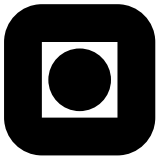
NORGES TEKNISK-NATURVITENSKAPELIGE UNIVERSITET

FAKULTET FOR INFORMASJONSTEKNOLOGI, MATEMATIKK OG ELEKTROTEKNIKK



MASTEROPPGAVE

Kandidatenes navn: Bjørn-Erik Sæther Stenbakk og Gunnar René Øie

Fag: Datateknikk

Oppgavens tittel (norsk):

Oppgavens tittel (engelsk): Role-Based Information Ranking and Access Control

Oppgavens tekst:

The healthcare sector is very information intensive. Large quantities of information from different sources are used by healthcare workers with different specialities and requirements. Therefore, it could be useful if the most important information was high-lighted. Also, having sufficient access control methods is a necessary precondition for any computer-based and networked use of confidential patient information.

This thesis examines using roles for both relevance ranking of, and access control to, healthcare information.

Oppgaven gitt:	17. januar 2005
Besvarelsen leveres innen:	18. juli 2005
Besvarelsen levert:	24. juni 2005
Utført ved:	Institutt for datateknikk og informasjonsvitenskap
Veileder:	Øystein Nytrø

Trondheim, 24. juni 2005

Øystein Nytrø
Faglærer

Abstract

This thesis presents a formal role-model based on a combination of approaches towards role-based access control. This model is used both for access control and information ranking.

Purpose: Healthcare information is required by law to be strictly secured. Thus an access control policy is needed, especially when this information is stored in a computer system. Roles, instead of just users, have been used for enforcing access control in computer systems. When a healthcare employee is granted access to information, only the relevant information should be presented by the system, providing better overview and highlighting critical information stored among less important data. The purpose of this thesis is to enable efficiency and quality improvements in healthcare by using IT-solutions that address both access control and information highlighting.

Methods: We have developed a formal role model in a previous project. It has been manually tested, and some possible design choices were identified. The project report pointed out that more work was required, in the form of making design choices, implementing a prototype, and extending the model to comply with the Norwegian standard for electronic health records.

In preparing this thesis, we reviewed literature about the extensions that we wanted to make to that model. This included deontic logic, delegation and temporal constraints. We made decisions on some of the possible design choices. Some of the topics that were presented in the previous project are also re-introduced in this thesis. The theories are explained through examples, which are later used as a basis for an illustrating scenario. The theory and scenario were used for requirement elicitation for the role-model, and for validating the model.

Based on these requirements a formal role-model was developed. To comply with the Norwegian EHR standard the model includes delegation and context based access control. An access control list was also added to allow for patients to limit or deny access to their record information for any individual. To validate the model, we implemented parts of the model in Prolog and tested it with data from the scenario.

Results: The test results show rankings for information and controls access to it correctly, thus validating the implemented parts of the model. Other results are a formal model, an executable implementation of parts of the model, recommendations for model design, and the scenario.

Conclusions: Using the same role-model for access control and information ranking works, and allows using flexible ways to define policies and information needs.

Keywords: role-based access control, information ranking, healthcare informatics

Preface

This thesis is written for the subject TDT4900 Master Thesis, Technology Studies (Sivilingeniør), at the Norwegian University of Science and Technology (NTNU), spring 2005. The thesis carries the study credit of one semester, and is the final subject of the MSc. level degree *Master i teknologi / Sivilingeniør*.

The assignment is given by the Department of Computer and Information Science (IDI,NTNU), and the assignment is formulated by Associate professor Øystein Nytrø. Nytrø has also been our supervisor during the work on this thesis.

For follow-up, contacting the authors, and further work, see
http://gunnarre.nvg.org/studies/rmhc/.

We wish to thank the Department for providing the facilities, and the following for support and inspiration:

- Øystein Nytrø, our supervisor
- Students sitting in room itv060
- Torbjørn Nystadnes at KITH
- Tone Kravdal Mathiesen

.....
Bjørn-Erik Sæther Stenbakk

.....
Gunnar Rene Øie

Trondheim
24. juni 2005

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The healthcare sector is very information intensive. By law, information is generated and recorded every time a patient is examined or treated. This information may be acquired from different equipment and persons, and then shared among and interpreted by healthcare workers. These professionals need and want different subsets of the medical information. They should not be compelled to spend excessive amounts of time or effort finding the information, lest they miss critical information or waste time better spent on patient care. E.g. a doctor would be interested in a patient's present medical conditions and medication, while a secretary would care more about information pertaining to administrative tasks.

All information about patients must be protected to ensure its confidentiality and correctness. On the social and legal level, information is protected by law and the morals of healthcare workers. As availability is increased by the use of information technology, there is an increase in both the opportunity for malicious activities and in their potential magnitude. Because of this heightened risk, information technology must take part in managing security.

There is a lot of work going on to provide decision support in healthcare, and structuring and standardizing healthcare information. Decision support is researched and applied at various levels of advancement. Scholars are seriously asking each other whether the benefits of higher levels of advancement are outweighed by more stringent demands on information entry. An issue related to traditional decision support is the emphasizing of information. If the decision support system is not advanced enough to make a diagnosis or noticing a mistake on its own, it could still be strong enough to emphasize information that is important. It is this level of decision support that is the focus of this thesis.

In this thesis, we want to show that a model for ranking and access control developed by us earlier actually could be implemented in computer-executable form, and then advancing this model just enough to fulfill requirements in the Norwegian laws and standards.

The rest of this introduction is devoted first to express the research goals and method of work, and second to an outline of this thesis.

1.1 Research Goals and Examples

From the problem description we have developed a few research goals. The research goals are used to guide and evaluate this thesis. They are also a means to make the thesis relevant and interesting. Because the research goals stated here cover a relatively large area, some design choices are made later on in this thesis. The research goals are also the basis for the discussion in chapter 9. A prerequisite to be able to perform these research goals is a literature study of the existing role-models. In addition we have to study the Norwegian electronic health record standard.

Research goal number one: Create a modularized role-model for access control and information ranking. It is important that the model is compliant to the Norwegian EHR standard. The possibility and value of a modularized approach is discussed and the model is verified both from formal definitions and a simple prototype role-model implemented in Prolog.

Research goal number two: We will clarify the concepts of information ranking, both through formal expression and through given examples. The concepts of information ranking is also to be represented through the Prolog prototype.

Research goal number three: We choose to use deontic logic to formalize the model we develop, thus we need to motivate and argue for this choice. It is outside the scope of this thesis to develop a new language, but we present a suitable formalism from the literature.

Research goal number four: The final research goal is to state the value of this thesis and how this can be used in future development.

1.1.1 Method

To illustrate different problems and possible solutions, we are using examples based on situations that can occur in the healthcare service. These examples are running throughout the thesis and be extended with more information and context to demonstrate various concepts and choices. Example 1 introduces how these examples look like and also introduces the main problem this thesis addresses.

Example 1
When the patient Elisa is put in contact with a hospital for a problem, the hospital secretary needs to book an appointment. To book the appointment, the secretary needs to know name and personal security number of Elisa. When the appointment takes place the doctor needs to have information about the patient's past medical history, the current problem and the doctor may need to order some tests or prescribe some drugs. Both the permissions given to and the information needed for the secretary and doctor is different. A solution to this is assigning the different users to different roles, for instance assigning Alice to the doctor role, and assigning permissions to these roles. For instance the doctor role has access to most of the clinical record information.

Based upon the examples we give, we create a scenario. The scenario helps to define some of the general requirement for the model. These general requirements and a set of requirements that we create to limit the scope of the model, form the basis for the model design. The model we develop is a combination of different approaches towards RBAC and we try to develop the model in different modules in such a way that each module adds functionality to the model. The model is subsequently implemented in Prolog and the implementation is tested in order to validate the model. This validation is done by using the scenario as input to the Prolog implementation of the model.

The implementation of the formal model provides the ability to express and test different qualities of the model. Thus an assumption is that the language may be implemented, thus making logic expression like:
$assigned\_users : (r : ROLES) \rightarrow 2^{USERS}$, the mapping of role r onto a set of users. Formally:
$assigned\_users(r) = u \in USERS | (u, r) \in UA$.

can later be written as a Prolog rule like:
$user\_assigned\_roles(user, role\_list)$.

The model is able to give a suggestion for how to support access control and how to handle the enormous amount of information in healthcare industry. Important to the model is that it complies with the described rules for access control in the Norwegian EHR standard [Nys05].

1.2 Thesis Outline

The rest of this thesis is organized as follows:
Chapter 2 presents a modal (deontic and action) first order many sorted language named $\mathscr{L}_{\mathscr{DA}}$. We also briefly motivate for the choice to use deontic logic in this thesis. The formalism is later used to express different properties of the role-model.

Chapter 3 introduces different approaches to access control. The chapter then focuses on different approaches and models for role-based access control, especially the NIST standard. The chapter also motivates for how RBAC can be used in healthcare and a review of the Norwegian EHR standard, which is very important to this thesis.

Chapter 4 motivates and explains the concepts of information ranking. In particular it describes how information can be ranked both as relevance and detail. The chapter also introduces information classes and information hierarchies, which are very important to make administration of access control and information ranking efficient.

Chapter 5 presents a scenario, which is used to test the model. From the scenario we describe some general requirements for an access control model. The chapter also presents a set of requirements for the model, that place a greater limitations on the development.

Chapter 6 presents a formalized role-model. The model is based on theories described in the thesis and is developed to comply with the Norwegian EHR standard. The model is able to support delegation, context dependent access and information ranking. The model is completed with rules and functions, that evaluate the access and ranking a user is given.

Chapter 7 is a presentation of the Prolog implementation of the model. The chapter presents the code structure and how the facts in the scenario are implemented and it presents the most important predicates of the implementation.

Chapter 8 presents the results and key findings of the testing and validation of the model.

Chapter 9 contains the discussion and evaluation of this thesis. It discusses the design choices we make and how successful they are. The chapter also presents possible application of the model and suggestions for future work.

Chapter 10 presents the conclusions for this thesis.

Chapter 2

The Formalism: Deontic logic

The possibility to express the role-model in a formal language is the key to proving the correctness and quality of the model. We have chosen to present the model in deontic logic and it is therefore necessary to present a formalism based on deontic logic. Deontic logic is originally developed for use in an ethical and legal context, but has later been found useful to express norms in any intelligent systems including artificial intelligence systems. The application of deontic logic in role-based access control as discussed Kolaczek [Kol02], concludes that deontic logic, as it formalizes the notions of obligation, prohibition and permission, corresponds in a natural way to specificity of access control activities.

This chapter presents a formalism for deontic logic and how we intend to use this formalism in the development of the model.

2.1 Language

This is a partial quotation of $\mathscr{L}_{\mathscr{DA}}$ as described by Pacheco & Santos in *Delegation in a Role-Based Organization* [PS04].

$\mathscr{L}_{\mathscr{DA}}$ is a modal (deontic and action) first order many sorted language. The non-modal component of $\mathscr{L}_{\mathscr{DA}}$ is used to express factual descriptions, and properties and relationships between agents. It contains a finite number of sorts, not related with agents or roles, and three special sorts: Ag (the agent sort), R (the role sort) and AgR (the agent in a role sort).

Those sorts have variables and constants, but there are no moment variables of the AgR sort.There may be functions between these sorts, but no function with Ag as co-domain is considered (the terms of sort Ag are either variables or constants). The terms of each of these sorts are defined as usual.
$\mathscr{L}_{\mathscr{DA}}$ also contains a finite number of role generators, generically denoted by rg, of sort $(\rightarrow R)$. There is always a role generator , denoted by *itself*.Moreover, for each role generator rg, there

exists a predicate (qualification predicate), denoted by $is-rg$ of sort (Ag) and denotes a property that an agent may have.

The terms of the sorts R and AgR are built as follows:

1. if rg is of sort $(\rightarrow R)$, then $rg()$ is a term of sort R (rg is written instead of $rg()$);
2. if t is a term of sort Ag and r is a term of sort R, then $t : r$ is a term of sort AgR.

$r, r_1, \ldots,$ is used to generically refer to roles, and $a, a_1, \ldots,$ to generically refer to a term of sort Ag (either a constant or a variable), and $t, t_1, \ldots,$ are used generically refer to terms of the appropriate sorts. Finally , $a : a$ is used as an abbreviation of *a:itself*, and $qual(a : rg)$is an abbreviation of $is - rg(a)$, and intuitively means that agent a is qualified to play the role rg.

The formulas of $\mathscr{L}_{\mathscr{D}\mathscr{A}}$ are inductively defined as follows:

1. if p is a predicate symbol of sort $(s_1, \ldots, s_n)$ and $t_1, \ldots, t_n$ are terms of sort $s_1, \ldots, s_n$, then $p(t_1, \ldots, t_n)$ is an atomic formula;
2. if B is a formula, then $\neg B$ is a formula;
3. if B_1 and B_2 are formulas then $(B_1 \wedge B_2)$ is a formula;
4. if B is a formula and x^s is a variable of sort s, then $(\forall_{x^s})B$ is a formula;
5. if B is a formula and $a : r$ is a term of sort AgR, then $E_{a:r}B, O_{a:r}B$ and $P_{a:r}B$ are formulas;

The other standard logical connectives $(\vee, \rightarrow \; and \; \leftrightarrow)$ and the existential quantifiers are introduced through the usual abbreviation rules, and parentheses may be omitted assuming the following priorities: first $\wedge$; then $\vee$; and finally $\rightarrow$ and $\leftrightarrow$. The forbidding operator is defined as follows: $F_{a:r} \stackrel{abv}{=} \neg P_{a:r}B$

2.1.1 Axiomatizations

We list some of the principles discussed by Pacheco & Camaro [PC03], where a more details can be found. It is assumed that all tautologies are axioms of this logic, and that the rule of Modus Ponens is valid.
The general properties of quantifiers and the generalization rule (if $\vdash B$ then $\vdash (\forall_x)B$), and the following axioms take part:

$(\forall_x)(B_1 \rightarrow B_2) \rightarrow ((\forall_x)B_1 \rightarrow (\forall_x)B_2)$
$B \rightarrow (\forall_x)B$, if x does not occur free in B
$(\forall_x)B \rightarrow B[xfree/t]$, for t a constant of sort s or a variable x_1 such that x does not occur free in B within the scope of $(\forall_{x_1})$.

($B[x^s free/t]$ denotes the formula we obtain when we replace (in B) the free occurrences of x^s by t.)
The formal properties of the action operator $E_{a:r}$ are described [below]:

Axioms	
(T_E)	$E_{a:r}B \rightarrow B$
(C_E)	$E_{a:r}A \wedge E_{a:r}B \rightarrow E_{a:r}(A \wedge B)$
$(Qual)$	$E_{a:r}B \rightarrow qual(a:r)$
$(\textit{Itself})$	$(\forall_x) qual(x : \textit{itself})$
Proof rule:	
(RE_E)	If $\vdash A \leftrightarrow B$ then $\vdash E_{a:r}A \leftrightarrow E_{a:r}B$

With respect to the formal properties of the deontic operators, and of the relationships between each other and with the action operator, the following axioms and proof-rules are considered:

Axioms	
(C_O)	$O_{a:r}A \wedge O_{a:r}B \rightarrow O_{a:r}(A \wedge B)$
$(O \rightarrow P)$	$O_{a:r}B \rightarrow P_{a:r}B$
$(O \rightarrow \neg P \neg)$	$O_{a:r}B \rightarrow \neg P_{a:r} \neg B$
$(O \wedge P)$	$O_{a:r}A \wedge P_{a:r}B \rightarrow P_{a:r}(A \wedge B)$
Proof rules:	
(RE_O)	If $\vdash A \leftrightarrow B$ then $\vdash O_{a:r}A \leftrightarrow O_{a:r}B$
(RM_P)	If $\vdash A \rightarrow B$ then $\vdash P_{a:r}A \leftrightarrow P_{a:r}B$
(RM_{EP})	If $\vdash E_{a_1:r_1}A \rightarrow E_{a_2:r_2}B$ then $\vdash P_{a_1:r_1}A \rightarrow Pa_2 : r_2 B$

2.1.2 Extensions to the Formalism

The formalism described needs to be extended in order to be able to handle delegation. The extensions dealing with delegation is already described by Pacheco & Santos and we quote a brief summary of this: The deontic characterization of a role in an organization is part of the identity of the organization and does not depend on the agent that holds that role in a particular moment. To capture this idea, deontic notions are attached to roles, but they are actually interpreted as applied to the holders of such roles, when acting in such roles(...). Thus no new operators are introduced, but just new abbreviations:

$$(O_r B) \stackrel{abv}{=} (\forall_x) qual(x : r) \rightarrow O_{x:r} B$$
$$(P_r B) \stackrel{abv}{=} (\forall_x) qual(x : r) \rightarrow P_{x:r} B$$
$$(F_r B) \stackrel{abv}{=} (\forall_x) qual(x : r) \rightarrow F_{x:r} B$$

[PS04, p. 213–215] We have not extended this formalism any further, but we believe it is sufficient to express the model we develop as it is.

2.2 Application of the Language

Although the language $\mathscr{L}_{\mathscr{D}\mathscr{A}}$ are explained pretty closely, the model we develop will not follow it completely. This is because there are some differences between the application of the language as presented by Pacheco & Santos [PS04] and how we intend to express the model. We have tried to identify where the model will be using the language and where there are important differences.

The formulas and standard logical connectives are of course valid for the model. The deontic properties of the language are also important to why we chose this language and are therefore a subject for use in the model. The language we have presented are used to express agents behavior in normative systems. Even if this should hold for the model, we want to limit the conceptual application to only include access control and information ranking in a healthcare organization. This implies for instance that we use the concept of users in a healthcare organization instead of agents in an organization. In addition the model have to express some context dependency and this also have to be solved.

A proof of how to translate a particular class of role-based access control modal model formulae into a form of the first order Horn clauses can be found in Kolaczek's article on deontic logic in RBAC [Kol02]. That proof is not included in this thesis.

The application of the language for expressing the model and how well suited the language actually is in addition to the discussion of the implementation of the language is thoroughly examined in chapter 9.

2.3 Summary

This chapter introduces the language $\mathscr{L}_{\mathscr{D}\mathscr{A}}$ which is a modal (deontic and action) first order many sorted language. This language has the properties we stated in research goal number three on page 1. We have also tried to foresee how well suited this language is to express the model and potential limitations or problems with the langauge, though this is thourghly discussd in chapter 9.

Chapter 3

Access Control

Role-based access control (RBAC) has become a well-accepted and well-known approach for authorization and access control. RBAC is a way of encoding security policies, where privileges are granted to roles instead of directly to subjects [FKC03]. RBAC cannot be standardized as a single model; it is more appropriate to think of it as a family of models, and this is reflected in the standardization work [San01].

This chapter begins with a short overview of access control. We then present the NIST standard for role-based access control, which is later used as the basis for the role-model in chapter 6. Thereafter we present some possible extensions to the NIST standard, which are needed in order to make a role-model compliant to the Norwegian EHR-standard. At the end of this chapter we present, in short, the Norwegian EHR standard. This presentation motivates for the use of RBAC and the extensions we have presented.

3.1 Access Control Types

This is a short overview of access control types, found as section 4.1 of the specialization project [SØ04, p. 17].

Access control is the process of granting subjects (users or processes) access to perform operations on objects (files, processes or data fields) [Gol99, pp. 30–45]. Role-based access control is an alternative to the more traditional form of access. There are three basic access control models [Fra03],[PHS03, pp. 565–589]:

- Discretionary access control (DAC), usually identity based. An access control matrix gives full control over what operations any subject can perform on any object; but a full matrix may be too complex to be practical for security management. To make access grants simpler, permissions are often given per object, as an access control list, or per subject, as a capability. The access control list is used by the owner of the object to give other subjects

access to the object. Capabilities are defined by the system administrator and give subjects access to objects based on the needs of subjects.

- Mandatory access control (MAC), based on levels, and found primarily in the military or other highly sensitive systems. This is based on classifying objects according to the sensitivity of the data and giving subjects a clearance, or authorization level. A user is granted access only when the user and object have corresponding clearance levels. The access method may permit reading from lower levels, but not reading from higher levels. It may permit writing to a higher level, but not writing to a lower level.
- Non-discretionary access control, usually role-based, centrally administrated with authorization decisions based on the roles individuals have within the organization. The system administrator grants and revokes system privilege based on the user's role.

The basic models may be used in pure form, or be combined with each other, to provide more fine-grained access control. On the flip-side, one could reduce granularity in other ways, e.g. by defining a privilege as a ordered set, where a higher privilege includes all lower privileges.

3.2 Role-based Access Control

The preferred information system of use for RBAC would exhibit the following characteristic [RC99]: For users; a large number of users, few security administrators, and frequent change of job responsibility. For data and applications; there are large numbers of data and sharing objects based on job functions. For enterprises; the data is owned by the enterprise, controlled by security administrators, before and after the fact audit, and periodic assessment of access control policy enforcement necessary. Ferraiolo, Cugini and Kuhn [FCK95] believe the principal motivation behind RBAC is the ability to express and enforce enterprise-specific security policies and streamline the typical burdensome process of security management.

The essence of role-based access control is that system permissions are assigned to defined roles rather than to individual users. And therefore the basis of RBAC is the concept of a role. A role is a type grouping that categorizes subjects based on various properties. These properties pertain to the functional responsibilities of the user in the organization.

3.2.1 RBAC Characteristics and Policies

RBAC policies are described in terms of users, subjects, roles, role hierarchies, operations and protected object. We have listed some of the characteristics stated by Ferraiolo, Cugini and Kuhn[FCK95]:

- Role-hierarchy defines roles that have unique attributes and that may contain other roles.
- Role authorization can be subject to the following:

1. The user can not be given more privilege than necessary in order to perform his/her job (principle of least privilege).
2. The role, in which the user is gaining membership, is not mutually exclusive with another role, for which the user already possesses membership (static separation of duty).
3. The numerical limitation that exists for role membership cannot be exceeded (cardinality property).

- Role activation involves the mapping of a user to one or possibly many roles. A particular role for a user can be activated if:
  1. The user is authorized for the role being proposed for activation.
  2. The activation of the proposed role is not mutually exclusive with any other active role(s) for the user.
  3. The proposed operation is authorized for the role that is being proposed for activation.
  4. The operation being proposed is consistent within a mandatory sequence operation.
- Role execution of an operation can take place only if the subject is acting within an active role.
- Dynamic separation of duty can be provided in RBAC if a subject can become active in a new role only if the proposed role is not mutually exclusive with any of the roles, in which the subject is currently active.
- Operation authorization can only be granted to a subject if the operation is authorized for the subject's proposed active role.
- Operational separation of duty requires that for all the operations associated with a particular business function, no single user can be allowed to perform all of these operations.
- Object access authorization requires subject access to RBAC objects to be controlled. A subject can access an object only if:
  1. The role is part of the subjects current active role set.
  2. The role is allowed to perform the operation.
  3. The operation to access the object is authorized.

3.2.2 RBAC Constraints

RBAC also enables administrators to place constraints on role authorization, role activation and operation execution. In RBAC separation of duty is a well known control principle in management. Separation of duty can be seen as mission critical combination of tasks required to

be performed by different people and it prevents accidental or malicious violation of business requirements [Cra03]. Separation of duty can be divided into static, dynamic and historical separation of duty. A more detailed description of static and dynamic separation of duty is given in the description of the NIST standard. Historical separation of duty is, as the name implies, based on historical or logged states and events in the system. Historical constraints are more difficult to enforce than static and dynamic constraints, but a proposed method of enforcing this would be to create a blacklist of request, that would cause a constraint to be violated. This method raises some new matters to the prior, for instance a poorly specified set of constraints may lead to situations where no user can invoke a particular method on a particular object.

3.3 The NIST Standard for Role-Based Access Control

The NIST standard is a proposed standard for RBAC. The proposed standard tries to resolve a situation where no single authoritative definition of RBAC exists. The standard solves this by unifying ideas from a base of frequently referenced RBAC models, commercial products and research prototypes. It does not try to standardize RBAC features beyond those that have achieved acceptance in the commercial marketplace and research community. The NIST standard is organized in two main parts: the RBAC reference model and the RBAC functional specification. These two main parts are in turn organized into four RBAC components; core elements, hierarchical RBAC, static separation of duties relations and dynamic separation of duties relations.

The reference model provides a rigorous definition of RBAC sets and relations. It has two primary objectives: to define a common vocabulary of terms for use in consistently specifying requirements and to set the scope of the RBAC features included in the standard. There is a reference model for each of the four RBAC components.

> “Each model component is defined by the sub components:
>
> - A set of basic elements sets;
> - A set of RBAC relations involving those element sets [...]; and
> - A set of Mapping Functions, which yield instances of members from one element set for a given instance from another element set.
>
> ”

[FSG+01, p. 232]

3.3.1 Core RBAC

Core RBAC, as defined in figure 3.1 from [FSG+01], covers the essential parts of RBAC and is required for any RBAC system. The core elements include five basic data elements, which are

users, roles, objects, operations and permissions. In addition the core RBAC model includes a set of sessions. The basic concept of RBAC is that users are assigned to roles, permissions are assigned to roles (as illustrated in example 2) and users acquire permissions by being members of roles. It is required that user-role and permission-role assignment is a many-to-many relationship. It is also required that users can be able to simultaneously exercise permissions of multiple roles. To introduce the concept of user sessions allows selective activation and deactivation of roles.

Example 2
Typical roles in a scenario are "Discharging doctor for the patient Elisa", "doctor on call on ward B" or a "secretary on ward B". For instance, the discharging doctor have read access to all record information for the current period of care and update access to recent information created by him-/herself. The doctor on call may only have read access to recent information. Both the actual role assignment and activation are discussed later in chapter 6.

A user is defined as a human being in the NIST standard. Although the concept of a user can be extended to include other machines or systems, we find this definition appropriate for this thesis. A role can be a job function or job title within the context of an organization with some associated semantics regarding the authority and responsibility conferred on a member of the role. Permissions in RBAC are defined as the operations that are permitted to be performed on an object. Operations are functions that is performed on the object. What types of operations that exist depend upon what kind of application is implemented, i.e. read, write and execute for a file system and create, update, delete for a DBMS. An object is an entity that contains or receives information. This can for instance be files or directories in an operating system or columns, rows, tables or views in a DBMS. It can also be some other system resource like printers, disk space or CPU cycles.

3.3.2 Hierarchical RBAC

This model component adds requirements for supporting role hierarchies. Figure 3.2 shows how role hierarchy are included in the references model. The NIST standard recognizes two types of role hierarchies: general hierarchical RBAC and limited hierarchical RBAC. General role hierarchies provide multiple inheritance, while limited role hierarchies are restricted to a single immediate descendant. Multiple inheritances provide important hierarchy properties. The first is the ability to compose a role from multiple subordinate roles, second is the ability to provide uniform treatment of user-role assignment relations and role-role inheritance relations.

The hierarchies are means for reflecting the organizational structure and responsibilities. Role hierarchies may also define inheritance relationships between roles. Because of this, role hierarchies, the security policy is easier to administer. An example of a role hierarchy for staff members in a hospital is given in figure 3.3. In this figure user membership is inherited top-down and role-permissions are inherited bottom up. We emphasize that this is not a complete staff member hierarchy for a hospital. In example 3 we illustrate how users can be placed in a hierarchy.

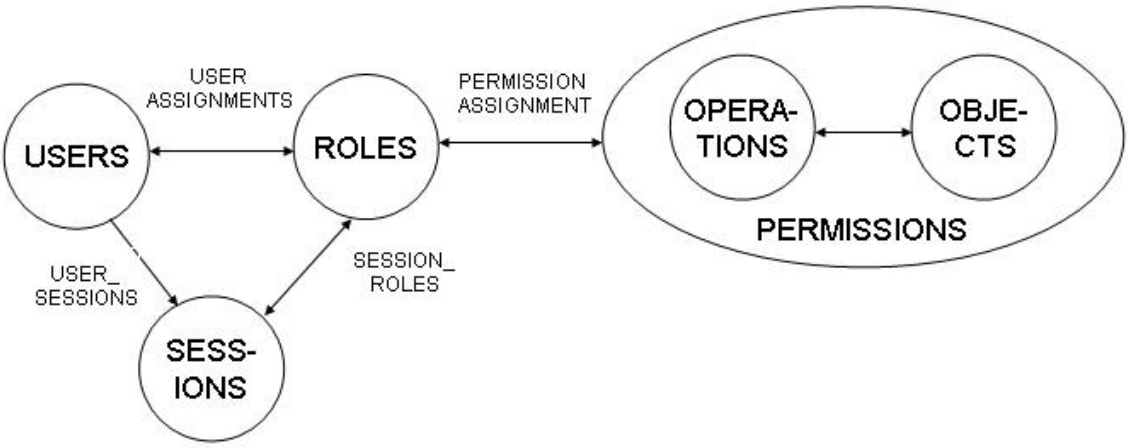


Figure 3.1: Core RBAC in NIST standard [FSG+01]

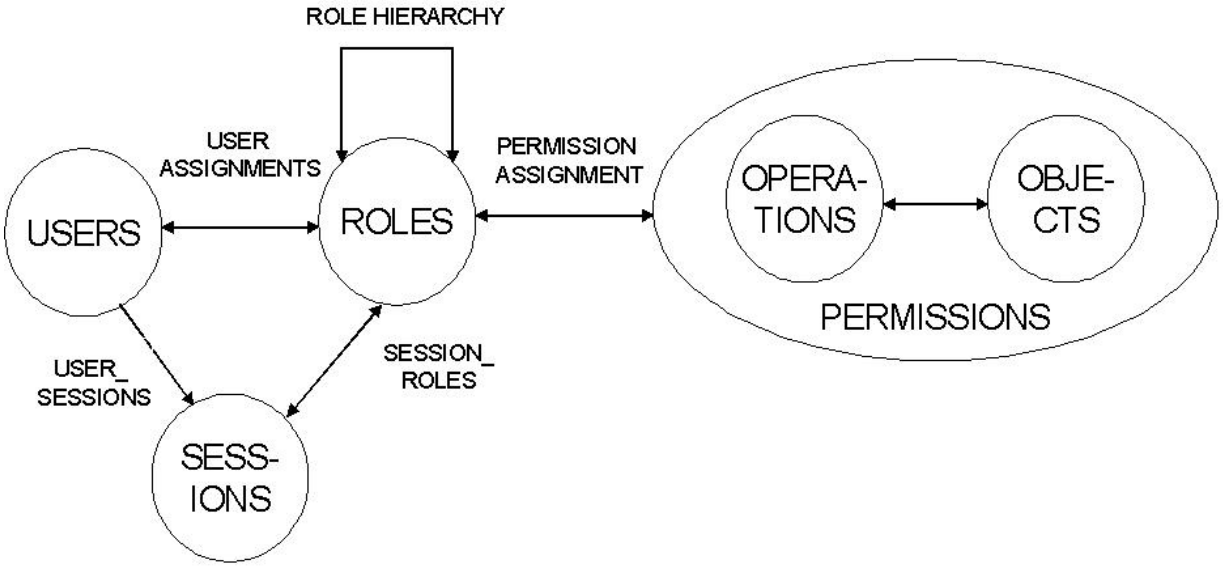


Figure 3.2: RBAC with role hierarchy in NIST standard [FSG+01]

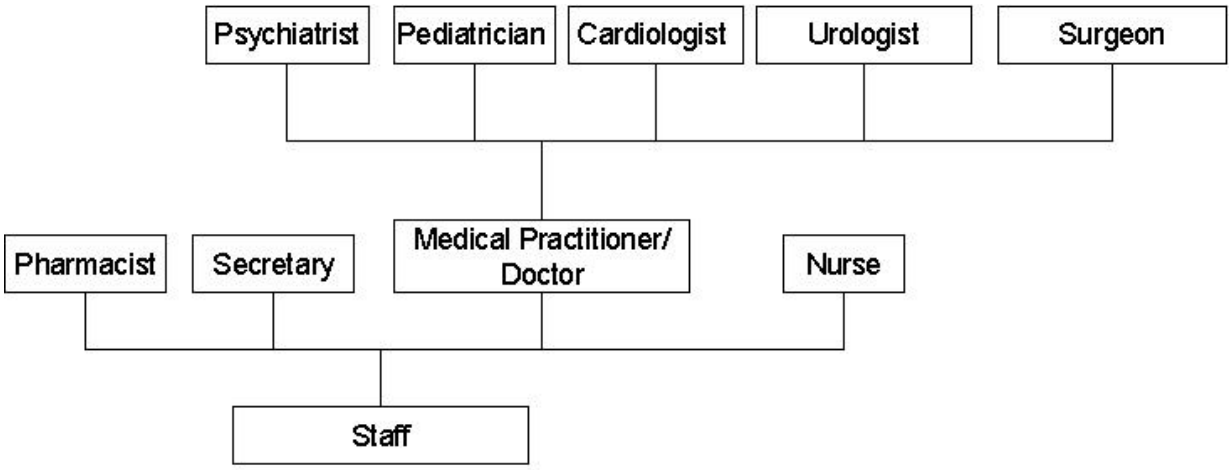


Figure 3.3: Example role hierarchy for staff members

Example 3
In figure 3.4 users are put into the role hierarchy. Alice is assigned to the cardiologist role and is authorized as a doctor and a staff member. Bob is assigned to the secretary role and is authorized as a staff member. Darth is assigned to the physiatrist role and is authorized as a doctor and a staff member. Ken is assigned both the urologist role and the surgeon role and is authorized as a doctor and a staff member. Although the role assignments for Alice, Bob and Darth could be represented in a limited hierarchy the role assignment for Ken could not. In a limited role hierarchy users would be omitted because core RBAC requires a many-to-many relationship for user-role assignment.

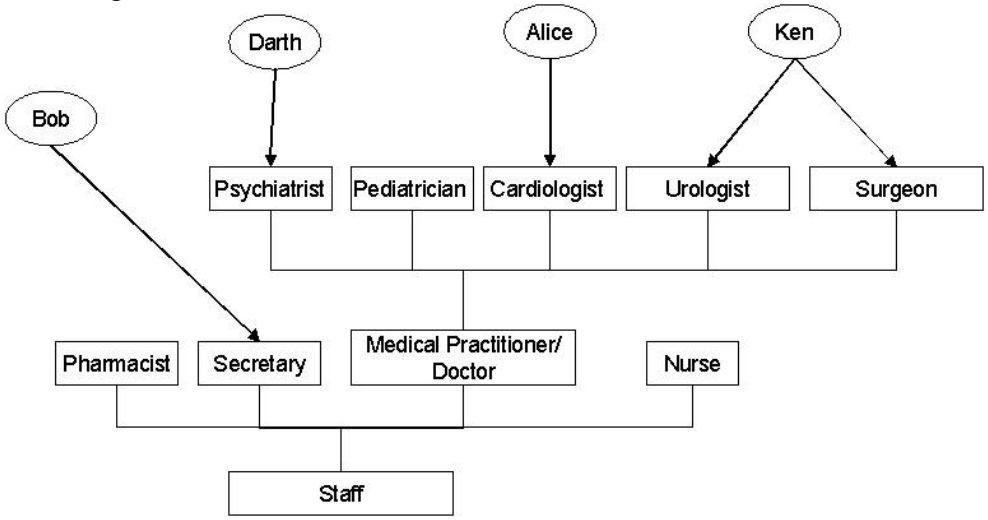


Figure 3.4: Users included in the hierarchy

3.3.3 Constrained RBAC

The NIST standard adds two types of constraints to the RBAC model, namely static separation of duties and dynamic separation of duties.

Static separation of duty is one way to enforce conflict of interests policies, that organizations may employ to prevent users from exceeding a reasonable level of authority for their positions. The simple explanation of static separation of duty, is that if a user is assigned to one role, the user is prohibited from being member of a second role, i.e. the two roles are mutual exclusive. Example 4 illustrates how two roles can be mutually exclusive. The static constraints defined in the NIST standard are limited to those relations that place restrictions on a set of roles and in particular on their ability to form user-assignment relations. SSD relations define and place constraints on a user's total permission space. As seen in figure 3.5 the static separation of duties is enforced on user-role assignment and the role hierarchy directly. SSD constraints can be defined both in the presence and absence of role hierarchies.

> *Example 4*
> A user that activates a role as a doctor signing a discharge paper can never activate a role as a doctor who countersigns on the same discharge paper. this task places a static separation of duty constraints on the two roles, because the signing and counter signing of the discharge paper has to be done by two different users.

Dynamic separation of duty is another way of enforcing conflict of interest, but differs from static separation of duty by the context, in which the limitations are imposed. DSD introduces properties that limit the availability of the permissions over a user's permission space. This is done by placing constraints on the roles that can be activated within or across a user's session. DSD extends the support for the principle of least privilege. We believe, that in a healthcare organization, this type of separation of duty is more relevant. This is because a user may require to act in several roles, but it may be necessary to prevent the user to act in these roles during the same session. Example 5 tries to illustrate this. As seen in figure 3.6, dynamic separation of duties is not enforced on user-role assignment, but on the roles acquired during a session.

> *Example 5*
> Alice will in a particular situation activate the role as an doctor on call on ward B, thus she has access to information regarding all patients treated at ward B. She is unable to access information regarding the patient on ward C, if the doctor on call on ward C is required to be another user, thus enforcing the principle of least privilege. Alice can still be both doctor on call on ward C, she just have to deactivate her role as doctor on ward B.

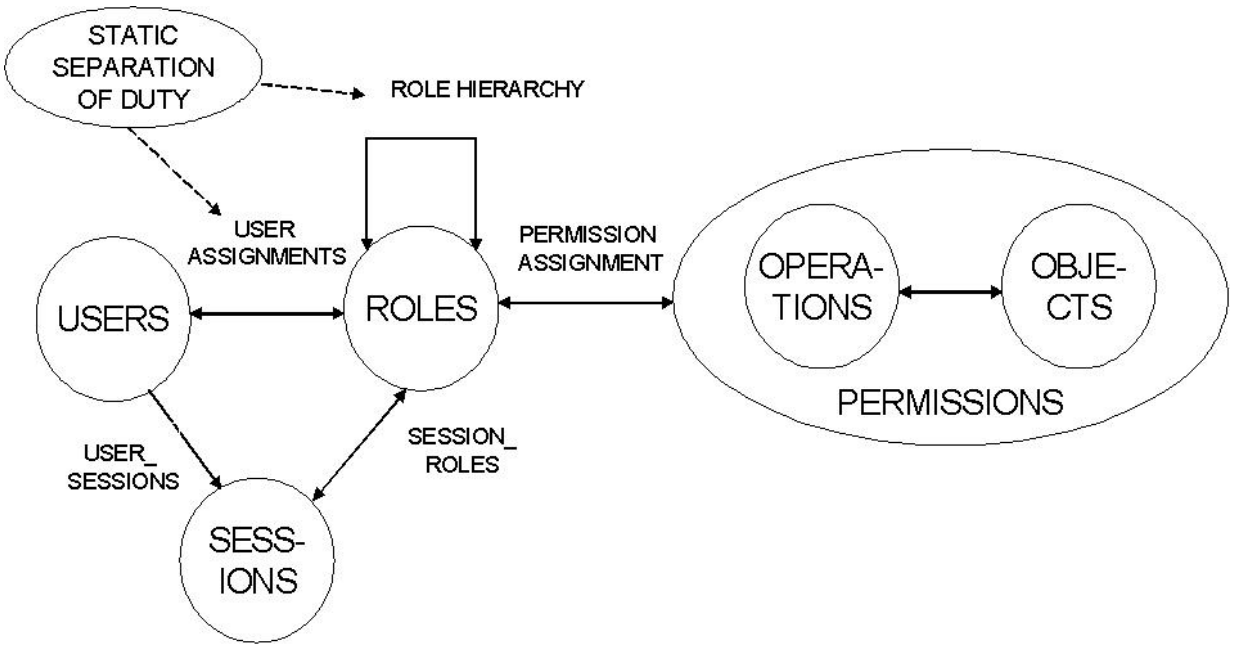


Figure 3.5: RBAC with static separation of duty in NIST standard [FSG+01]

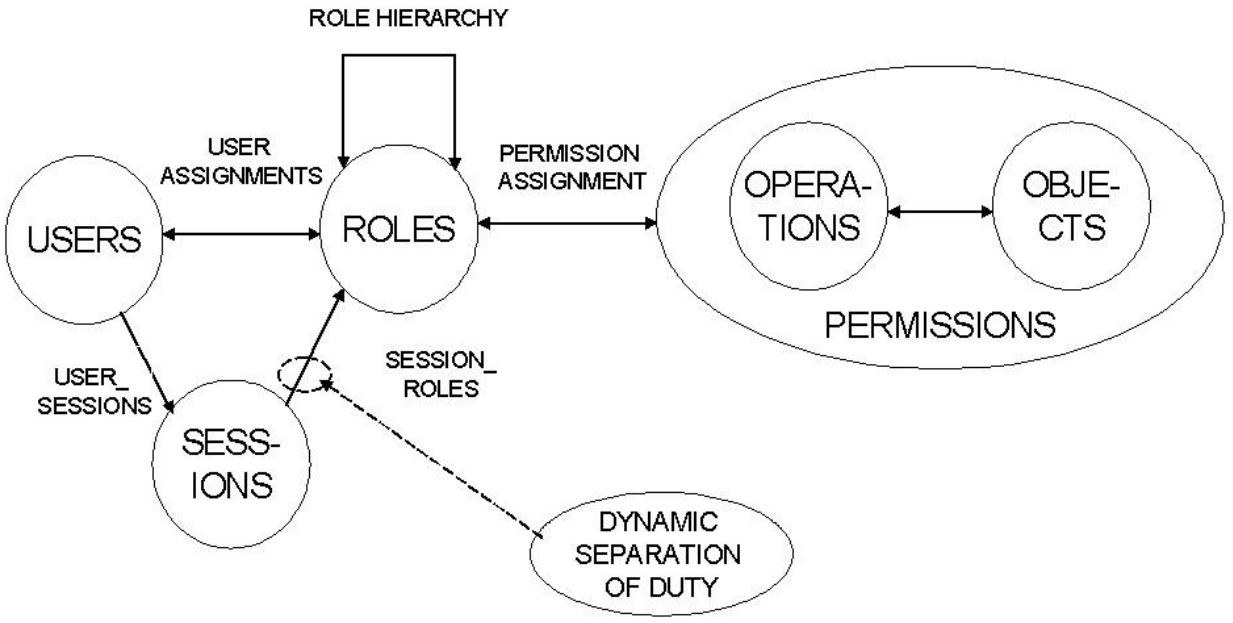


Figure 3.6: RBAC with dynamic separation of duty in NIST standard [FSG+01]

3.3.4 Functional Specification

In order to meet the requirements for each of the components defined in the earlier sections, the standard provides an overview of the functionality involved. There are three categories of functions in the specification, and their purpose are:

- **Administrative functions:** creation and maintenance of elements sets and relations for building the various components in RBAC models;
- **Supporting system functions:** functions that are required by the RBAC implementation to support the RBAC model constructs.
- **Review functions** review the results of the actions created by administrative functions.

We have not made use of the functional specification in a noticeable degree later in this thesis and therefore the reader is referred to the NIST standard [FSG+01] for a complete specification of these functions. The prolog implementation is made for a purpose that not requires a complete implementation of this specification.

3.4 Other Role-models

Several different other solutions to role-based models exist. We present some of these here and use them later to provide a reasonable extension to the NIST standard, which is used as a basis for the role-model. This improvement focuses on role-based delegation and context based access control. Role-based access control, in associating privileges with roles, provides the means to express access control, that is scalable to a large number of principals. Pure RBAC associates privileges only with roles, whereas applications, and indeed the Norwegian EHR-standard, often require more fine-grained access control. We present here both temporal RBAC and role-based delegation, as approaches to this problem. In addition we present in short some of the work done by Bacon [BMY02] which approaches context through parametrization of roles and delegation through an appointment model.

3.4.1 RBAC and Delegation

There are several different approaches to role-based delegation described in the literature, but it typically refers to delegation like this: One active entity in a system delegates its authority to another entity, to carry out some functions on behalf of the former [ZAC02]. In a system, this activity can be performed between:

- user and user
- user and machine

- machine and machine

Here we focus on user to user delegation, which is most relevant for a healthcare organization. In situations where personnel are on vacation, have too many patients or are ill themselves, they need to be able to delegate their obligations and permissions to other professionals. In addition there are some tasks, hence obligation and permissions, which are always delegated to achieve effectiveness in the organization; this might be a secretary writing a record entry on behalf of a doctor. This gives this informal definition of delegation, freely after Pacheco and Santos [PS04]

"user X, acting in role R1, delegates on user Y, acting in role R, the obligation to bring about $\varnothing$ *and permission to bring about* ψ *"*

There are two views of delegation summarized by Linn and Nystrøm [LN99]: In administratively directed delegation; an administrative infrastructure outside the direct control of a user mediates delegation, which means that a security officer must mediate delegation. In user directed delegation; any user's system may mediate delegation to resources under the user's control. In both situations it is necessary to enforce predefined delegation policies to prevent power abuses by individual users. Moreover there are several characteristics related to delegation. Barka and Sandhu [BS00] identify a set of these characteristics, which include permanence (refers to delegation in relation to time duration); totality (refers to the completeness of the delegation of roles); and levels of delegation (refers to the transitivity of delegation, i.e. whether a delegated permission can be passed on to others). Figure 3.7 illustrates how delegated roles are related to users and where delegation restriction occurs. Similar to this, Pacheco and Santos identify [PS04] some choices that have to be made when setting the security policy for delegation, including how delegation takes place and what is delegated. They formalize this using the language we adopted from them earlier in chapter 2.1, hence giving a deontic approach towards delegation.

Delegation and Revocation

Another issue relating to delegation is revocation. Revocation is important if for instance a professional should abuse his or her delegated permission. Then the delegator could revoke these permissions. The paper *A rule-based framework for role-based delegation* [ZAC01] suggests two ways of revoke delegation, using duration-restriction constraint or allowing user revocation. The former approach is vulnerable to the setting of time periods, but it is important when it comes to delegating obligations. The latter suggestion may be split into grant-dependent and grant-independent revocation. Grant-dependent revocation allows only the user, which has delegated the obligations and/or permissions, to revoke them, whereas grant-independent revocation allows any user in the original role to revoke any delegated role. User revocation is illustrated in example 6.

Example 6
If Alice is assigned the doctor role, she may delegate the task of writing a record entry to her secretary. If it is only Alice who may revoke this delegation, it is grant dependent revocation. If any user acting in the doctor role can revoke this delegation, it is grant independent revocation.

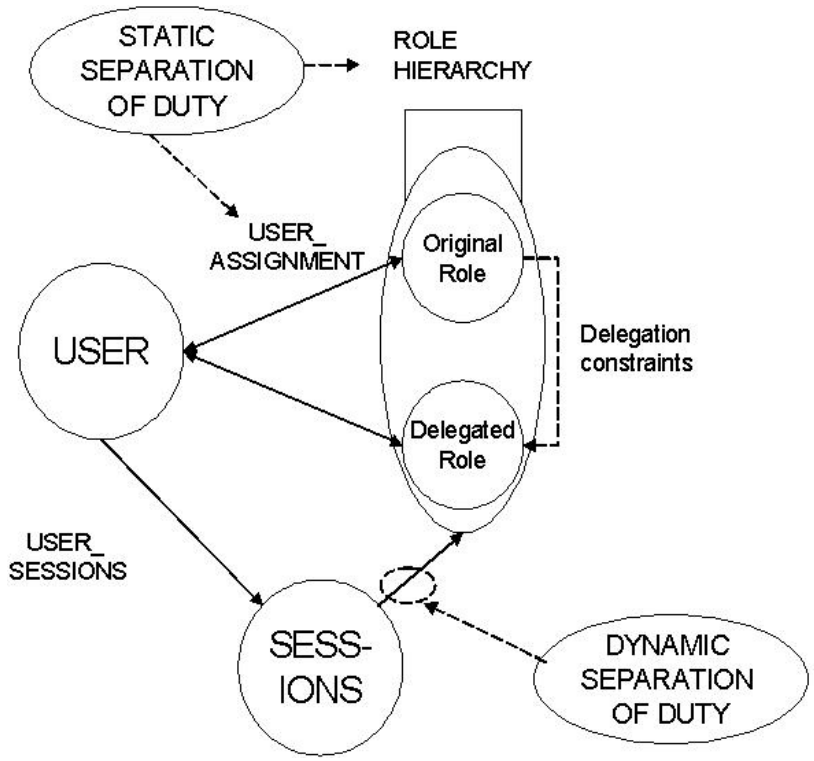


Figure 3.7: User-role relation with delegation

Delegation Depth and Width
To control role proliferation the paper [ZAC01] suggests to put constraints on delegation depth (transitivity) and width. The set depth dictates how many, if any, times the obligation or permission to be delegated may be delegated on to others. The width dictates how many persons may receive the delegated permissions at the same time. This can be done using boolean control or integer control. Having boolean control can impose restriction both on depth and width, letting the delegating user decide if the obligation and permissions can be further delegated. The drawback is that boolean control depends on individual users themselves to set constraints, if or if not further delegation may take place. By using integer control it is possible to decide the maximum depth, but there is not possible to control the width of delegation. A solution to this, is using a combination with integer control on high level delegation and boolean control on lower levels. Example 7 illustrates how boolean and integer control can be used.

Example 7
If Alice has the role as a senior resident, she can delegate the responsibility to write a record entry to an intern. If there is boolean control, Alice can decide if the delegation she makes can be further delegated (true or false). If she decides true, she is no longer in control of who receives the delegated responsibilities. If she decides false, the responsibility of writing the record entry has to be borne by the intern and no one else. If there is integer control, let's say of value 2, Alice may still delegate the task of writing the record entry to the intern, but the intern may delegate this further down to a secretary. The secretary can not delegate this any further regardless of what the intern might say.

When we develop and formalize the model, we try to identify which restrictions on delegation that are required in order to comply with the Norwegian EHR standard. Here, an important aspect of delegation is that all actions may be regarded as transitive delegations from the patient or a patient guardian. This is referred to as patient consent. Another problem we encounter, when formalizing delegation is informal or implicit delegation, which have not been addressed in the referenced literature.

3.4.2 Temporal RBAC

Temporal constraints are formal statements of access policies, that involve time-based restrictions on access to resources. They can be used to control time sensitive activities in workflow management systems, or they can be used to limit role availability and activation capability in designated periods. Depending upon the application requirement, there can be defined different temporal constraints. Ferraiolo, Kuhn and Chandramouli [FKC03] define these categories as:

- Temporal constraints on roles;
- Temporal constraints on user-role assignments;
- Temporal constraints on role-permission assignments.

To support temporal RBAC it is required to represent time. Two time-related concepts, periodicity and duration, are presented in Generalized TRBAC (GTRBAC) Part I [JBGL01]. The two concepts are used to represent periodicity-type temporal constraints and duration-type temporal constraints, respectively. The representation is as follows:

- **Periodicity:** A periodic time is represented by the pairs ([Begin, End], P), where P is the periodic expression, denoting a set (possible infinite) of periodic time instants, and [Begin, End] is a time interval, denoting the lower and upper bounds that are imposed on instants in P. For example the representation([17.01.2005, 18.07.2005], Friday 08.30) represents all the scheduled supervisor meetings for our master thesis.
- **Duration:** A duration is simply represented by a symbol D, which stands for either an expression or a numeral.

In GTRBAC temporal constraints have the generic form (X, E), where X is a periodic time or duration and E is an event. Constraints on roles in GTRBAC [JBGL01] are specified making a clear distinction between the concepts of role enabling and role activation, where an enabled role is said to be activated, when a user acquires the permissions assigned to that role. Role-enabling and -disabling are examples of events in GTRBAC. Example 8 illustrates how weekend shifts may be defined in an organization.

> *Example 8*
> An event of role-enabling for a periodicity-type will be (([01.01.2005, 31.12.2005], Friday 1600 - Monday 0800), enable weekend-shift), which states that in the year 2005 weekend shifts is from Friday at 1600 to Monday at 0800.

Because role-activation is done at the user discretion, there is no periodicity-type constraints, but there may be a duration constraint on how long a user can activate a role. Temporal constraints on user-role assignments are constructs to express the specific interval or duration a user is assigned to a role. Temporal constraints on role-permission assignment are constructs to express the specific interval or duration a permission is assigned to a role. There may also be specified other requirements associated with temporal constraints. For example that an event requires another event to occur. Such an event can be expressed by role-triggers.

Having temporal constraints allow for runtime requests as well. Bertino, Bonatti and Ferrari only describe [BBF01] run-time request for security administrators, but it is described both for users and administrators in GTRBAC. A user may make a request for activation of a role at run-time within a user session, for instance the emergency access – if emergency access is regulated with roles.

Supporting temporal constraints in the RBAC model affect the semantics of the role hierarchies specified in the model. The presence of various timing constraints, relating to role-enabling or role-activation and role-permission assignments, result in dynamic hierarchies called temporal hierarchies. These hierarchies do not possess the complete permissions inheritance found in the

role hierarchies of RBAC models, without the associated constraints.

Using temporal expressions in a model, allow for access control based on the context in which access is needed, and in addition it provides the means for ranking information based on this context. Run-time requests allow the context to change dynamically, and therefore support the policy of least privilege. In addition it could be very useful for changing relevance and detail ranking, described later in chapter 4. Having periodicity-type and duration-type expressions also make it possible to have temporal constraints on delegation.

Based on [JBGL01] and [BBF01] the periodic expressions and run-time requests are included in the model in chapter 6. We do not include temporal constraints in the implementation. This is left outside the scope of the model implementation, because this requires several supporting data stores and functional modules to be tested and verified. We also want to keep the complete permissions inheritance, found in the role hierarchies of the NIST standard, in the model to avoid any ambiguity.

3.4.3 Role Parametrization and Appointment Model

A model of OASIS RBAC is described by Bacon [BMY02]. Central to this approach is the idea of credential-based role activation. The credentials that the user possesses, together with states dependent on the environment, authorize a user to activate a set of roles. In OASIS, a role activation rule specifies the conditions that a user must meet, in order to activate a role. The OASIS architecture models these conditions in three categories: Prerequisite roles, appointments and environmental constraints. The prerequisite roles are roles needed in order to activate another role; appointments are a way to indirectly allocate privileges persistently through a secure persistent capability. This is illustrated in example 9.

Example 9
If Alice, who is assigned the doctor role, asks a nurse to place an order for drugs on her behalf (a privilege the nurse usually do not have). To enable this type of task assignment , the security administrator could introduce a role pharmacy nurse, who has the right to order drugs, and allow activation of pharmacy nurse to a principal, who is acting as a nurse and who has a valid recommendation from a doctor. Here Alice, who wishes to assign this task, is an appointee, the recommendation is an appointment certificate, and the nurse who accepts this recommendation is an appointee.

In OASIS, parameters may be included in the rules, that cover both role activation and access to an object or service. Parameters may be bound to such items, as the time of a role activation, the user id of a file owner, or an attribute of the object that is being accessed. The values that instantiate parameters are therefore context-dependent. Example 10 illustrates how user id can be parameter in a role. A special trait of the OASIS architecture is that it is possible to test context predicates during role activation, as well as at the time of access.

Example 10
If Alice activates the role doctor on duty, the activated role contains Alice's id as a parameter like this: doctor_on_duty(alice_id).

The technology used in OASIS can be used both for delegation, administrative roles, constraints and context based RBAC. In addition to this, it is possible to check both parameter values and relationships between those values; so evaluation of role activation conditions may express exceptions by default access control.

The OASIS architecture can be used to solve many of the same problems as traditional delegation and TRBAC solves. Parameterized roles could fundamentally change the underpinnings of the entire role-model. In addition most of the benefits from parametrization can be provided by traditional delegation and RBAC. It may be possible to combine the three models, but in our view the increased complexity of such a combination would outweigh the benefits.

With this reasoning, and the time limitations for the thesis in mind, we choose to follow up on traditional delegation and TRBAC, and leave parametrization and the OASIS architecture out of the scope of the model we develop. We further discuss role parametrization in chapter 9.

3.5 RBAC in Healthcare

The motivation for role-based access control in healthcare information systems, is that the characteristics stated in section 3.2.1 on page 10 are very close to what we find in a healthcare organization. There are large numbers of users, relatively few security administrators, frequent change of job responsibility and there are large numbers of data and applications to be shared. In addition, healthcare personnel are dependent on communication with other people for performing their responsibility. This dependency is based on the others' role, not identity. The identity is irrelevant as long as the person concerned has the knowledge associated with the particular role.

In Norway there are three major electronic health records (EHRs) in use in hospitals [Pro03], these are DocuLive, DIPS and Infomedix. DocuLive and DIPS use role-based access control. Roles in DocuLive are based on profession, ward assignment and clinical rights, such as the right to create new documents or the right to accept them. Unlike DocuLive, roles in DIPS are based on the relations between a user and its workplace and work tasks, and a patient and its period of care.

The discussion of how roles are used in different systems, is outside the scope of this thesis; we only note that the concept of roles have been in use in healthcare information systems for some time. The Norwegian EHR standard, which is reviewed in the next section, also acknowledge roles as means for access control policies, using action and role templates.

3.5.1 The Norwegian EHR Standard

The Norwegian EHR standard [Nys01] – Elektronisk pasientjournal standard (sic) - is developed by KITH as part of standardization program run by the Norwegian Ministry of Social Affairs and Health.

The Norwegian EHR standard contains requirements, recommendations, and general information models. The requirements stated are based on Norwegian laws and regulations [1] , while recommendations based on best practice are also given. The information models are very general with regard to specific medical cases.

The Norwegian EHR standard is being revised as we are writing this thesis, but we have received a draft of this standard [Nys05]. Here we focus on what this new edition says about access control. The standard divides the requirements into two major groups in relation to the Norwegian laws. These are:

- General principles, which are effective in most cases. The combination of professional secrecy and the use of consent is the basis for the legislation.
- Exceptions. For instance when a patient, or someone entitled to act on behalf of the patient, has required special conditions for access to recorded information. In addition, there are emergency situations and other situations, where the required consent has to be departed.

The Norwegian EHR standard describes over one hundred different requirements for access control, which need to be taken into consideration. To quote all of these would be tedious, so instead we present a short summary, which motivates for some of the design choices regarding the model. It is important to note that the access control model should have a high granularity and it must also support the many ways in which access may be granted.

Actions and Role Templates
Access is given as a consequence of a decision to bring about some action. This action must have an expressed and valid goal. The standard specifies that the EHR must have several action templates. Each of these action templates must clearly specify what information the action requires and what information the action produces. Thus, when a healthcare worker is tasked with performing an action, the action is entered into the EHR system, and the system automatically grant the necessary access (least privilege),according to the action template. There is a minimum number, required by the standard, of action templates that must exist. One of these is the permanent action, that allows emergency access to the record. Another is an action that gives some roles the ability to order new actions.

Actions may be specified to be performed by a specific person or role, or the action may be specified to be performed by any person. The action template specifies which roles and authorizations are needed to perform the action and thus also the access the record components. The

[1] Relevant laws and regulations are [opd00], [Hd03e], [Hd03a], [Hd03c], [Hd04], [okd01], [Hd03b], [Hd03f], [Hd03d] and [Hd02].

standard also supports healthcare workers performing actions on behalf of another, i.e. delegated obligation and permissions.

The standard also specifies role templates. These role templates have the right to create a new patient record, specify who is responsible for the record, restricting access to it on behalf of the patient, granting access on behalf of the patient and changing personal details. The person responsible for the record of the patient, usually has full access to the contents of the record, except when access to parts of it is restricted by the patient.

Administrative Access
Access to read and update guidelines and coding is also handled by the standard. This information is not privileged patient information, but it must be protected from unauthorized tampering. There may also be licensing agreements in place, limiting who gets to read the information. Special role templates grant this access. Access for system and maintenance work must also be granted through the access control model, without sacrificing security for expedience. System and maintenance work should rarely require access to the records of real patients. Some role templates also give access to change the organizational structure, and create and change role templates, action templates and roles, and assigning healthcare workers to roles.

Time Constraints and Time Control
A user's right to perform in a role have to has the possibility to be limited to one or more time intervals. Actions may also be given time limits for beginning and ending, thus reducing the timeframe for potential misuse, and reminding users/healthcare workers of actions, that have been forgotten. For larger organizations, the standard requires the EHR system to produce reports about actions, that have yet to be started, and about those actions that are yet to be completed. The system must also report all emergency access, and self-approved actions possibly used to extract information illegally.

Patient Access and Preference
The record must reflect a patient's request to see the record, whether or not this request is approved, or whether the record is seen by the patient or somebody acting on the patient's behalf.

The patient has a right to limit access to certain parts of the record, and may specify for which service provider groups, roles, or specific individuals the information is available or unavailable for. This may apply to the entire record, or just to certain topics. If the patient has given no instruction regarding access, the general rule applies: Access is granted to the role performing an action.

Implementation Consideration
The access control model in the Norwegian EHR standard takes this into consideration:

- Which patient the access concerns;
- Who is attempting access (role, abilities, authorizations, and unique identity);
- Which unit the patient is being treated at, and whether it's the same unit from which the access is attempted;

- Why the access should be granted (actions, action templates, goals);
- What data is accessed (which components are accessed);
- What mode of access is attempted (reading, writing or changing information);
- whether or not the patient has granted special access to, or restricted certain parts of the record.

3.6 Summary

This chapter gives a short introduction to different access control types. The chapter then focuses on different approaches and models for role-based access control, especially the NIST standard, which is used as the basis of the model we develop. The chapter also reviews different approaches towards delegation and context based access control. The model in chapter 6 applies parts of the approaches for traditional delegation and temporal RBAC. The chapter also includes a short review of the OASIS architecture, which uses role parametrization, but this approach is omitted due to incompatibility. The chapter finally describes how RBAC can be used in healthcare and a review of the Norwegian EHR standard, which is very important to the work we are doing.

Chapter 4

Information Ranking

In the repost we wrote for the specialization project, [SØ04], we tried to reason about how to rank information. The idea is to let the system choose what kind of information a user in a particular role would require, or at least being able to tell what the most important information to that role is. The system is going to do this based on predefined levels of relevance and detail a particular information object is given for a particular role. The source of our work on relevance ranking is intelligent user interfaces.

In this chapter we first present a brief overview of intelligent user interfaces and relevance ranking and how these two relate to one another. The concept of information ranking is then further developed with regards to relevance and detail. The last part of this chapter is used to give a detailed presentation of information classes and hierarchy.

4.1 Information Ranking and User Interfaces

There is a recognized problem associated with information overload and limited human memory in the healthcare industry. Therefore several different implementations of computer-based systems, which help healthcare providers use information to make better decisions, exist. The purpose is to improve the quality and reduce the cost of healthcare. Some of these systems also provide decision support to the healthcare employee. The success of these systems is not unconditional, some have ”made it” and some have not.

Although user interfaces are outside the scope of this thesis, we find it necessary to address this topic briefly because of its importance to the application of the model we develop. The reason for this is that having a dynamic ranking of record information could be used in the design of intelligent user interfaces. Having data about what pieces of information are most important to the user, the user interface can be used to only display these. The interface may allow the user the option to also view the less important information. A key feature of intelligent user interfaces

is that they may show a different "face" to the user, depending on which part of the working process the user is in.

The computer systems, that would be conventionally classified as intelligent, are those that aim to develop models of their users, i.e. the user interface has information about the users' behavior and acts thereafter. User modelling has traditionally analyzed the goals of an individual user, and identified what operations the user needs to perform in the accomplishment of that goal [Kie99], with the purpose of improving user interfaces. The concept may be expanded to also include descriptions of human-computer interaction styles and for presenting information on computer displays.

4.1.1 Proposed Framework for Ranking Record Information

Bayegan [Bay02] with Nytrø and Grimsmo [BØNG01], propose a framework to be used in ranking record information. *Decision frames* are the selections of what information should be shown at each time. Bayegan proposes classifying record information in a content ontology called *CareActType*, and that each decision frame would be a selection of classes from this ontology. Thus, after selecting a medical problem, each information object related to that problem would be selected for display, depending on which class it belongs to. The top super-classes in the CareActType information classification are actually not information types in themselves, but are named by phases in the care process. By relating information classes to simple phases, the content ontology contains a choice of what information is important in what phase.

To know which particular decision frame to show, the record system needs to know the phase that the user is in. Bayegan proposes using *traces* to establish process knowledge without disturbing the user. The record system could analyze traces, action patterns, in the information that has already been entered into the system. Trace recognition could become more advanced depending on the quality of artificial intelligence, becoming able to recognize not just what phase of care the user is in, but whether treatment is working and the patient is getting better. This is a difficult task, but one that becomes simpler if we satisfy ourselves with a small number of defined phases of care.

Another way, that the system can keep up with phases, is having the user explicitly change phase. A common example of this is configuration *wizards* used when installing software applications.

4.2 Relevance and Detail Ranking

Our view of information ranking is related to decision support, but differs in the way that it does not give any support regarding decisions. It rather sort out what information the user should concentrate on and hide seemingly unimportant information. Similar to what is described by Bayegan [Bay02], the information that is hidden is not unavailable to the user, unless the user

are denied access according to the access control policy. In order to do this we have found it necessary to expand the concept of relevance ranking to include two different parameters, namely relevance and detail. We also modelled this concept in the specialization project, [SØ04], for a more limited model and we now try to include this concept in a extended role-model.

4.2.1 Relevance Ranking

The relevance of information varies among the different positions and different situations in the organization. Because we want to use the role-model to express the context of access we may also use this to express the relevance in a particular context. This is important because relevance ranking has to be able to answer questions such as: "To whom is what information relevant?" And, "at what time is this relevant?"

The information is ranked with integers to have a more fine grained relevance ranking than boolean ranking would give. Example 11 illustrates how integer control would affect the view of information. The same approach was also used in the specialization project, [SØ04]; the difference to the extended model in this thesis is that it is more suited to rank information according to context..

Example 11
The information of a patients allergies has different relevance level for Billy acting in a internist role and for Ben acting in a radiologist role. Even though both roles are senior to the doctor role and therefore have access to patient information. The patients allergies are of a very high relevance for the internist when performing the task of prescribing medicine. For the radiologist this information would probably not affect the work at all, but it can not be ignored, so it is given a lower relevance level. Using relevance levels makes it possible to answer questions like these: *What information is relevant and how relevant is it for Alice acting in a internist role?*

The empirical work of investigating and make suggestions for the relevance of different information is outside the scope of this thesis, but we use the scenario to make our own guesses and test the role-model according to this. Relevance is the measure of how crucial the information is and what information is most likely to be needed for the particular role. Every role defined in the role set is given a particular relevance ranking for every information class see, 4.2.3. Because a user can activate multiple roles in one session, also referred to as the functional role of the user, relevance and detail are ranked according to several roles. Just like permissions are assigned to roles, relevance and detail are assigned to roles as shown i figure 4.1. The concept of relevance ranking is formally defined in the role-model in chapter 6.

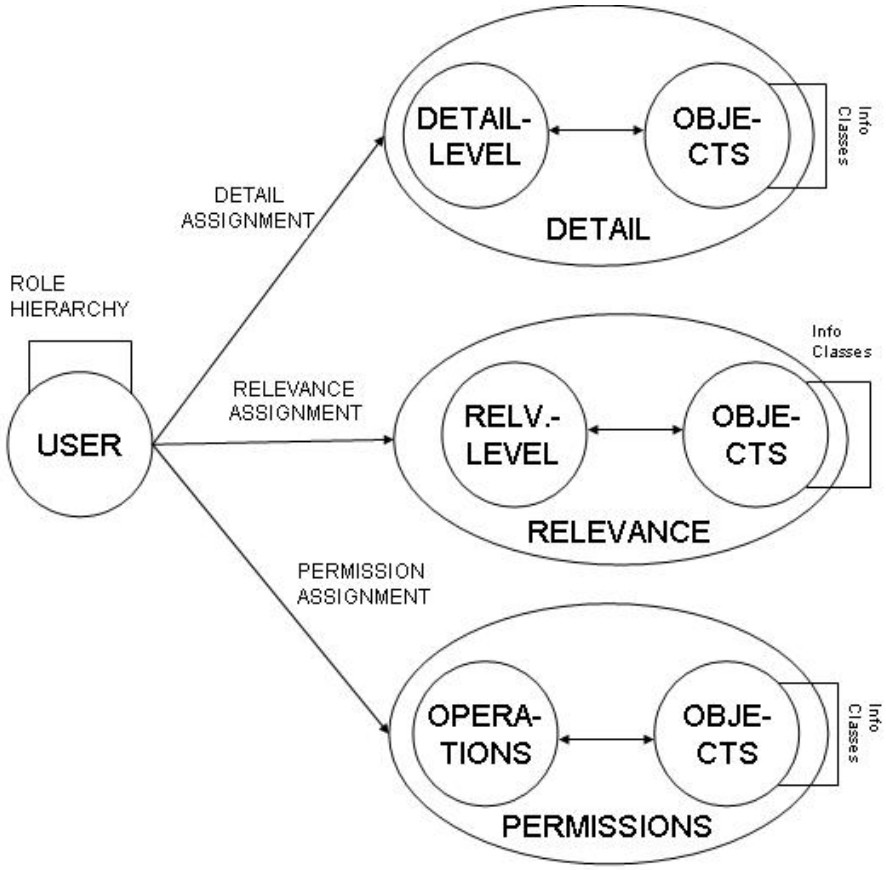


Figure 4.1: Relevance and detail in relation to roles and permissions

4.2.2 Detail Ranking

Detail ranking has similar properties as relevance ranking, but detail ranking differs in that it tries to measure the voluminously of the information that should be presented. Detail ranking is therefore an important mechanism of ”hiding” information. This allows the user to have different views of the information. Example 12 illustrates what we mean by different detail levels and how we can make use of this.

> *Example 12*
> It may be sufficient for a nurse to know that a patient has received medication, but if needed, the nurse can further investigate what kind of medication and the dosage. The doctor treating the patient, may be interested in the type and dosage of the drugs without having to ”investigate” further.
> Using detail levels make it possible to answer questions like these: *Given the patient Elisa, at what detail levels does Alice see Elisa's information objects; and at what detail level does Bob see those objects?*

The detail level is also ranked with integers. Here, as with relevance ranking, an important extension from the earlier work we have done is the possibility to do a more precise detail ranking of information in relation to the context.

4.2.3 Information Classes and Hierarchy

We stated in the specialization project, [SØ04], that it is necessary to put objects into information classes to allow for effective ranking of information. These classes can be put into hierarchies for even more effective ranking. Figure 4.2 shows information classes that are put in different hierarchies. Each object is related to at least one information class. An example of how a hierarchy of information classes might look like is given in figure 4.3. Note that the inheritance is top down with the most general information class in top of the hierarchy.

An object is, as mentioned in section 3.3.1 on page 12, an entity that contains or receives information. This can for instance be files or directories in an operating system or columns, rows, tables or views in a DBMS. It can also be some other system resource, like printers, disk space or CPU cycles. In general the set of objects that are covered by a RBAC, are all objects listed in the permissions assigned to roles. When we discuss objects in relation with an information class, treat objects as pieces of information about patients. This is because the main purpose of the model is to regulate access to record information. We have not found it relevant to describe the model used for access control to other types of objects. Also, the Norwegian EHR standard [Nys05] does not consider access to anything but information components and uses the term *EPJ fragment* to describe the smallest possible piece of stored information. Thus, in the model we develop objects are equal to EHR fragments.

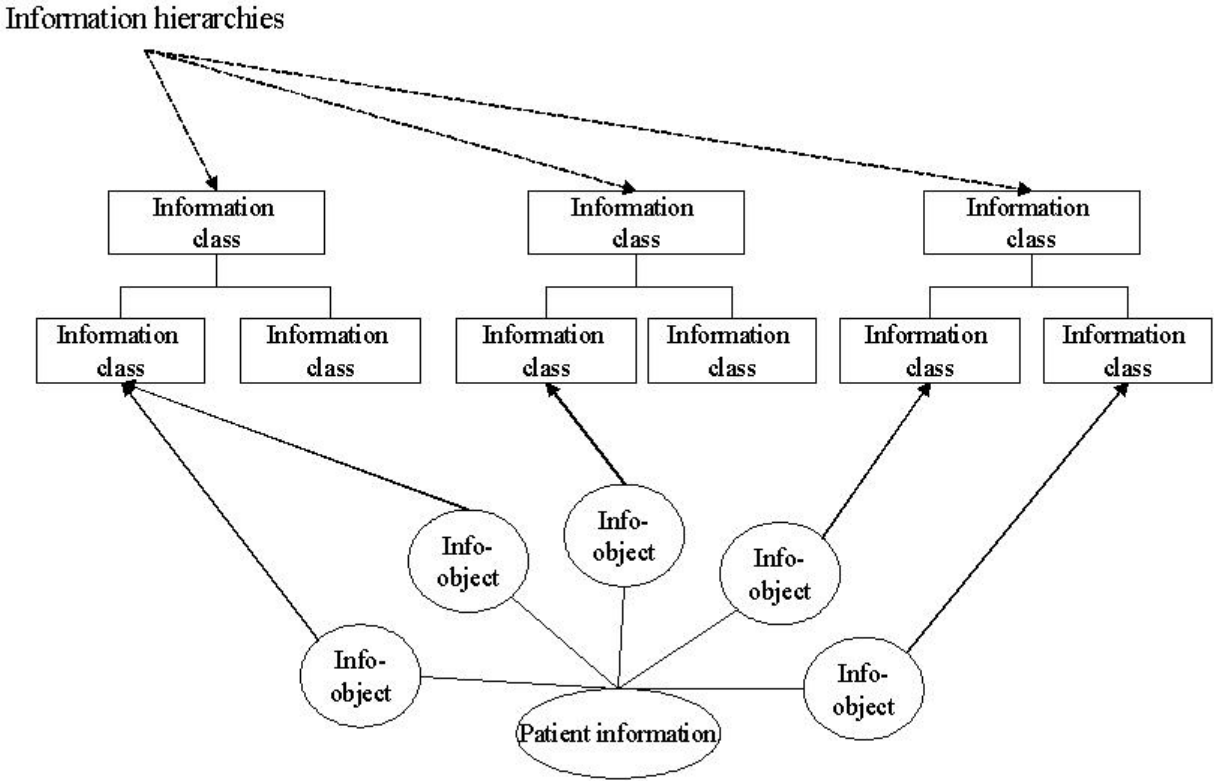


Figure 4.2: Information, information classes and info-objects

The use of information classes are similar to the classifying of record information done by Bayegan, [Bay02]. It is also stated by Guan [Gua04] as an practical mean to provide permission on classes of information, instead of single objects. Modelling of patient data is also well known from decision support systems, where the patient information model is concerned with the representation of patient data and its mapping to institutional electronic medical record data models. A difference here is that all we are concerned with, is the administration and implementation benefits from having a hierarchy and not the exact representation of the data.

Classifying objects in information classes and putting these classes in hierarchies provide the practical mean to present information with the correct permission, relevance and detail. We have illustrated this in example 13.

Example 13
A secretary, unless delegated some responsibility, may only access a patient personalia, thus access need to be denied for every other information class. In the hierarchy in figure 4.3 denying the secretary access to the ”Medical History” automatically deny the secretary access to the junior class ”Treatment”. For a doctor having the relevance level ranked for the class ”Test Data” all the classes junior to ”TestData” are ranked with at least the same value.

4.3 Summary

This chapter includes two major parts, both relating to information ranking. The first part considers information ranking in relation with user interfaces. Although user interfaces is outside of the scope of this thesis this is included because it is an application of information ranking. The chapter also includes a short review of the work of Bayegan with Nytrø and Grimsmo, [BØNG01]. The framework they present to use for ranking relevance of record information, is part of the motivation for this thesis. The second part of this chapter is used to explain how we want to use information ranking in the model we develop. In particular we describe how information can be ranked both as relevance and detail. The chapter also introduces information classes and information hierarchies, which are very important to make administration of access control and information ranking efficient.

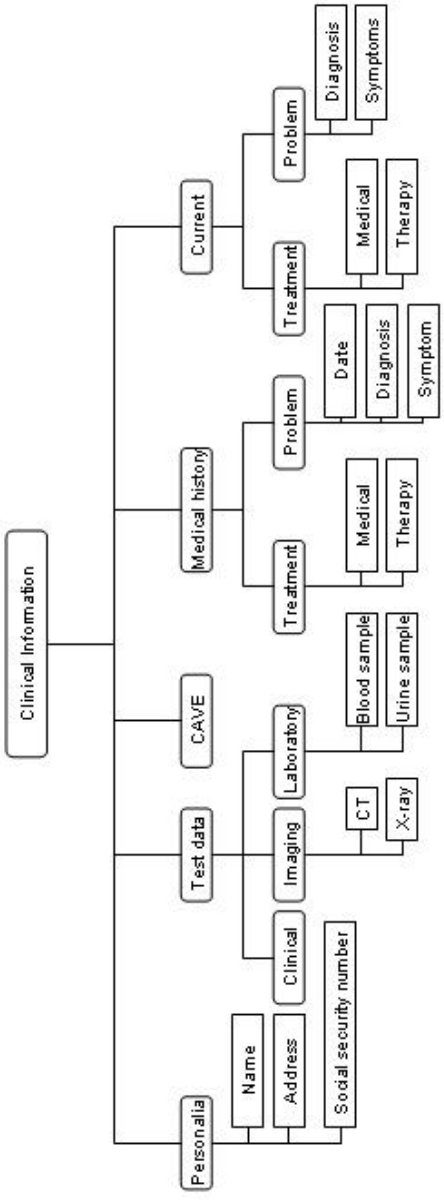


Figure 4.3: Example hierarchy

Chapter 5

Prerequisites for the Model

In this chapter we present a scenario from a healthcare organization. The central point to this scenario is what information is most important to whom, and what information is to be kept confidential. The scenario is important to test and validate the model. As already indicated in this thesis RBAC is useful in a healthcare organization and the scenario is used to support this indication. From the scenario we derive the general requirements for a RBAC model, that is used in a healthcare organization. The scenario is partially based on a scenario described by Thomas [Tho97], but is further developed to meet this thesis requirements and match the examples given earlier. It involves several different actors with different roles in the care process, thus the actors have different goals and information needs. These goals and information needs are described for each actor, but we note that this is done according to our own beliefs for the purpose of making the model. The scenario is not based on interviews of healthcare personnel or other empirical research. Making the rules and hierarchies, that are useful and correct for specific users, is not a part of this thesis.

In the second part of this chapter we present the requirements, that set constraints on how we are supposed to build the role-model. These requirements help identify what the model is supposed to demonstrate and how it is supposed to demonstrate it.

5.1 An Illustrative Scenario

This patient is a 28 year old female named Elisa. She suffers from diabetes mellitus and hypertension. For these conditions she is prescribed with insulin, a beta blocker, and a thiazide. Elisa is brought to the hospital and requires acute medical attention for hypoglycemia. At the hospital the scenario starts out with a visit to the emergency room (ER). At the ER the patient is quickly screened by an intern, who determines that the patient needs to be admitted to the internal medicine ward. The intern that admitted the patient, subsequently prescribes a medicine to stabilize the blood sugar, and in addition requests a CT, because the patient has suffered from

a head trauma when passing out because of the hypoglycemia. The internist, who is consulted at the internal medical ward, decides to prescribe a new dosage of insulin. The insulin is delivered by a nurse, who also keeps track of the blood sugar and blood pressure of Elisa. After being stabilized, Elisa is sent to radiology for CT and then returned to the medical ward. During the night the patient suffers a heart attack. She is immediately transferred to the cardiology unit, where a cardiology team undertake the care of the patient. When the patient's heart condition is stabilized, the patient is transferred back to the medical ward. After spending a few more days in the hospital, the patient recovers, is discharged, and is told to see his/her primary care doctor for follow-up care. The discharge paper is dictated and signed by one of the involved doctors, but is written by a secretary and is counter signed by a senior resident.

5.1.1 Actors and Information Requirements

An important aspect of role-based access control is the identification of roles with belonging access permissions. This process, called role engineering [Coy96], depends on thorough organizational knowledge and specific work flow understanding. Though role engineering is important to system design, it is not a major focus of this thesis. Instead we solve this by identifying the roles that actually are in the scenario and make a very short description of their responsibilities, access privileges and information needs in the current situation. We note that this description is NOT based on medical experience.

For any given situation each user may activate several different roles, just like in this scenario. Thus, in order to have the access privileges and information ranking described here, there is a need for information regarding the context. The information about the context can be expressed as roles that the user can activate. Thus, the roles the users have activated are in reality intern_on_call_at_ER, nurse_at_medical_ward and so on. For simplicity we have not done this here, but in chapter 8 the role activation with several roles is done.

Patient (Elisa) Is the "owner" of the information about herself.

Intern (Roger) Is the first in contact with the patient. He needs to do a screening of Elisa and decide on the appropriate action. He needs the medical history of the patient, but is denied access to certain information such as psychiatric information. Important information is current medical conditions and medication.

Internist (Billy) Is contacted after the first screening and is interested in the blood pressure and blood sugar levels in particular and also need to be informed about current medication.

Nurse (Betty) It may be sufficient for a nurse to know that Elisa has received insulin. If Elisa hasn't received her insulin, the nurse also needs to know the correct dosage in order to prescribe Elisa the insulin.

Radiologist (Ben) Has to perform a CT on the patient, but only needs information regarding the head trauma the patient is suffering from. He later has to examine the CT image.

Cardiologist (Alice) She needs information about all patient allergies, medical conditions and medications to get a overview of the situation.

Secretary (Bob) Writes the discharge papers and is therefore delegated permissions to view all information regarding Elisa's stay at the hospital. These permissions are delegated from the doctor dictating the discharge papers .

5.1.2 General Requirements

We notice the following about this scenario:

- A number of clinical staff (users) are involved in various roles in providing care at various points in the care process. These include interns, internists, cardiologists, nurses, etc.
- The staff is associated with different wards in the organization.
- Staff members that are providing care, are often dynamically put into the care process.

For access control and information ranking in particular, we notice the following requirements:
General Requirement 1 (Role in care)**.** *As already anticipated the permissions a clinical staff member has to clinical records (documents) should reflect the staff member's role in providing care.*
General Requirement 2 (Treating patient)**.** *Only staff members that actually treat the patient should be able to get access to the patient's records.*
General Requirement 3 (Changing wards)**.** *Depending on the care process, various clinical staff members need access to patient records at different points in the overall process. Therefore, when a patient is transferred to a new ward, the staff members at the new ward should only be given access to the patient's information after the staff at the old ward has requested the transfer.*
General Requirement 4 (Joining the care process)**.** *Requirements 1 and 2 above should hold for any staff member who dynamically joins the care process.*
General Requirement 5 (Delegate)**.** *Certain staff members may delegate duties and associated permissions to other staff members.*
General Requirement 6 (Discharge)**.** *Once the patient is discharged, all permissions to the patient's medical records should be deactivated.*

The model can be developed to meet these requirements, but needs some extensions from an ordinary RBAC model. If we look at the requirements one by one we notice the following:

Requirement 1 clearly calls for RBAC and can indeed be handled by RBAC in a straightforward manner. The exact nature of permissions in the model, presented in chapter 6, have certain ramifications. In other words, are permissions specified on classes (types) of objects, or are they specified on individual objects?

Requirement 2 cannot be met completely by a basic RBAC model. If we examine this requirement closely, we see that it has three parts:

2.1 The type of permissions allowed to a staff member should be determined by their role.

2.2 Permissions should be restricted to individual user instances of roles treating a specific patient

2.3 Permissions should be restricted to specific instances of objects (records) that pertain to a specific patient.

These three requirements call for access control to be specified at a much finer level than ordinary RBAC. They require access control on individual users and object instances. We note that RBAC, with roles being assigned permissions to individual object instances, can meet requirement 2.1 in isolation. Specifying permissions at this level add to security administration overhead, and possibly confusion and error, which is precisely the problem RBAC is trying to avoid. In summary, requirement 2 calls for a hybrid model of access control that incorporates aspects of RBAC, as well as user identity and object-based access control. In other words, we want RBAC-based permission assignment on object types at an enterprise level, and yet able to activate and control permissions on individual users and object instances at a more detailed level of administration.

Requirement 3 calls for permission activation to progress from staff members in one ward to staff members in another for a specific patient.

Requirement 5 calls for the ability to delegate a permission to a specific instance of a role. This essentially means that a permission needs to be assigned and activated for the role instance. We note that it is a problem relating to delegation that permissions cannot be delegated to roles, because doing so give additional permission to all instances of the role. Instead roles need to be delegated to users that may not be assigned to such a role, and in addition such a delegation should only include one specific patient.

Requirement 6 calls for deactivation of all permissions that various staff members may hold on a patient's record. This implies that users have their assigned roles deactivated.

In summary, the above scenario require role-based permission assignment for users and information classes, and activation of permissions for individual users and object instances.

5.2 Model Requirements

The scenario revealed some requirements for a role-model to be used in a healthcare organization, but still there is a lot of options to comply with these requirements. We have therefore sorted out a set of requirements that place a greater limitations on the development of the model. We note that the requirements listed are not supposed to be requirements viewed as system specific requirements. The requirements are instead to be seen as design choices, constraints on the

surrounding and on what the model has to represent. At the end these requirements also help us validate the model.

The first and most important model requirement is that the model we develop is built with a formal logic. This is done using the formal language $\mathscr{L}_{\mathscr{DA}}$ described in chapter 2. Having the model built with a formal language helps us test and validate it. This requirement is given in table 5.1.

Name	Formal language
Description	The model shall be described in a formal language that is able to express deontic logic.
Goal	When expressed in a formal language the model can be tested and validated accordingly.

Table 5.1: Formal language

5.2.1 The Components Required

The second model requirement we set for the model is that it shall be based on the NIST standard for RBAC. This is because the NIST standard is regarded as authoritative work on role-based access control and we like the model we develop to be compatible with this standard. The core elements are natural to include, when we are to follow the NIST standard. In accordance with the needs in the healthcare sector, which have multiple roles and many of them have similarities, hierarchical RBAC is included. The need for setting constraints on role assignment and on role activation is an argument for both static and dynamic separation of duty constraints. These model requirements related to the NIST standard are listed in table 5.2.

Name	NIST standard components
Description	Using the NIST standard as a starting point, we include: 1. The core RBAC elements 2. Hierarchical RBAC component 3. Static separation duty 4. Dynamic separation of duty
Goal	The model is compatible with a known RBAC standard.

Table 5.2: NIST standard components

Various tasks in healthcare can be delegated, and these tasks are therefore carried out on behalf of someone. Accordingly the model must handle delegation. To be more precise the model have to determine the transitivity of delegation, what can be delegated and who may receive

delegation. The model also have to determine what happen to the permissions of the receiver of a delegation and what happen to the permissions of the delegator. Table 5.3 and 5.4 list the model requirements needed for delegation in the model.

Name	Delegation
Description	The model must give the opportunity for performing task on behalf of another user, and authorizing another user to perform tasks on one's behalf. The model shall formally define: 1. Transitivity of delegation 2. Completeness of delegation 3. Permanence of delegation
Goal	The model shall support the delegation that occur in a healthcare organization.

Table 5.3: Delegation

Name	Delegation of role units
Description	Users may only want to delegate certain permissions to other users and not permissions of the entire role.
Goal	The model must give the opportunity to delegate a smaller atomic unit than a role.

Table 5.4: Delegation of role units

The access to and ranking of information in a healthcare organization is heavily dependent upon the context of the access and ranking. Important to this context is the location and the time. This can be expressed as roles such as on_duty and ward_B. To express constraints on the time dependent roles we need to be able to express a period of time. It is also necessary to be able to change the context during run-time, such as revoking access right, because the patient is transferred to another ward, and therefore need a formalization of a run-time expression. The two model requirements listed in table 5.5 and 5.6 cover the model requirements for periodic time and run-time request.

Name	Periodic time
Description	In order to achieve access and information ranking based on the context of time we need the model to support periodic expressions.
Goal	Periodic time is formalized in the model.

Table 5.5: Periodic time

Name	Run-time requests
Description	Dynamic changes occur in healthcare and these changes need to be reflected in the model.
Goal	Run-time requests make it possible to change a given context, thus changing access privileges and information ranking.

Table 5.6: Run-time requests

5.2.2 Information Ranking Requirements

To allow for information ranking there are a few model requirements. First of all when having both delegation and time expressions added to the NIST standard, there has to be added only two additional components in the model. These two components, which are relevance and detail, have to be defined similar to permission. Relevance and detail ranking should be based on the same policy that grants and denies access. This is because it is important that the ranking of information has to be done without any extra input from the user, only relying on information already entered into the EHR system. Table 5.7 covers the model requirement for adding relevance and detail.

Name	Information ranking components
Description	We need to include relevance and detail components to support information ranking.
Goal	The model is made to support information ranking based on the information used for access control.

Table 5.7: Information ranking components

For both access control and information ranking, we need an efficient way to give access to and rank relevance and detail for a particular object. We believe information classes and information hierarchies are helpful in this regard. Table 5.8 covers the model requirement for information classes and information hierarchies.

Name	Information classes and hierarchy
Description	Information can be put into classes that can be placed in hierarchy. This is a way of modelling patient data in order to make administration of the security policy
Goal	The model is made to support information classes and information hierarchy.

Table 5.8: Information classes and hierarchy

5.2.3 Activation Functions

Roles are activated for users, either explicitly by themselves or implicitly by a system observing their authentication, location, work flow, or other context. Since several roles may be activated in the same session, the model must act deterministically when giving the user permissions and ranking information. This requirement is found in table 5.9. If this is not required, the information ranking and permissions available to a user with more than one activated role is indeterminate. Without studying the implementation code, there is no way of knowing whether the resulting ranking and permissions are a sum, difference, minimum, maximum, random, or other value from the corresponding ranking and permissions in the roles. Implementations of the same model should not vary from each other on this point. If a different behavior when combining roles is desired, this should be clearly defined in the model prior to implementation. Alternatively, various behavior could be part of the same implementation, and chosen by configuration.

Table 5.10 covers the model requirement for having a function, that evaluates which roles may be activated. This function enforces the separation of duty and time constraints on access. In addition, the function evaluates the type of authorization given, i.e. read, update etc., and the ranking given each role.

The last model requirement is that patients may deny or extend access to the whole or part of the record for one or several healthcare employees, by identity, profession, or other criteria. If the patient denies access to someone it overrules everything else. This model requirement is covered in table 5.11. Without going into detail about other access control mechanisms we find it necessary to use an access control list to support this model requirement. An access control list may be used to revoke access from particular users that the patient has denied access instead of specifying which user has access to which patient. The latter design introduces a security administration overhead, which is precisely the problem RBAC is trying to avoid.

Name	Combining ranking and access from several roles
Description	Because users may activate several roles the model needs to evaluate how these roles behave in combination with each other.
Goal	To make a function in pseudo code that evaluates access and ranking for a user, which has activated a set of roles.

Table 5.9: Combining ranking and access from several roles

Name	Legal role activation
Description	Because users may activate several roles the model needs to express how the roles can be activated by a user in a way that makes sure no constraints are broken.
Goal	Express a role construction function in pseudo code.

Table 5.10: Legal role activation

Name	Patient preference
Description	Patients may deny specific users from accessing the patient record, and extend access to others; thus the model have to express this, but at the same time avoid unnecessary security administration overhead.
Goal	An access control list is used to revoke access from particular users, that the patient has denied access.

Table 5.11: Patient preference

5.3 Summary

This chapter describes a scenario, which is used in chapter 8 to test the model. The scenario includes a general description of the actors, which is important when we test the model. From the scenario we derived some general requirements for an access control model. These requirements are complementary to the research goals on page 1 and the motivation for using RBAC in healthcare on page 25. The second part of this chapter is used to describe requirement for the model, that place a greater limitations on the development. These requirements are not system specific requirements. They are instead to be seen as design choices, constraints on the surrounding and on what the model has to represent.

Chapter 6

Model

In this chapter we present the model that we have developed based on the theories presented in the preceding chapters. The first section of the chapter is used to give an overview of the model. The following sections express the formal model. The last sections describes the role construction function and the access-ranking function for the model.

6.1 Model overview

To show how the different parts of the model fit into each other, we use a simple diagram. Figure 6.1 shows this diagram. It is a new version of the diagram in the specialization project [SØ04, p. 50].

The *basic definitions* are presented in section 6.2. These definitions, based on the NIST standard [FSG+01], are used in the other parts of the model.

Several other parts of the model come from the NIST standard. The model therefore supports *role hierarchies* (section 6.4), *static separation of duty* and *dynamic separation of duty* (*SoD*, section 6.5). This basic model is extended with the notion of information classes (6.3) and information ranking. In addition, the model is designed to support delegation (6.6) and context based RBAC. These additional extensions are based on the work described in section 3.4.

Section 6.7 contains the *temporal* definitions. *Access rules* defined in section 6.8 include the general rules for granting access to roles, as well as the rules that are used when patients want to deny or extend access to their personal information.

Role combination and *Ranking and access control*, marked with a darker background in figure 6.1, are the central components of our model. Section 6.9 contains the role construction function and the ranking and granting function. *Additive methods*, methods using sources of relevance information external to the role rules, are commented in section 6.10.

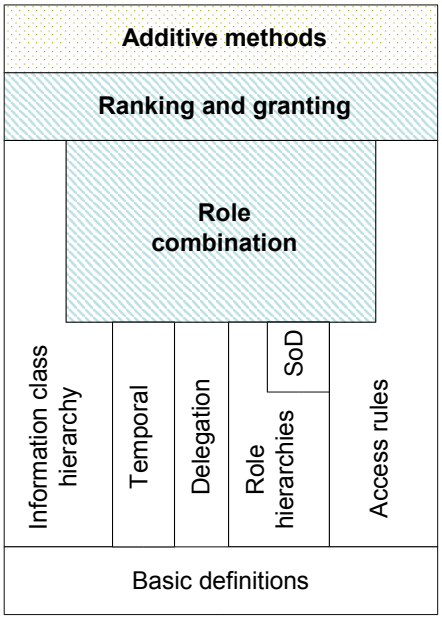


Figure 6.1: Model architecture

In the formal definitions, we have highlighted our own work like this: to separate that from the work done by others. There is no such highlighting of the functions (role construction and ranking), as they are entirely designed by us.

6.2 The Basic Definitions

The role-model is made from the basic sets defined in the NIST standard, but in addition the basic model includes both relevance ranking and detail ranking.

Basic sets:
The basic sets similar to the NIST standard.

- $U =$ a set of users, $\{u_1, \ldots, u_i\}$
- $OBJ =$ a set of information objects, $\{obj_1, \ldots, obj_j\}$
- $OP =$ a set of operations, $\{op_1, \ldots, op_k\}$
- $ROLES =$ a set of roles, $\{r_1, \ldots, r_i\}$
- $SESSIONS =$ a set of sessions, $\{s_1, \ldots, s_r\}$

The two basic sets that we have included to support information ranking are:

- $REL =$ a set of relevance levels, $\{rel_1, \ldots, rel_i\}$
- $DEL =$ a set of detail levels, $\{del_1, \ldots, del_i\}$

As mentioned in section 4.2.3, page 33, objects in the model are EHR fragments or collections thereof.

Basic Relations:
The relation and mapping between the basic sets. First we present the NIST standard, which already now has to be extended to handle delegation, this is because of the need to separate between original user-to role assignment relation and delegated user-to-role assignment relation.

- $UAO \subseteq USERS \times ROLES$ a many to many mapping original user-to-role assignment relation
- $UAD \subseteq USERS \times ROLES$ a many to many mapping delegated user-to-role assignment relation
- $UA = UAO \cup UAD$ the union of original and delegated user-to-role assignment relation
- $\text{assigned\_users} : (r : ROLES) \rightarrow 2^{USERS}$, the mapping of role r onto a set of users. Formally: $\text{assigned\_users}(r) = \{u \in USERS | (u, r) \in UA\}$.
- $PRMS = 2^{OP \times OBJ}$ set of permissions.
- $PA \subseteq PRMS \times ROLES$, a many to many mapping permission-to-role assignment relation.
- $\text{assigned\_permissions}(r : ROLES) \rightarrow 2^{PRMS}$, the mapping of role r onto a set of permissions. Formally: $\text{assigned\_permissions}(r) = \{p \in PRMS | (p, r) \in PA\}$
- $OB(p : PRMS) \rightarrow op \subseteq OPS$, the permission to operation mapping, which gives the set of operations associated with permission p.
- $OB(p : PRMS) \rightarrow ob \subseteq OBJ$, the permission to object mapping, which gives the set of objects associated with permission p.
- $\text{user\_sessions}(u : USERS) \rightarrow 2^{SESSIONS}$, the mapping of users u onto a set of sessions.
- $\text{session\_roles}(s : SESSIONS) \rightarrow 2^{ROLES}$, the mapping of session s onto a set of roles. Formally: $\text{session\_roles}(s_i) \subseteq r \in ROLES | (\text{session}(s_i), r) \in UA$.
- $\text{avail\_session\_perms}(s : SESSEIONS) \rightarrow 2^{PRMS}$, the permissions available to a user in a session, $\bigcup_{r \in \text{session\_roles}(s)} \text{assigned\_permissions}(r)$.

The basic relations and mapping for relevance are:

- $RRMS = 2^{REL \times OBJ}$ set of relevances.

- $RA \subseteq RRMS \times ROLES$, a many to many mapping relevance-to-role assignment relation.
- $\text{assigned\_relevance}(r : ROLES) \rightarrow 2^{RRMS}$, the mapping of role r onto a set of relevance. Formally: $\text{assigned\_relevance}(r) = \{p \in RRMS | (p, r) \in RA\}$
- $OB(rr : RRMS) \rightarrow rel \subseteq REL$, the RRMS to relevance mapping, which gives the set of relevance associated with RRMS rr.
- $OB(rr : RRMS) \rightarrow ob \subseteq OBJ$, the RRMS to object mapping, which gives the set of objects associated with RRMS rr.
- $\text{avail\_session\_perms}(s : SESSEIONS) \rightarrow 2^{RRMS}$, the relevance for a user in a session, $\bigcup_{r \in \text{session\_roles}(s)} \text{assigned\_relevance}(r)$.

The basic relations and mapping for detail are:

- $DRMS = 2^{DEL \times OBJ}$ set of details.
- $DA \subseteq DRMS \times ROLES$, a many to many mapping detail-to-role assignment relation.
- $\text{assigned\_detail}(r : ROLES) \rightarrow 2^{DRMS}$, the mapping of role r onto a set of detail. Formally: $\text{assigned\_detail}(r) = \{p \in DRMS | (p, r) \in DA\}$
- $OB(dd : DRMS) \rightarrow det \subseteq DET$, the DRMS to detail mapping, which gives the set of detail associated with DRMS dd.
- $OB(dd : DRMS) \rightarrow ob \subseteq OBJ$, the DDMS to object mapping, which gives the set of objects associated with detail dd.
- $\text{avail\_session\_perms}(s : SESSEIONS) \rightarrow 2^{DRMS}$, the detail for a user in a session, $\bigcup_{r \in \text{session\_roles}(s)} \text{assigned\_detail}(r)$.

6.3 Information Classes

In the definitions for information ranking we have included definitions for information classes and the information class hierarchy, in order to make implementations for larger organizations possible. The information classes define a general hierarchy; a limited hierarchy is not defined in this role-model. Each information object is a member of an information class and these classes form a hierarchy.

- $CLASSES =$ a set of information classes for the objects, $\{class_1, \ldots, class_o\}$
- $CH \subseteq CLASSES \times CLASSES$, is a partial order on classes called the class inheritance relation, called a information classification hierarchy or class hierarchy, written as $\succeq_{CH}$, where $class_1 \succeq_{CH} class_2$ means that $class_1$ is either one of the ancestors of $class_2$, or that $class_1 = class_2$

- $CA \subseteq OBJ \times CLASSES$, a many-to-one mapping object-to-class assignment relation. An object has only one class.
- $instanceof(obj \in OBJ) \rightarrow CLASSES$, the mapping of object obj onto a single class. Formally: $instanceof(obj) = \{class \in CLASSES \mid (obj, class) \in CA\}$

6.4 Role Definitions and Role Hierarchy

In section 6.2 we have defined the set of roles according to the NIST standard. In addition we want to define two other concepts regarding roles. The concept of a role unit is needed to deal with delegation of smaller units than a role. Thus a role unit is a subset of the obligation and permission given a role. The concept of a functional role are used when evaluating the total permission space, relevance and detail ranking for user. The functional role is defined as the set of roles and role units the user has activated in a particular session.

- Role unit is a set of obligation and permissions.
  $\text{role\_unit}(r \in ROLES) \rightarrow PRMS \times RRMS \times DRMS$ Formally:
  $\text{role\_unit}(r) = \{p_2, rr_2, d_2) | r = (p_1, rr_1, d_1), p_1 \subseteq PRMS, rr_1 \subseteq RRMS,$
  $d_1 \subseteq DRMS, p_2 \subseteq p_1, rr_2 \subseteq rr_1, d_1 \subseteq d_2, r \subseteq ROLES\}$
- A functional role is the set of roles that a user has activated in a particular session and therefore describes the permission the user have in that session.
  $\text{fur}(role\_list : ROLES) \rightarrow ROLES \times PRMS \times RRMS \times DRMS$ Formally:
  $\text{fur}(r) = \{(role\_list, priva, rrmsa, drmsa) | role\_list \in ROLES\ , priva \in PRMS,$
  $rrmsa \in RRMS\ , drmsa \in DRMS\}$

The Role Hierarchy Definitions We have chosen to include general role hierarchies, which allow for multiple inheritance. We believe we don't have to limit the inheritance for the role hierarchies, therefore we do not define inheritance limitations in the model. The $\succeq$ relation is subscripted to differentiated role hierarchy from information class hierarchy.

- $RH \subseteq ROLES \times ROLES$, is a partial order on ROLES called the class inheritance relation, written as $\succeq_{RH}$, where $r_1 \succeq_{RH} r_2$ means that r_1 is either one of the ancestors of r_2, or that $r_1 = r_2$
- $\text{authorized\_users}(r : ROLES) \rightarrow 2^{USERS}$, the mapping of role r onto a set of users in the presence of a role hierarchy. Formally: $\text{authorized\_users}(r) = \{u \in USERS | r' \succeq_{RH} r(u, r') \in UA\}$.
- $\text{authorized\_permissions}(r : ROLES) \rightarrow 2^{PRMS}$, the mapping of role r onto a set of permissions in the presence of a role hierarchy. Formally: $\text{authorized\_permissions}(r) = \{p \in PRMS | r' \succeq_{RH} r(p, r') \in PA\}$.

6.5 Separation of Duty

The NIST standard includes both static and dynamic separation of duty, we found both of these types of SoD useful for the model and therefore define both.

Static Separation of Duty Rules and Definitions:
To enforce separation of duty rules in a static context we use SSD rules. The NIST standard provides definitions for static separation of duty. We include this in the model despite of the fact that most, but not all, constraints in healthcare applications are probably dynamic.

- $SSD \subseteq (2^{ROLES} \times N)$ is a collection of pairs(rs, n) in static separation of Duty, where each rs is a role set, t is a subset of roles in rs, and n is a natural number ≥ 2, with the property that no user is assigned to n or more roles from the set rs in each (rs, n) $\in$ SSD. Formally: $\forall(rs, n) \in SSD, \forall t \subseteq rs : |t| \geq n \Rightarrow \cap_{r \in t} \text{assigned\_users}(r) = \varnothing$
- in the presence of a role hierarchy the static separation of duty needs a redefinition to formally: $\forall(rs, n) \in SSD, \forall t \subseteq rs : |t| \geq n \Rightarrow \bigcap_{r \in t} \text{authorized\_users}(r) = \varnothing$

Dynamic Separation of Duty Rules and Definitions:
Dynamic separation of duty constraints is as mentioned in 3.3.3 are constraints placed on the roles activated within or across a user's session. Dynamic separation of duty provides extended support for the principle of least privilege in that at user may have different levels of permissions at different times. Because dynamic separation of duty relations are constraints on the roles that are activated in a user's session, it does not differ in the presence of a role hierarchy.

- $DSD \subseteq 2^{ROLES} \times N$ is collection of pairs (rs, n) in Dynamic Separation of duty, where each rs is a role set and n is a natural number ≥ 2, with the property that no subject may activate n or more roles from the set rs in each $dsd \in DSD$. Formally:
  $\forall rs \in 2^{ROLES}, n \in N, (rs, n) \in DSD \Rightarrow n \geq 2 \wedge |rs| \geq n$, and $\forall s \in SESSIONS$,
  $\forall rs \in 2^{ROLES}, \forall \text{role\_subset} \in 2^{ROLES}, \forall n \in N, (rs, n) \in DSD, \text{role\_subset} \subseteq rs$,
  $\text{role\_subset} \subseteq \text{session\_roles}(s) \Rightarrow |\text{role\_subset}| < n$.

6.6 RBAC Delegation Relations and Functions

We repeat from section 6.2 that UAO is a many to many mapping between original user-to-role assignment relation, UAD is a many to many mapping between original user-to-role assignment relation, and $UA = UAO \cup UAD$. We use this to define the delegation relation.

The delegation relation in the model have to be defined for both original user delegation relation and the delegated user relation. These two relation together form the delegation relation. Formally:

- $DLGT \subseteq UA \times UA = U \times R \times U \times R$ is a many to one delegation relation
- $ODLGT \subseteq UAO \times UAD$ is an original user delegation relation

- $DDLGT \subseteq UAD \times UAD$ is a delegated user delegation relation
- $DLGT = ODLGT \cup DDLGT$

In section 3.4.1 we mentioned that there are several characteristics related to delegation. one of these characteristics is transitivity. The function Prior maps each user to role assignment to another user to role assignment (or possibly an empty set). The function Path determines the path of delegated user-role assignment. The function Depth is a simple function that determines the number of elements in the delegation path. Together the three functions Prior, Path and Depth are used to formalize the depth of transitive delegation.

- $\mathrm{Prior} : U \cup R \rightarrow U \times R$ is a function that maps each UA to another UA or $\varnothing$
  $\mathrm{Prior}(u, r) = \{(u', r') | (u, r) \in UAD, (u', r', u, r) \in DLGT\}$
  $\mathrm{Prior}(u, r) = \{\varnothing | (u, r) \in UAO\}$
- A delegation path is a set of ordered user-role assignment
  $\mathrm{Path}(u_0, r_0) = \{(u_0, r_o), (u_1, r_1), \ldots, (u_i, r_i), \ldots, (u_n, r_n) | (u_i, r_i) = \mathrm{Prior}(u_{i-1}, r_{i-1}) \in UA$ when $i > 0\}$
  $Path(u, r) = \{\varnothing | (u, r) \in UAO\}$
- $Depth : U \cup R \rightarrow N$. A delegation depth is the number of elements in a delegation path minus one
  $\mathrm{Depth}(u, r) = \{n | n = |\mathrm{Path}(u, r)|, (u, r) \in UAD\}$
  $\mathrm{Depth}(u, r) = \{0 | (u, r) \in UAO\}$

In addition to transitivity the model take into account what may be delegated, who may receive delegation, how the receiver should respond to delegation and the transfer of responsibilities. These formal definition place restrictions on how delegation may take place, but we believe they are according to the Norwegian EHR standard.

- A holder of a role may only delegate obligation or permissions that is part of his/her own functional role. Formally:
  $Delegate(x : r1, y : r2, r3) \rightarrow r3 \leq r1$
- Any user within the organization can receive any delegation rights as long as it does not interfere with the separation of duty constraints. Formally:
  $Delegate(x : r1, y : r2, r3) \rightarrow (r2, r3) \notin dsd \cap (r2, r3) \notin ssd$
- A user receiving obligation or permissions adds this role unit to his/her functional role. Formally:
  $Delegate_c(x : r1, y : r2, r3) \stackrel{def}{=} is - r2 + r3(y) \wedge \ldots$
- The user who delegates obligations or permission still continues to have the same obligations and permissions. Formally:
  $Delegate(x : r1, y : r2, r3) \stackrel{def}{=} is - r2 + r3(y) \wedge r3 : REP(x : r1, *)$

When introducing delegation we also need delegation revocation. Different types of revocation are discussed in section 3.4.1, but here we only allow the user who have delegated the permissions to revoke them, known as grant-dependant revocation. We do this because we belive this is the common practice in a healthcare organization. This revocation also have to be cascading, meaning that if the delegated role or role unit is further delegated it is revoked here as well.

- $Revoke(x : r1, y : r2, r3,) \stackrel{def}{=} is - r2(y) \wedge is\_not - r3(y)$

6.7 Temporal RBAC Rules and Definitions

An important aspect with this extension is the need for the model to represent different responsibilities and needs at different times. This implies that we have to be able to specify different activation times. We do this by formalizing periodic expression and run-time request as described in 3.4.2. We make this extension as simple as possible to avoid unnecessary complexity and ambiguity.

We include periodic expressions from [BBF01] in the model. These definitions are a calendar expression, which is defined as a countable set of contiguous intervals, and periodic expression, which combines calendar expression to express more general periodic expressions. In addition the function $\Pi()$ denotes the infinite time instants corresponding to a periodic expression, and the function $Sol()$ define the set of time instants denoted by $< [\text{begin}, \text{end}], P >$. These three functions allow us to place some temporal constraints on user-role activation.

Periodic expression Given calendars $C_d, C_1, \ldots, C_n$, a periodic expression P can be defined as: $P = \sum_{i=1}^{n} Q_i.C_i \rhd x.C_d$.
where $Q_1 = all$, $Q_i \in 2^{\mathbb{N}} \cup \{all\}$, $C_i \subseteq C_{i-1}$ for $i = 2, \ldots, n$, $C_d \subseteq C_n$, and $x \in \mathbb{N}$. Example: all.Years + $\{3, 7\}$.Months $\rhd 2$. Months represent the set of intervals starting at the same instant as the third and seventh month every year and having a duration of two months.

Function $\Pi()$ Let $P = \sum_{i=1}^{n} Q_i.C_i \rhd x.C_d$ be a periodic expression, then $\Pi(P)$ is a set of time intervals whose common duration is C_d, and whose set S of starting points is computed as follows:

- If $n = 1$, S contains all the starting points of the intervals of calendar C_1
- If $n > 1$, and $Q_n = n_1, \ldots, n_k$, then S contains the starting points of the $n_1^{th}, \ldots, n_k^{th}$ intervals (all intervals if $Q_n = all$) of calendar C_n included in each interval of $\Pi(\sum_{i=1}^{n} Q_i.C_i \rhd 1.C_{n-1}$.

Function $Sol()$ Let t be a time instant , P a periodic expression, ,and $begin$ and end two date expression. Define $t \in Sol([begin, end], P)$ iff there exists $\tau \in \Pi(P)$ such that $t \in \tau, t_b \leq t \leq t_e$ are the instants denoted by $begin$ and end, respectively.

In addition we include run-time requests in the model. The motivation to include run-time requests is requirement K7.41 in the Norwegian EHR standard, [Nys05], which specifically states that it should be possible to limit a person's right to act in a role to one or several time intervals. This implies that a role may be deactivated after a while. In addition run-time request allow for context changes for instance when a patient are moved from one ward to another.

- a user's run-time request expression to activate a role has the form: s : activate r for u after $\triangle t$, or s : deactivate r for u after $\triangle t$ where $r \in ROLES$ and $u \in USERS$, s is the session attached to the request and $\triangle t$ is the duration.
- an administrator's run-time request expression has the form: $pr : E$ after $\triangle t$ where $pr : E$ is a prioritized event expression and $\triangle t$ is the duration expression

6.8 Access Rules

The model developed so far supports basic functions such as separation of duties and also supports delegation, temporal constraints and some context dependency. The model still lacks rules for ranking and access in roles; it also lacks the ability to deny particular users access to patient information based on this patient's request. To support this we include an access control list in the model. The definitions for this is found in section 6.8.2.

In order to include the patient access control list, we first need to define various rules for permissions, relevance and detail.

6.8.1 Rules in Roles

The functional role is given access rights and a relevance and detail ranking based on some general principles. We make these granting and ranking rules and functions partially independent of each other. This meaning that the rules and functions for each of access, relevance and detail are given as separate rules.

We also define a triplet of privilege, relevance and detail, called an *access-ranking*, which supports the patient access control list and the ranking and granting function. A *privilege* is simply a set of operations. Finally, we define a set of rules for how objects should be given an *access-ranking*.

Access control
Each object belonging to a particular class has a set of rules for how it should be given permissions:

- $PRIV \subseteq OP$, is a set of privileges. Each privilege can contain from none to all operations.

- $PRULES \subseteq PRIV \times CLASSES$, is a set of rules for how an object belonging to a particular information class should be given permission.

Ranking
Each object belonging to a particular class has a set of rules of how it should be given a relevance and detail ranking:

- $RRULES \subseteq REL \times CLASSES$, is a set of rules for how an object belonging to a particular information class should be given relevance level.
- $DRULES \subseteq DEL \times CLASSES$, is a set of rules for how an object belonging to a particular information class should be given detail level.

Accrank and crule
Relevance, detail and privilege together form a triplet we have called an *access-ranking* ($ACCRANK$) which consists of relevance, detail and privilege (list of operations) for a particular rule. Using this triplet, we can make a *construction rule* ($CRULES$). A construction rule can be split into a $PRULE$, $RRULE$ and $DRULE$.

- $ACCRANKS \subseteq REL \times DEL \times PRIV$, a set of access and rankings, each accrank consists of privilege (list of operations), one relevance level and one detail level.
- $CRULES \subseteq ACCRANKS \times CLASSES$, is a set of rules for how an object belonging to a particular information class should be given an $ACCRANK$.

6.8.2 Rules for Patient's Preferences

As we have stated in table 5.11, the model needs to reflect a patient's preferences, for denying particular users access to his/her record information, and also to grant additional access to particular users. Thus the model needs to support this. We support this requirement by including a *patient access control list* ($PACL$). This is where deontic operators, permit ($Permit$) and forbid ($Forbid$) first are used in the model.

The patient access control list is able to express access rules applying to both classes and objects, depending on both roles and users:

- $PATIENTS =$ a set of patients, $\{p_1, \ldots, p_s\}$
- $concerns(obj \in OBJ) \rightarrow PATIENTS$, maps each object to the patient which the record belongs to.
- $PACLRULE \subseteq \{Permit, Forbid\}((U \cup COR) \times (CLASSES \cup OBJ) \times ACCRANKS)$, a set of patient-specific access control rules.
  $pacrule = operator(subject, object, accrank)$, $subject \in U \cup COR$, $info \in CLASSES \cup OBJ$,$accrank \in ACCRANKS$, $operator \in \{Permit, Forbid\}$.

- $PACL \subseteq PATIENTS \times PACLRULE$, a set of patient-specific access control lists where the patients can give more access or less access, than specified globally, to users and construction roles.
- $aboutpatient(obj \in OBJ) \rightarrow PATIENTS$, a mapping of all information objects into the patient that they concern.

6.9 Model functions

The whole process of ranking and access control, starting with a user activating roles and ending with objects being given relevance, detail and permissions, is split into two functions. The first is called the role construction function, which makes a valid functional role depending on the user and session, and is found in section 6.9.1. The second function, in section 6.9.2, is the rank function that assigns relevance, detail, and permissions to objects based on the functional role constructed in the first function.

6.9.1 Role Construction

Because different roles are given different access privileges, the functional role have to reflect the access rights for each of the roles it contains. If all the different roles contained in the functional role are concerned with access on different objects, the access control becomes fixed in a predictable fashion, because the output is uniquely determined by the input. Because of *conflicting rules* this is hardly ever the case; this means that a policy choice have to be made. Conflicting rules are rules that concern the exact same information class. If two or more conflicting roles are activated in the same session, a policy choice has to be made as to what kind of rule should result from the combination. In the specialization project [SØ04], we suggested some of the possible choices which can be made in this situation, one is handling it as an SoD-violation, the second is ignoring it, which leads to a non-deterministic role combination, and the third is granting full access (of every role contained) to the functional role. Similar to this the functional role have to reflect the ranking for each of the roles it contains. It is important that any denied access take precedence over ranking, which means ranking of relevance and detail is only done for the objects the functional role have access.

As defined earlier a functional role consists of several roles and role units, but it is not a union of these nor does have the same sample space. The functional role is built from all the roles that the user has legally activated in the session, i.e. it is put together in such a way that it does not contain any conflicting roles. This is done by enforcing separation of duty constraints. In addition the ranking rules in the functional role contain only one rule about each direct class. This way, once the functional role is known, all that is needed for ranking an information object is knowing to which class it belongs.

The role construction function is a function that constructs valid functional roles FUR based on who the user is, and what roles the user has available and activated in a session.

We repeat the definition of the functional role and add two components for describing rules for the functional role and which class the rule concerns:

- $\text{fur}(role\_list : ROLES) \rightarrow ROLES \times PRIV \times RRMS \times DRMS$ Formally: $\text{fur}(r) = \{(role\_list, prmsa, rrmsa, drmsa) | role\_list \in ROLES$,$prmsa \in PRIV$, $rrmsa \in RRMS$,$drmsa \in DRMS\}$
- $FURULE \subseteq ACCRANKS \times CLASSES$, rules for the functional roles, built with he role-construction function.
- $\text{func\_rule\_about}(furule \in FURULE) \rightarrow CLASSES$, which class this rule concerns.

We have chosen to resolve conflicting rules by having the resulting functional rules contain the maximum relevance and detail levels of the roles that are activated in the session. The permissions of the functional role are a privilege union of the privileges of the activated roles. This was suggested in the specialization project [SØ04, p. 58] for handling conflicing rules from different hierarchies. For conflicting rules from the same hierarchy, that version of the role construction function picked the rule from the most direct role. In this thesis version of the function, handle conflicting rules within a role hierarchy in the same way as we handle conflicting rules accross role hierarchies. This version of the function is thus simpler than the one from the specialization project.

Figure 6.2 shows the role construction function. Within the role construction function we have used several other functions. The three most important of these are:

- $\text{decide\_transitivety}$ is made to illustrate the use of the previously defined functions Prior, Path and Depth.
- can_delegate decides that the user may delegate parts of his/her functional role.
- $\text{check\_temporal\_constraints}$ illustrates that we check temporal constraints. We have not found it useful to give a more thorough specification of this function.

We also note the functions rule_about, parent, isroot and rulesof. They return the class the rule concerns, the predecessor of an element, the root of an hierarchy and the rules that is contained in a role respectively. Also $role\_list$ is used in the function, but is not previously defined. It is only a list of the roles in the functional role. The design choices made for the role construction function are further discussed in section 9.2.6.

6.9.2 Function for access control and information ranking

The actual ranking and granting of access is given in the model as a function in pseudo code. The first part of the function is given in figure 6.3 and the second part is given in 6.4. This function evaluates the functional role, also handling role and information hierarchies, and then gives the

function $roleconstruction(roles \subseteq ROLES) \rightarrow FUR$
{
  if $\forall roles$,$\forall dsd \in 2^{ROLES}$,$\forall role\_subset \in 2^{ROLES}$,
$dsd \in DSD$,$role\_subset \subseteq dsd$,
$role\_subset \subseteq roles$, $\mid role\_subset \mid < n$

    $newfuncrole = (role\_list_{new}, furules_{new})$
    $role\_list_{new} = \bigcup_{c \in roles} role\_id(c)$
    $furules_{new} = \{\}$
    for each $role_i \in roles$
      if $role_i \in UAD$ then
        decide_transitivety($role_i$)
      endif
      check_temporal_constraints($role_i$)
      $a = role_i$
      while $\neg isroot(a)$ do
        for each $crule_k \in rulesof(a)$
          $crule_k = ((rel_k, del_k, priv_k), class_b)$
          for each $f \in furules_{new}$
            $f = ((rel_f, del_f, priv_f), class_f)$
            if $class_f \neq class_b$ then
              $furules_{new} = crule_k \cup furules_{new}$
            else
              $g = ((\max(rel_f, rel_g), \max(del_f, del_g), (priv_f \cup priv_g)), class_f)$
              $furules_{new} = g \cup furules_{new}$
            endif
          next f next $crule_k$
      $a = parent(a)$
      endwhile
    next $role_i$
    can_delegate($newfuncrole$)
    return $newfuncrole$
  else
    DSD violation.
  endif
}

Figure 6.2: Role construction function

correct access and ranking. The function then checks if the patient has forbidden information from being accessed by this user, or permitted extended access, by checking the access control list.
In the ranking and granting function we make use of a function called $mathrmdirectaccrank$. This function is used to evaluate the Permit and Forbid deontic operators of patient consent. The function is described in pseudo code in figure 6.5. Like the role construction function the function for ranking and granting is further discussed in section 9.2.6.

function $rank(obj \in OBJ, fur \subseteq FUR, u \in U, p \in PATIENTS, pacls \subseteq PACL)\_$
$\rightarrow ACCRANKS$
$\{$
$accrank_{result} = (0, 0, \{\})$ (initial value)
$if aboutpatient(obj) = p$
  $class_b = instanceof(obj)$
  do
    for each $rfurule_f \in furules_j, (role\_list_j, furules_j) \in fur$
      if $class_b = func\_rule\_about(rfurr)$
        $accrank_{result} = accrank_f, where rfurule_f = (accrank_f, class_b)$
        $match = TRUE$
      endif
    next $rfurule_f$
    $class_b = parent(class_b)$
  while $(\neg isroot(class_b)) \wedge (match = FALSE)$
  $\forall rpacl \subseteq pacls, rplacl = (p, rpaclrules)$
    if $\exists paclr_m \in rpaclrules, paclr = deontoper_m(u \in U, info_m, accrank_m)$
      if $obj \in info_m$
        if $\exists paclr_k \in rpaclrules$ where $deontoper_k = Forbid$ and $info_k = obj$
          if $deontoper_m = Forbid$ then
            $accrank_{result} = directaccrank(accrank_m, Forbid, accrank_{result})$
          endif
        else
          if $deontoper_m = Permit$ then
            $accrank_{result} = directaccrank(accrank_m, Permit, accrank_{result})$
          endif
        endif
      else if $info \in \{info_1 \succeq_{CH} \ldots \succeq_{CH} info_n\}$and $info = info_n$
        if $\exists paclr_k \in rpaclrules$ where $deontoper_k = Forbid$ and $info_k \in classes$
          if $deontoper_m = Forbid$ then
            $accrank_{result} = directaccrank(accrank_m, Forbid, accrank_{result})$
          endif
        else
          if $deontoper_m = Permit$ then
            $accrank_{result} = directaccrank(accrank_m, Permit, accrank_{result})$
          endif
        endif
      endif

Figure 6.3: Ranking and Granting function I

else if $\exists paclr_m \in rpaclrules, cmpaclr = (class \in ROLES, info, ,)$
$\forall caclr \in rpaclrules$,$caclr = deontoper_m(role\_list, info, accrank_m)$, $role_m \in ROLES\ role\_id(role_m) \in role\_list$, $fur = (role\_list, rpaclrules)$
if $obj \in info_m$
if $\exists paclr_k \in rpaclrules$ where $deontoper_k = Forbid$ and $info_k = obj$
if $deontoper_m = Forbid$ then
$accrank_{result} = directaccrank(accrank_m, Forbid, accrank_{result})$
endif
else
if $deontoper_m = Permit$ then
$accrank_{result} = directaccrank(accrank_m, Permit, accrank_{result})$
endif
endif
else if $info \in \{info_1 \succeq_{CH} \ldots \succeq_{CH} info_n\}$ and $info = info_n$
$accrank_{result} = directaccrank(accrank_m, deontoper_m, accrank_{result})$
if $\exists paclr_k \in rpaclrules$ where $deontoper_k = Forbid$ and $info_k = info_n$
$if deontoper_m = Forbid$ then
$accrank_{result} = directaccrank(accrank_m, Forbid, accrank_{result})$
endif
else
if $deontoper_m = Permit$ then
$accrank_{result} = directaccrank(accrank_m, Permit, accrank_{result})$
endif
endif
next caclr
endif
next rpacl
endif
return $accrank_{result}$ }

Figure 6.4: Ranking and Granting function II

$directaccrank(accrank_a \in ACCRANKS$,$deonticoperator_a \in \{Permit, Forbid\}$,-
$accrank_b \in ACCRANKS) \rightarrow ACCRANKS$
{
  $(rel_b, del_b, priv_b) = accrank_b$
  $(rel_a, del_a, priv_a) = accrank_a$
    $rel_c = max(rel_a, rel_b)$
    $del_c = max(del_a, del_b)$
  if $direction_a = Permit$
    $priv_c = priv_a \cup priv_b$
  else if $direction_a = Forbid$
    $priv_c = priv_b \setminus priv_a$
  endif
  return $accrank_c(rel_c, del_c, priv_c)$
}

Figure 6.5: Applying access, relevance and detail rules from a PACL

6.10 Additive Methods

From the perspective of the user, the ranking assigns each information object a relevance level, a detail level, and gives the user a set of operations that the user is allowed to perform on the object. This ranking does not need to be the end of ranking. We may include methods that add or subtract relevance and detail after the "rank and grant" function is finished with the object.

We proposed some such methods in the specialization project [SØ04, pp. 60-63], and have also considered new ones. Here's a non-exhaustive list of possible ranking methods whose results can be added to the role model ranking:

- Information that is new since the last time the user used the system, could be given an added relevance. (A trivial example of this is how new messages are given a different color when viewing an e-mail inbox.)

- Users may have different preferences for what information they need, and may assign a different relevance or detail level to some information classes. Working habits differ, and the same class of information thought too highly ranked by one user could be thought too lowly ranked by another. Permissions should of course not be set in preferences.
  The additive preferences could be used by administrators and designers: Whenever a majority of the users agree, through the preferences, that a class of information is wrongly ranked, role definitions can be updated. A caveat here is that a majority of the users could be wrongly downgrading critical information, so experts and guidelines should be consulted before adopting these collective preferences when updating role definitions.

- Guideline-derived rankings could be added. A knowledge base having more specific knowledge about the relationships between information, could add relevance and detail to an object. As an example, the role model might assign an relevance 3 and detail 2 to an object with information about "current drug treatment" based on roles, but a guideline-based knowledge system may detect a drug interaction with a new prescription, and add +5 to the relevance. The resulting information object gets a relevance of $3 + 5 = 8$ and a detail of $2 + 0 = 2$.

- Past problems could possibly be ranked according to the same rules, and considered to be instances of the same information classes in a particular implementation of the model. This could result in all past problems being shown at the same relevance and detail level. Additive rules could differentiate ranking and access to these past problems by for example

  - Considering guidelines, like giving higher ranking to problems sharing the symptoms or target organs with the problem currently under review; or

  - by giving a lower ranking to older problems and problems not under treatment.

6.11 Summary

In this chapter we presented a model that includes several components. Most important is the four components of the NIST standard, which includes core RBAC, hierarchical RBAC, static separation of duty and dynamic separation of duty. The NIST standard is extended with components of information ranking. The model is further extended with components that can support delegation, including three functions to express transitivity, restrictions on how delegation can take place and a definition of role units. To support the ability to express different context the model is extended with periodic expressions and run time requests. An access control list is also added in order to express an individual patients requests. The model is completed with rules and functions that evaluate the access and ranking a user is given. Thus the final model is presented in this chapter as a composite of several different models described in the literature, with additional extensions of information ranking.

Chapter 7

Implementation

To test and validate the model we make a simple implementation in the Prolog programming language. This implementation is also used to see whether we can transform the deontic logic into Prolog without loosing any of the semantics expressed.

This chapter first gives a short introduction of the Prolog programming language and the motivation for using this programming language. The chapter then describes the code structure of the implementation, before it shows how the implementation is built. The implantation examines different parts of the model and is presented in stages of progressively higher advancement.

7.1 About the Implementation Language

Prolog [CM03],[SSW94] is a logic programming language. We chose to use Prolog for the implementation, because the clause and list form that are used to make systems in Prolog resembles the logic expressions of the formal model.

We begun implementing the model in *Visual Prolog* [Vis05], because we wanted the implementation to be easily fitted with a prototype graphical user interface (GUI) for role administration. We later decided to move the implementation into *SWI Prolog* [SWI03] to accommodate a more standardized Prolog environment. The implementation does require the presence of some clauses provided by the *SWI Prolog library*, particularly list manipulation clauses. The library is free software under an LGPL-style license. Thus, if the implementation doesn't appear to work on a Prolog variant, the problem might be solved by obtaining that library.

If one wants to use the model with a user interface, various GUI tool kits for Prolog can be used with the implementation. Also, the implementation can easily be moved back into Visual Prolog. In order to do this, it is required to explicitly type-declare the types of the variables, and also to provide the SWI Prolog library with declared types.

7.2 Code Structure

To keep the implementation code manageable, we keep the Prolog statements sectioned according to their level of generality. Very general rules, that we expect to apply in most policy domains, are found in the model file.

The information in the scenario is implemented in a facts file separate from the model file. These facts include users, roles, role hierarchies, patients, information objects and information class hierarchies. The facts file also contains the role- and rank-assignments, that is the codified representation of the of the local security- and relevance policy.

The facts file is used in the verification of the model, in chapter 8. The general model does place certain limitations on how the information in the facts file is represented. That is why the rest of this chapter is not only concerning the model file implementation, but also describes how the facts are structured in the facts file.

7.3 Implementation of Facts from the Scenario

In the implementation, users are defined as structures with only a name for identification, e.g. user('Bob'). The model can contain more information about each user, but we consider that unnecessary to demonstrate in this implementation. If one wants to extend the user structure with new fields, like "user(name,gender,address,phone)", the only clauses that need to be changed are the ones where the "name" variables in the "user" structure needs to be bound. Statements in these clauses would be changed from something like "user(NAME)", to user "user(NAME,_,_,_)".

Likewise, the exact content structures of the information objects are exchangeable. It is not important to the model exactly how the data fields in each information object are structured. The only things that matter in this regard are that the information object must be uniquely identifiable, and that it belongs to the record of a single patient.

Roles are defined with both a name and an integer identifier, e.g. $role(15,'HomeNurse')$. The integer identifier is used in the role hierarchy. Thus the parent-child relationship in the hierarchy is implemented as $roleparent(Child, Parent)$, with $Child$ and $Parent$ as integer identifiers. For instance $roleparent(3, 1).$ means that role 1 is the parent of role 3.

7.4 Implementation of General Rules in the Model

We chose to do the implementation of the ranking and access model in incremental stages. In this way, we can validate each part of the model, and each added function, before moving on to the next stage.

In the following code examples, comments are enclosed between /* and */. In these comments, argument variables are preceded by a '+', '-' or '?', in a style similar to the documentation for SWI Prolog. These variables indicate how the arguments should be used. '+' denotes an input argument, '-' denotes an output argument, while '?' denotes an argument that may be either input to or output from the clause. The code does contain more comments, which can be found in section D.1.

The rules defined in section 6.8.1, $PRULES$, $RRULES$ and $DRULES$ are all referred to as *construction rules*, to keep them conceptually separate from the functional rules mentioned in section 6.9.1.

We make a "bare-bones" role construction function before we consider role hierarchies. Then we proceed with the access and ranking functions with and without information hierarchy, and with and without considering user identities.

7.4.1 Functional Role Construction, Without Role Hierarchies

The first part of regulating access with the model, is to select rules for classes based on who the user is, and what roles the user has activated in the session. For simplicity, we first construct a functional role without considering role hierarchies. Figure 7.1 contains the clause for a functional role without the consideration of hierarchies. Following the figure, some of the called clauses are explained in detail. The tactic used is to first extract a list of all construction rules ($CRULEs$) from the roles that are activated. The activated roles are given a list of construction roles as the input parameter $CONSTLIST$. After flattening this list, the construction rules have to be *learned* - converted into valid *functional rules*, $FURULEs$.

```
/*
fur_nonhier(+CONSTLIST, -FURULEs) :-
*/
fur_nonhier(CONSTLIST, FURULEs) :-
   dsd_ok(CONSTLIST),
   crulesinconstlist_nonhier(CONSTLIST, COR_CRULES),
   flatten(COR_CRULES,CRULEs),
   learnallcrules(CRULEs),
   collect_furules([],FURULEs).
```

Figure 7.1: Constructing a functional role without hierarchies

Collecting construction rules is a rather trivial task in the non-hierarchical version of the function. The clause for this is shown in figure 7.2, in order to compare the clause with the hierarchical version of the clause in section 7.4.2.

```
crulesincor_nonhier(CORID,CRULEs) :- findall(CRULE,cor(CORID,CRULE),CRULEs).
```

Figure 7.2: Collect all construction rules from the roles

The most important rule in 7.2 is the rule for learning the functional rules, *learncrule*. It is in this clause, see figure 7.3, that the choices for treating conflicting construction rules are made. The clause *learnallcrules* simply invokes *learncrule* for each of the construction rules in a list. As mentioned in section 6.9.1, conflicting construction rules are rules about the same class that happen to activated at the same time. If one of the previously learned rules is about the same class, the functional rule is retracted and reasserted with new values. If not, the construction rule is merely learned into the database. After the clause *learnallcrules* is invoked on a list of construction rules, the Prolog database contains valid functional rules.

The valid functional rules may now be collected into a list by *collect_furules* in figure 7.1, and returned as an output parameter. The database is thus cleared and ready for the next role construction.

```
/*
learncrule(crule(accrank(+REL,+DEL,+PERM),+CLASS)) :-
*/

:- dynamic furule/2.

learncrule(crule(accrank(REL,DEL,PERM),CLASS)) :-
   retract(furule(accrank(OLDREL,OLDDEL,OLDPERM),CLASS)),
   max(REL,OLDREL,NEWREL),
   max(DEL,OLDDEL,NEWDEL),
   addperm(PERM,OLDPERM,NEWPERM),
   asserta(furule(accrank(NEWREL,NEWDEL,NEWPERM),CLASS)).

  learncrule(crule(accrank(REL,DEL,PERM),CLASS)) :-
   asserta(furule(accrank(REL,DEL,PERM),CLASS)).
```

Figure 7.3: Learn a single construction rule into the database

7.4.2 Functional Role Construction, With Role Hierarchy

The hierarchical role construction, shown in figure 7.4, is similar to the non-hierarchical role-construction, shown in section7.4.1. The only difference is the call to a hierarchical version of

the *crulesinconstlist* clause, *crulesinconstlist_hier*. The clause *crulesinconstlist_hier* is shown together with the clauses *dirindir* and *ancestor* in figure 7.5. The rules in COR_CRULES are CRULEs from not only the roles contained in the roles in $CONSTLIST$, but also all CRULEs from the ancestors of those roles. Compare these clauses to *crulesincor_nonhier* in figure 7.2.

```
/*
fur(+CONSTLIST, -FURULEs) :-
*/
fur(CONSTLIST, FURULEs) :-
   dsd_ok(CONSTLIST),
   crulesinconstlist_hier(CONSTLIST, COR_CRULES),
   flatten(COR_CRULES,CRULEs),
   learnallcrules(CRULEs),
   collect_furules([],FURULEs).
```

Figure 7.4: Constructing a functional role with role hierarchies

7.4.3 Ranking Without Information Class Hierarchy

The valid functional role, that is constructed, has only one $FURULE$ and $ACCRANK$ assigned to each class. Thus, the final ranking becomes a matter of finding the class the object is a member of, and then assigning the rankings and permissions associated with that class to the object. First, we do this only with direct classes, without the class hierarchy.

Figure 7.6 shows the non-hierarchical ranking clause, without the user argument. $FUNCROLE$ is a valid functional role, and $OBJECT$ is the object to be ranked. $ACCRANK$ is the access-ranking triplet that is the result and output parameter from the ranking function.

7.4.4 Ranking With Information Class Hierarchy

In this section, the clauses in section 7.4.3 are extended with class hierarchies. Figure 7.7 contains a hierarchical version of the ranking clause. The clause is still without the user argument. As with the role construction, there is only a slight difference between the hierarchical and non-hierarchical versions of the function.

The final ranking function is shown in figure 7.8. The final ranking does check for available roles and static separation of duty. It then constructs a functional role using the role construction function, and ranking objects with the *ranknou* clause. We note that this version of the Prolog implementation does not consider patient access control lists as defined in section 6.8.2.

```
crulesincor_hier(CORID,CRULEs) :-
  findall(CRULE,(dirindir(CORID,D),cor(D,CRULE)),CRULEs).

/* dirindir(?Direct, ?DirIndir)
   */

dirindir(Direct, DirIndir) :-
   Direct=DirIndir.

dirindir(Direct, DirIndir) :-
   ancestor(Direct,DirIndir).

/* ancestor(?ChildID, ?AncestorID) :-
   */
ancestor(ChildID, AncestorID) :-
   objectParent(ChildID, AncestorID).

ancestor(ChildID, AncestorID) :-
   objectParent(ChildID, A1),
   ancestor(A1, AncestorID).
```

Figure 7.5: Listing construction rules from roles and ancestors

```
/* ranknou_nonhier(+FURULEs,+OBJECTID,-ACCRANK) :-
  */

ranknou_nonhier(FURULEs,OBJECTID,ACCRANK) :-
   obj(OBJECTID,CLASS,_,_),
   member(furule(ACCRANK,CLASS),FURULEs).
```

Figure 7.6: Ranking an object without class hierarchy

```
/* ranknou(+FURULEs,?OBJECTID,-ACCRANK) :-
  */

ranknou(fur(CFURULEs,OBJECTID,ACCRANK) :-
   obj(OBJECTID,CLASS,_,_) ,
   rankrecursive(FURULEs,CLASS,ACCRANK).
```

Figure 7.7: Ranking an object with class hierarchy

```
/* rank(+USER,+SESSION,?OBJ,-ACCRANK) :- :-
  */

rank(USER,SESSION,OBJ,ACCRANK) :-
   ssd_ok(USER) ,
   roles_available(USER,SESSION) ,
   fur(SESSION,FURULEs) ,
   OBJ=obj(OBJECTID,_,_,_) ,
   ranknou(FURULEs,OBJECTID,ACCRANK).
```

Figure 7.8: Ranking an object with class hierarchy

7.4.5 Implementation of Separation of Duty

The implementation contains the *ssd_ok* and *dsd_ok* clauses for checking static and dynamic separation of duty, respectively. Static separation of duty takes role hierarchies into account, just as specified in section 6.5. In the implementation SSD is checked at role activation, rather than at role assignment. This should be changed, so that instead of the regular user, it is the administrator assigning roles that is alerted of the violation. Dynamic separation of duty does not take role hierarchies into account, as in the formal model.

7.5 Summary

In this chapter, we introduce the programming language and the technical environment in which we implement the model. We then present the implementation, which has clauses for role construction, ranking, separation of duty, and various support clauses. After implementing parts of the model, the implementation can be tested.

Chapter 8

Testing and Results

This chapter contains the results from testing the model that is presented in chapter 6. We have tested this model with the scenario presented in chapter 5. The testing is done with the Prolog implementation of the model presented in chapter 7. The testing of the model is split into sections similar to the ones in the implementation chapter, 7. The sections can roughly be divided into role construction, ranking of and access to information and separation of duties. There is also a small section showing testing the possibility of using the model with user interfaces.

8.1 Testing the Prolog Implementation

The thesis from the specialization project [SØ04] describes a manually executed application of the model on two test cases. In this thesis, we use the Prolog implemented version of the model, presented in chapter 7, for testing.

The first part of the test is done with a implementation of the core model. This means that there is no role hierarchy, information hierarchy or separation of duty. We then proceed by adding role hierarchy, information hierarchy and separation of duty. We act as security administrators by instantiating users, roles, operations, objects and patients, and assigning permissions, relevance and detail to the roles.

When we test the model we try the different rules to see what objects the user has access to and what ranking the object has according to the user. Because the EHR system will authenticate the user, know which roles the user has activated, and which patient and objects that are to be access and ranked, this testing would show that the permissions and information rankings are interpreted as intended in the model.

In the implementation, the $CONSTLIST$ and FUR that are constructed are similar to a login request and role activation. Trying to access an object takes FUR and $OBJECT$ in as input

parameters and produces $ACCRANK$ as an output parameter. To see the code of the actual functions being used, see the appendix section D.1.

8.1.1 Functional Role Construction, Without Role Hierarchies

The first property we test is role construction without role hierarchies. The test consists of three parts. The first is activating two roles that have no rules about the same class, figure 8.1, and the second is activating two roles that have rules about the same class, figure 8.2. The third part is for testing that the roles in the second part actually produce different results when activating them separately, 8.3.

When activating two roles (role 1: Staff and 5: Nurse) that have no rules about the same class, we get the expected result of a list of rules from both roles, as shown in figure 8.1

```
?- fur_nonhier([1,5],FURULEs).

FURULEs = [furule(accrank(1, 1, [2]), 7), furule(accrank(1, 1, [2]), 9),
furule(accrank(1, 1, [2]), 4), furule(accrank(4, 1, [2]), 26)]
```

Figure 8.1: Activating roles 1 and 5 without role hierarchy

When activating two roles (role 10: Internist and 5: Nurse) that have rules about the same class, we get the expected result, as shown in figure 8.2

```
?- fur_nonhier([10,5],FURULEs).

FURULEs = [furule(accrank(5, 5, [2]), 19), furule(accrank(4, 6, [1, 2, 3]), 26)]
```

Figure 8.2: Activating roles 5 and 10 without a role hierarchy

If we look at the rules (construction rules) for roles 10: Internist and 5: Nurse in the case file, we see that they both have rules about class 26. We demonstrate this by separately constructing functional roles of each construction role in figure 8.3. We get the expected results and this result also confirms that the result from the second test (figure 8.2) is correct.

```
?- fur_nonhier([5],FURULEs).

FURULEs = [furule(accrank(4, 1, [2]), 26)]

?- fur_nonhier([10],FURULEs).

FURULEs = [furule(accrank(5, 5, [2]), 19), furule(accrank(3, 6, [1, 2, 3]), 26)]
```

Figure 8.3: Activating roles 10 and 5 individually, without role hierarchy

The results from all three parts of the test are as expected. The results show that the rule about class 26 in the construction of roles 5 and 10 is given the maximum relevance and detail levels of the rules in each of the construction roles.

8.1.2 Functional Role Construction, With Role Hierarchy

After testing that the role construction works without hierarchies, the second test make sure that introducing role hierarchies has the desired effect and that the role construction still works as expected. The role hierarchies we have used in this case are presented in figure 8.4, which shows the staff member role hierarchy, and figure 8.5, which shows the location role hierarchy. We note that the roles in the hierarchies are given numbers, which are corresponding to the numbers used in the scenario facts file.

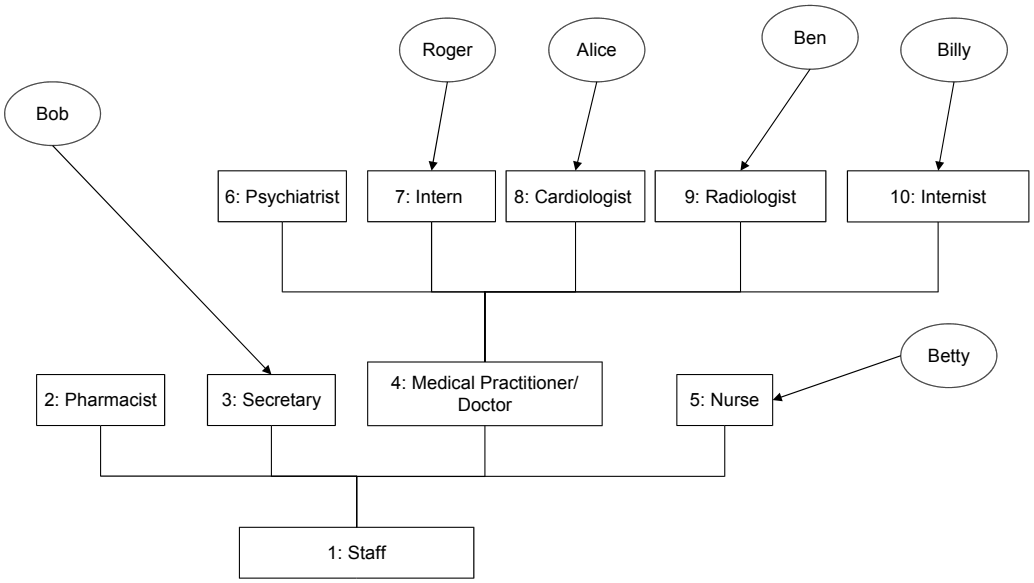


Figure 8.4: Staff member role hierarchy

From the hierarchy in figure 8.4 we can see that the functional rules resulting from activating roles 1 and 5 are the same as those activated without role hierarchies. The only difference is that the list of functional rules is ordered differently, something that is rather irrelevant. The reason for the functional rules being identical, is that role 1 is the parent role of role 5.

Activating role 7 (Intern), shown in figure 8.6, makes functional rules based on the construction rules in roles 7 (Intern), 4 (Medical practitioner) and 1 (Staff). If the rules have a conflict, the maximum of relevance and detail is chosen, and permissions are accumulated. In the scenario, there are no construction rules for role 7 itself, so the rules are all from roles 4 and 1.

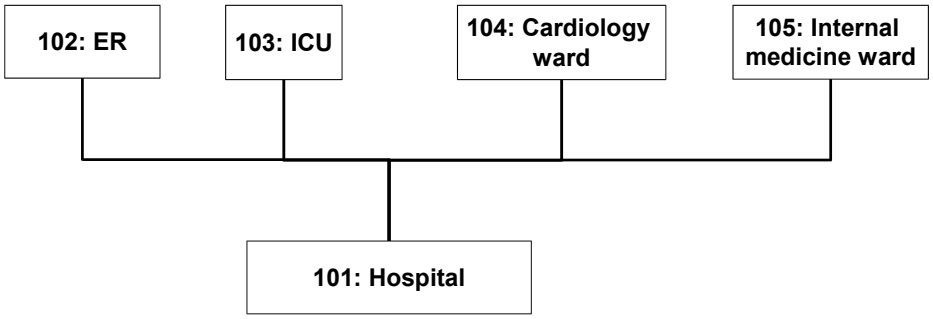


Figure 8.5: Location role hierarchy

```
fur([7],FURULEs).

FURULEs = [furule(accrank(3, 2, [2]), 5), furule(accrank(4, 4, [2]), 6),
furule(accrank(1, 1, [2]), 7), furule(accrank(1, 1, [2]), 9),
furule(accrank(4, 2, [2]), 4)] ;
```

Figure 8.6: Activating role 7 with role hierarchy

In figure 8.7 the roles 7 (Intern) and 102 (ER) are activated. This makes functional rules based on roles 102 (ER) and 101 (Hospital), in addition to 7,4 and 1. There are no rules defined for role 101 in the scenario. The results in 8.7 show that role 102 (Location ER) gives the information class 4 relevance 6, detail 6 and read access. (furule(accrank(6, 6, [2]), 4)). These are higher relevance and detail levels than those obtained by just having activated role 7 (Position Intern) furule(accrank(4, 2, [2]), 4).

```
?- fur([7,102],FURULEs).

FURULEs = [furule(accrank(3, 2, [2]), 5), furule(accrank(4, 4, [2]), 6),
furule(accrank(1, 1, [2]), 7), furule(accrank(1, 1, [2]), 9),
furule(accrank(6, 6, [2]), 4)] ;
```

Figure 8.7: Activating roles 7 and 102 with role hierarchy

In figure 8.8 role 10 (Internist) is activated. The result of activating role 10 (Internist) is functional rules based on the construction rules in roles 10 (Internist),4 (Medical practitioner) and 1 (Staff). If the rules have a conflict, the maximum of relevance and detail is chosen, and permissions are accumulated.

```
?- fur([10],FURULEs).

FURULEs = [furule(accrank(5, 5, [2]), 19), furule(accrank(3, 6, [1, 2, 3]), 26),
furule(accrank(3, 2, [2]), 5), furule(accrank(4, 4, [2]), 6), furule(accrank(1, 1, [2]), 7),
furule(accrank(1, 1, [2]), 9), furule(accrank(4, 2, [2]), 4)]
```

Figure 8.8: Activating role 10 with role hierarchy

8.1.3 Ranking Without Information Class Hierarchy

The results in 8.1.2 and 8.1.2 demonstrate that the functional role is correctly constructed both in presence and absence of role hierarchy. The next part to be tested is the ranking of the information objects. This test will also consist of two steps; first we are testing ranking without information class hierarchy, then we are testing with information class hierarchy.

In figure 8.9 we are using the result of activating role 7 (Intern) and 102 (ER) at the same time, and we ask the question ”What information objects does the Intern in the ER have access to, and at what relevance and detail levels?”. The answer given in figure 8.9 shows two objects with corresponding access and ranking.

In figure 8.10 the contents of the information fields are shown. We extract them by re-stating the question, explicitly asking Prolog to bind variables like $CONTENT$.

The results show that the Intern in the ER has access to objects numbered 20 (the name ’Elisa Eliasen’) and 22 (the social security number ’078-05-1120’) with various *accranks*. These are only objects that are of direct information classes found in the functional rules. To rank objects

```
 ?- rank_nonclasshier('Roger',[7,102],OBJ,ACCRANK).

OBJ = obj(20, _G1036, _G1037, _G1038)
ACCRANK = accrank(1, 1, [2]) ;

OBJ = obj(22, _G1036, _G1037, _G1038)
ACCRANK = accrank(1, 1, [2]) ;

No
```

Figure 8.9: Ranking identifiers with roles 7 and 102, without information class hierarchy

```
?- rank_nonclasshier('Roger',[7,102],obj(OBJID,_,_,_),ACCRANK),obj(OBJID,CLASS,PATIENT,CONTENT).

OBJID = 20
ACCRANK = accrank(1, 1, [2])
CLASS = 7
PATIENT = 1
CONTENT = 'Elisa Eliasen' ;

OBJID = 22
ACCRANK = accrank(1, 1, [2])
CLASS = 9
PATIENT = 1
CONTENT = '078-05-1120' ;

No
```

Figure 8.10: Ranking contents with roles 7 and 102, without information class hierarchy

that don't have direct information classes in the functional rules, we need to consider information class hierarchies.

8.1.4 Ranking With Information Class Hierarchy

The results from ranking information in the absence of an information hierarchy are as expected, but to avoid the work of making rules for every object for every role we introduce an information hierarchy in section 7.4.4. The figure 8.11 shows how we place the clinical information classes used in the testing in a hierarchy. We note that access and ranking to users are given to objects, which are members of an information class.

In figure 8.12 the use of information class hierarchy is demonstrated. In particular, figure 8.12 shows the objects that the user 'Roger' acting as an intern in the emergency room (role 7 and 102) has access to, with their corresponding relevance levels, detail levels, permissions and content. The results show that the Roger the intern in the ER has access to objects numbered 1,2,3,4,5,6,7,8,11,14,20 and 22 with various relevance levels, detail levels and permissions. In this case, the permissions are all of type 2 (read).

Important to the results in presence of a information class hierarchy, are that when the user is denied access to a object that is member of a class that is senior to another class, the user is denied access to objects of the junior class as well. The ranking in the hierarchy also has the desired effect with that the object that is member of a class is given at least the ranking of the senior class if nothing else is stated in the scenario. The results of testing the ranking with hierarchy also show that using information hierarchy is more effective, because there is less information to be ranked.

Denying Access
Figure 8.13 and figure 8.14 demonstrate the difference between users trying to activate the same role. In figure 8.13 we try to let the user 'Roger' access objects as an internist (role 10) in the internal medicine ward (role 105). This access fails, because 'Roger' does not have the internist role assigned. In figure 8.14 the user 'Billy' succeed in this role activation, thus he has access to the objects.

Comparing 8.12 and figure 8.14 we also see that there is a difference between the access given to Billy (Internist) and Roger (Intern). Billy gets a higher detail level on the current insulin treatment (object number 11), and also gets permission to do operations 1 (create) and 2 (write) for that object.

Simulating a User Interface
To test a practical application of information ranking we simulate a user setting a minimum relevance level in a user interface. Figure 8.15 shows predicate *shown* by setting the minimum relevance level to 4 for the user 'Billy'. Only four objects are shown in the interface compared to twelve in figure 8.14. In figure 8.16 'Billy' lowers the relevance level in the user interface to 2. This shows all the objects shown in figure 8.14, except for the name and social security number.

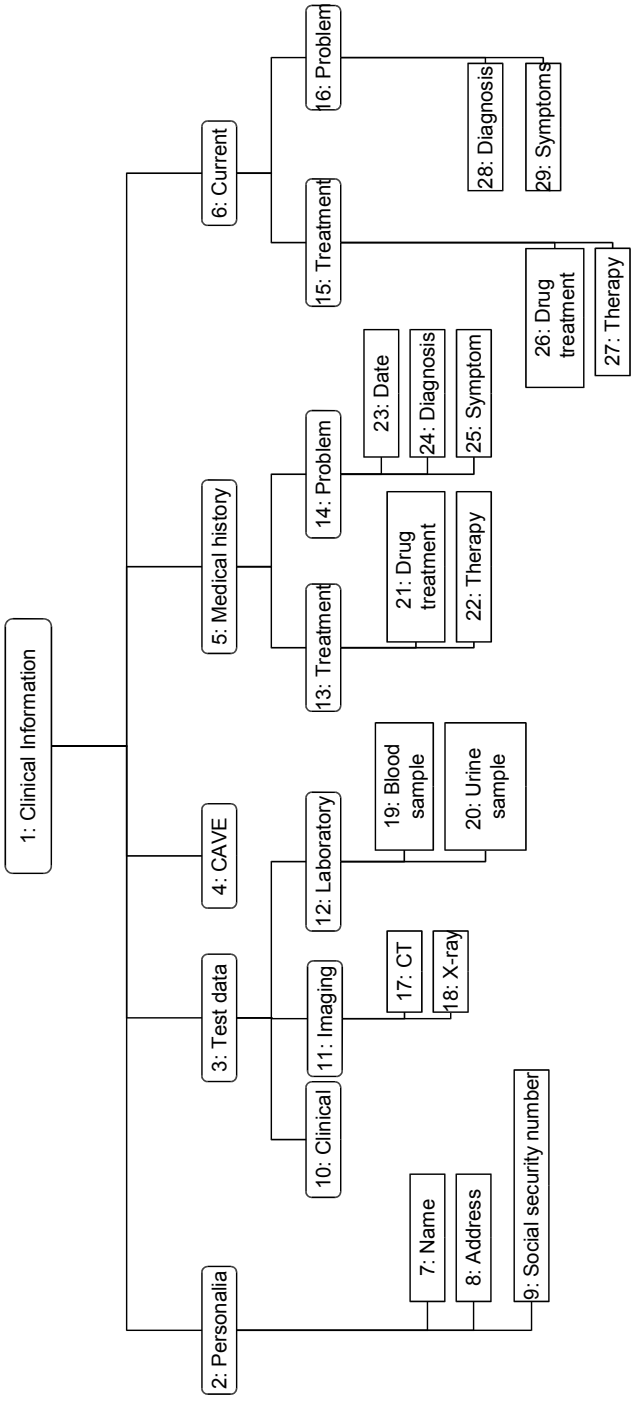


Figure 8.11: Clinical information classes in the scenario

```
?- rank('Roger',[7,102],obj(OBJID,_,_,_),accrank(REL,DEL,PRIV)),obj(OBJID,_,_,CONTENT).

OBJID = 1,REL = 3,DEL = 2,PRIV = [2],CONTENT = 'diabetes mellitus' ;
OBJID = 2,REL = 3,DEL = 2,PRIV = [2],CONTENT = insulin ;
OBJID = 3,REL = 3,DEL = 2,PRIV = [2],CONTENT = hypertension ;
OBJID = 4,REL = 3,DEL = 2,PRIV = [2],CONTENT = 'beta blocker' ;
OBJID = 5,REL = 3,DEL = 2,PRIV = [2],CONTENT = tiazide ;
OBJID = 6,REL = 4,DEL = 4,PRIV = [2],CONTENT = hypoglycemia ;
OBJID = 7,REL = 4,DEL = 4,PRIV = [2],CONTENT = 'syncope (fainting)' ;
OBJID = 8,REL = 4,DEL = 4,PRIV = [2],CONTENT = 'trauma to head' ;
OBJID = 11,REL = 4,DEL = 4,PRIV = [2],CONTENT = insulin ;
OBJID = 14,REL = 4,DEL = 4,PRIV = [2],CONTENT = 'heart attack' ;
OBJID = 20,REL = 1,DEL = 1,PRIV = [2],CONTENT = 'Elisa Eliasen';
OBJID = 22,REL = 1,DEL = 1,PRIV = [2],CONTENT = '078-05-1120' ;
No
```

Figure 8.12: Ranking contents with roles 7 and 102, with class hierarchy

```
?- rank('Roger',[10,105],obj(OBJID,_,_,_),accrank(REL,DEL,PRIV)),obj(OBJID,_,_,CONTENT).
Roles not available for activation!

No
```

Figure 8.13: Trying to activate a role that is not assigned

```
?- rank('Billy',[10,105],obj(OBJID,_,_,_),accrank(REL,DEL,PRIV)),obj(OBJID,_,_,CONTENT).

OBJID = 1,REL = 3,DEL = 2,PRIV = [2],CONTENT = 'diabetes mellitus' ;
OBJID = 2,REL = 3,DEL = 2,PRIV = [2],CONTENT = insulin ;
OBJID = 3,REL = 3,DEL = 2,PRIV = [2],CONTENT = hypertension ;
OBJID = 4,REL = 3,DEL = 2,PRIV = [2],CONTENT = 'beta blocker' ;
OBJID = 5,REL = 3,DEL = 2,PRIV = [2],CONTENT = tiazide ;
OBJID = 6,REL = 4,DEL = 4,PRIV = [2],CONTENT = hypoglycemia ;
OBJID = 7,REL = 4,DEL = 4,PRIV = [2],CONTENT = 'syncope (fainting)' ;
OBJID = 8,REL = 4,DEL = 4,PRIV = [2],CONTENT = 'trauma to head' ;
OBJID = 11,REL = 3,DEL = 6,PRIV = [1, 2, 3],CONTENT = insulin ;
OBJID = 14,REL = 4,DEL = 4,PRIV = [2],CONTENT = 'heart attack' ;
OBJID = 20,REL = 1,DEL = 1,PRIV = [2],;CONTENT = 'Elisa Eliasen' ;
OBJID = 22,REL = 1,DEL = 1,PRIV = [2],CONTENT = '078-05-1120' ;
No
```

Figure 8.14: Billy the internist in the internal medicine ward

```
?- shown('Billy',[10,105],4,obj(OBJID,_,_,_),DEL),obj(OBJID,_,_,CONTENT).

OBJID = 6,DEL = 4,CONTENT = hypoglycemia ;
OBJID = 7,DEL = 4,CONTENT = 'syncope (fainting)' ;
OBJID = 8,DEL = 4,CONTENT = 'trauma to head' ;
OBJID = 14,DEL = 4,CONTENT = 'heart attack' ;
```

Figure 8.15: Billy wants just an overview

```
?- shown('Billy',[10,105],2,obj(OBJID,_,_,_),DEL),obj(OBJID,_,_,CONTENT).
```

Figure 8.16: Billy wants to see more information

8.1.5 Separation of Duty

The facts file has the rule *ssd([3,4],2)*, meaning "A user must not have both roles Secretary and Medical practitioner available for activation". For the purpose of testing this rule we can temporarily assign internist Billy to also be a secretary with this rule: *userAssignedRole('Billy', [10,3])*.. When now trying to access information as Billy, the implementation denies access. Figure 8.17 shows that even though Billy doesn't attempt to activate role 3 (secretary), SSD is violated because he has the role available. Billy is not a direct member of role 4, but is an indirect member via role 10.

```
?- shown('Billy',[10,102],2,obj(OBJID,_,_,_),DEL),obj(OBJID,_,_,CONTENT).
Static Separation of Duty violation!
```

Figure 8.17: An SSD violation

As mentioned in section 7.4.5, checking SSD at role activation is not ideal. It should be done at user-role assignment, administration operations is not included in the model. Administration involves changing the facts file and re-consulting it. We change the user-role assignment back to *userAssignedRole('Billy', [10]).* to remove the SSD violation.

Among the rules in the facts file is the dynamic separation of duty rule `dsd([101,102,103,104,105],2).`. This rule implies user can not activate 2 or more of the location roles at the same time. This encodes the policy that a user can not act as being at more than one ward at the time. As shown in figure 8.18 ,trying to activate roles 102 and 105 at the same time triggers an error message and gives no access.

```
?- shown('Billy',[10,102,105],4,obj(OBJID,_,_,_),DEL),obj(OBJID,_,_,CONTENT).
Dynamic Separation of Duty violation!

No
```

Figure 8.18: A DSD violation, trying to be two places at once

8.2 Key Findings

The findings, when testing the model, are as expected and quite obvious. The key findings from the test results are as follows:

- Role construction works in both the presence and the absence of role hierarchy.
- The results illustrated the difference in access, relevance and detail between using a role hierarchy, and not using it.
- Information is ranked by class depending on roles, both in the presence and the absence of a information hierarchy.

- The results illustrated the difference in access, relevance and detail between using an information hierarchy, and not using it.
- Roles are used to express both job function and location, but
- the model is heavily dependent upon the role hierarchy and information hierarchy, and the correct assignment of permissions, relevance and detail thereto, to be correct.

We note that the following properties are expressed by the model in the current form, but is not implemented and therefore not tested:

- Delegation.
- Periodic expressions and run-time requests.
- Patient access control list.

Both the results and limitations of the implementation are further discussed in chapter 9 (Discussion).

8.3 Summary

This chapter presents the results of the testing of the model. The testing is done with the Prolog implementation and is done for role construction, ranking of and access to information and separation of duties. The testing resulted in some key findings, which are further discussed in chapter 9.

Chapter 9

Discussion

This chapter is a discussion of the design choices we made in this thesis. Important to this discussion is the desired effect versus the actual effect. We first consider the choice of the formalism, in which we present the model, and second we consider the different architectural components we have used. Subsequent to this the accomplishments and limitations of Prolog implementation of the model are discussed.

The end of this chapter provides a discussion of possible applications of the model, and it gives some directions for improvements and future work.

9.1 Using a Formal Language to Express the Role-model

Although not a very large part of this thesis, the choice of a formal language is definitely important for the thesis. The properties we set for the language are stated in research goal number three on page 2. These properties are:

- It has to be a formal and expressive language
- It should be based on deontic logic
- It should already be defined in the literature

It is also important, though not stated explicitly in the research goals, that the language should be computer interpretable for the purpose of making an executable model. The translation and implementation of the model is further discussed in section 9.3.

As systems become more complicated, and safety becomes a more important issue, the formal approach to system design offers another level of insurance against design faults. By using a formal language we have hoped to clearly define the problems, goals and solutions of the model, and we can by the means of mathematical and logical techniques express, investigate, and analyze the model and its behavior.

The reason for choosing deontic logic is that it formalizes the notions of obligations, prohibition and permissions corresponds to access control activities, , as stated in chapter 2. By using a language, that is a first order many sorted logic, an implementation of the model can be done relatively simply in a logic programming language.

It saves us a lot of time to use an already defined language and we could also review how the language had been applied to other problems.

The language we chose to use is both formal and expressive. It is based on deontic logic and has already proved to be useful for expressing delegation in a role-based organization. Thus, the language is especially useful for representing the delegation part of the model. The language is also well suited to express the rest of the model. We only take full advantage of the deontic characterization of the language, when it comes to expressing the patient access control list in the function for ranking and granting in figure 6.3 and 6.4. We also note that we have not used the agent term in the model and the model slightly deviates from the formalism. This and other deviations from the language the model might have, is not essential to the fact that the use of a formalism has made us able to develop a unambiguous model.

9.2 The Different Architectural RBAC Components

The primary aim for this thesis is to create a role-model, which is compliant to access control activities in the Norwegian EHR standard. In addition to this the model should be used for information ranking. With several different models for RBAC in the literature and with each of them solving different aspects of access control, we needed to find those architectural components, which together solve the access control activities related to the EHR standard and yet avoid ambiguity and unnecessary complexity.

9.2.1 Using the NIST Standard

The main reason for choosing the NIST standard as a foundation for the role-model is that it is considered authoritative work on RBAC. The fact that we used this standard in the specialization project [SØ04] and therefore are familiar with the standard is also significant for this choice. The core elements from the NIST standard are of course included in the model, but we had to decide whether including role hierarchies, static separation of duty and dynamic separation of duties would be accurate for the Norwegian EHR standard.

9.2.1.1 Adding Role Hierarchies

We found it very useful to include role hierarchies as several different roles in a healthcare organization are similar, but not equal, in both responsibilities and information needs. It is also

important that a user may be assigned several roles in a hierarchy, and if users are a part of the hierarchy it is necessary to include general role hierarchies. When working with the scenario we also found that it could be useful to define several role hierarchies.

Although the model was only tested and validated on role hierarchies created from the scenario, the formal definition of role hierarchies in the model supports any implemented hierarchy. To find complete and correct role hierarchies for a typical hospital in Norway would require separate effort with empirical work, interviews and analysis (see also future work, section 9.5).

9.2.1.2 Adding Separation of Duties

The reason for including separation of duties in the model is quite obvious. There are several roles that may not be assigned to single user or activated by a single user in the same session. Nevertheless it is difficult to conclude when static separation of duties should be used and when dynamic separation of duties should be used. Although we stated in chapter 3.3.3 that dynamic separation of duties probably is the most common to occur in a healthcare organization, we included both static and dynamic separation of duty in the model.

The scenario we used to test the model did not include any obvious cases of separation of duty, except object-related separation of duty (i.e. creator, owner, approver etc. of a single information object). Since the model considers separation of duty between roles independently of the objects, we had to leave that kind of SoD to possible further work, and rather introduce some other rules for SSD and DSD. With these rules, the results in section 8.1.5 show that separation of duties have the intended effect.

9.2.2 Extending the NIST Standard

To comply with the Norwegian EHR standard the model has to be extended in order to handle delegation, context and patients preferences. We looked at two different approaches to make these extensions. One is to extend the model with traditional delegation, temporal RBAC and an access control list for patient preferences. The other is the OASIS architecture, which uses role parametrization to achieve a more fine grained access control. The OASIS approach supports delegation and it could also be used for a context representation. Because traditional delegation and temporal RBAC are more compatible with the NIST standard, we chose to follow the first approach rather than the OASIS-approach.

9.2.2.1 Delegated Roles and Role Units

One of the requirements in the Norwegian EHR standard is that a user should be able to act on behalf of another user. Thus, this user would require some of the same access privileges as the

user it is acting on behalf of. The model we have developed reflect this requirement by introducing delegation. One important reason for separating the delegated roles and the assigned roles, is that it makes revocation of delegation possible. A second reason is that it makes it possible to configure, based on local policy, different properties of delegation, such as transitively, completeness and permanence.
Another design issue related to delegation is the introduction of role units, which are collections of access privileges smaller than roles. This makes it possible to choose which permissions one wants do delegate. This is also in accordance to the delegation requirement in the EHR standard.

The formal approach to delegation gives a precise way of interpreting delegation in the model. The only problem we encounter when formalizing delegation, is the fact that delegation in a healthcare organization can be informal or implicit. If implicit delegations are to be used in the access control model, they must be formalized and handled in some deterministic way, even if they might not need to be turned into explicit delegation.

9.2.2.2 Representing the Context

Access control activities with the Norwegian EHR standard as a basis, are heavily dependent on the context, in which access is given. This is reflected in the model by having roles related to both location and to care phase[1], in addition to existing roles that relate to job position and job functions. The run time request and periodicity expressions are made to support these roles with temporal constraints and a changing environment.

Although the model expresses roles as points in the care process, periodicity and run time request, the model's time representation is quite simple. This is to keep the model compatible with the NIST standard and to avoid unnecessary complexity. The actual implementation of the time concept is omitted due to the need for a lot of support data and several databases. The limitations of the implementation are more thoroughly discussed in section 9.3.

9.2.3 Access Relating to the Patient

To gain access to healthcare information there are two key requirements that need to be fulfilled. The first is; that access is given as a consequence of a decision to bring about some action, usually providing care to the patient. The second is; that the patient has to approve that his or her healthcare information can be accessed. Both of these requirements introduce some difficulties that have to be solved.

The first requirement makes access control context and patient dependent. When the patient is already admitted to the hospital we assume that he or she requires healthcare, so access to healthcare information is granted to someone. We have assumed in the testing of the model, that the users trying to access a patient's record actually are performing some kind of care. Thus, if

[1] Phase in the care process that the patient is in.

a user is on duty on the same ward as the patient is admitted, the user has a right to that patient information in accordance to his/her job position, i.e. nurse or doctor. This is a reasonable assumption, because healthcare personnel often have to obtain information immediately; a more restrictive access control could complicate the care process.

The second requirement is not a problem as long as all patients approve that anyone who fulfills the first requirement, gain access to their patient records. Because the patients are entitled to deny anyone access to their information, the model must support this. To use a purely role-based approach would be possible, i.e. coding patients' preferences in each and every role, but this would introduce security administration overhead; which is precisely the problem RBAC is trying to avoid. Thus, we introduced an access control list, which revokes access from users and roles, which originally have access based on their role, but that are not allowed to access information because of patient preferences.

Another solution to these two difficulties is to use role parametrization, this would perhaps be a way of making object-related roles and patient-related roles. This is probably a more elegant solution, but we have not studied this option any further.

We manually validated the use of a patient access control list in the specialization project [SØ04] In this version we have not tested this use. Neither did we include it in the Prolog implementation, discussed in section 9.3.

9.2.4 Role Dependencies

Having different roles to express both job functions and the context, make the activation of one role dependent on the activation of another role. For instance being doctor_on_call requires the user both to activate the role as a doctor and on_call. We suggest three ways of solving role dependency:

The first is that we can treat each role like doctor_on_call or nurse_on_call as individual roles. This solution yields one giant role hierarchy, where the number of roles is $M * N$, where M and N are the numbers of roles in two distinct hierarchies. With additional hierarchies this solution is inefficient.

The second option, and included in the model, is letting each role have access privileges and information ranking regardless of the other roles, and only increasing access privileges if necessary. This is also in accordance to the EHR standard, which states that a user may activate several roles. A down side to this solution is that the solution can not force a user to activate several roles at once to gain privileges. For instance, a user in the scenario should need to activate at least one role in the staff member hierarchy, one role in the location hierarchy and one role in the responsibility hierarchy to make this solution work adequately [2]. This solution lets the user

[2] In our model, SoD rules are expressed as (rs, n), where maximum $n-1$ roles in rs may be used at once (section 6.5). If SoD is expressed with (rs, n, m), where maximum $n-1$ and minimum $m+1$ roles in rs may be used, SoD rules could be used to require that one role from each role hierarchy is activated.

gain some privileges by just activating one of those roles.

A third solution to this is using parametrization of roles, which make it easier to introduce dependencies among roles. This is similar to the second solution, but have the advantage that it can make role dependencies independent of hierarchies, see section 9.5 for future work.

9.2.5 Information Ranking and Access Control Model

Another one of the research goals for this thesis is to use the role-model we developed for both access control and information ranking. The reason for this approach is that access control and information ranking are closely related. The main goal of access control is to prevent users from obtaining information they are not authorized for. Access control can also be used to prevent users from obtaining a lot of unnecessary information. This is where information ranking can be used, by giving ”more” access to some objects that are believed to be important and hiding other unimportant information. Thus the user can access the information, but is not bothered with it in an already information intensive work environment.

To allow for effective information ranking we have done some important design choices. The first choice is to use information classes and information hierarchies. The second choice is to rank information with two parameters. The third and final design choice is how we wanted to relate information ranking to the model.

We note that some of the design choices were already made and discussed in the specialization project [SØ04], but we discuss them here because they are important to the understanding of the design.

9.2.5.1 Information Classes and Hierarchies

It is necessary to include information classes and hierarchies in order to model patient data for efficient access control and information ranking. The concept of placing information objects in information classes is partly obtained from the Norwegian EHR standard and partly from other approaches to patient data modelling. The concept of information hierarchies is adopted from the role hierarchies in the NIST standard. The reason for modelling patient data like this, is that it is easier to administrate both access permissions and information ranking for particular roles.

All test data supports that using information classes and hierarchies is a correct choice.

9.2.5.2 Using Two Parameters

The choice to rank information with both relevance and detail is a design choice we discussed thoroughly with our supervisor when working with the specialization project [SØ04, pp. 50–53].

The two parameters were chosen, because they are both complementary and relatively independent. Relevance supports prioritizing of information and detail ranking shows information with different precisions.

The relevance and detail ranking should be done for each role similarly to how access permissions are given, making use of both role hierarchy and information hierarchy.

The outcome we hoped for from the test results was that the amount of information would be reduced because of a lower relevance level, the information with the highest relevance would be displayed regardless of the detail ranking, and that this ranking was according with the scenario. The results are positive to both reducing the amount of information shown and assigning detail levels, though we realize that the result may be biased because the scenario is made by us, for the purpose of demonstrating the model.

9.2.5.3 Relating Information Ranking to the Model

Because information ranking is closely related to access control we discussed several options for combining relevance ranking, detail ranking and permissions in the specialization project [SØ04, pp. 50–53]. The two main options discussed were combining detail ranking with permissions and having relevance ranking independent of these; or having permission, detail and relevance all independent of each other. The latter design was chosen because it is a much simpler and a more flexible design.

9.2.6 Functions

The role construction function, in section 6.8, is made to reflect the access rights, for each of the roles it contains, with role hierarchies. The results from the tests shows that this is achieved. Because we are unable to test delegation and temporal constraints with the implementation, we decided not to advance delegation and temporal constraints in the function. Thus, the function has a trivial solution for treating this.

The ranking and granting function, in section 6.8, evaluates the functional role, make use of information hierarchies, and then gives the correct access and ranking to objects. The first goal of this function is to illustrate the purposes and effect of information hierarchies. The second is to illustrate how an access control list can be used in the model. These goals are met in the model, but not in the implementation. Only the evaluation of hierarchies and the subsequent access permissions and information ranking are implemented, (see section 9.3). The evaluation of the patient access control list was tested and found reasonable in the specialization project [SØ04]. The patient access control is used to reflect the patient preferences, and is therefore an absolute necessity in order to comply with the Norwegian EHR standard.

9.3 The Prolog Implementation of the Model

As stated in section 9.1, the implementation is done in order to test different qualities of the model. This implied that the formal model could be made executable in a programming language. Because the formalism is based on deontic logic, we assumed that a logic programming language would be adequate for this task. Thus, the implementation would be a translation from one logic language into another.

The implementation is only useful for testing certain qualities of the model. Because the implementation is only meant to show that the formal model could be made executable and provide test results for validating it, the discussion of the practical use of the implementation must be seen from that perspective. Thus, the implementation should be useful to show that the formal model could be made executable, and that this is useful for testing and validating the model.

9.3.1 Translation From One Language To Another

Implementing the model in Prolog meant that we had to transform the model's formulae into conjunctions of Horn clauses. We have done this translation, but only with partial success. This is because only parts of the model are actually implemented. In addition the representation of the model deviated slightly from the original language. Although the partial implementation worked appropriately and the model is expressed formally, we cannot conclude whether all parts of the model, or if deontic logic can be implemented.

9.3.2 Limitations of the Implementation

As mentioned in section 8.2, delegation, periodic expressions and patient access control lists are not part of the implementation.

In order to complete the implementation some compromises had to be made between the necessities of engineering and the goals for formal design. The implementation does not support delegation nor does it support periodic expression and run time expressions. Implementing delegation could have demonstrated how the delegated role would be handled as an assigned role, with some added attributes that would support revocation and restriction of how delegation would take place. Implementing periodic expressions and run-time request would not have changed much of the model, because this can be tested by only changing the facts for placing time constraints and environment changes and then run the implementation once more. In effect, the implementation supports the most essential parts of the model and is effective for testing and validating the model.

The implementation also lacks the ability to express patient access control lists. This is a significant limitation, but this is still not essential to the value of the implementation. We showed that

parts of the model can be implemented, and the implementation can be extended with the parts missing. (See section 9.5).

SSD is implemented for hierarchies, while DSD is implemented in a non-hierarchical manner. This corresponds to the NIST standard definitions, and is not necessarily something that should be changed, but one should be aware of this when defining and encoding SoD policy.

9.4 Application of the Model

When we look at the application of the model we have developed, there are two closely related areas that have to be discussed. The first is whether to use the model in a centralized or a distributed method for RBAC. The second area is whether to use the model on top of many systems or integrating the model within single systems.

9.4.1 Centralized versus Distributed RBAC

Many existing methods of using RBAC are centralized, but as many services are distributed across numerous servers, it becomes more attractive to use a distributed RBAC method.

The main argument for choosing a distributed approach, is that access control models must keep up with how the architectures of most computing systems are shifting from being centralized to being distributed. This require a de-centralized fine-grained access control model, instead of a centralized coarse grained access control model. Also when a system becomes larger, the RBAC control system becomes larger. Because every operation of the user is controlled by the RBAC system, a centralized approach can become the bottleneck of the system.

The main argument for choosing a centralized approach, is that administration of roles in a distributed environment is more intricate. All the systems may not support role hierarchies and every role may not be present on every server. Permission-role assignment is typically done independently at each local system. One example is how a PACS (picture archive and communication system) server and an EHR server can independently determine what privileges are available to the same doctor role. In general, user-role assignment is a logically centralized service, since the same enterprise role may occur on multiple servers. Here, the treatment of the role hierarchy centralized imposes the same seniority relationship at each server.

This version of the model is heavily dependent upon reasonable role engineering (see section 9.5.3). In a healthcare organization it would probably be preferred to do this centrally, because access to information may literally be a matter of life and death. It is therefore not likely that ”distributed” security administrators may decide who may or may not access information.

Even though the system might use a centralized method, availability and resilience would benefit from having local caches for access control, and having redundancy in the centralized servers.

This would help make available potentially life-saving information during power and communications failures. Just providing full access under those situations is not ideal, since all a malicious user would have to do to gain access would be to cut the communication.

9.4.2 On Top of Systems or Integrated in a System

The model we have developed is a general access control model, which can be used for relevance and detail ranking as well. There are two different ways of applying the model. The first is using the model for access control on top of existing systems in a healthcare organization and provide access control and information ranking for all the different systems. The second is integrating the access control model within each system and provide access control for the specific system.

We belive the model can be used for both purposes. The first has the advantage that the access control policy for the entire organization is placed in one application. The drawback is that it is too simple with large granulated role hierarchies or too complex with fine granulated role hierarchies. The second application has the advantage that it can implement access control policies for single applications, thus keeping the fine grained role hierarchies, and avoid unnecessary complexity; but the access control policy has to be implemented several places, complicating consistency and updates.

9.4.3 Adding Relevance and Detail

A related question to the application of the model, is whether the role model should be the only source of relevance values for objects. See section 6.10 for some potential sources of additional relevance and detail level values. These, possibly negative, values could be added to the relevance and detail that is output from the role model.

We observe that with a class hierarchy, the model can not encode or detect drug interactions, abnormalities, or highlight a possible diagnosis based in the information in the record. This is the domain of guideline based decision support. On the one hand, the ranking from the role model is simpler in concept and encoding. On the other hand, using guidelines is one of the significant potential advantages of using electronic health records. Thus, it is essential to prepare the model for additive functions. We also see the possibility of traditional decision support systems using the role model.

9.5 Future work

More work is required in order to make this model sufficient and for covering the security policy for a healthcare organization. This work belongs in three categories. The first is improvement

on the model itself, the second is improvement on the implementation, and the third is the work that relates to building role hierarchies and information hierarchies.

9.5.1 Model Improvements

The model, which is developed, does support the requirements from the Norwegian EHR standard, but by using parametrization the design could have been slightly more elegant. Unfortunately, we believe that the OASIS model, which uses parametrization, is unsuitable to use together with the NIST standard. Parametrization could provide a simpler and more elegant solution to special issues, like role dependencies, object-related roles and patient-related roles. Thus including parametrization in such a way that it is suitable with the rest of the model, could prove to be beneficiary.

9.5.2 Implementing the Model Within a Healthcare Information System

We have implemented a simple prototype of the model in Prolog. The next step for an implementation would be to implement the entire model. This means, as mentioned in section 9.3, to include delegation, periodic expressions and patient access control lists in the implementation. It would also require implementing the support functions for periodic expressions and run-time requests.

The next step is to implement the model in relation to a healthcare information system. This would control access to the system in addition to have relevance and detail ranking tested with a user interface.

9.5.3 Role Engineering

Before all the benefits of RBAC can be realized, all the role-permission assignment have to be correctly defined. This process, called role engineering, aims to define a set of roles that is complete, correct, and efficient. The role engineering process must capture the organization's business rules, as these relate to access control, and reflect these rules in defining, naming, structuring, and constraining a valid set of roles. In short, this means that for the model to be efficient, the components to be defined as part of a role engineering process are:

- Roles
- Permissions
- Relevance
- Detail
- Constraints

- Hierarchies

In addition to this, the patient data has to be modeled so that it is complete, correct, and efficient, and in a form that is compatible with the model.

We have not included role engineering as part of this thesis, because it is a research area of its own. It is a very important part of applying the model. If the role engineering process is done inappropriately, a role-based access control scheme and indeed the model, which we have developed, would also seem inappropriate.

9.6 Summary

This chapter is a discussion of the design choices we have made in this thesis. The language we chose to use is both formal and expressive. It is based on deontic logic and has already proved to be useful for expressing delegation in a role-based organization. Thus, the language is especially useful for representing the delegation part of the model. The language is also well suited to express the rest of the model. We have augmented for the different architectural components used in the model, including the NIST standard components, delegation component, temporal components and the access control list used to support patient preferences. We have also made some suggestions for application of the model. In the future work we have presented, we suggest role parametrization as a possible alternative solution for making roles that are related to patients. The discussion is used for stating the conclusions of this thesis.

Chapter 10

Conclusion

This thesis presents a formal model for access control and information ranking. The model is developed with the healthcare domain in mind. Thus, the model is equipped with the necessary definitions to encode a policy that fulfills the requirements for delegation, time dependency, and role dependency in the access control part of the Norwegian standard for electronic health records.

An important discovery is that traditional role based access control models are insufficient on their own in order to support patient preferences. The model in this thesis supports patient preferences by adding an access control list, but the thesis also indicates that role parametrization could be used to make a more elegant solution to this problem.

The concepts of information ranking were clarified through formal expressions and through given examples. Information ranking can be used in relation with access control to prevent users from obtaining a lot of unnecessary information. Thus, the user is not bothered with unimportant information in an information intensive work environment. Two very important properties of information ranking, are that it does not require any input from the user and it does not prevent the user from obtaining any information the user requires.

The formal model is partially implemented in Prolog. The implementation has two major objectives. The first is to test and validate the model using data from a scenario. The results of the tests show that the implementation behaves consistently with the formal definition and with our expectations. The implementation lacks the ability to express delegation, time dependency and patient preferences. The implementation is thus insufficient for testing wether or not the model complies with the Norwegian EHR standard, but it did successfully validate the parts that are implemented.

The second objective for implementing the model is to see whether deontic logic can be implemented using a logic programming language. The implementation as it is, is not an implementation of deontic logic. For the deontic operations to be useful with our model, we first had to implement the parts of the model on which the final ranking function was dependent. With

those in place, the implementation can be further extended with deontic operations and patient preferences.

Combining information ranking with access control is not only feasible, but also efficient. With multiple hierarchies of roles and information classes, policies for access and information ranking may be flexibly encoded. A few rules placed in general roles apply to all their descendants, and the descendant roles only need rules about how they differ from their ancestors. With the exception of patient preferences, all rules about how information is shown are collected in one place, namely in the role.

When fully implemented, a hybrid role model would be an effective means for granting access to and ranking of healthcare information, preferably assisted by a traditional decision support system.

Appendix A
References

[Bay02] Elisabeth Bayegan. *Knowledge Representation for Relevance Ranking of Patient-Record Contents in Primary-Care Situations*. EngD thesis. Norwegian University of Science and Technology, 2002. This thesis contains several separately published articles, [BØNG01] among them. 30, 35

[BBF01] Elisa Bertino, Piero Andrea Bonatti, and Elena Ferrari. Trbac: A temporal role-based access control model. *ACM Trans. Inf. Syst. Secur.*, 4(3):191–233, 2001. 23, 24, 54

[BMY02] Jean Bacon, Ken Moody, and Walt Yao. A model of oasis role-based access control and its support for active security. *ACM Trans. Inf. Syst. Secur.*, 5(4):492–540, 2002. 19, 24

[BS00] Ezedin Barka and Ravi Sandhu. A role-based delegation model and some extensions. In *Proceedings of 23rd National Information System Security Conference*, pages 89–100, 2000. 20

[BØNG01] Elisabeth Bayegan, Øystein Nytrø, and Anders Grimsmo. Ranking of information in the computerized problem-oriented patient record. In Vimla L. Patel, Ray Rogers, and Haux Reinhold, editors, *Proceedings of the 10th World Congress on Medical Informatics (MEDINFO2001)*. IOP Press, 2001. 30, 35, 101

[CM03] William F. Clocksin and Christopher .S. Mellish. *Programming in Prolog*. Springer, 2003. 67

[Coy96] Edward J. Coyne. Role engineering. In *RBAC '95: Proceedings of the first ACM Workshop on Role-based access control*, page 4, New York, NY, USA, 1996. ACM Press. 38

[Cra03] Jason Crampton. Constraints in role-based access control(foil-set). http://www.isg.rhul.ac.uk/~jason, June 2003. 12

[FCK95] David Ferraiolo, Janet Cugini, and Richard Kuhn. Role-based access control. http://hissa.ncsl.nist.gov, November 1995. 10

APPENDIX A
REFERENCES

[FKC03] David F Ferraiolo, Richard Kuhn, and Ramaswamy Chandramouli. *Role-based Access Control*. Artech House Publishers, April 2003. 9, 22

[Fra03] Steve Frame. Role-based access control. http://www.giac.org, November 2003. 9

[FSG+01] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, Richard Kuhn, and Ramaswamy Chandramouli. Proposed nist standard for role-based access control. http://www.nist.org, August 2001. xi, 12, 14, 15, 18, 19, 47

[Gol99] Dieter Gollmann. *Computer Security*. Worldwide series in computer science. Wiley, Chichester, 1999. 9

[Gua04] Wei Guan. Improvement on role based access control model. http://www.ir.iit.edu/~ophir/cs529/slides/wei-proposal.pdf, 2004. 35

[Hd02] Helsedepartementet. Lov om elektronisk signering. http://www.lovdata.no, December 2002. 26

[Hd03a] Helsedepartementet. Forskrift om pasientjournal. http://www.lovdata.no, February 2003. 26

[Hd03b] Helsedepartementet. Helsepersonelloven. http://www.lovdata.no, August 2003. 26

[Hd03c] Helsedepartementet. Helseregisterloven. http://www.lovdata.no, August 2003. 26

[Hd03d] Helsedepartementet. Lov om psykisk helsevern. http://www.lovdata.no, August 2003. 26

[Hd03e] Helsedepartementet. Pasientrettighetsloven. http://www.lovdata.no, December 2003. 26

[Hd03f] Helsedepartementet. Spesialisthelsetjenesteloven. http://www.lovdata.no, December 2003. 26

[Hd04] Helsedepartementet. Helseforetaksloven. http://www.lovdata.no, May 2004. 26

[JBGL01] James B D Joshi, Elisa Bertino, Arif Ghafoor, and U Latif. Generalized temporal role based access control model (gtrbac) (part i)- specification and modeling. In *Technical Report CERIAS TR 2001-47*, 2001. 23, 24

[Kie99] David Kieras. A guide to goms model usability evaluation using gomsl and glean3, 1999. 30

[Kol02] Grezwgorz Kolaczek. Application of deontic logic in role-based access control. *Int. J. Appl. Math. Comput. Sci.*, 12(2):269–275, 2002. 5, 8

[LN99] John Linn and Magnus Nyström. Attribute certification: an enabling technology for delegation and role-based controls in distributed environments. In *RBAC '99: Proceedings of the fourth ACM workshop on Role-based access control*, pages 121–130, New York, NY, USA, 1999. ACM Press. 20

[Nys01] Torbjørn Nystadnes. Elektronisk pasientjournal standard. http://www.kith.no/epj_undermapper/19547/, June 2001. 26, 103

[Nys05] Torbjørn Nystadnes. Elektronisk pasientjournal standard, 2005. Draft of revised [Nys01]. 3, 26, 33, 55

[okd01] Kultur og kirke departementet. Arkivloven. http://www.lovdata.no, May 2001. 26

[opd00] Justis og politi departementet. Personopplysningsloven. http://www.lovdata.no, April 2000. 26

[PC03] Olga Pacheco and José Camaro. A role based model for the normative specification of organized collective agency and agents interaction. *Autonomous Agents and Multi-Agent Systems*, 6:145–184, 2003. 6

[PHS03] Josef Pieprzyk, Thomas Hardjono, and Jennifer Seberry. *Fundamentals of computer security*. Springer, Berlin, 2003. 9

[Pro03] The Kvalis Project. Kvalis webpage. http://kvalis.ntnu.no/, 1999-2003. 25

[PS04] Olga Pacheco and Filipe Santos. Delegation in a role-based organization. In Alessio Lomuscio and Donald Nute, editors, *DEON*, volume 3065 of *Lecture Notes in Computer Science*, pages 209–227. Springer, 2004. 5, 8, 20

[RC99] Anthony. Rhodes and William. Caelli. A review paper role based access control. http://www.isrc.qut.edu.au, 1999. 10

[San01] Ravi S. Sandhu. Future directions in role-based access control models. In *MMM-ACNS '01: Proceedings of the International Workshop on Information Assurance in Computer Networks*, pages 22–26. Springer-Verlag, 2001. 9

[SSW94] Leon Sterling, Ehud Shapiro, and David H.D. Warren. *The art of Prolog : advanced programming techniques*. MIT Press, 1994. 67

[SWI03] Swi prolog 5.4.7. http://www.swi-prolog.org, August 2003. Licensed under various Free and Open source licenses. 67

[SØ04] Bjørn-Erik Stenbakk and Gunnar René Øie. Role models in healthcare. http://gunnarre.nvg.ntnu.no/studies/rmhc/rmhcStenbakkOie.pdf, November 2004. Specialization project leading up to this thesis. 9, 29, 31, 33, 47, 57, 58, 64, 75, 88, 91, 92, 93

[Tho97] Roshan K. Thomas. Team-based access control (tmac): a primitive for applying role-based access controls in collaborative environments. In *RBAC '97: Proceedings of the second ACM workshop on Role-based access control*, pages 13–19, New York, NY, USA, 1997. ACM Press. 37

[Vis05] Visual prolog 6.1. http://www.visual-prolog.com/vip6/product/, August 2005. No cost for academic and non-commercial work (personal edition). 67

[ZAC01] Longhua Zhang, Gail-Joon Ahn, and Bei-Tseng Chu. A rule-based framework for role based delegation. In *SACMAT '01: Proceedings of the sixth ACM symposium on Access control models and technologies*, pages 153–162, New York, NY, USA, 2001. ACM Press. 20, 22

[ZAC02] Longhua Zhang, Gail-Joon Ahn, and Bei-Tseng Chu. A role-based delegation framework for healthcare information systems. In *SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 125–134, New York, NY, USA, 2002. ACM Press. 19

Appendix B

Glossary

ACL Access Control List: Here a patient access preferences

Cardiology: A branch of medicine studying the heart.

CAVE: a caveat, a warning, from *cavere* (Latin).

CPR: Computerized Patient Record

Construction role: A role that preforms as a building block for a functional role

DAC: Discretionary Access Control

DSD: Dynamic Separation of Duty

Electronic patient record/Electronic health record (EPR/EHR): A set of information about patients and their treatment

Epicrisis: Discharge letter

ER: Emergency Room

Functional role: The active role a user requires in a session.

GP: General Practitioner

HCO: Healthcare organization

Health care authority: An enterprise in the healthcare sector.

ICU: Intensive Care Unit

Intern: A recently graduated physician practicing medicine under supervision.

Internist: A physician specializing in internal medicine

ICD-10: International Classification of Diseases A systematic classification of all diseases. Used primarily in hospitals.

LGPL: { Lesser | Library } GNU Public License.
See http://www.fsf.org/licensing or http://creativecommons.org for more information.

MAC: Mandatory Access Control

Pediatrician: A physician who specializes in treating infants and children up to adolescence.

RBAC: Role-Based Access Control

SoD: Separation of Duty

SSD: Static Separation of Duty

Subject: Something or somebody attempting to access information.

Urology: The study and treatment of diseases that affect the urinary system.

User: A human user of a computer system. A human Subject.

Appendix C

Detailed Information From the Scenario

This appendix contains some of the detailed information from the scenario, described in section 5.1. Patient data is in section sec:appcasedata. (The class and role hierarchies are found in chapter 8.1.3.) The Prolog-encoded facts are found in section C.2.

C.1 Patient Data

Existing problem: Diabetes mellitus.
Existing problem: Hypertension.

Existing med: Insulin.
Existing med: Beta blocker.
Existing med: Tiazide.

Current problem: Hypoglycemia.
Current problem: Syncope (fainting).
Current problem: Trauma to head.

Location 1: The emergency room (ER).

Intern prescribes a medicine to stabilize the blood sugar.
Intern requests a CT, because the patient has suffered from a head trauma when passing out, because of the hypoglycemia.

After having stabilized Elisa she is sent to radiology for CT and then returned to the medical ward.

Location 2: Internal medicine ward.

Internist prescribe a new dosage of insulin.
Nurse who deliver insulin dosage.
Nurse who also keeps track of the blood sugar and blood pressure of Elisa.

Current problem: Heart attack

Location 3: Cardiology unit

Cardiology team takes over the care of the patient.

Location 4: Internal medicine ward.

Discharged
Told to see his/her primary care doctor for follow-up care.

An involved doctor dictated discharge papers.
Secretary writes discharge papers.
Same involved doctor signed discharge papers.
Senior resident counter signed discharge papers.

C.2 Case Code (Facts)

```
/* Regular facts CASE (defined in accordance with our model, and the
case described in the thesis. Used to test the model*/

/*Users*/

user('Roger').
user('Billy').
user('Betty').
user('Ben'). /* The radiologist */
user('Alice').
user('Bob').

/*userAssignedRoles*/
userAssignedRole('Elisa', []).
userAssignedRole('Roger', [7]).
userAssignedRole('Billy', [10]).

/* userAssignedRole('Billy', [10,3]).*/ /* Violate SSD */
```

```
userAssignedRole('Betty', [5]).
userAssignedRole('Ben', [9]).
userAssignedRole('Alice', [8]).
userAssignedRole('Bob', [3]).

userAssignedRole(_, [101,102,103,104,105]).

/* Information objects */

/* obj(objectid, classid, patientid, information) */

obj(1, 24, 1,'diabetes mellitus').
obj(2, 24, 1,'insulin').
obj(3, 24, 1, 'hypertension').
obj(4, 21, 1, 'beta blocker').
obj(5, 21, 1, 'tiazide').

obj(6, 28, 1, 'hypoglycemia').
obj(7, 29, 1, 'syncope (fainting)').
obj(8, 28, 1, 'trauma to head').

obj(9, 17, 1, image('ct','MMUgzvzxAHc')).
obj(10, 18, 1, 'glucose solution').

obj(11, 26, 1, 'insulin').
/* obj(12, 10, 1, test('blood glucose',)). Skal skrives..*/
/* obj(13, 10, 1, test('blood pressure',,)). */

obj(14, 28, 1, 'heart attack').

/* obj(, , , ''). */

obj(20, 7, 1, 'Elisa Eliasen').
obj(21, 8, 1, 'Metroville').
obj(22, 9, 1, '078-05-1120').

/*Roles*/

/* Staff */
role(1,'Staff').
role(2,'Pharmacist').
role(3,'Secretary').
role(4,'Medical practitioner').
role(5,'Nurse').
role(6,'Psychiatrist').

role(7,'Intern'). /* Perhaps belongs in a different hierarchy, like Resident,
On trial, etc. with regards to countersigning */

role(8,'Cardiologist').
role(9,'Radiologist').
role(10,'Internist').

/* Location */
role(101,'Hospital').
role(102,'ER').
role(103,'ICU').
role(104,'Cardiology ward').
role(105,'Internal medicine').

/*roleparent (Child, Parent)*/
roleparent(2,1).
roleparent(3,1).
roleparent(4,1).
roleparent(5,1).
```

```
roleparent(6,4).
roleparent(7,4).
roleparent(8,4).
roleparent(9,4).
roleparent(10,4).

roleparent(102,101).
roleparent(103,101).
roleparent(104,101).
roleparent(105,101).

/*SSD*/
/*ssd(RoleSet,N)*/
:- dynamic ssd/2.
ssd([3,4],2).

/*DSD*/
/* dsd(RoleSet,N)*/
:- dynamic dsd/2.

dsd([101,102,103,104,105],2).

/*Information classes*/
/* infoclass(classid,classname) */
infoclass(1, 'Clinical Information').

infoclass(2, 'Personalia').
infoclass(3, 'TestResults').
infoclass(4, 'CAVE').
infoclass(5, 'Medical history').
infoclass(6, 'Current').

infoclass(7, 'Name').
infoclass(8, 'Adress').
infoclass(9, 'SocialSecurityNumber').

infoclass(10, 'Clinical').
infoclass(11, 'Imaging').
infoclass(12, 'Laboratory').

infoclass(13, 'Historic Treatment').
infoclass(14, 'Historic Problem').
infoclass(15, 'Current Treatment').
infoclass(16, 'Current Problem').


infoclass(17, 'CT').
infoclass(18, 'X-ray').
infoclass(19, 'Blood sample').
infoclass(20, 'Urine sample').

infoclass(21, 'Historic Drug treatment').
infoclass(22, 'Historic Therapy').
infoclass(23, 'Historic Problem date').

infoclass(24, 'Diagnosis').
infoclass(25, 'Symptom').

infoclass(26, 'Drug treatment').
infoclass(27, 'Therapy').

infoclass(28, 'CurrentDiagnosis').
infoclass(29, 'CurrentSymptoms').
```

```
/*Information hierarchy objectParent(Child, Parent)*/
objectParent(2,1).
objectParent(3,1).
objectParent(4,1).
objectParent(5,1).
objectParent(6,1).

objectParent(7,2).
objectParent(8,2).
objectParent(9,2).

objectParent(10,3).
objectParent(11,3).
objectParent(12,3).

objectParent(13,5).
objectParent(14,5).

objectParent(15,6).
objectParent(16,6).

objectParent(17,11).
objectParent(18,11).

objectParent(19,12).
objectParent(20,12).

objectParent(21,13).
objectParent(22,13).

objectParent(23,14).
objectParent(24,14).
objectParent(25,14).

objectParent(26,15).
objectParent(27,15).

objectParent(28,16).
objectParent(29,16).


/*operations*/
operation(1, 'create').
operation(2, 'read').
operation(3, 'write').
operation(4, 'approve').
operation(5, 'invalidate').
operation(6, 'correct').


/* Rules for roles

Rules about aclass can both be deined in a single rule
cor(CORID,crule(accrank(REL,DEL,PRIV),CLASS))

or individual rules about ranking or class*/

/* A combined construction role that puts single assignments together.
Remember that permissions really are lists.*/
/*cor(CORID,crule(accrank(REL,DEL,PRIV),CLASS)).*/

cor(CORID,crule(accrank(REL,DEL,PRIV),CLASS)) :-
roleAssignedPermission(CORID,CLASS,PRIV),
roleAssignedRelevance(CORID,CLASS,REL),
roleAssignedDetail(CORID,CLASS,DEL).
```

```
/* Staff */
roleAssignedPermission(1,7,[2]).
roleAssignedRelevance(1,7,1).
roleAssignedDetail(1,7,1).

roleAssignedPermission(1,9,[2]).
roleAssignedRelevance(1,9,1).
roleAssignedDetail(1,9,1).

roleAssignedPermission(1,4,[2]).
roleAssignedRelevance(1,4,1).
roleAssignedDetail(1,4,1).

/* Medical practitioner */
roleAssignedPermission(4,5,[2]).
roleAssignedRelevance(4,5,3).
roleAssignedDetail(4,5,2).

roleAssignedPermission(4,6,[2]).
roleAssignedRelevance(4,6,4).
roleAssignedDetail(4,6,4).

roleAssignedPermission(4,4,[2]).
roleAssignedRelevance(4,4,4).
roleAssignedDetail(4,4,2).

/* Internist */
cor(10,crule(accrank(5,5,[2]),19)).
cor(10,crule(accrank(3,6,[1,2,3]),26)).

/* Nurse */
roleAssignedPermission(5,26,[2]).
roleAssignedRelevance(5,26,4).
roleAssignedDetail(5,26,1).

/* Radiologist */
roleAssignedPermission(9,11,[1,2,3,4]).
roleAssignedRelevance(9,11,5).
roleAssignedDetail(9,11,6).

/* Secretary */
roleAssignedPermission(3,2,[2]).
roleAssignedRelevance(3,2,4).
roleAssignedDetail(3,2,5).

/* Location role rules */

/* ER */
roleAssignedPermission(102,4,[2]).
roleAssignedRelevance(102,4,6).
roleAssignedDetail(102,4,6).

/* ICU */
roleAssignedPermission(103,4,[2]).
roleAssignedRelevance(103,4,5).
roleAssignedDetail(103,4,5).
```

Appendix D

Prolog Code

This appendix contains a listing of the code in the model implementation, described in chapter 7. The code may also be downloaded from the thesis homepage at http://gunnarre.nvg.org/studies/rmhc.

D.1 Model Code

```
/* A role model for ranking and access control.
Copyright ©2004-2005  Bjørn-Erik Stenbakk and Gunnar René Øie
All rights reserved.

http://gunnarre.nvg.org/studies/rmhc/

Made for SWI prolog. Some of the predicates in the SWI prolog library is used.
See http://www.swi-prolog.org for licensing


In the comments in the following code examples, argument variables preceeded by a
'+', '-' or '?', in a style similar to the documentation for SWI Prolog. These
indicate how the arguments should be used. '+' denotes an input argument, '-'
denotes an output argument, while '?' denotes an argument that may be either input
to or output from the predicate.

*/

/**********
 * Setup  *
 **********/

/*
Setup rules that may be manually run.

set_prolog_flag(toplevel_print_options,[quoted(true), portray(true),
attributes(portray), max_depth(20)]).
*/


/****************************************
 * Genereic rules used in the other rules  *
```

```
 ****************************************/
 /* List and graph search rules etc.*/

/* max(+X,+Y,-MAX) */
max(X,Y,X) :- X >= Y.
max(_,Y,Y).



/****************************************
 * Model rules apply to all organizations and policies*
 ****************************************/
 /* The main focus of our assignment.*/

/* Creating a "functional" role

Post-condition:
A set of rules for this composite role.
No rules about the exact same class.
But overlapping hierarchies of object classes allowed.

Build a list of all rules that apply to this role.
If two rules concern the exact same class or object, then use
max of REL and DEL, and add privileges.

Then, rewrite the thing for deontic operation.

*/

/*
fur_nonhier(+CONSTLIST, -FURULEs) :-
The functional role fur is made up of CONSTLIST, which is a list of the
construction roles that took part in building the fur, and FURULEs, which is a
flat list of rules.
This version does not take role hierarchies into account.
*/

fur_nonhier(CONSTLIST, FURULEs) :-
dsd_ok(CONSTLIST),/* Check DSD */
crulesinconstlist_nonhier(CONSTLIST, COR_CRULES),
/* Make a list of lists of CRULEs */
flatten(COR_CRULES,CRULEs), /* Flatten the list of CRULES */
learnallcrules(CRULEs), /* Learn/resolve rules from the roles */
  collect_furules([],FURULEs). /* Collect the furules into a list.*/


/*
fur(+CONSTLIST, -FURULEs) :-
The functional role fur is made up of CONSTLIST, which is a list of the
construction roles that took part in building the fur, and FURULEs, which
is a flat list of rules.
This version uses role hierarchies.
*/

fur(CONSTLIST, FURULEs) :-
dsd_ok(CONSTLIST),/* Check DSD */
crulesinconstlist_hier(CONSTLIST, COR_CRULES),
/* Make a list of lists of CRULEs*/
flatten(COR_CRULES,CRULEs), /* Flatten the list of CRULES */
learnallcrules(CRULEs), /* Learn/resolve rules from the roles */
  collect_furules([],FURULEs). /* Collect the furules into a list.*/


crulesincor_nonhier(CORID,CRULEs) :- findall(CRULE,cor(CORID,CRULE),CRULEs).
```

```
crulesincor_hier(CORID,CRULEs) :-
findall(CRULE,(dirindir(CORID,D),cor(D,CRULE)),CRULEs).

/*
learnallcrules(+CRULEs) :-
Given a list of CRULEs, send them all to the predicate "learncrule".
*/
learnallcrules([]).

learnallcrules([CRULE|CRULEsTail]) :-
learncrule(CRULE),
!,
learnallcrules(CRULEsTail).

/*

crulesinconstlist_nonhier(+CONSTLIST, -COR_CRULES) :-
The crulesinconstlist is made up of CONSTLIST, which is a list of the
construction roles that took part in building this particular fur, and
COR_CRULES, which is a list of lists of crules. [[CRULESs],[CRULEs]]
*/

crulesinconstlist_nonhier([CORID|CONSTLIST2], [CRULES|COR_CRULES2]) :- %Listbuild
crulesincor_nonhier(CORID,CRULES),
crulesinconstlist_nonhier(CONSTLIST2, COR_CRULES2).


crulesinconstlist_nonhier([CORID], [CRULES]) :- %Limit
crulesincor_nonhier(CORID,CRULES).


/*

The name of this predicate might change.

crulesinconstlist_hier(+CONSTLIST, -COR_CRULES) :-
The crulesinconstlist is made up of CONSTLIST, which is a list of the
construction roles that took part in building this particular fur, and
COR_CRULES, which is a list of lists of crules. [[CRULESs],[CRULEs]]
*/

crulesinconstlist_hier([CORID|CONSTLIST2], [CRULES|COR_CRULES2]) :- %List builder
crulesincor_hier(CORID,CRULES),
crulesinconstlist_hier(CONSTLIST2, COR_CRULES2).


crulesinconstlist_hier([CORID], [CRULES]) :- %Limit
crulesincor_hier(CORID,CRULES).


/*

learncrule(crule(accrank(+REL,+DEL,+PRIV),+CLASS)) :-
Remember the rankings for CLASS in a FURULE in the Prolog database. If CLASS
has been present in a previousely considered CRULE, then retract the old
FURULE, and make a new FURULE with the appropriate ranking and permissions..

*/

:- dynamic furule/2.

learncrule(crule(accrank(REL,DEL,PRIV),CLASS)) :-
retract(furule(accrank(OLDREL,OLDDEL,OLDPRIV),CLASS)),
/* A furule about this class allready existed, and it is retracted to be
replaced with a new one*/
```

```
max(REL,OLDREL,NEWREL),
max(DEL,OLDDEL,NEWDEL),
addperm(PRIV,OLDPRIV,NEWPRIV),
asserta(furule(accrank(NEWREL,NEWDEL,NEWPRIV),CLASS)).

learncrule(crule(accrank(REL,DEL,PRIV),CLASS)) :-
asserta(furule(accrank(REL,DEL,PRIV),CLASS)).
/* A furule about this class did not exist. It was asserted.*/

/* addperm(NEW,OLDPRIVS,RESULT) :-
Add the NEW permission(s) to the existing permissions.
Predicate used by learncrule.*/

addperm(OP,OLDPRIVS,RESULT) :-
atom(OP),
union([OP],OLDPRIVS,RESULT).

addperm(OP,OLDPRIVS,RESULT) :-
number(OP),
union([OP],OLDPRIVS,RESULT).

addperm(NEWLIST,OLDPRIVS,RESULT) :-
union(NEWLIST,OLDPRIVS,RESULT).

/* collect_furules(?Templist,-Result) :-
Collect all the FURULEs that have been asserted for this functional role,
and put them all in the list Result.*/

collect_furules(Templist,Result):-
FURULE=furule(_,_),
retract(FURULE),
!,
collect_furules([FURULE|Templist],Result).

collect_furules(Result,Result).

/* Ranking an individual object

Post-condition:
For each object, a single accrank.
Precedence to
- forbid (vs. permit).
- more detailed object
- more detailed role
*/


/* ranknou_nonhier(+FURULEs,?OBJECTID,-ACCRANK) :-
ranking without knowing who the user is, and without hierarchy*/
ranknou_nonhier(FURULEs,OBJECTID,ACCRANK) :-
obj(OBJECTID,CLASS,_,_), /* Get the direct class */
member(furule(ACCRANK,CLASS),FURULEs). /* Find the class among the furules,
and pick the accrank from that furule.*/

/* ranknou(+FURULEs,+OBJECTID,-ACCRANK) :-
ranking without knowing who the user is*/

ranknou(FURULEs,OBJECTID,ACCRANK) :-
obj(OBJECTID,CLASS,_,_), /* Get the direct class */
rankrecursive(FURULEs,CLASS,ACCRANK). /* Follow the hierarchy
to find the class closest to the direct class. */


/* try to find in this class */
rankrecursive(FURULEs,CLASS,ACCRANK):- /* Singleton CONSTLIST */
```

```
member(furule(ACCRANK,CLASS),FURULEs),!. /* Find the class among the furules,
and pick the accrank from that furule.*/ /* Red cut */


/* find parent class, and call recursively */
rankrecursive(FURULEs,CLASS,ACCRANK):-
objectParent(CLASS,PARENT),
rankrecursive(FURULEs,PARENT,ACCRANK).
/* Find the closest parent among the furules,
and pick the accrank from that furule.*/



/* rank(+USER,+SESSION,?OBJ,-ACCRANK) :-
Actual ranking with a choice of rules to use.*/

rank(USER,SESSION,OBJ,ACCRANK) :-
ssd_ok(USER),
roles_available(USER,SESSION),
fur(SESSION,FURULEs),!,
OBJ=obj(OBJECTID,_,_,_),
ranknou(FURULEs,OBJECTID,ACCRANK).

rank_nonclasshier(USER,SESSION,OBJ,ACCRANK) :-
ssd_ok(USER),
roles_available(USER,SESSION),
 fur(SESSION,FURULEs),!,
OBJ=obj(OBJECTID,_,_,_),
ranknou_nonhier(FURULEs,OBJECTID,ACCRANK).

/* dirindir(?Direct, ?DirIndir)
Given a Direct role return DirIndir: Is this the Direct role or
one of its ancestors?*/
dirindir(Direct, DirIndir) :-
Direct=DirIndir.

dirindir(Direct, DirIndir) :-
ancestor(Direct,DirIndir).

/* ancestor(?ChildID, ?AncestorID) :-
Checing for ancestor for roles. ChildID and AncestorID are of the same
type as the arguments to the structure roleparent. */
ancestor(ChildID, AncestorID) :-
    roleparent(ChildID, AncestorID).

ancestor(ChildID, AncestorID) :-
    roleparent(ChildID, A1),
    ancestor(A1, AncestorID).


/*Checing for ancestor for information objects*/
       ancestorInfo(ChildID, AncestorID) :-
            objectParent(ChildID, AncestorID).
       ancestorInfo(ChildID, AncestorID) :-
            objectParent(ChildID, A1),
            ancestorInfo(A1, AncestorID).

/* Checking for roles available*/

roles_available(USER,SESSION) :-
findall(ROLEsAssigned,userAssignedRole(USER, ROLEsAssigned),Nonflat),
flatten(Nonflat,ROLEsAvailable),
subset(SESSION, ROLEsAvailable),!.
```

```
roles_available(_,_) :- !,write('Roles not available for activation!\n'),fail.

/*Implementing dynamic seperation of duties*/


    /* ssd_ok(U) :-
        ssd_ok if the User has no SSD conflicts
       WARNING: TODO: SSD should be checked when assigning roles,
       not when ranking.
       WARNING: Will fail on first violation.*/
    ssd_ok(User) :-
    not((ssd(RS,N),
     findall(DirIndirRole,(rolemember(User,DirectRole),
     dirindir(DirectRole, DirIndirRole)),AvailRoleList),
     list_to_set(AvailRoleList,AvailRoleSet),
     intersection(RS,AvailRoleSet,T),
     length(T,TLength),
     TLength >= N,
     write('Static Separation of Duty violation!\n'))).

rolemember(User,Role):-
userAssignedRole(User, USERROLEs) ,
member(Role,USERROLEs).

/* dsd_ok(+SESSION) :-
       dsd_ok if the session has no SSD conflicts
       WARNING: Will fail on first violation.*/
dsd_ok(SESSION) :-
     not((dsd(RS,N),
     intersection(RS,SESSION,T),
     length(T,TLength),
     TLength >= N,
     write('Dynamic Separation of Duty violation!\n'))).



    /* shown(+USER,+SESSION,+MINREL,?OBJ,-DEL) :-
    Given a user USER that has set a preferred minimum relevance MINREL in
    the user interface,
        what OBJects are shown, and with what DEL (detail level)*/
    shown(USER,SESSION,MINREL,OBJ,DEL) :-
     rank(USER,SESSION,OBJ,accrank(REL,DEL,PRIV)),
        member(2,PRIV),
        REL >= MINREL.




/************************************
 *  Policy rules defined for our case              *
 ************************************/

/************************************
 *  Test rules just for testing...........               *
 ************************************/


       reconsult(FileName) :-
            retractall(caseDB),
            consult(FileName).
```