, Chang2001b]; PRBAC is a Parameterized RBAC model [Bacon2002]; GRBAC is a Generalized RBAC model [Covington2000]. So the figure shows that as all elaborations are based on RBAC reference model and partly in some cases on each other then it is possible to create a set of related ontologies. If a new elaboration of the RBAC reference model or any other extension of a RBAC based model, which are already specified in OWL, needs to be included in upper level policy specification language of EAC model then expressing this particular extension is possible through reusing already specified concepts and models to provide compatibility and integrity of EAC policy specification language structure and RBAC correspondence.

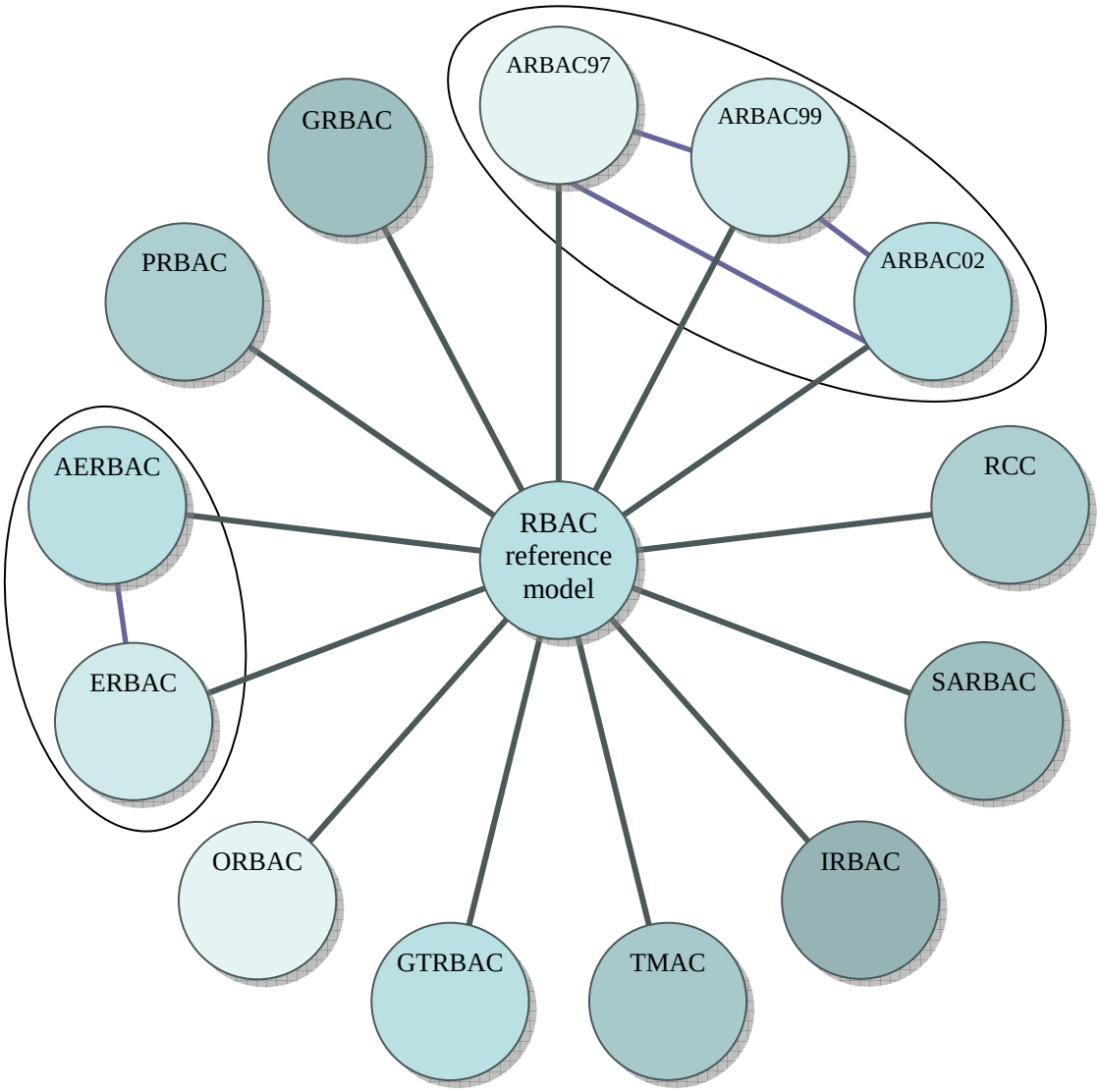

Figure 4.3 – RBAC elaborations and extensions integration

Let us consider in detail how such integration can be done. First of all RBAC reference model has to be specified using OWL in some namespace. Namespace is URI [URI] to the OWL ontology space which contains the definition of RBAC reference model concepts and relations. Use of URI's guarantees unique identification and interpretation of ontology terms. I will refer the namespace for RBAC reference model ontology as "rbac:" in a real case the namespace can be any arbitrary valid URI.

Each RBAC concept becomes an OWL class in ontology. Relations between concepts are expressed by properties of corresponding classes. For instance the class rbac:User has the property rbac:roles which points to all the roles that were assigned to the user. Class rbac:Role has the property rbac:users which points to all users that were assigned to the role. Properties rbac:users and rbac:roles have the type of owl:inverseOf. This definition supports consistency of encoded data and expresses association between classes in a way that if a value of one property is set then value of inverse property is also set (must be set) to appropriate value. The InverseOf feature of OWL provides same metadata logic for RDF based data representation as relationship in relational model for databases. Appendix B contains OWL ontology for RBAC reference model.

Assume that a RBAC administration task arises. Then some model for RBAC data administration can be chosen to support the enterprise needs. Specification of this model must be based on specification of RBAC ontology in the case when this model itself relies on RBAC reference model. Thus ontology specification of any elaboration or extension of the RBAC reference model has to create semantic links to the ontology of RBAC reference model. These extensions also have to create semantic links to other extensions and elaborations to avoid misunderstanding between them. Several features exist in OWL to create a set of ontologies as a semantic network of different RBAC based models:

- owl:subClassOf property allows to specify concepts in namespace of elaboration ontology as subclasses of previously defined concepts of RBAC model or any other already defined model. For example in the case of ARBAC97, concepts Group, Abilty, UP-role, which are specified as classes arbac97:Group, arbac97:Ability, arbac97:UP-role, are direct subclasses of rbac:Role class.

- owl:range property allows to point to an already defined class in property definition for ontology of elaboration of the RBAC model. For example of ARBAC97 model specification, range of arbac97:fromRole property is rbac:Role class.

- In the case when a new class has a previously defined property, usage of owl:domain property allows to extend the definition of the already defined property

in RBAC based model ontology to include a new class in the property domain. For example almost all classes of elaborations have rbac:name property by extending the domain of rbac:name into each ontology namespace.

- owl:intersectionOf, owl:disjointWith, owl:unionOf, owl:complementOf properties allow to specify additional features of problem domain that are more sophisticated then in RDFS.

- An owl:equivalentClass points on definition of equivalent class. An owl:equivalentProperty points on definition of equivalent property. These are useful properties for example when the same concepts have new names in a new elaboration.

By using the above listed features of OWL, all RBAC based model ontologies can be linked into a semantic network of concepts. This semantic network which is formed using OWL provides the definition of the language for policy of access control specification.

4.3.3 Specification of functional semantics

To define the first component of the semantic EAF it is not enough to create only the ontology. Specification of a set of the functions which will support all required manipulations with semantically encoded EAC data is also an important aspect and has to be considered within specification of the EAC model.

RBAC standard provides an administrative functional specification to manipulate with RBAC reference model. Functions are defined using the theory of sets and first-order logic predicates. Semantic Web language based on RBAC ontology is used to encode RBAC data. Thus the meaning of administrative functions of RBAC standard has to be enriched and redefined in terms of Semantic Web language for encoding of RBAC data. Table 4.1 describes the OWL support of the Core RBAC model functions. System support functions are omitted because EAC model does not support sessions.

Core RBAC function	RBAC meaning	OWL meaning
Administrative functions		
AddUser	creates new RBAC user	creates an instance of rbac:User class
DeleteUser	deletes an existing user	deletes an instance of rbac:User class
AddRole	creates a new role	creates an instance of rbac:Role class
DeleteRole	deletes an existing role	deletes an instance of rbac:Role class
AssignUser	assigns user to a role	specifies a property rbac:usersToRoleAssigned of rbac:Role instance to an instance of rbac:User (inverse property rbac:rolesToUserAssigned of an instance of a rbac:User class has to be assigned to an instance of rbac:Role class)
DeassignUser	deletes the assignment of the user to the role	deletes specification of a property rbac:usersToRoleAssigned of rbac:Role instance to an instance of rbac:User (reassigning of inverse property)
GrantPermission	grants the permission to perform an operation on an object to a role	specifies a property rbac:permissionsToRoleAssigned and owl:inverseOf property rbac:rolesToPermissionAssigned to link instances of rbac:Role and rbac:Permission
RevokePermission	revokes the permission to perform an operation on an object from set of permissions assigned to role	Deletes link between two instance of rbac:Role and rbac:Permission specification
Review functions		
AssignedUsers	returns the set of users assigned to a given role	returns the values of a property rbac:usersToRoleAssigned for an instance of rbac:Role class

AssignedRoles	returns the set of roles assigned to a given user	returns the values of a property rbac:rolesToUserAssigned for an instance of rbac:User class
Advanced review functions		
RolePermissions.	returns the set of permissions granted to a given role	returns the values of a property rbac:permissionsToRoleAssigned for an instance of rbac:Role class
UserPermissions	returns the permissions a given user gets through user's assigned roles	returns the values of a property rbac:permissionsToRoleAssigned for instances of rbac:Role class which are values of a property rbac:rolesToUserAssigned of a given instance of rbac:User class
RoleOperationsOnObject	returns the set of operations a given role is permitted to perform on a given object	Union of all instances of rbac:Operation class for all instances of rbac:Permission which have given instance of rbac:Object in property rbac:objects and which are in property rbac:permissionsToRoleAssigned for a given instance of a class rbac:Role
UserOperationsOnObject	returns the set of operations a given user is permitted to perform on a given object	Union of all instances of rbac:Operation class for all instances of rbac:Permission which have given instance of rbac:Object in property rbac:objects and which are in property rbac:permissionsToRoleAssigned for instances of a class rbac:Role which are in property rbac:rolesToUserAssigned of a given instance of rbac:User class.

Table 4.1 – OWL support of Core RBAC model functions

Administrative functions of Core RBAC remain valid for Hierarchical RBAC. Hierarchical RBAC introduces four additional functions to support role hierarchy. All review functions of Core RBAC stay the same and Hierarchical RBAC defines additional review functions. Advanced review functions of Core RBAC remain the same for Hierarchical RBAC, but formal description of RolePermissions, UserPermissions, RoleOperationsOnObject and UserOparationOnObject changes because users and permissions became authorized to each other through role inheritance. For limited role hierarchy all description above remains

valid, except that AddInheritance function is redefined. Table 4.2 represents new functions of Hierarchical RBAC.

Hierarchical RBAC function	RBAC meaning	OWL meaning
Administrative functions		
AddInheritance	creates new immediate inheritance relationship between existing two roles	specification of an inheritance between two instances of a class rbac:Role by assigning values to instance's properties which are owl:inverseOf each other rbac:ascendants and rbac:descendants
DeleteInheritance	deletes an existing immediate inheritance relationship between two roles	deletes cross linking for two instances of rbac:Role by revoking values of instance's properties rbac:ascendants and rbac:descendants
AddAscendant	creates a new role and inherits it in the role hierarchy as an immediate ascendant of the existing role	creates new instance of rbac:Role and set it's rbac:descendants property to existent given instance of rbac:Role class
AddDescendant	creates a new role and inherits it in the role hierarchy as an immediate descendant of the existing role	creates new instance of rbac:Role and set it's rbac:ascendants property to existent given instance of rbac:Role class
Review functions		
AuthorizedUsers	returns the set of users authorized to a given role, the users that are assigned to a roles that inherit the given role	returns union of all instances of rbac:User which are values in property rbac:usersToRoleAssigned for a given instance of rbac:Role and for all instances of roles which are values of transitive property rbac:descendants
AuthorizedRoles	returns the set of roles authorized for a given user	returns union of all instances of rbac:Role which are values in property rbac:rolesToUserAssigned for a given instance of rbac:User and then all instances which are values in transitive property rbac:ascendants

Table 4.2 – OWL meaning of new functions of Hierarchical RBAC

To use SSD relations with core RBAC function AssignUser from core RBAC function set is formally redefined. All other functions of core RBAC remain valid. New administrative and review functions are introduced in Table 4.3.

Constrained RBAC function	RBAC meaning	OWL meaning
Administrative functions		
CreateSsdSet	creates a named SSD set of roles and sets the cardinality of its subsets that cannot have common users	creates an instance of rbac:Ssd class, assigns properties rbac:SSDs and rbac:staticExclusiveRoles, and sets property rbac:cardinality
AddSsdRoleMember	adds a role to a named SSD set of roles	sets property rbac:SSDs and rbac:staticExclusiveRoles values for instances of rbac:SSD and rbac:Role classes
DeleteSsdRoleMember	removes a role from named SSD set of roles	deletes link to a given instance of rbac:Role from specification of a property rbac:staticExclusiveRoles of a given instance of rbac:SSD
DeleteSsdSet	deletes a SSD role set completely	deletes instance of class rbac:SSD
SetSsdSetCardinality	sets cardinality associated with a given SSD role set	specifies value for property rbac:cardinality of an instance of rbac:SSD
Review functions		
SsdRoleSets	returns the list of all SSD role sets	returns instances of rbac:Role which are values of a property rbac:staticExclusiveRoles of a given instance of class rbac:SSD
SsdRoleSetCardinality	returns the cardinality associated with a SSD role set	returns value of a property rbac:cardinality of a given instance of class rbac:SSD

Table 4.3 – OWL meaning of new functions of Constrained RBAC

Using of SSD RBAC with general role hierarchy of Hierarchical RBAC: supporting system functions remain as defined for general role hierarchy; review functions remain as

for general role hierarchy and which additionally defined for SSD RBAC; advanced review functions remain the same as defined for general role hierarchy of Hierarchical RBAC.

4.3.4 Encoding and using semantic EAC data

Now, the EAC data can be encoded using RDF according to the ontology which defines the language of policy specification. In the real case, the policy specifications are distributed across the enterprise systems but managed centrally. For simplicity assume that the whole specification is one RDF document. All instances are uniquely identified by their URIs. RBAC model is defined as OWL ontology and encoding of RBAC data follows RDF and OWL ontology.

The practical value of the EAC data in an RDF document can only be realized if there exist tools to automatically interpret, extract and map it into the access control informations in the native formats of the several application platforms in the enterprise. There is a variety of RDF parsers whose links can be found from RDF activity site [RDF]. An RDF parser provides a library of routines or methods that implement either a DOM [dom] or SAX application program interface (API) [sax] and those in turn are then callable from a procedural language. For example, the library of methods in an RDF parser for Java can be utilized by Java programs to access the structure and content of RDF documents because they follow XML syntax.

DOM, Document Object Mode is a programming interface specification being developed by the World Wide Web Consortium [W3C] that lets a programmer to create and modify HTML pages and XML documents as full-fledged program objects

SAX (Simple API for XML) is an API that allows a programmer to interpret a Web file that uses the XML - that is, a Web file that describes a collection of data. SAX is an alternative to using the DOM to interpret the XML file. As its name suggests, it's a simpler interface than DOM and is appropriate where many or very large files are to be processed, but it contains fewer capabilities for manipulating the data content. SAX is an event-driven interface. The programmer specifies an event that may happen and, if it does, SAX gets control and handles the situation.

It is not convenient to use the document model of DOM API or the event model of SAX API in the upper level software design of the application for managing EAC. The definition of RBAC ontology can be mapped to UML representation to allow reusing RBAC model in software development. Figure 4.4 illustrates the UML class diagram for RBAC model. An UML model can be easily obtained from OWL ontology by mapping OWL classes and properties to UML classes and properties; OWL object properties to UML class associations; OWL inverse properties to UML bidirectional associations. In such a way an arbitrary RBAC ontology can be mapped to UML class diagram.

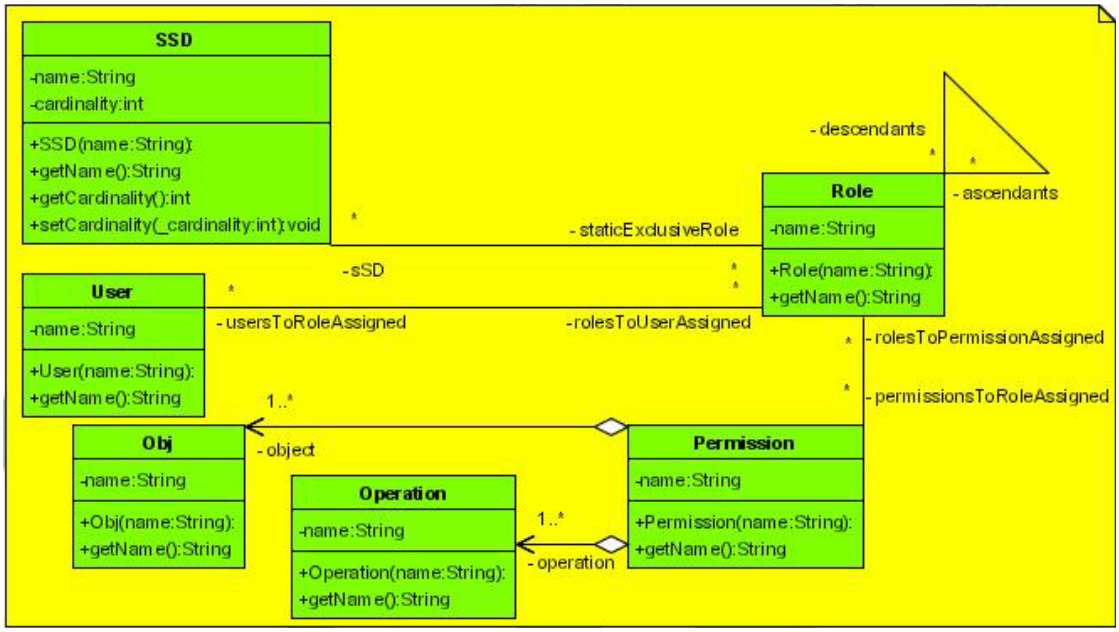

Figure 4.4 – UML class diagram for RBAC reference model

4.4 Chapter summary

Design of an EAF is complicated as it requires integration of native platform-dependent mechanisms for AC and integration of data representation on enterprise level. Chapter provides some ideas how to make such kind of integration easer using Semantic Web family of standards.

First of all any EAC model follows RBAC reference model and its elaborations. Basically language for AC data specification on enterprise level is needed. OWL and RDFS

languages give ability to model the RBAC domain and to define an RBAC ontology which plays the role of meta level formal language for specification of AC data.

EAC data could be encoded according to RBAC ontology using RDF and XML languages. Tool with such kind of functionality is part of EAF. It has to provide UI for administrators of AC in enterprise level and modules for AC data mapping to target systems. A Protégé tool for creating ontologies using RDFS and OWL languages can be used for creation of RBAC ontology in particular enterprise. This tool has been used for the examples of this thesi. Thus EAF looks like table 4.4 summarizes.

EAF component	Solution	Description
EAC model	RBAC model	formal model for AC in enterprise
	OWL, RDFS, RDF and XML specifications	define domain modeling languages to capture semantic of RBAC reference model and it's elaborations in RBAC ontology
	RBAC ontology	defines the meta language for EAC policy specification
EAC data	RBAC ontology	encoded EAC data corresponds to RBAC ontology
	RDF and XML specifications	define syntax for EAC data encoding and grammar-based constraints
Tool set	DOM API, SAX API	API to process EAC data
	Protégé [Protégé]	Tool to create RBAC ontology
	AC policy specification tool	Such tool has to be developed to provide UI for AC policy specification on enterprise level

Table 4.4 – Semantic approach components for EAF

5 RBAC in a University

5.1 Motivation

In this chapter we consider the access control in the context of a University to provide an example that illustrates the issues described in the previous chapters. The main goal of an EAC model is to provide a solid framework for

- Centralized specification of high level university policy
- Explicit specification of organizational requirements
- Explicit specification of structural and domain specific constraints
- Direct mapping of abstract EAC model concepts to concepts of native access control modules
- Strict division of authority and responsibilities of users within IT infrastructure
- Support of arbitrary policy types (mandatory, discretionary, etc)
- Maintenance of organizational process view on access control
- Approach to delegate administrative tasks among university level admins, system admins, end users (self-support) and users with naturally administrative functions (personnel officer)

Successful implementation of RBAC model leads to

- Improvement of user provisioning and deprovisioning
- Reduction of cost of day-to-day administration activity. Permissions are provided to roles which are more stable in organization than user occupation of roles. Partial automation of administrative activities can be achieved

Special requirements can be specified only for defined host systems and types of access to objects. Architecture and functionality of the ERBAC system is highly correlated with the level of integration of ERBAC system into native access control modules. Three possible alternatives are

- Just auditing of permissions to user granting and revoking process
- Direct mapping of abstract ERBAC concepts to native system enforcement mechanism data, and using this mechanism for online access control
- Changing the native mechanism algorithm by implementing role based online access control

5.2 Overview of roles and structures of University of Jyväskylä

The actors of the university are the students (N=15 000), staff (N=2 500) and affiliated people. We can assume that of each actor we have (automatically) some basic attributes (personal data, department/main topic, status as student (regular, visiting, graduate, etc), status as employee (position/title)). This data is fed from student and personnel systems and can be assumed to be available in a relational database or as a LDAP-view.

There are several systems and applications that are used by all or by many users. Some systems support (and require) quite evolved role structure whereas some others have relatively flat role structures that are in most cases based on the general attributes of the user.

- Korppi: Aims to cover all the teachers and all the students of all the courses at the university (plus relevant administration). Rich variety of roles needed. The student/staff attributes and department/topic attribute are relevant as a basis. Likely to be the home base for any role management applications. Own application.
- Optima: Learning environment that should encompass all the students and teachers as users and provide a variety of work spaces to different groups. Hopefully linked

to Korppi (creation of groups in Korppi, allocation of work environment in Optima). Commercial.

- Travel: Used by the staff to report business trips. Each staff member has at least the basic traveler role (within the own department) that is to be managed automatically. Also administrative roles (verification, approval). Verification (secretary) attached to named administrative persons, approval mainly by faculty in formal positions (heads, deans). Commercial.

- Tutka: Used by the staff to report scientific activities (articles, projects, etc). Staff allocated to departments, one administrative role (on department level). Own product.

- Common and local computers: Three different families (Unix, Novell, MS AD). Local systems expected to join the centralized user/password management and authentication by the end of the year. Student/staff and department are relevant attributes.

There are a number of systems that are used mainly by certain administrative staff. In addition Korppi and Travel have administrative roles. The most important applications are:

- AdeEco: Accounting and bookkeeping system. Used mainly in read only mode by named secretaries in the departments. Only few roles/persons for entering the data (all in the central administration).

- Fortime: Personnel system. Read only at departments (by named secretaries). Entering of data in central administration.

- Rondo: System for electronic management of bills. Coming to operation within a year. It will require electronic approval/signature at department level (verification and approval roles for secretaries and dept. heads). User management and access control will be critical.

- Jore: System for accounting the student's credits. Used both in read-only and data entering modes at departments (by secretaries and officers in study affairs). Own product.

There seems to be a need to flexibly create small groups and allocate resources to them, either in some application or in common computer systems.

- Teaching: It can be useful to be able to identify the students (or the teachers) of a certain course or an exercise group and to provide them either a work space or access to some material (like electronic books with limited licenses) - or to give some administrative rights to the teachers over the student group. Typically, one should allow the student to join the group on his/her own initiative and the teacher could manage the group after that.

- Research: The different research projects have currently no system support at all. Common working place for sharing documents would be needed, with possibility to manage the access rights within the research group.

- Administration: In administration there are quite a number of different groups and task forces (boards, steering groups, ad hoc committees, etc) that could use a common working place and whose members might benefit from access to sensitive information that can not be made public to the whole community.

The biggest masses are in relatively simple roles that are naturally defined as intersections of some simple classes (say, being member of staff and belonging to certain department) and the corresponding rights are limited accordingly. In pure RBAC this seems to require systematic creation of tens of similar basic roles and similar basic permissions. Can/should this be avoided? Perhaps by systematically requiring department attribute to be a part of the user data so that it does not have to be included in the roles.

Many of the administrative systems are commercial and beyond our control. Hence the user/permission administration can not be delegated to a separate application that would communicate directly machine to machine. Hence, tools must be provided to assist and

systematize the communication between the actual allocation of roles and permissions (on the level of decision making – that can be made in RBAC application) and actual management of users in the commercial application. (Now this is based on set of ad hoc forms (one for each application) that are signed and circulated between different offices).

One big question is how to provide support to distributed creation of ad hoc roles and allocation of permissions to them (in learning environments, document management systems or file servers)

5.3 Meta roles and role engineering in the University

In order to design an RBAC model for the university, a role hierarchy has to be developed. Role engineering should follow structured, methodical development, modification and maintenance of roles in role-based systems. The life-cycle of a role provides an abstract description of the role engineering process. Figure 5.1 illustrates the role life-cycle [Kern2002a].

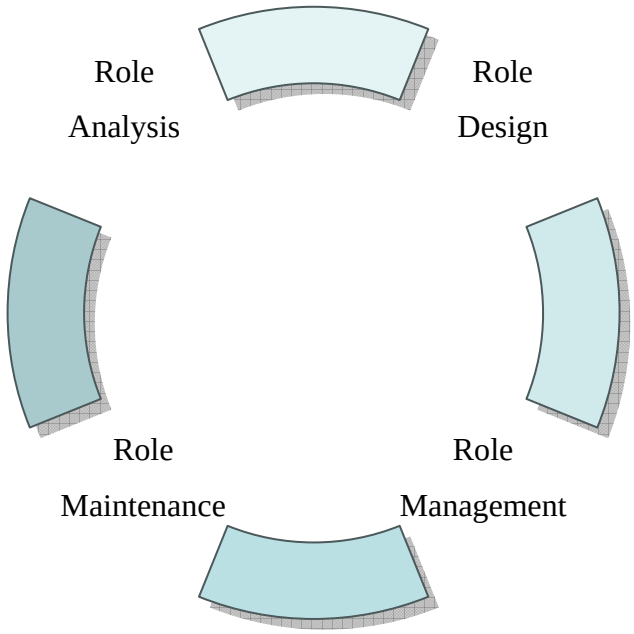

Figure 5.1 – The Role Life-Cycle

Role analysis is a stage when domain should be analyzed to identify meta roles or formal criteria for role design. This stage can be performed only by domain expert who knows well enterprise roles in the target domain.

The role analysis process should ideally be a mixed bottom-up and top-down approach. The top-down approach starts from the business process oriented role descriptions. Very often, there is no policy at all, so we have to start from scratch by defining all privileges that a particular job function must have [Roeckle2000]. A bottom-up approach means that, starting from the total set of permissions which exist in the organization, one tries to find clusters of permissions which represent roles. While a top-down approach ignores existing permissions, a pure bottom-up approach would not take organizational structures into account. Organizational structure, positions of users, locations of users and operational duties of users within IT systems can serve as areas of role design in university. Where operational duties within IT systems are used in bottom-up approach, others are used in top-down approach.

Role hierarchy reflecting organizational structure is more “natural” and stable decomposition than for instance role hierarchy of operational duties within IT systems. Figure 5.2 shows the hierarchy of organizational meta roles.

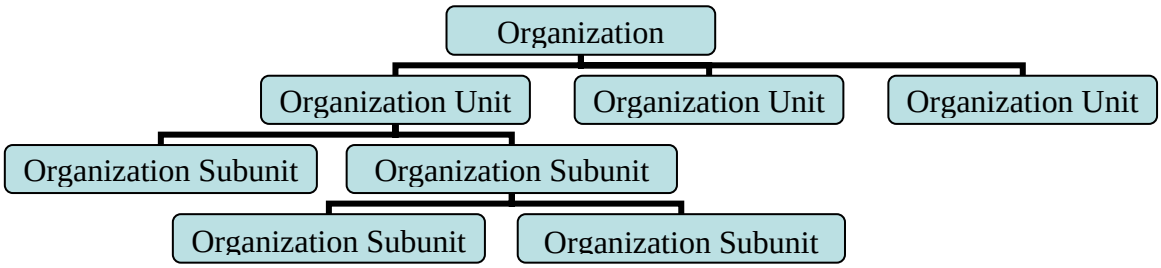

Figure 5.2 – The hierarchy of organizational meta roles

Organizational structure of the University looks like it is shown on figure 5.3. Organizational meta role hierarchy is up side down organizational structure because “University” role has more generic permissions and role of the “department” has more specific ones.

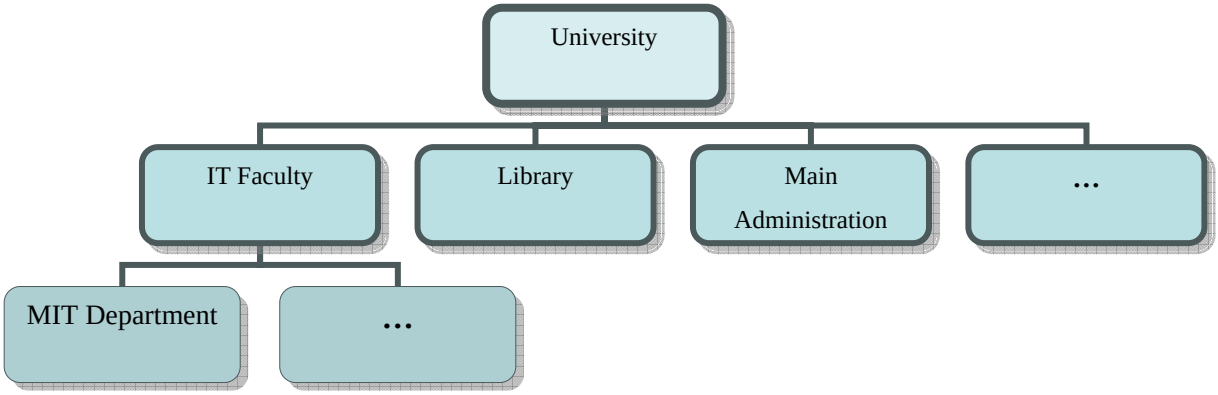

Figure 5.3 - Organizational structure of the University

Role hierarchy which is formed by using information about positions of users and types of positions makes great impact on user AC. This role hierarchy follows structure of meta roles staff – type of position – position. Figure 5.4 shows concrete sample of taxonomy for positions in university.

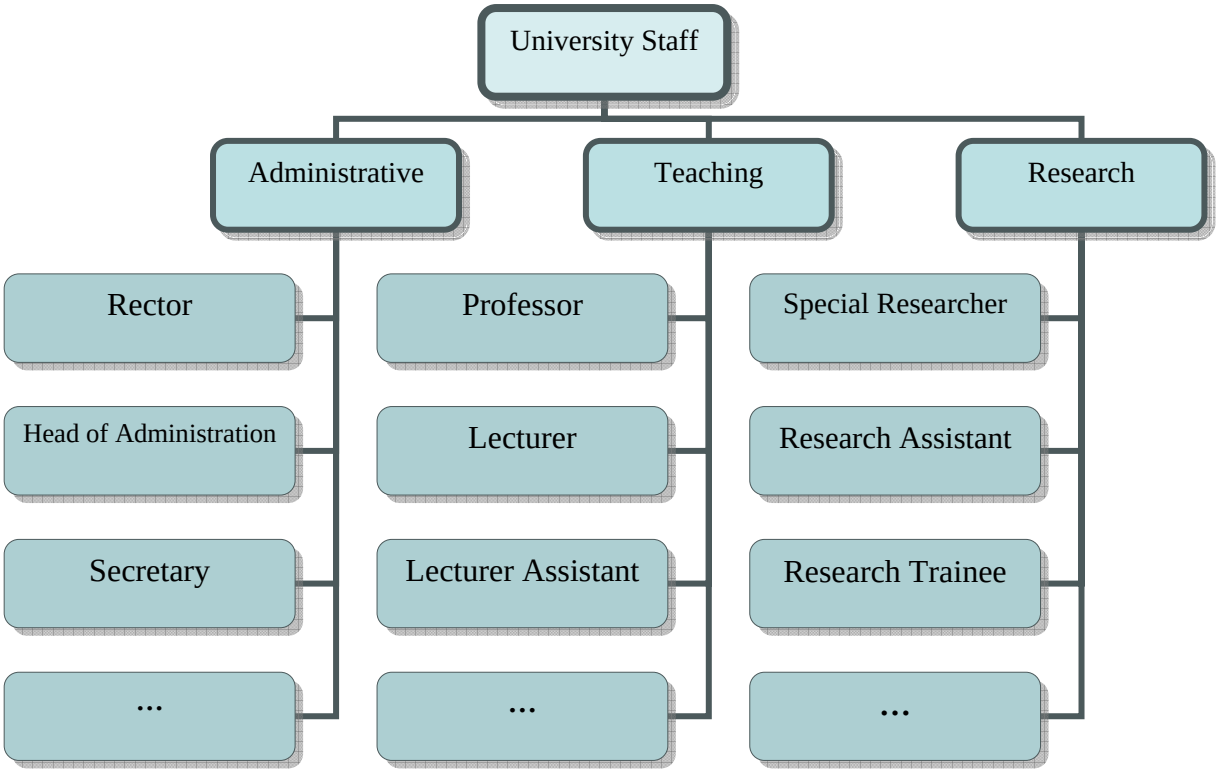

Figure 5.4 – Taxonomy of positions in the University

Location dependent role hierarchy consists of geographical branches of the company for instance. In the case of University location points on the work place of the user in terms of buildings, floors, working areas. The location concepts associate to each other mainly by part-of relation.

Top-down approach uses considered above organizational, position, location criteria for meta role analysis. Bottom-up approach requires to have IT system specific roles first. These specific roles can be collected using whatever metaphor even without any enterprise sense just by set-subset relation between collections of permissions. But it is more appropriate to use for instance operational criteria. Role hierarchy can follow duty-task-activity distribution of permission. Operational roles like job position or named responsibility can be used to link together enterprise view and system specific view. Figure 5.5 illustrates meta roles for collecting permissions and operational roles as roles from enterprise view.

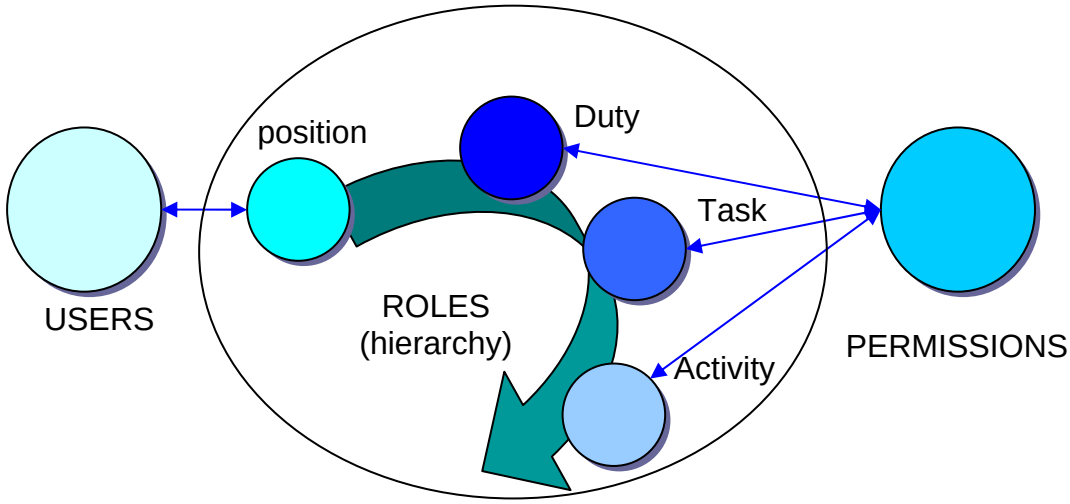

Figure 5.5 – System specific metaphor for meta roles

Stanford University model and metaphor for building role hierarchy serves as an example. Figure 5.6 depicts conceptual view on implementation of ERBAC model.

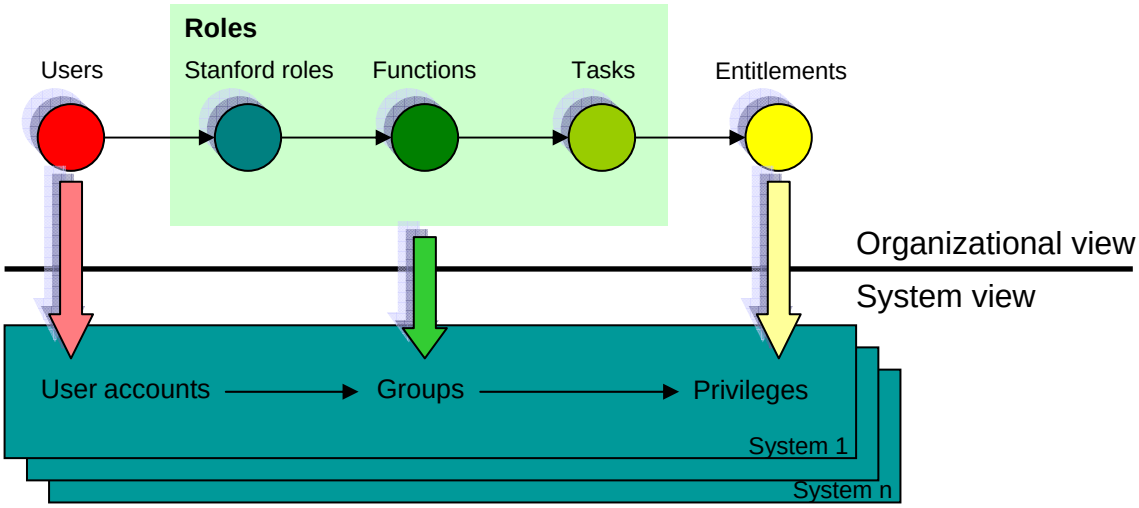

Figure 5.6 – Stanford University ERBAC roles

After analysis of different meta role structures in the University, the role hierarchies for different criteria have to be designed and finally one consolidated role hierarchy should be obtained. The role hierarchy is a result of merging analyzed and designed role hierarchies. For the case of the University, the role hierarchy on enterprise level is a composition of the organizational structure and the taxonomy of positions.

Role hierarchy can consists of disjoint subgraphs for organizational structure, location and position in the case when decision of access permission relies separately on organizational unit, location or position of the user. But in the general case information from several organizational structures is needed to define a single role for some business application. In a situation when access decision relies on location, position and organizational unit the resulting role hierarchy is the direct product of the involved hierarchies. Problem of high complexity of resulting role hierarchy arises. Formalization of the process of the merging the structures into one role hierarchy is also a difficult task. For the construction of a merged graph, standard methods from graph theory could be used. However, the resulting graph will be very complex, unless most of the structures are very small.

Another approach which avoids increasing complexity has proven to be successful. Only one structure is chosen as the role structure and the other structures are used as constraints. For the University case the organizational structure which is merged with positions is the

most appropriate role hierarchy. Location and other user characteristics can be formalized as parameters of roles [Bacon2002, Kern2002b].

5.4 Parameterized RBAC

ERBAC provides a good basis for user and security administration. However, the use of this model as it is would lead to a large number of roles and thus to a high administration effort. There are basically two reasons for this: multiple factors defining the roles and the need for fine-grained control of application security.

As mentioned before, the access rights a person receives are normally based on a number of factors. These may be organizational unit, position, location or others. As the combination of these factors defines the rights, one cannot simply build separate role hierarchies based on organization, position etc. Instead, a role must be defined for every valid combination of these factors. The resulting role structure would obviously be very complex and difficult to maintain.

The solution for these problems is to parameterize roles. Therefore ERBAC model was enhanced with attributes and rules [Kern2002b]. Enhanced ERBAC is described briefly in the forth chapter of the thesis. Here I describe parameterized ERBAC with more details.

In the enhanced ERBAC attributes can be assigned to the following entities:

- users,
- roles,
- user assignments,
- permission assignments,
- role-to-role assignments.

These attributes may specify constraints or other values relevant for access control decisions. Rules specify what happens when attributes are changed or assignments are given or removed.

The user in the ERBAC model contains a rich set of standard and company-specific attributes. In the University of Jyväskylä user's attributes are stored in database which is called AMAN. These attributes can be used for a number of important functions.

A set of attributes such as name, title, telephone number describes the user. They can be propagated to a target system when an account is created for the user.

Several attributes describe the user's organizational unit, position(s) and so on. These attributes provide the basis for automation of user administration as they normally define the roles a user will receive. If these attributes change, the roles a user receives or loses are computed using rules and automatically assigned or deassigned. Of course, automation can considerably reduce administration costs and is therefore the main goal of the University when implementing a universitywide user administration tool.

A further possibility is to use user attributes for specifying user-specific information which can be used as constraints for roles and permissions or for other administration tasks. Let us consider this possibility on example.

The example is about managing the access control to printers in the University. Assume that we have a general role to use printers. Every instance of a printer in enhanced ERBAC has attributes of its location (room, floor, department, etc). User has the attribute of the location of his or her work place. User is assigned to a role for using printers with the constraint that he or she has particular work place. Resulting permission set printers of really accessible by user will be obtained according to a rule of choosing concrete printers based on their and user location information. Figure 5.7 illustrates this example. Here the attribute of user location serves as a constraint on user to role assignment, particular "real" permission can be computed accordingly to rule which defines permission to role assignment.

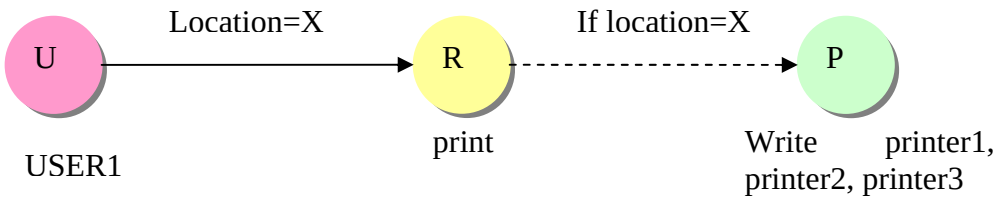

Figure 5.7 – User assignment to general role, rule based permission computation

The concept of roles with generic permissions helps to prevent building separate role structures. Normal roles are collections of permissions defined for specific target resources. Generic roles allow the assignment of generic permissions defined for a set of target systems. When such a role is assigned to a user, one or more target resources from this set are specified. The user then receives these permissions only in the specified target resources.

When assigned to a user, the actual permission is computed using the attributes of the user and/or user-role connection on the basis of rules. The permission is then granted to the user. The rules use attributes of the user (e.g. organizational unit, location, position) to compute the actual permission.

User-specific constraints constitute a further aspect that occurs often – especially in business applications. People doing principally the same job may have different restrictions. Some examples include:

- Lectures in Korppi system perform the same set of operations but only with their courses and student (working) groups.

- People, who are responsible to maintain organizational unit's homepages, have the same set of permission to access to Web Server content but only for the workspace that has the right organizational attributes.

5.5 Chapter summary

Parameterized RBAC seems to be the most appropriate EAC model for the University, because it enables integrating in one role hierarchy several taxonomies of factors which influence on access control. In the case of pure RBAC model for the University the role hierarchy would be too complicated because one would have to apply multiple inheritance to all structures of factors to construct the role hierarchy. It seems that role hierarchy has to be merged from the structure of organizational units and the structure of user positions within the University. Other factors such as location, status characteristics, etc become parameters of users, roles, permissions and relations of assignment between these concepts. Constraints of access control in target systems and rules, for defining particular permission in target system from general permission in policy specification, rely on parameters. Formalisms have to be elaborated for specification of rules.

The view concept of RCC or the scope concept of SARBAC model might be used in administrative RBAC to define administrative authority over data of parameterized RBAC model. Concept of exclusion has to be studied to support sufficient fine granularity of specification and delegation of administrative authority.

6 Specification of functionality of the prototype application

6.1 Description of the prototype

Approach of prototyping research ideas is well known and is widely used in software system design and development. Prototype itself provides a possibility to study the important features and verify the concepts of a complex and usually expensive system before investing to the main development project. Some methodologies of software development (Extreme Programming) use prototype as a result of each iteration of the development process. So a prototype usually has only a limited set of most important features of the target system.

In academic computer science research prototyping constitutes the process of practical verification of theoretical concepts. In our case the first goal of the prototype is to provide a workspace to RBAC model design for different cases of business systems and operational environments in the University. From this point of view the prototype has to implement a user interface for creation of the main concepts of RBAC model to check whether the RBAC model meets the needs of the University.

Second goal of the prototype of RBAC model is to give possibility for university people to try RBAC approach for managing access control in different domains. So the goal is to familiarize administrators with RBAC approach in practical use instead of theoretical consideration.

Next practical goal of the prototype is to implement basic functionalities to manipulate the concepts of RBAC model. Core functionality could be reused in project of development of real system.

Prototype application has also the goal to implement the functionality for decision flow management. Main idea is to send messages to people who are decision makers for access rights management and who are actual executors of activities for access rights granting and revoking. Crucial component of prototype application is a module for auditing all changes

of RBAC data. Thus the prototype has its own value and can be used for tracking access control decisions in real cases.

6.2 Functionality of the prototype

Prototype provides mainly three subsets of functionality:

- Functionality and user interface to specify data accordingly to RBAC model.
- User interface to specify and functionality to run scenarios of decision making of granting or revoking of access rights to user.
- Functionality to audit manipulation with RBAC data and user interface to browse and search through audit trail.

Figure 6.1 illustrates an upper level vision of prototype functionality. It has typical real world objects that are concepts of target domain of access control in the University (user, object, and operation). The figure 6.1 depicts also Web Mail Server as an external system, Decision Maker and Executor as people to whom system should send email messages.

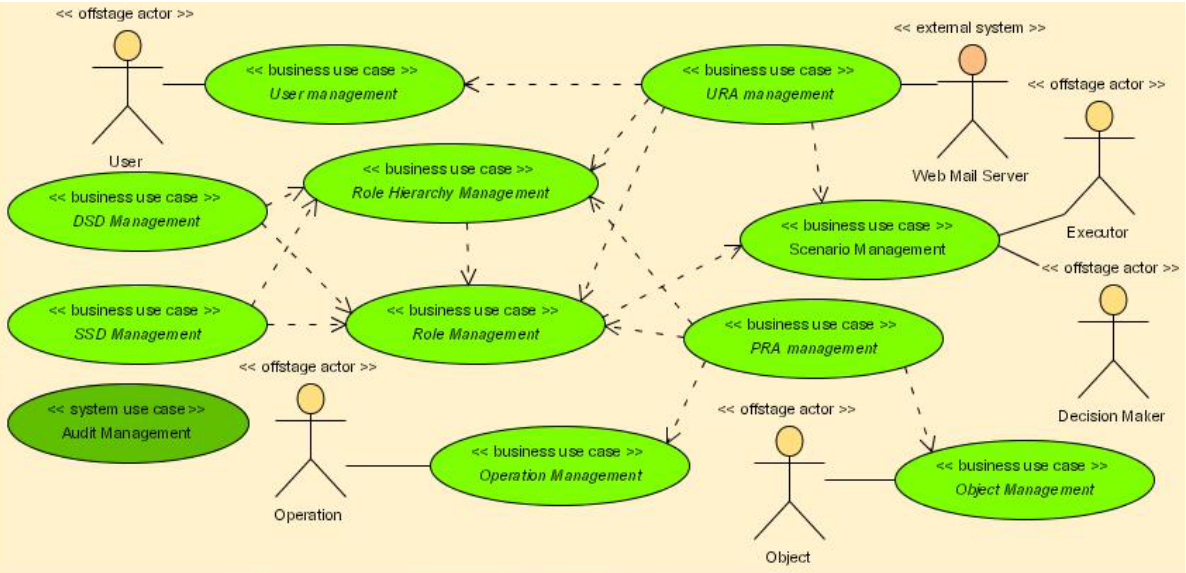

Figure 6.1 – Use Case UML diagram of business view on system functionality

Next subchapter describes particular use cases of the prototype using UML notation of use case diagramming [UML].

6.3 Analysis of use cases of the prototype

6.3.1 User Management use case

User Management use case is described in table 6.1.

Use case:	User Management.
Summary:	Information about user has to be created and maintained in up-to-date state to allow manipulation of access rights of the user.
Frequent:	Whenever user changes his or her status.
Purpose:	Administration can manage access rights of the user.
Precondition:	Prototype is running, administrator is authenticated, up-to-date information about user exists.
Description:	Administrator as a primary actor can add new user to system, delete existing user or provide new information about user. User is an offstage actor which does not use system, but system uses information about user. System can be connected to human resources database to run operation of this use case automatically. In such case participation of administrator is not needed. Figure 6.2 illustrates operations and actors of this use case.
Postconditions:	Entry of the user in RBAC database is up-to-date and available for further access control management.

Table 6.1 – User Management use case

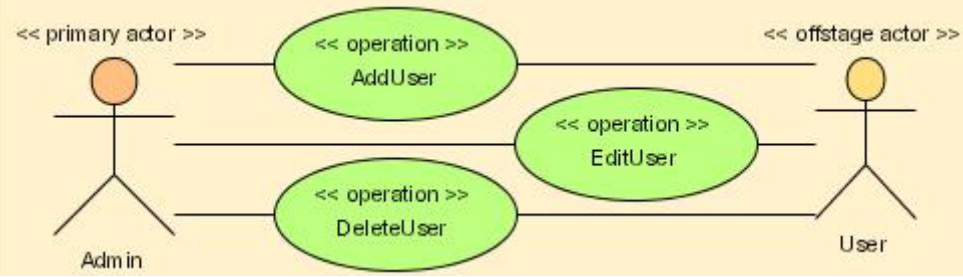

Figure 6.2 – UML diagram of User Management use case

6.3.2 Role Management use case

Role Management use case is described in table 6.2.

Use case:	Role Management.
Summary:	Basic element of RBAC is role. Roles have to correspond to organizational units and staff positions within the University.
Frequent:	Whenever role related organizational concepts change.
Purpose:	Administration can manage a set of roles corresponding to the University state.
Precondition:	Prototype is running, administrator is authenticated, up-to-date information about roles exists.
Description:	Administrator as a primary actor can add new role to system, delete existing role or provide new information about role (name). Figure 6.3 illustrates operations and actors of this use case.
Postconditions:	Entry of the role in RBAC database is up-to-date and available for further access control management.

Table 6.2 – Role Management use case

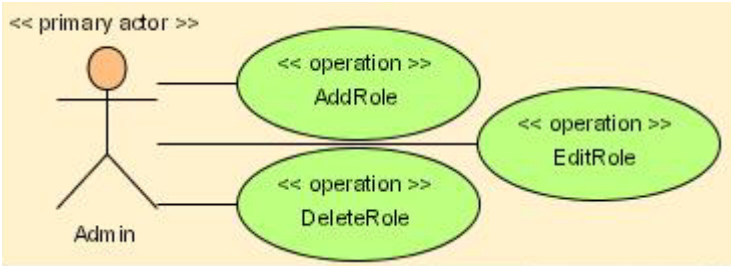

Figure 6.3 – UML diagram of Role Management use case

6.3.3 Object Management use case

Object Management use case is described in table 6.3.

Use case:	Object Management.
Summary:	Information about object has to be created and maintained in up-to-date state to allow access control to this object.
Frequent:	Whenever object changes its status.
Purpose:	Administration can manage access rights to resources.
Precondition:	Prototype is running, administrator is authenticated, up-to-date information about object exists.
Description:	Administrator as a primary actor can add new object to system, delete existing object or provide new information about object. Object is an offstage actor which does not use system, but system uses information about it. Additional research is needed to find possible sources of information about resources in the University (Microsoft Active Directory, Novel Directory, other registries). Figure 6.4 illustrates operations and actors of this use case.

Postconditions:	Entry of the object in RBAC database is up-to-date and available for further access control management.

Table 6.3 – Object Management use case

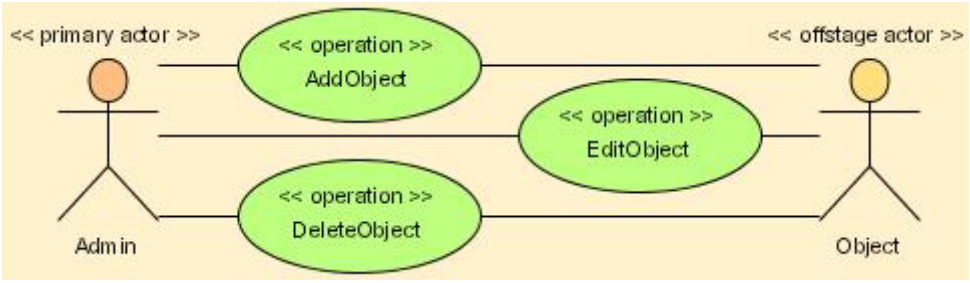

Figure 6.4 – UML diagram of Object Management use case

6.3.4 Operation Management use case

Operation Management use case is described in table 6.4.

Use case:	Operation Management.
Summary:	Information about operation has to be created and maintained in up-to-date state to allow access control through using this operation to the objects.
Frequent:	Whenever operation changes its status. Most likely once when object is added.
Purpose:	Administration can manage access rights of operating with resources.
Precondition:	Prototype is running, administrator is authenticated, up-to-date information about operation exists.

Description:	Administrator as a primary actor can add new operation to system, delete existing operation or provide new information about operation. Operation is an offstage actor which does not use system, but system uses information about it. Every object typically has set of applicable operations. Operation should have link to type of resources to which it is applicable. Figure 6.5 illustrates operations and actors of this use case.
Postconditions:	Entry of the operation in RBAC database is up-to-date and available for further access control management. Set of operation applicable for object types is known.

Table 6.4 – Operation Management use case

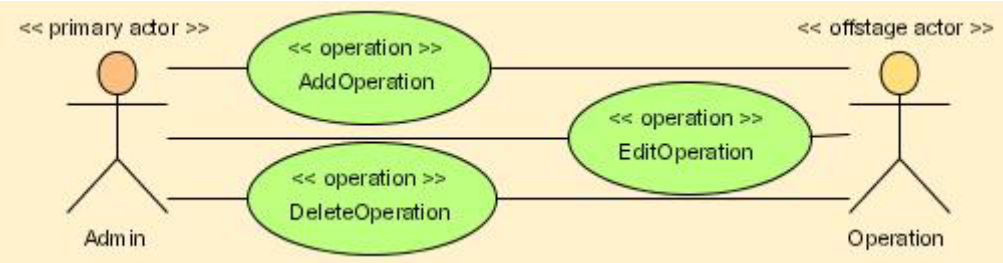

Figure 6.5 – UML diagram of Operation Management use case

6.3.5 Permission to Role Assignment Management use case

Permission to Role Assignment Management use case is described in table 6.5.

Use case:	PRA Management.
Summary:	Assignment of a set of operations over a set of objects to a role accordingly to duties and responsibilities which this role represents is a core functionality of any RBAC system.

Frequent:	Whenever role meaning changes. Most likely when role is created or business processes change.
Purpose:	Administration can provide access rights for users by simple assigning users to roles.
Precondition:	Prototype is running, administrator is authenticated, and information is in the system about role, required operations and objects.
Description:	Administrator as a primary actor can grant new permission to role or revoke assigned permission from role. Permission is a set of operations over a set of objects. Administrator defines a set of operations, a set of corresponding objects and a role. Remark: there is not named permission in a system, pair of operation and object can be considered as atomic permission. Figure 6.6 illustrates operations and actors of this use case.
Postconditions:	Permission to role assignment is in RBAC database. Role has permissions to access resources.

Table 6.5 – Permission to Role Assignment Management use case

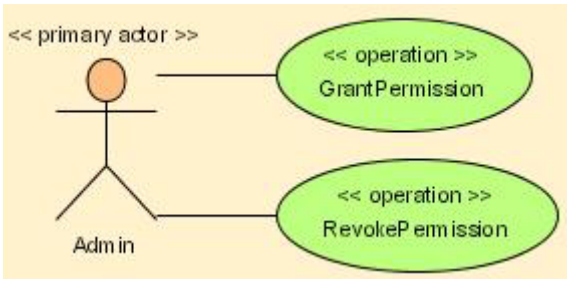

Figure 6.6 – UML diagram of Permission to Role Assignment Management use case

6.3.6 Role Hierarchy Management use case

Role Hierarchy Management use case is described in table 6.6.

Use case:	Role Hierarchy Management.
Summary:	Ordering of roles to hierarchy based on organizational structure and positions for top-down approach; based on permission and user inheritance relations for bottom-up approach of role hierarchy design.
Frequent:	Whenever role meaning changes. Iterative process which initiates accordingly to changes in the University and information systems.
Purpose:	All advantages of organizing role to hierarchy can be achieved.
Precondition:	Prototype is running, administrator is authenticated, and information is in the system about roles.
Description:	Administrator as a primary actor can design role hierarchy accordingly to desired permission and user distribution among roles. Figure 6.7 illustrates operations and actors of this use case.
Postconditions:	Role Hierarchy exists in RBAC database.

Table 6.6 – Role Hierarchy Management use case

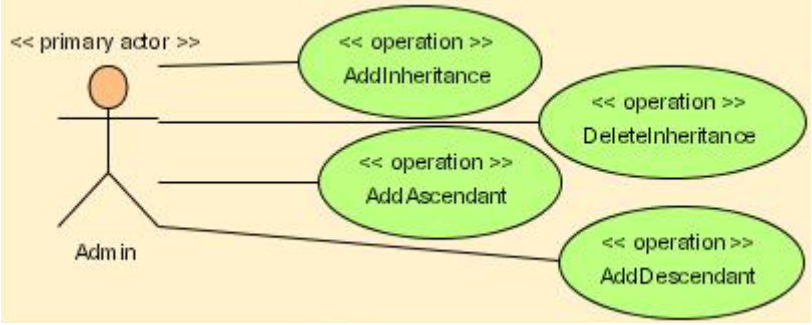

Figure 6.7 – UML diagram of Role Hierarchy Management use case

6.3.7 Scenario Management use case

Scenario Management use case is described in table 6.7.

Use case:	Scenario Management.
Summary:	System has to provide support for decision workflow management when series of approvals are needed to assign permissions for user. System has to provide support for provisioning and deprovisioning of access rights by local executors.
Frequent:	Whenever role is created for which user cannot be assigned by administrator without gathering approvals from external organization authorities.
Purpose:	Support simple email messaging for gathering approvals to grant or revoke access right to sensitive resources and to initiate provisioning or deprovisioning of access rights for user by local executors.
Precondition:	Prototype is running, administrator is authenticated, and information about scenario, decision makers, local executors and template messages is available.
Description:	Administrator as a primary actor can design scenario as a sequence of email messages to decision makers and executors. Administrator has decision maker's and executor's email addresses. Administrator manages email templates of email messages. Administrator can attach scenario to a role. Figure 6.8 illustrates operations and actors of this use case.
Postconditions:	Scenarios might be run in order to assign users to sensitive roles.

Table 6.7 – Scenario Management use case.

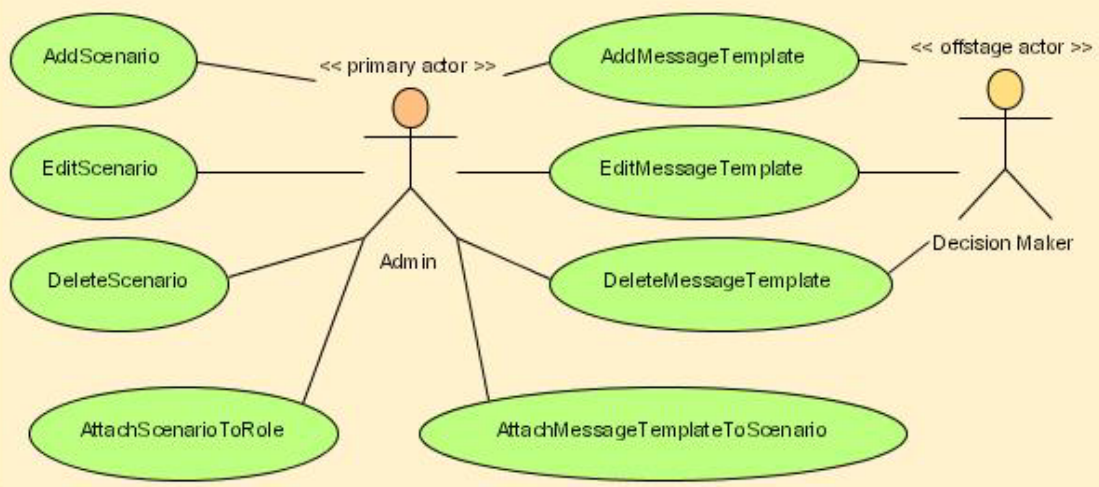

Figure 6.8 – UML diagram of Scenario Management use case.

6.3.8 User to Role Assignment Management

User to Role Assignment Management use case is described in table 6.8.

Use case:	URA Management.
Summary:	Administrator performs provisioning and deprovisioning of access rights by assigning user to a role mainly.
Frequent:	Whenever user characteristics change which are sensitive for access control.
Purpose:	Reduce administration efforts and complexity of user provisioning and deprovisioning.
Precondition:	Prototype is running, administrator is authenticated, and information about roles, permissions and scenario of user assignment is in the system.

Description:	Administrator as a primary actor can assign user to role or deassign user from role. Assignment process becomes long-living transaction for time of execution of scenario. After scenario finishing user is assigned. Figure 6.9 illustrates operations and actors of this use case.
Postconditions:	Access rights belong to appropriate users.

Table 6.8 – User to Role Assignment Management use case

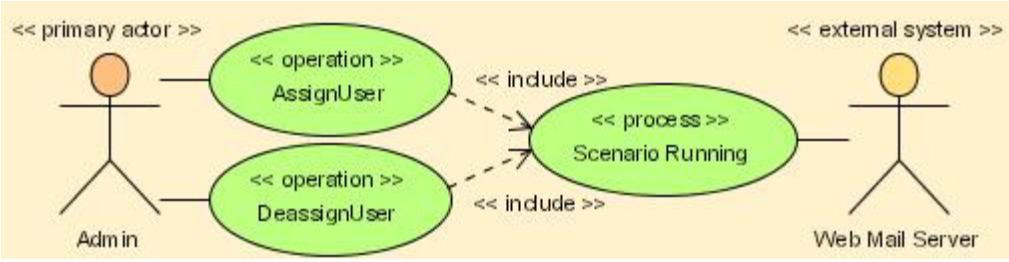

Figure 6.9 – UML diagram of User to Role Assignment Management use case

6.3.9 Static Separation of Duties Management use case

Static Separation of Duties Management use case is described in table 6.9.

Use case:	SSD Management.
Summary:	Administration can enforce static separation of duties constraints on access rights distribution among users.
Frequent:	Whenever logic of separation of duties changes.
Purpose:	Ensure access rights to be in match with constraints of separation of duties on organizational level.
Precondition:	Prototype is running, administrator is authenticated, information about roles is in the system and separation of duty rule can be explicitly specified.

Description:	Administrator as a primary actor can manipulate with SSD element to meet requirements of access control. Figure 6.10 illustrates operations and actors of this use case.
Postconditions:	SSD can be enforced in use cases of URA management and Role Hierarchy management.

Table 6.9 – Static Separation of Duties Management use case

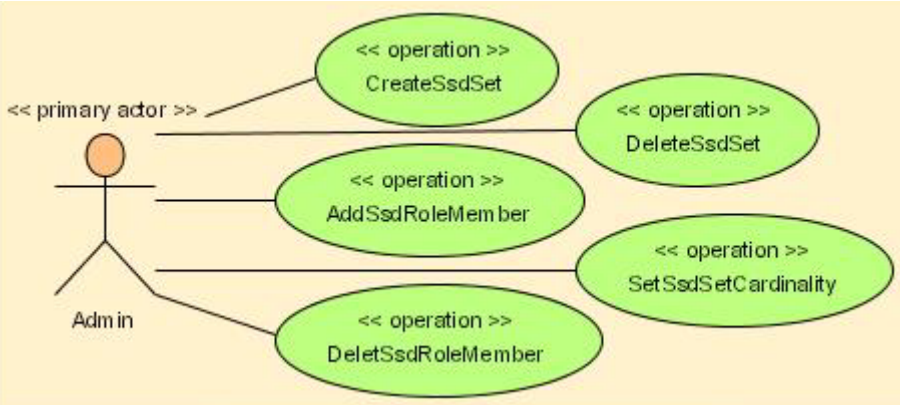

Figure 6.10 – UML diagram of Static Separation of Duties Management use case

6.3.10 Dynamic Separation of Duties Management use case

Dynamic Separation of Duties Management use case is described in table 6.10.

Use case:	DSD Management.
Summary:	Administration can specify dynamic separation of duties constraints on access rights set within user sessions.
Frequent:	Whenever logic of separation of duties changes
Purpose:	Ensure access rights to be in match with constraints of separation of duties on system level.

Precondition:	Prototype is running, administrator is authenticated, information about roles is in the system and separation of duty rule can be explicitly specified.
Description:	Administrator as a primary actor can manipulate with DSD element to meet requirements of access control. Figure 6.11 illustrates operations and actors of this use case.
Postconditions:	DSD is specified and target systems executors can be notified about this constraint.

Table 6.10 – Dynamic Separation of Duties Management use case

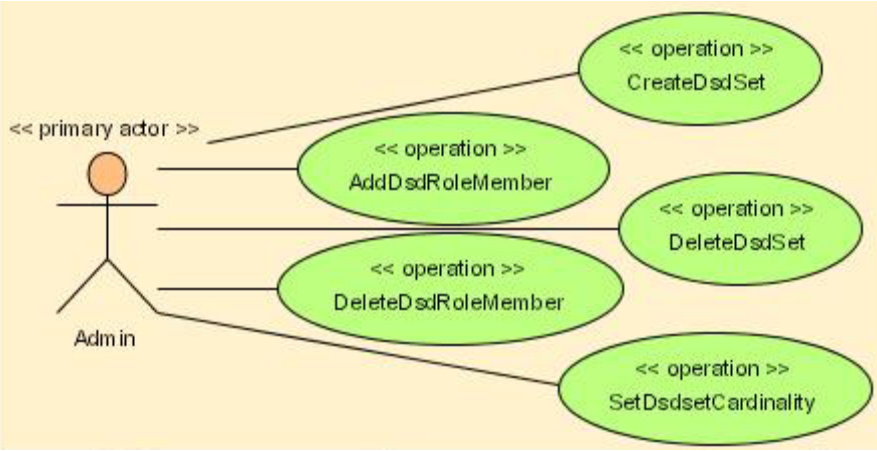

Figure 6.11 – UML diagram of Dynamic Separation of Duties Management use case

6.3.11 Audit Management use case

Audit Management use case is described in table 6.11.

Use case:	Audit Management.
Summary:	Audit is crucial element in any security system. System has to log all activities of manipulating with RBAC data and scenario execution.

Frequent:	Whenever changes to RBAC data occur.
Purpose:	Keep track of all manipulations within the system.
Precondition:	Prototype is running, administrator is authenticated, administrator performs some operation.
Description:	It is system level use case. System has to log information about who, what and when do within the system. Figure 6.12 illustrates operations and actors of this use case.
Postconditions:	Audit trail is available.

Table 6.11 – Audit Management use case

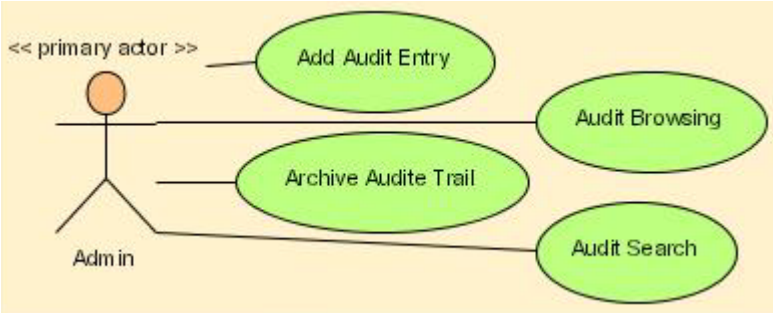

Figure 6.12 – UML diagram of Audit Management use case

6.4 Example of specification of EAC policy

This subchapter figures out steps of EAC policy specification for concrete case. It describes the example itself and encoding of the EAC policy in OWL accordingly to RBAC ontology.

Let us start with description of the example. Assume that we have to specify EAC policy for AC of the files of some Web Server. User Bob should have permissions to access files of www-pages on the Web Server of the organization. Bob is head of some department and he has access only to the files in a directory of his department.

First of all we need to add information about our user to specification of EAC policy. The document of EAC policy to represent Bob contains information that OWL statement has unique RDF identification by symbolic name “Bob” and that it is instance of class “rbac:User” of OWL ontology of RBAC:

```
<rbac:User rdf:ID="Bob"/>
```

Next step is to specify explicitly instances of an object and of operations of access. In our example the object is the folder on Web Server and operations are from a set of regular operations to access content of an arbitrary folder.

```
<rbac:Object rdf:ID="folder"/>
<rbac:Operation rdf:ID="execute"/>
<rbac:Operation rdf:ID="specialPerm"/>
<rbac:Operation rdf:ID="modify"/>
<rbac:Operation rdf:ID="write"/>
<rbac:Operation rdf:ID="read"/>
<rbac:Operation rdf:ID="list"/>
```

When we have object and operations specified in EAC policy, we can design and specify roles and their relations of inheritance in a way like it is shown below for roles of content modification and full access.

```
<rbac:Role rdf:ID="ModifyContent">
        <rbac:descendants>
          <rbac:Role rdf:ID="Head">
            <rbac:asscendants rdf:resource="#ModifyContent"/>
          </rbac:Role>
        </rbac:descendants>
</rbac:Role>
```

Then we have to assign pairs of operation and object to bottom roles in our role hierarchy. Intermediate roles, which are called also as connector roles, collect bottom roles to sets of permissions.

```
<rbac:Role rdf:ID="Readonly">
   <rbac:permissionsToRoleAssigned>
    <rbac:Permission rdf:ID="Read">
     <rbac:operations>
      <rbac:Operation rdf:resource="#read"/>
     </rbac:operations>
     <rbac:objects rdf:resource="#folder"/>
     <rbac:rolesToPermissionAssigned rdf:resource="#Readonly"/>
    </rbac:Permission>
    <rbac:Permission rdf:ID="List">
     <rbac:rolesToPermissionAssigned rdf:resource="#Readonly"/>
     <rbac:objects rdf:resource="#folder"/>
     <rbac:operations>
      <rbac:Operation rdf:resource="#list"/>
     </rbac:operations>
    </rbac:Permission>
   </rbac:permissionsToRoleAssigned>
</rbac:Role>
```

Finally, we need to make specification of Bob assignment to a role which represents his position within organization. Assume that we have top roles in our hierarchy to reflect positions of users within the organization.

```
<rbac:User rdf:resource="#Bob">
```

<rbac:rolesToUserAssigned rdf:resource="#Head"/>

</rbac:User>

Figure 6.13 illustrates whole picture of specification of EAC policy for the given example. It contatins all operations, object as a folder on Web Server, the role hierarchy and additional user Alice who is secretary of the department.

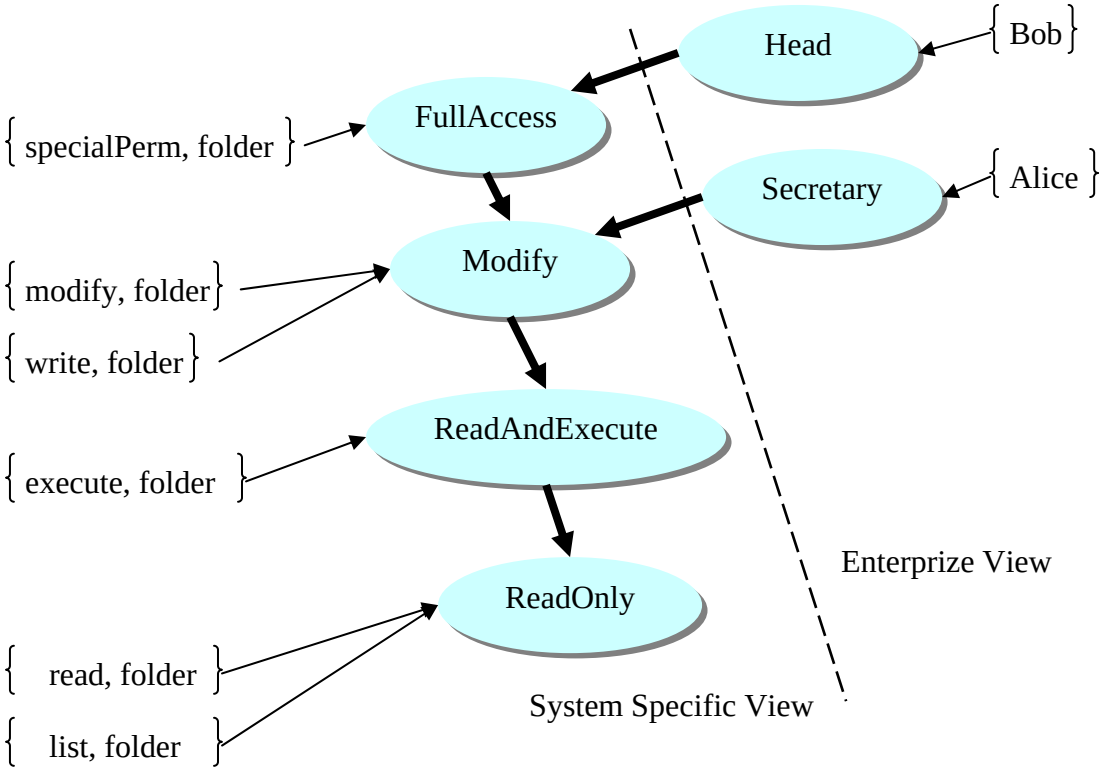

Figure 6.13 – Conceptual view of the example

6.5 Chapter summary

With specification of prototype functionality given in this chapter the first milestone of design review can be reached. It is called requirements review and takes into analysis the use case model, user interface prototypes and the domain model. Stakeholders of the project should revise the functionality of the prototype and provide corrections if it is needed. Next step would be to perform specification of operations of use cases, data model design, and system architecture design. This goes, however, beyond the scope of the work.

CONCLUSIONS

Design of an EAF is complicated as it requires integration of native platform-dependent mechanisms for AC and integration of data representation on enterprise level. The thesis provides some ideas how to make such kind of integration easer using Semantic Web family of standards. OWL and RDFS languages give ability to model RBAC domain and to define RBAC ontology which plays the role of meta level formal language for specification of AC data.

OWL is more appropriate for specification of RBAC models than RDFS. On the other hand RDF is older technology for semantic description and thus has more tools to work with it. Anyway, RDFS and OWL can be used together and supplement each other. However, on my opinion the better the semantics are defined the better will be unification of understanding and the easier will be integration and further extension of RBAC ontology. Thus OWL is to be preferred in case when abstract, flexible and expressive language is needed.

One of the existing approaches Parameterized RBAC is the most promising EAC model for a large organization, because it supports integration several taxonomies of factors which influence on decision of access control in one role hierarchy. In the pure RBAC model the role hierarchy would be too complicated as a result of using multiple inheritance to merge all structures of factors in one role hierarchy. It seems that the role hierarchy has to be merged from the structure of organizational units and the structure of user positions within the University. Other factors such as location, status characteristics, etc become parameters of users, roles, permissions and relations of assignment between these concepts. Constraints of access control in target systems and rules for defining particular permission in target system from general permission in policy specification rely on parameters. Formalisms have to be elaborated for specification of rules.

The view concept of RCC or the scope concept of SARBAC model might be used as concepts of administrative RBAC to define administrative authority over the data of parameterized RBAC model. Concept of exclusion has to be researched to support fine granularity of specification and delegation of administrative authority.

Basically all RBAC administrative models follow RBAC reference model. Some of elaborations introduce new concepts and change RBAC reference model. The main differences between administrative models are in approaches to express the sets of RBAC concepts to delegate authority among administrative roles. My opinion is that none of the above mentioned models looks like ideal model for RBAC data administration. Main reason for this conclusion is unclear definition of the functional part of administrative models. The concepts and relation in the models are defined rigorously but still for each model something with functional specification remains unspecified or the model fails to support the RBAC standard as a whole. Thus development of ideal administrative RBAC model and its functional specification remains as a task for further research. And it is attractive to design such model in a level of semantic description to allow high scale of integration possibilities for RBAC elaborations.

Specification of the prototype functionality and analysis of RBAC model constitute the first milestone of software design. The proposed design should e reviewed to move further. This, so called requirements review, should analyses use case model, user interface prototypes and the domain model. Stakeholders of the project should revise the functionality of the prototype and provide necessary corrections. Next step is to specify operations of use cases and to design data model and system architecture.

Further research of the semantics of the RBAC model and its concepts is needed. A Semantic RBAC model could extend and enhance the regular RBAC model. The possibilities and advantages of semantic approach for specification of AC data have to be studied as well. Main benefits to be expected of a semantic RBAC model are formalisms for specification of taxonomies of operations, objects, users and roles using class-subclass relations and formalization also relations between classes of RBAC concepts which are missed in nowadays RBAC models. Thus granularity and expressiveness can be achieved for definition of AC policy in arbitrary level of abstraction.

References

[Bacon2002] J. Bacon, K. Moody, and W. Yao. A model of OASIS role-based access control and its support for active security. ACM Transactions on Information and System Security (TISSEC), 5(4):492–540, November 2002.

[Bertino2001] E. Bertino, P. A. Bonatti, and E. Ferrari. TRBAC: A temporal role-based access control model. ACM Transactions on Information and System Security, 4(3):191–233, 2001.

[Chang2001a] Chang. N. Zhang and Cungang Yang, "Specification and Enforcement of Object-oriented RBAC Model", 2001 IEEE Canadian Conference on Electrical and Computer Engineering, Toronto, May 2001, pp. 128--135.

[Chang2001b] Chang. N. Zhang , Cungang Yang, An Object-Oriented RBAC Model for Distributed System, Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA'01), p.24, August 28-31, 2001

[Covington2000] Michael J. Covington, Matthew J. Moyer, and Mustaque Ahamad. Generalized role-based access control for securing future applications. In Proceedings of the National Information Systems Security Conference (NISSC), October 2000.

[Crampton2002] Crampton, J. and G. Loizou. Administrative Scope and Role Hierarchy Operations. in 7th ACM Symposium on Access Control (SACMAT). 2002. Naval Postgraduate School, Monterey, CA, USA.

[Crampton2003] J. Crampton and G. Loizou. Administrative Scope: A Foundation for Role-Based Administration Models. ACM Transactions on Information and System Security (TISSEC), 6(2):201–231, 2003.

[dom] Document Object Model activity site, http://www.w3.org/DOM/

[Ferraiolo1999] D. Ferraiolo, J. Barkley, and D. Kuhn. A Role-Based Access Control Model and Reference Implementation within a Corporate Intranet. ACM Transactions of Information and System Security, 2(l) (1999) 34-64

[Ferraiolo2001] D. Ferraiolo, R. Sandhu, S. Gavrila, D. Kuhn, and R. Chandramouli, “Proposed NIST standard for role-based access control”, ACM Transactions on Information and Systems Security, vol. 4, no. 3, pp. 224–274, August 2001.

[Ferraiolo2003a] D. Ferraiolo, D. Kuhn, and R. Chandramouli, “Role-Based Access Control”, Artech House, 2003, 316 pages, ISBN:1580533701

[Ferraiolo2003b] D. Ferraiolo, G. Ahn, and S. Gavrila, "The Role Control Center: Features and Case Studies," Proc. of the 8th ACM Symposium on Access Control Models and Technologies, June 2003

[Joshi2001] J. B. D. Joshi, Elisa Bertino, Usman Latif, Arif Ghafoor, "Generalized Temporal Role Based Access Control Model (GTRBAC)", Submitted to IEEE Transaction on Knowledge and Data Engineering. Available as CERIAS tech. report 2001-47.

[Joshi2002] J. B. D. Joshi, Elisa Bertino, Arif Ghafoor, “Temporal hierarchies and inheritance semantics for GTRBAC”, In proceedings of 7th ACM Symposium on Access Control Models and Technologies, June 2002

[Joshi2003] J. Joshi, E. Bertino, B. Sah.q, and A. Ghafoor. Dependencies and separation of duty constraints in GTRBAC. In Proceedings of the 8th ACM Symposium on Access Control Models and Technologies (SACMAT 2003), 2003.

[Kern2002a] A. Kern, M. Kuhlmann, A. Schaad, and J. Mofett. Observations on the Role Life-Cycle in the Context of Enterprise Security Management. In Proceedings of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT 2002), Monterey, California, USA, pages 43–51, June 2002.

[Kern2002b] A. Kern. Advanced Features for Enterprise-Wide Role-Based Access Control. In Proceedings of the 18th Annual Computer Security Applications Conference, Las Vegas, Nevada, USA, pages 333–342, December 2002.

[Kern2003] Kern, A., A. Schaad, and J. Moffett. An Administration Concept for the Enterprise Role-Based Access Control Model. in 8th ACM Symposium on Access Control Models and Technologies (SACMAT). 2003.

[OASIS] Organization for the Advancement of Structured Information Standards, http://www.oasis-open.org/

[Oh2002] S. Oh and R. Sandhu. A Model for Role Administration Using Organization Structure. In Proceedings of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT 2002), Monterey, California, USA, pages 155–168, June 2002.

[OWL] Web Ontology Language specification site, http://www.w3c.org/2004/OWL/

[Powers2003] S. Powers. ”Practical RDF”, O’Reilly, 2003, 350 pages, ISBN 0-596-00263-7

[Protégé] Protégé – Ontology Editor and Knowledge Acquisition System, http://protege.stanford.edu/

[RDF] Resource Description Framework specification site, http://www.w3c.org/RDF/

[Roeckle2000] Roeckle H., G. Schimpf, and R. Weidinger, "Process- Oriented Approach for Role-Finding to Implement Role-Based Security Administration in a Large Industrial Organisation." presented at Fifth ACM Workshop on Role-Based Access Control, Berlin, Germany, 2000.

[Sandhu1996] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstain, and Charles E. Youman. 1996. Role-based Access Control Models. IEEE Computer, 29 (2): 38-47

[Sandhu1997] Ravi Sandhu and Venkata Bhamidipati, “The ARBAC97 model for Role-based administration of Roles: Preliminary Description and Outline”, In Proceedings of second ACM Workshop on Role-Based Access Control. November 1997.

[Sandhu1998] Ravi Sandhu and Qamar Munawer, “The RRAC97 model for Role-based administration of Role hierarchies”, In Proceedings of 14th Annual Computer Security Applications Conference, December 1998

[Sandhu1999] Ravi Sandhu and Qamar Munawer, “The ARBAC99 model for administration of roles”, In Proceedings of the Annual Computer Security Applications Conference. 1999.

[sax] SAX—Event-Driven API for XML, http://www.megginson.com/SAX

[SemanticWeb] Semantic Web activity site, http://www.w3c.org/2001/sw/

[Thomas1997] Thomas R.K. Team-Based Access Control (TMAC): A Primitive for Applying Role-Based Access Controls in Collaborative Environments, Proceedings of the Second ACM workshop on Role-based Access Control, Fairfax, VA USA, 1997.

[UML] Unified Modeling Language resources site, http://www.uml.org/

[URI] “Naming and Addressing: URIs, URLs, ... “ site, http://www.w3c.org/Addressing/

[W3C] World Wide Web Consortium site, http://www.w3c.org/

[XML] Extensible Markup Language specification site, http://www.w3c.org/XML/

Appendix A. RDFS ontology of RBAC reference model

```
<?xml version='1.0' encoding='Cp1252'?>
<!DOCTYPE rdf:RDF [
                <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
                <!ENTITY kb 'http://rbac.jyu.fi/kb#'>
                <!ENTITY rdfs 'http://www.w3.org/TR/1999/PR-rdf-schema-19990303#'>
]>
<rdf:RDF xmlns:rdf="&rdf;"
                xmlns:kb="&kb;"
                xmlns:rdfs="&rdfs;">
<rdfs:Class rdf:about="&kb;DSD" rdfs:label="DSD">
                <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdf:Property rdf:about="&kb;ID" rdfs:label="ID">
                <rdfs:domain rdf:resource="&kb;Session"/>
                <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdfs:Class rdf:about="&kb;Object" rdfs:label="Object">
                <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Operation" rdfs:label="Operation">
                <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Permission" rdfs:label="Permission">
                <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;PermissionRoleAssignment"
                rdfs:label="PermissionRoleAssignment">
                <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Role" rdfs:label="Role">
```

```
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;RoleInheritance" rdfs:label="RoleInheritance">
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;SSD" rdfs:label="SSD">
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Session" rdfs:label="Session">
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;User" rdfs:label="User">
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;UserRoleAssignment" rdfs:label="UserRoleAssignment">
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdf:Property rdf:about="&kb;cardinality" rdfs:label="cardinality">
        <rdfs:domain rdf:resource="&kb;DSD"/>
        <rdfs:domain rdf:resource="&kb;SSD"/>
        <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;fromRole" rdfs:label="fromRole">
        <rdfs:range rdf:resource="&kb;Role"/>
        <rdfs:domain rdf:resource="&kb;RoleInheritance"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;name" rdfs:label="name">
        <rdfs:domain rdf:resource="&kb;DSD"/>
        <rdfs:domain rdf:resource="&kb;Object"/>
        <rdfs:domain rdf:resource="&kb;Operation"/>
        <rdfs:domain rdf:resource="&kb;Permission"/>
```

```
                <rdfs:domain rdf:resource="&kb;Role"/>
                <rdfs:domain rdf:resource="&kb;SSD"/>
                <rdfs:domain rdf:resource="&kb;User"/>
                <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;object" rdfs:label="object">
                <rdfs:range rdf:resource="&kb;Object"/>
                <rdfs:domain rdf:resource="&kb;Permission"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;operation" rdfs:label="operation">
                <rdfs:range rdf:resource="&kb;Operation"/>
                <rdfs:domain rdf:resource="&kb;Permission"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;permissions" rdfs:label="permissions">
                <rdfs:range rdf:resource="&kb;Permission"/>
                <rdfs:domain rdf:resource="&kb;PermissionRoleAssignment"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;roles" rdfs:label="roles">
                <rdfs:domain rdf:resource="&kb;DSD"/>
                <rdfs:domain rdf:resource="&kb;PermissionRoleAssignment"/>
                <rdfs:range rdf:resource="&kb;Role"/>
                <rdfs:domain rdf:resource="&kb;SSD"/>
                <rdfs:domain rdf:resource="&kb;Session"/>
                <rdfs:domain rdf:resource="&kb;UserRoleAssignment"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;session" rdfs:label="session">
                <rdfs:range rdf:resource="&kb;Session"/>
                <rdfs:domain rdf:resource="&kb;User"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;toRole" rdfs:label="toRole">
                <rdfs:range rdf:resource="&kb;Role"/>
```

```
        <rdfs:domain rdf:resource="&kb;RoleInheritance"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;users" rdfs:label="users">
        <rdfs:range rdf:resource="&kb;User"/>
        <rdfs:domain rdf:resource="&kb;UserRoleAssignment"/>
</rdf:Property>
</rdf:RDF>
```

Appendix B. OWL ontology of RBAC reference model

```
<?xml version="1.0" encoding="windows-1252" ?>
<rdf:RDF
   xmlns="http://rbac.jyu.fi/ontology#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
   xmlns:dc="http://purl.org/dc/elements/1.1/"
  xml:base="http://rbac.jyu.fi/ontology">
  <owl:Ontology rdf:about="RBAC"/>
  <owl:Class rdf:ID="SSD"/>
  <owl:Class rdf:ID="User"/>
  <owl:Class rdf:ID="Permission"/>
  <owl:Class rdf:ID="Role"/>
  <owl:Class rdf:ID="DSD"/>
  <owl:Class rdf:ID="Object"/>
  <owl:Class rdf:ID="Operation"/>
  <owl:Class rdf:ID="Session"/>
  <owl:ObjectProperty rdf:ID="SSDs">
    <rdfs:range rdf:resource="#SSD"/>
    <rdfs:domain rdf:resource="#Role"/>
    <owl:inverseOf>
      <owl:ObjectProperty rdf:about="#staticExclusiveRoles"/>
    </owl:inverseOf>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="rolesToUserAssigned">
    <owl:inverseOf>
      <owl:ObjectProperty rdf:about="#usersToRoleAssigned"/>
    </owl:inverseOf>
    <rdfs:range rdf:resource="#Role"/>
```

```
  <rdfs:domain rdf:resource="#User"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="user">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#sessions"/>
  </owl:inverseOf>
  <rdfs:range rdf:resource="#User"/>
  <rdfs:domain rdf:resource="#Session"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="permissions">
  <rdfs:range rdf:resource="#Permission"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="activeInSessions">
  <rdfs:range rdf:resource="#Session"/>
  <rdfs:domain rdf:resource="#Role"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#activeRoles"/>
  </owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="sessions">
  <rdfs:domain rdf:resource="#User"/>
  <owl:inverseOf rdf:resource="#user"/>
  <rdfs:range rdf:resource="#Session"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="dinamicExclusiveRoles">
  <rdfs:domain rdf:resource="#DSD"/>
  <rdfs:range rdf:resource="#Role"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#DSDs"/>
  </owl:inverseOf>
```

```
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="ascendants">
  <rdfs:range rdf:resource="#Role"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#descendants"/>
  </owl:inverseOf>
  <rdfs:domain rdf:resource="#Role"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#TransitiveProperty"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="descendants">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#TransitiveProperty"/>
  <owl:inverseOf rdf:resource="#ascendants"/>
  <rdfs:range rdf:resource="#Role"/>
  <rdfs:domain rdf:resource="#Role"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="staticExclusiveRoles">
  <rdfs:domain rdf:resource="#SSD"/>
  <rdfs:range rdf:resource="#Role"/>
  <owl:inverseOf rdf:resource="#SSDs"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="activeRoles">
  <owl:inverseOf rdf:resource="#activeInSessions"/>
  <rdfs:range rdf:resource="#Role"/>
  <rdfs:domain rdf:resource="#Session"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="permissionsToRoleAssigned">
  <rdfs:domain rdf:resource="#Role"/>
  <rdfs:range rdf:resource="#Permission"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#rolesToPermissionAssigned"/>
  </owl:inverseOf>
```

```
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="operations">
  <rdfs:range rdf:resource="#Operation"/>
  <rdfs:domain rdf:resource="#Permission"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="DSDs">
  <rdfs:range rdf:resource="#DSD"/>
  <owl:inverseOf rdf:resource="#dinamicExclusiveRoles"/>
  <rdfs:domain rdf:resource="#Role"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="usersToRoleAssigned">
  <owl:inverseOf rdf:resource="#rolesToUserAssigned"/>
  <rdfs:range rdf:resource="#User"/>
  <rdfs:domain rdf:resource="#Role"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="rolesToPermissionAssigned">
  <rdfs:domain rdf:resource="#Permission"/>
  <rdfs:range rdf:resource="#Role"/>
  <owl:inverseOf rdf:resource="#permissionsToRoleAssigned"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="objects">
  <rdfs:domain rdf:resource="#Permission"/>
  <rdfs:range rdf:resource="#Object"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="ID">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdfs:domain rdf:resource="#Session"/>
</owl:DatatypeProperty>
<owl:FunctionalProperty rdf:ID="cardinality">
  <rdfs:domain>
```

```
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#SSD"/>
          <owl:Class rdf:about="#DSD"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="name">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#User"/>
          <owl:Class rdf:about="#Role"/>
          <owl:Class rdf:about="#Permission"/>
          <owl:Class rdf:about="#Object"/>
          <owl:Class rdf:about="#Operation"/>
          <owl:Class rdf:about="#SSD"/>
          <owl:Class rdf:about="#DSD"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:FunctionalProperty>
</rdf:RDF>
```

Appendix C. RDFS ontology of ARBAC family of models

```
<?xml version='1.0' encoding='Cp1252'?>
<!DOCTYPE rdf:RDF [
                <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
                <!ENTITY kb 'http://arbac.jyu.fi/kb#'>
                <!ENTITY rdfs 'http://www.w3.org/TR/1999/PR-rdf-schema-19990303#'>
]>
<rdf:RDF xmlns:rdf="&rdf;"
                xmlns:kb="&kb;"
                xmlns:rdfs="&rdfs;">
<rdfs:Class rdf:about="&kb;APA"  rdfs:label="APA">
                <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;ARA" rdfs:label="ARA">
                <rdfs:subClassOf rdf:resource="&kb;AdminObject"/>
                <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;ARH" rdfs:label="ARH">
                <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;AUA" rdfs:label="AUA">
                <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Ability" rdfs:label="Ability">
                <rdfs:subClassOf rdf:resource="&kb;AdminObject"/>
                <rdfs:subClassOf rdf:resource="&kb;Role"/>
                <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;AdminObject" rdfs:label="AdminObject">
                <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
```

```
<rdfs:Class rdf:about="&kb;AdminOperation" rdfs:label="AdminOperation">
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;AdminPermission" rdfs:label="AdminPermission">
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;AdminRole" rdfs:label="AdminRole">
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;AdministrativeCommand"
         rdfs:label="AdministrativeCommand">
        <rdfs:subClassOf rdf:resource="&kb;AdminOperation"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;AdvancedReviewFunction"
         rdfs:label="AdvancedReviewFunction">
        <rdfs:subClassOf rdf:resource="&kb;AdminOperation"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;AuthorityRange" rdfs:label="AuthorityRange">
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;DSD" rdfs:label="DSD">
        <rdfs:subClassOf rdf:resource="&kb;AdminObject"/>
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;GRA" rdfs:label="GRA">
        <rdfs:subClassOf rdf:resource="&kb;AdminObject"/>
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Group" rdfs:label="Group">
        <rdfs:subClassOf rdf:resource="&kb;AdminObject"/>
        <rdfs:subClassOf rdf:resource="&kb;Role"/>
```

```
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdf:Property rdf:about="&kb;ID" rdfs:label="ID">
        <rdfs:domain rdf:resource="&kb;Session"/>
        <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdfs:Class rdf:about="&kb;Mobility" rdfs:label="Mobility">
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Object" rdfs:label="Object">
        <rdfs:subClassOf rdf:resource="&kb;AdminObject"/>
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Operation" rdfs:label="Operation">
        <rdfs:subClassOf rdf:resource="&kb;AdminObject"/>
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;OrgUnit" rdfs:label="OrgUnit">
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;PRA" rdfs:label="PRA">
        <rdfs:subClassOf rdf:resource="&kb;AdminObject"/>
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Permission" rdfs:label="Permission">
        <rdfs:subClassOf rdf:resource="&kb;AdminObject"/>
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;PermissionPool" rdfs:label="PermissionPool">
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
```

```
<rdfs:Class rdf:about="&kb;Range" rdfs:label="Range">
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;ReviewFunction" rdfs:label="ReviewFunction">
        <rdfs:subClassOf rdf:resource="&kb;AdminOperation"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Role" rdfs:label="Role">
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;SSD" rdfs:label="SSD">
        <rdfs:subClassOf rdf:resource="&kb;AdminObject"/>
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Session" rdfs:label="Session">
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;SystemFunction" rdfs:label="SystemFunction">
        <rdfs:subClassOf rdf:resource="&kb;AdminOperation"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;UP_RRA" rdfs:label="UP_RRA">
        <rdfs:subClassOf rdf:resource="&kb;AdminObject"/>
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;UP_Role" rdfs:label="UP_Role">
        <rdfs:subClassOf rdf:resource="&kb;AdminObject"/>
        <rdfs:subClassOf rdf:resource="&kb;Role"/>
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;URA" rdfs:label="URA">
        <rdfs:subClassOf rdf:resource="&kb;AdminObject"/>
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
```

```
</rdfs:Class>
<rdfs:Class rdf:about="&kb;User" rdfs:label="User">
        <rdfs:subClassOf rdf:resource="&kb;AdminObject"/>
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;UserPool" rdfs:label="UserPool">
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdf:Property rdf:about="&kb;ability" rdfs:label="ability">
        <rdfs:range rdf:resource="&kb;Ability"/>
        <rdfs:domain rdf:resource="&kb;PRA"/>
        <rdfs:subPropertyOf rdf:resource="&kb;role"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;adminObjects" rdfs:label="adminObjects">
        <rdfs:range rdf:resource="&kb;AdminObject"/>
        <rdfs:domain rdf:resource="&kb;AdminPermission"/>
        <rdfs:subPropertyOf rdf:resource="&kb;objects"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;adminOperations" rdfs:label="adminOperations">
        <rdfs:range rdf:resource="&kb;AdminOperation"/>
        <rdfs:domain rdf:resource="&kb;AdminPermission"/>
        <rdfs:subPropertyOf rdf:resource="&kb;operations"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;adminPermission" rdfs:label="adminPermission">
        <rdfs:domain rdf:resource="&kb;APA"/>
        <rdfs:range rdf:resource="&kb;AdminPermission"/>
        <rdfs:subPropertyOf rdf:resource="&kb;permission"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;adminRole" rdfs:label="adminRole">
        <rdfs:domain rdf:resource="&kb;APA"/>
        <rdfs:domain rdf:resource="&kb;AUA"/>
```

```
                <rdfs:range rdf:resource="&kb;AdminRole"/>
                <rdfs:domain rdf:resource="&kb;can-assign"/>
                <rdfs:domain rdf:resource="&kb;can-modify"/>
                <rdfs:domain rdf:resource="&kb;can-revoke"/>
                <rdfs:subPropertyOf rdf:resource="&kb;role"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;authorityRange" rdfs:label="authorityRange">
                <rdfs:range rdf:resource="&kb;AuthorityRange"/>
                <rdfs:domain rdf:resource="&kb;can-assign"/>
                <rdfs:domain rdf:resource="&kb;can-modify"/>
                <rdfs:domain rdf:resource="&kb;can-revoke"/>
</rdf:Property>
<rdfs:Class rdf:about="&kb;can-assign" rdfs:label="can-assign">
                <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;can-assigna" rdfs:label="can-assigna">
                <rdfs:subClassOf rdf:resource="&kb;can-assign"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;can-assigng" rdfs:label="can-assigng">
                <rdfs:subClassOf rdf:resource="&kb;can-assign"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;can-assignp" rdfs:label="can-assignp">
                <rdfs:subClassOf rdf:resource="&kb;can-assign"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;can-modify" rdfs:label="can-modify">
                <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;can-revoke" rdfs:label="can-revoke">
                <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;can-revokea" rdfs:label="can-revokea">
```

```
                <rdfs:subClassOf rdf:resource="&kb;can-revoke"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;can-revokeg" rdfs:label="can-revokeg">
                <rdfs:subClassOf rdf:resource="&kb;can-revoke"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;can-revokep" rdfs:label="can-revokep">
                <rdfs:subClassOf rdf:resource="&kb;can-revoke"/>
</rdfs:Class>
<rdf:Property rdf:about="&kb;cardinality" rdfs:label="cardinality">
                <rdfs:domain rdf:resource="&kb;DSD"/>
                <rdfs:domain rdf:resource="&kb;SSD"/>
                <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;contains" rdfs:label="contains">
                <rdfs:range rdf:resource="&kb;OrgUnit"/>
                <rdfs:domain rdf:resource="&kb;OrgUnit"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;fromAbility" rdfs:label="fromAbility">
                <rdfs:domain rdf:resource="&kb;ARA"/>
                <rdfs:range rdf:resource="&kb;Ability"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;fromAdminRole" rdfs:label="fromAdminRole">
                <rdfs:domain rdf:resource="&kb;ARH"/>
                <rdfs:range rdf:resource="&kb;AdminRole"/>
                <rdfs:domain rdf:resource="&kb;AuthorityRange"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;fromGroup" rdfs:label="fromGroup">
                <rdfs:domain rdf:resource="&kb;GRA"/>
                <rdfs:range rdf:resource="&kb;Group"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;fromRole" rdfs:label="fromRole">
```

```
                <rdfs:range rdf:resource="&kb;Role"/>
                <rdfs:domain rdf:resource="&kb;UP_RRA"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;group" rdfs:label="group">
                <rdfs:range rdf:resource="&kb;Group"/>
                <rdfs:domain rdf:resource="&kb;URA"/>
                <rdfs:subPropertyOf rdf:resource="&kb;role"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasMobility" rdfs:label="hasMobility">
                <rdfs:range rdf:resource="&kb;Mobility"/>
                <rdfs:domain rdf:resource="&kb;PRA"/>
                <rdfs:domain rdf:resource="&kb;URA"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;name" rdfs:label="name">
                <rdfs:domain rdf:resource="&kb;APA"/>
                <rdfs:domain rdf:resource="&kb;ARA"/>
                <rdfs:domain rdf:resource="&kb;AUA"/>
                <rdfs:domain rdf:resource="&kb;Ability"/>
                <rdfs:domain rdf:resource="&kb;AdminObject"/>
                <rdfs:domain rdf:resource="&kb;AdminOperation"/>
                <rdfs:domain rdf:resource="&kb;AdminPermission"/>
                <rdfs:domain rdf:resource="&kb;AdminRole"/>
                <rdfs:domain rdf:resource="&kb;GRA"/>
                <rdfs:domain rdf:resource="&kb;Group"/>
                <rdfs:domain rdf:resource="&kb;Mobility"/>
                <rdfs:domain rdf:resource="&kb;Object"/>
                <rdfs:domain rdf:resource="&kb;Operation"/>
                <rdfs:domain rdf:resource="&kb;OrgUnit"/>
                <rdfs:domain rdf:resource="&kb;PermissionPool"/>
                <rdfs:domain rdf:resource="&kb;Range"/>
                <rdfs:domain rdf:resource="&kb;UP_RRA"/>
```

```
                <rdfs:domain rdf:resource="&kb;UP_Role"/>
                <rdfs:domain rdf:resource="&kb;User"/>
                <rdfs:domain rdf:resource="&kb;UserPool"/>
                <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;orgUnitSet" rdfs:label="orgUnitSet">
                <rdfs:range rdf:resource="&kb;OrgUnit"/>
                <rdfs:domain rdf:resource="&kb;PermissionPool"/>
                <rdfs:domain rdf:resource="&kb;UserPool"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;partOf" rdfs:label="partOf">
                <rdfs:domain rdf:resource="&kb;OrgUnit"/>
                <rdfs:range rdf:resource="&kb;OrgUnit"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;prerequisiteCondition" rdfs:label="prerequisiteCondition">
                <rdfs:domain rdf:resource="&kb;can-assign"/>
                <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;range" rdfs:label="range">
                <rdfs:domain rdf:resource="&kb;AuthorityRange"/>
                <rdfs:range rdf:resource="&kb;Range"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;rbacObject" rdfs:label="rbacObject">
                <rdfs:range rdf:resource="&kb;Object"/>
                <rdfs:domain rdf:resource="&kb;Permission"/>
                <rdfs:subPropertyOf rdf:resource="&kb;objects"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;rbacOperation" rdfs:label="rbacOperation">
                <rdfs:range rdf:resource="&kb;Operation"/>
                <rdfs:domain rdf:resource="&kb;Permission"/>
                <rdfs:subPropertyOf rdf:resource="&kb;operations"/>
```

```
</rdf:Property>
<rdf:Property rdf:about="&kb;rbacPermision" rdfs:label="rbacPermision">
		<rdfs:domain rdf:resource="&kb;PRA"/>
		<rdfs:range rdf:resource="&kb;Permission"/>
		<rdfs:subPropertyOf rdf:resource="&kb;permission"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;role" rdfs:label="role">
		<rdfs:domain rdf:resource="&kb;DSD"/>
		<rdfs:domain rdf:resource="&kb;SSD"/>
		<rdfs:domain rdf:resource="&kb;Session"/>
		<rdfs:range rdf:resource="&rdfs;Resource"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;session" rdfs:label="session">
		<rdfs:range rdf:resource="&kb;Session"/>
		<rdfs:domain rdf:resource="&kb;User"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;toAbility" rdfs:label="toAbility">
		<rdfs:domain rdf:resource="&kb;ARA"/>
		<rdfs:range rdf:resource="&kb;Ability"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;toAdminRole" rdfs:label="toAdminRole">
		<rdfs:domain rdf:resource="&kb;ARH"/>
		<rdfs:range rdf:resource="&kb;AdminRole"/>
		<rdfs:domain rdf:resource="&kb;AuthorityRange"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;toGroup" rdfs:label="toGroup">
		<rdfs:domain rdf:resource="&kb;GRA"/>
		<rdfs:range rdf:resource="&kb;Group"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;toRole" rdfs:label="toRole">
		<rdfs:domain rdf:resource="&kb;UP_RRA"/>
```

```
                <rdfs:range rdf:resource="&kb;UP_Role"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;users" rdfs:label="users">
                <rdfs:domain rdf:resource="&kb;AUA"/>
                <rdfs:domain rdf:resource="&kb;URA"/>
                <rdfs:range rdf:resource="&kb;User"/>
</rdf:Property>
</rdf:RDF>
```