Authorization schema for electronic health-care records

For Uganda

ALEXIS MARTÌNEZ FERNÁNDEZ

KTH Information and
Communication Technology

Degree project in
Communication Systems
Second level, 30.0 HEC
Stockholm, Sweden

Authorization schema for electronic health-care records: For Uganda

Alexis Martínez Fernández

Master thesis

23 August 2012

Examiner & Supervisor: Gerald Q. Maguire Jr.

KTH Royal Institute of Technology
School of Information and Communication Technology
Stockholm, Sweden

Abstract

This master's thesis project began at the Karolinska University Hospital. This thesis discusses how to design an authorization schema focused on ensuring each patient's data privacy within a hospital information system. It begins with an overview of the current problem, followed by a review of related work. The overall project's goal is to create and evaluate an authorization schema that can ensure each patient's data confidentiality.

Authorization has currently become a very important aspect in information systems, to the point of being a necessity when implementing a complete system for managing access control in certain complex environments. This requirement lead to the approach that this master thesis takes for effectively reasoning about authorization requests in situations where a great number of parameters could affect the access control assessment.

This study is part of the ICT4MPOWER project developed in Sweden by both public and private organizations with the objective of improving health-care aid in Uganda through the use of information and communication technologies.

More concretely, this work defines an authorization schema that can cope with the increasing needs of sophisticated access control methods where a complex environment exists and policies require certain flexibility.

Sammanfattning

Detta examensarbete projektet startade vid Karolinska Universitetssjukhuset. Denna avhandling diskuterar hur man designar ett tillstånd schema fokuserat på att säkerställa varje patients dataskydd inom ett sjukhus informationssystem. Det börjar med en översikt över det aktuella problemet, följt av en genomgång av arbete. Projektets övergripande mål är att skapa och utvärdera ett tillstånd schema som kan garantera varje patient data sekretess.

Bemyndigande har för närvarande blivit en mycket viktig aspekt i informationssystem, till den grad att vara nödvändigt att genomföra komplett system för hantering av åtkomstkontroll i vissa komplexa miljöer. Detta är i själva verket den strategi som detta examensarbete tar för att effektivt resonemang om en ansökan om godkännande i situationer där ett stort antal parametrar kan påverka i åtkomstkontroll bedömningen.

Denna studie är en del av ICT4MPOWER projektet utvecklades i Sverige av både offentliga och privata organisationer i syfte att förbättra stödet sjukvård i Uganda med användning av informations-och kommunikationsteknik.

Mer konkret definierar detta arbete ett tillstånd schema som kan hantera de ökande behoven av sofistikerade metoder för åtkomstkontroll där en komplex miljö finns och politik kräver en viss flexibilitet.

Acknowledgements

I would like to start thanking my supervisor Gerald Q. Maguire, who supported me throughout the completion of this master thesis work and gave me his opinion and advice when I needed it. He also helped me in improving my report and providing me additional information that could enrich my research quality.

Furthermore, it has been really interesting to be involved in such complex project because it has truly given me a good skillset in problem solving and reasoning. In addition to this, I was forced to deeply delve into quite sophisticated technologies that may be useful in my professional future when facing similar difficulties.

I would like to finish by expressing my gratitude to the colleagues that were working on their thesis projects at the Communication Systems Department along with me. Also, I would like to mention the people from Karolinska that collaborated in this project, as their commitment was necessary for initiating and developing this project - which may really make a difference to someone's life.

Table of Contents

List of Figures

List of Tables

List of acronyms and abbreviations

ACMM	Access Control Matrix Model
ANSI	American National Standards Institute
AOP	Aspect Oriented Programming
ASTM	American Society for Testing and Materials
BTG	Break the glass
C-RBAC	Constrained Role-based Access Control
CCF	Care Comes First
CCHIT	Certification Commission for Healthcare Information
CDA	Clinical Document Architecture
CEN TC	Comité Européen de Normalization – Technical Committee
COTS	Commercial off-the-shelf
DAC	Discretionary Access Control
DHIS	District Health Information Software
DSD	Dynamic Separation of Duty
DSTU	Draft Standards for Trial Use
DTR	Draft Technical Report
EHCR	Electronic Health Care Record
EHR	Electronic Health Record
F-RBAC	Flat Role-based Access Control
GUI	Graphical User Interface
H-RBAC	Hierarchical Role-based Access Control
HIPAA	The Health Insurance Portability and Accountability Act of 1996
HIS	Hospital Information System
HISP	Health Information Systems Programme
HL7	Health Level 7
ICT	Information and Communication Technologies
ISO	International Standards Organization
MAC	Mandatory Access Control
NIST	National Institute of Standards and Technology
PMI	Privilege Management Infrastructure
POLA	Principle Of Least Authority
POLP	Principle Of Least Privilege
RBAC	Role-based Access Control
RBAC	Role-based Access Control
RIM	Reference Information Model
S-RBAC	Symmetric Role-based Access Control
SOD	Separation Of Duty
SSD	Static Separation of Duty
TM	Trust Management
UDS	Unified Data Schema
US	United States
USA	United States of America
XML	Extensible Markup Language
EIS	Enterprise Information System

MVC	Model-View-Controller
W3C	World Wide Web Consortium

1 Introduction

This chapter presents general information about this thesis project. The chapter starts by stating the problem's definition and continues with an overview of the project's context and proposed objectives. Finally, a short description of each of the subsequent chapters of this thesis is given.

1.1 Problem statement

The problem that this thesis project addresses is *how to ensure each patient's data privacy within a hospital information system*. The approach that will be used is to define an authorization schema which will be used to control who can access what patient's electronic health-care record (EHCR), when they can access this record, what access rights they have to this record, and how this access should be logged in order to create an audit trail.

1.2 Project overview

Many organizations in developed countries are currently working with projects that are designed to improve the living standards in developing countries. In the context of this thesis project, both private and public institutions and companies in Sweden are cooperating with several Ugandan governmental and educational organizations. This combination of actors is collaborating under an agreement with the government of Uganda to improve health-care system by using information and communication technologies (ICT).

The overall project is called *ICT4MPOWER.* This project aims to establish:

- The necessary infrastructure, such as networks, hardware, and electricity supply
- An e-health record management system
- A patient identification system
- An electronic patient referral and feedback system
- A mechanism to support remote-consultation
- A national drug and stock management system
- A human resource development system

The scope of the *ICT4MPOWER* project is the Isingiro district mainly because of its low health-care indices. However, if this project succeeds, it could be rolled out to the other 80 districts of Uganda. This master's thesis project began at the Karolinska University Hospital in Huddinge with a multidisciplinary team, including both health care and ICT professionals, from the Clinical Research Area situated at Novum Research Park.

1.2 Thesis overview

Chapter 2 provides some basic background information about hospital information systems, electronic health-care records, access control systems, and authorization schemes. It also summarizes some of the legal issues that must be addressed. Chapter 3 describes the method that will be used in this thesis project to solve the problem stated in section 1.1. Chapter 4 analyzes how well the proposed solution solves the problem. Finally Chapter 5 presents some conclusions and suggests future work.

1.3 Uncovered topics

This thesis specifically focuses on *data privacy of electronic health-care records within a hospital information system from* ***an authorization point of view***, therefore other aspects such as authentication, security protocols used for information transmission, system architecture, and hardware resources are not deeply discussed in this document. More concretely, this document explains, ***from a software point of view****, how to build an authorization system for a complex environment*, therefore issues that do **not** have a strong relationship with software engineering or software architecture are barely present in this thesis.

2 Background

This chapter provides some basic background information about hospital information systems, electronic health-care records, access control systems, and authorization schemas. The chapter describes related work which illustrates the state of the art in access control and authorization schemas, especially those applied to electronic health-care records. The chapter ends with a summary of some of the legal issues that an access control system for electronic health-care records must address, hence the authorization schema must capture these legal rules so that these rules will be properly (and automatically) enforced by the access control system.

2.1 Hospital Information Systems

This section is intended to give the reader a basic understanding of Hospital Information Systems. We begin with a general overview, the focus on what is special about a Hospital Information Systems in developing countries. Finally some conclusions are drawn about introducing such a system in a developing country.

2.1.1 Introduction

A Hospital Information System (HIS) is a computer-based system that manages a hospital's medical and administrative data and allows health-care professionals to carry out their tasks more effectively[1]. A HIS is generally a complex system composed of modules, as shown in Figure 2.1. Moreover, a wide range of software components could be part of a HIS. The emergence of information exchange standards, such as HL7 and DICOM, in the industry has made integration and administration of the integrated set of subsystems of the HIS much easier[1].

Usually, health-care information systems are geographically distributed over a certain area or region. These systems support different organizational levels of health-care management. Information systems situated in the individual medical facilities frequently function as a federation of autonomous systems. That means, they individually perform a specific task and collaborate, based upon conformity with standards and try to ensure data consistency, with other independent components inside a single functional unit that corresponds to the HIS. Furthermore, communication between the different modules is a key issue because there are many different medical facilities and a wide diversity of health-care units.

In [2], Fedele and Srl identify three different goals that an effective HIS should fulfill. First, it must correctly support individual medical facilities' activities, while interconnecting these individual facilities into a single functional unit. Second, the HIS should facilitate the users' daily tasks by exploiting information technology. Third, the HIS should provide a means for system evolution and distributed computing in order to take advantage of emerging health-care scenarios.

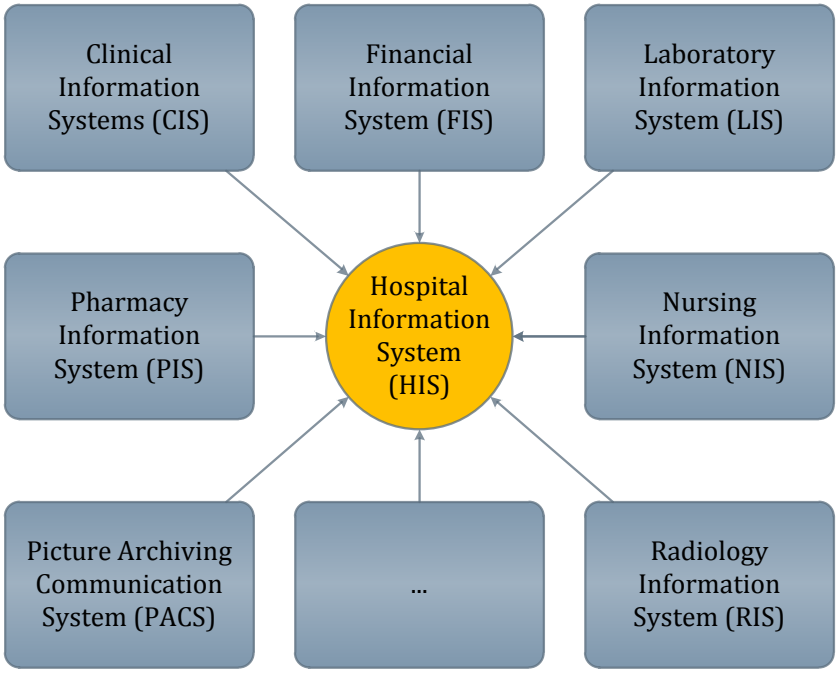


Figure 2.1: HIS Overview

2.1.2 HIS in developing countries

According to Venter, et al. there is a big difference between a HIS in developed countries and a HIS in developing countries in terms of economic resources, cultural influence, and political influences[3]. According to Malik and Khan consciousness of the benefits of a HIS in a healthcare system are not present in most developing nations, therefore there is an additional challenge in introducing a HIS in such a setting[4].

Igira, et al. have identified the main impediments to implementation of health-based software systems in developing countries as: organizational rupture and the absence of standardized methods in healthcare applications[5]. Moreover, social causes are more decisive than technical aspects when determining the successful end of projects of this nature [4].

Key factors that determined the success of implementing a HIS in developing countries are broad project planning, assuring governmental backing, and a special attention to the end user's needs. In contrast, factors that made projects fail are complex and out of scope proposals, inadequate graphical user interfaces (GUIs), ambiguous requirements, and insufficient awareness of the local systems. Additionally, other issues that negatively influenced the outcome are an absence of reliable electricity supply, poor computer infrastructure, inconsistent funding, and poor educational level of the technical department responsible for realizing the HIS[4]. The main reasons for success in health-care-software projects, particularly in the African continent, are simplicity, a low-budget, effectiveness, and efficiency[5].

Setting up of an HIS in developing countries should be done in a modular fashion, undertaken in a cooperative way, and incrementally scaled up. Success of a new HIS depends on the information system's design, the country's actual needs, and the actual health-care organization(s). Therefore it is crucial that system engineers are in close touch with the various users in order to understand their real needs [3].

Some strategies to close the design to reality gaps identified by Venter, Shaw, and Muyambo [3] are:

- **Hybrid profiles** are users that have knowledge from both technical design and managerial aspects. An individual that is educated in technical aspects of the system and that also understands the organizational factors and user requirements can make more appropriate decisions.
- **Participatory development** exploits users' feedback about the system's development progress and constantly auditing its adequacy with respect to the initial requirements.
- **Build local capacity** training new professionals that will be in charge of adapting changes when new requirements emerge in the future.
- **Recognize the complexity of the health system** due to external influences, both positive and negative, from donor organizations, political institutions, and medical centers.
- **Change management** is important in a possible and future roll-out of the system, **if** there is initial success in a low-scale network at a medical center or individual hospital.
- **Simple and flexible standards** that can be easily be adapted in a low-educated context and at the same time work in an integrated way over different medical units.

2.1.2.1 Health Information Systems Programme (HISP)

HISP[6] is a South African initiative that aims to enhance public health HIS in developing countries in Africa and Asia. In the context of this project, an open source and component-based application package called District Health Information Software (DHIS) has been developed. The principal objective of this software is to achieve data integration of multiple processes, while adjusting to existing standards. However, this integration could be a source of conflict in some countries with strong political control, because standardization is generally based on international cooperation.

Some of the factors, that lead to the failure of previous HIS designs and at the same time are key points for achieving success, are the lack of communication between different health initiatives and services within the same regional unit. This lack of communication is frequently based on inadequate data standardization. DHIS addresses this problem by establishing a finer granularity of health data, rather than trying to introduce complete health forms that contain multiple types of health information. In this way, forms can be dynamically built by composing multiple pieces of health data, thus allowing standardization of individual data blocks *within* forms that can individually be used (and reused) for a diverse range of medical purposes.

It is common to find inconsistent data, missing information, and different ways of collecting the same information between different people, while reporting directly on paper. This ultimately becomes a major obstacle, for example when there is an interest in analyzing this information in order to support executive decisions. Another difficulty is lack of computer skills, thus it is important to train the medical staff how to use the software[5].

2.1.2.2 Implementation of a HIS at Pakistan Institute of Medical Sciences

In [4], Malik and Khan stated that when the HIS project started at the Pakistan Institute of Medical Sciences (PIMS), there was a lack of enthusiasm for introducing a HIS. Moreover, there was a total absence of IT infrastructure. Furthermore, at the time this project took place there were no relevant open source software alternatives to consider and the institute could not afford to purchase commercial software.

At the technical department, a prevalent attitude was that working with a computer was a means to audit the staff's abilities or that this would revealing their insufficient mastery of skills, which could possibly lead to termination. As a result sabotage occurred with the objective of damaging the project's advancement.

When the system was introduced in the medical center, it functioned in parallel with the paper-based system. The graphical design of the HIS was made as similar as possible to the paper-based forms layout in order to enhance usability. Initially, medical personnel needed support from the technical staff to entering data into the computerized system, until the health staff started to acknowledge the obvious benefits of using the HIS. Moreover, only when the users assumed the control of the system did they became interested in helping to improve it and fix it in case of incidents. After the HIS was completely consolidated in the hospital, the paper-based method was discontinued. Moreover, training for both new and existing health staff became a key point in ensuring the continued success of the system.

After successfully demonstrating the system, the adoption by the rest of the center areas was predictable. However, there was a strong resistance to change the ineffective paper-based system. Only when the radiology and pathology departments started working in digital format did the rest of departmente realize the advantages of computerization. In addition, gaining the confidence of the political executives was an arduous task that was finally solved by organizing and conducting meetings and seminars.

In the context of PIMS, there was neither an initial planning nor a formal management board. However, an approach based on sequentially meeting every new system requirement at a departmental-level worked. As a result, the project's success was not ultimately dependent on a well-defined executive board.

User training was done individually, on-site, and with a special focus on practical aspects; rather than theoretical concepts. This last point seemed to be welcomed by the users, who became involved in the early development stages of the system. Therefore, an evolutionary approach was a wise decision as it allowed both system and users to progress together.

Malik and Khan concluded that when implementing HIS in developing nations it is important to be aware of country-specific contexts, restrictions, and necessities, where the introduction of information systems could be helpful in making a deep organizational change[4].

2.1.3 Conclusions about introducing HIS in developing countries

The previous section suggested that success was achieved by incremental development and education of the users. While this might have been good historically, the context has changed and there are now open source HIS packages. Hence it does not make sense to carry out an incremental development process; instead a more suitable approach is adapting current available software packages for local use and actively supporting the further development of these packages by providing feedback about failures and suggesting improvements.

In [7], Paul C. Webster states some of the reasons to use open source applications. One of the main reasons is to save a substantial amount of money. Additionally, open source software offers the chance of assessing the quality of code, and therefore it facilitates bug patching and source code improvement. In non-proprietary software there is more room for improvement, customization, and sometimes there is no need to sign a contract with a third-party company. These advantages of open source software make this approach an undisputed solution transferring advanced innovation to developing countries.

2.2 Electronic health records

An Electronic Health Record (EHR) is a collection of patient health information that contains demographic data, progress notes, problems, medications, past medical history, immunizations, laboratory results, and radiology data - among other health-based data[8][9].

The benefits of an EHR are the advantages that digitalized information commonly implies, such as an improvement in the precision of medical language and having the relevant and necessary information availability at the point of care. Moreover, EHR can provide a complete view of a patient's clinical record and naturally supports other analytical health-based activities, in this way enhancing the decision-making process and electronic communication[10]. EHRs allow systems to exploit the data in a variety of ways, while collecting the data only once. User interfaces that are appropriate for one medical area can be completely unsuitable for another medical specialty. Fortunately, EHRs can include additional features, such as alarms to medical staff and dynamic or customized presentation of data[9].

The most important secondary result of the adoption of EHRs is that the resulting data is available for analysis, management, and data-mining. Thus, researchers will be able to work with large datasets of information, which could help in understanding clinical effectiveness and in understanding the environmental and behavioral influences on diseases. For example, some commercial health insurance databases have been used to develop safety profiles of new diabetes drugs such as incretins. This type of research was largely unimaginable earlier[*]. Thus many believe that the wide-spread adoption of EHR systems can improve treatments and drugs, and consequently improve global health care. However, clear privacy and other ethical issues arise; for example, if a newly approved drug could benefit a set of patients, it would *not* be possible to communicate with them if these patients wish to remain anonymous [11].

Usually EHRs are created by different medical departments inside an organization. As Figure 2.2 illustrates, the main problems are that this information is not integrated and is stored separately[8]. Lack of integration of this information is currently one of the biggest barriers to overcome for the current EHRs and this barrier discourages the adoption of EHRs[10]. The main reason for this situation is that commercial companies use their own *de facto* standards and there is no single unified interface to each of the separate systems. Moreover, incompatibility of components from the same vendor is common when a vendor acquires smaller companies that each marketed their own software/hardware packages[8].

[*] This is not of course strictly true as there were pioneers such as Dr. Homer R. Warner, MD, Ph.D., who introduced medical informatics and decision support in the 1950s. See for example: http://intermountainhealthcare.org/hospitals/primarychildrens/about/news/pages/home.aspx?newsid=665

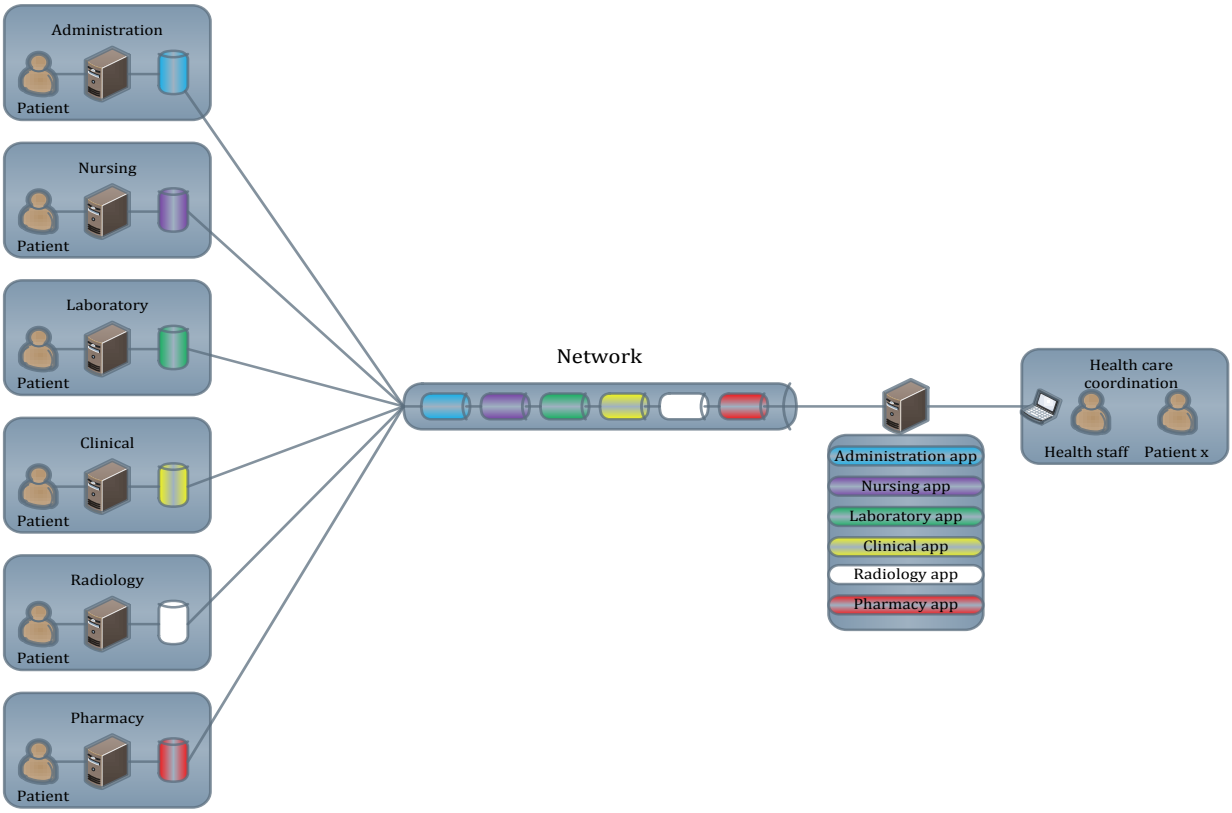


Figure 2.2: Unstandardized EHR

On the other hand, in a consolidated structure all the different systems can collaborate and access information that each one contains through the use of standardized interfaces. This model is shown in Figure 2.3. Although this is obviously a better approach, it is unfortunately **not** the predominant model in the current market.

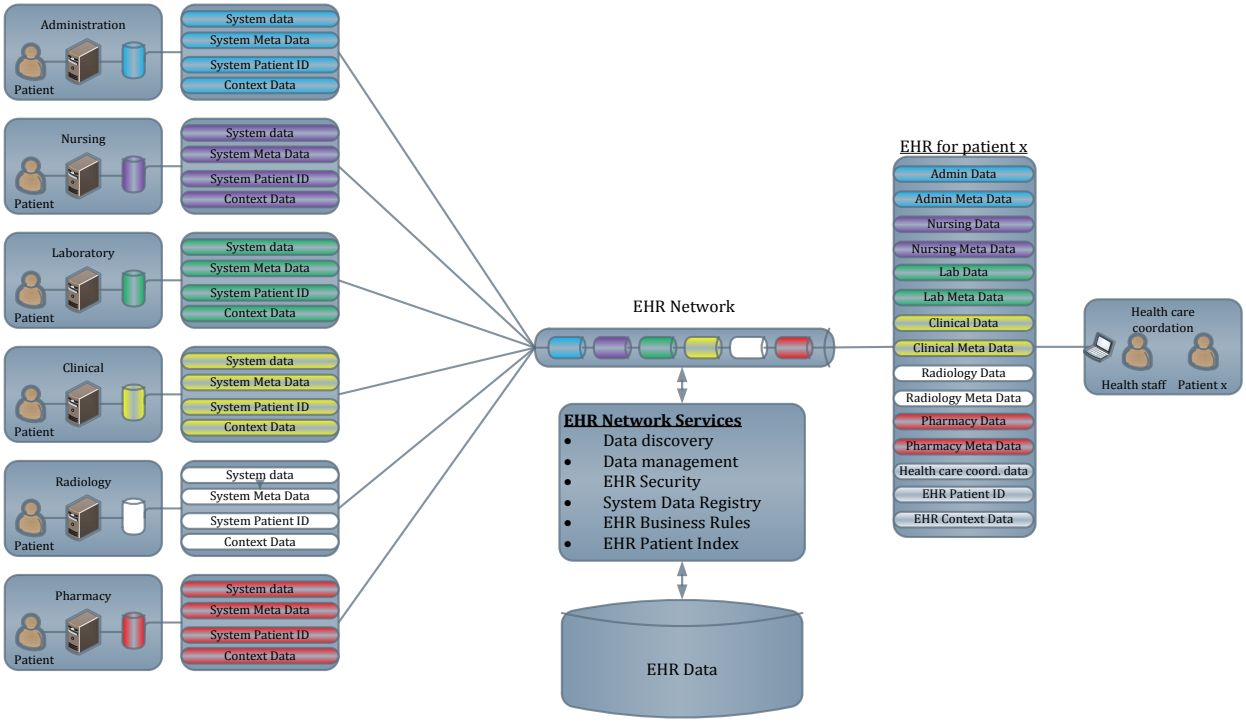


Figure 2.3: Standardized EHR

2.2.1 Use of EHR standards

Companies commercializing EHR have been implementing standards, but without a great deal of effort to ensure interoperability between systems from different vendors. As a consequence, there is still a lack of standardization of these kinds of systems. Initially, it might seem that this is due to the different vendors being unwilling to agree upon common standards. However, today this is mainly a problem of differences in automated medical vocabularies. Therefore, a complete and well-defined vocabulary and a clear and precise ontology are needed for an integrated and interoperable EHR system. Regulated vocabularies are useful for data exchange, comparison, and data aggregation.

Standards are required in EHRs to ensure a common vocabulary of medical terms, in order to exchange messages between different systems, and to have a common accepted ontology. Moreover, these standards must conform to existing privacy and security standards such as HIPAA in the US[11].

The most important entities that create EHR standards are [8]:

- American National Standards Institute (ANSI) Health Level Seven (HL7) in the US.
- Comité Européen de Normalisation – Technical Committee (CEN TC) 215 in Europe.
- American Society for Testing and Materials (ASTM) E31 in the US.
- International Organization for Standardization (ISO) worldwide.

The following subsections will describe the standards of these groups that are most relevant to HER.

2.2.2 ISO 18308 – "Requirements for an Electronic Health Record Reference Architecture"

ISO 18308[12] defines an architecture for using, sharing and exchanging EHR data across healthcare models, organizations and countries. This standard was created through three stages. Initially considerable research and contact with international domain experts was necessary to identify the most important requirements. After collecting more than 700 requirements, a hierarchical framework for organizing these requirements was created. Additionally, redundant requirements were excluded and the most important requirements were identified. Finally, a final set of requirements were developed and consolidated. This provided the starting point for an initial technical specification and an international standard.

The main users of this standard are EHR architecture developers, such as CEN 13606 and other architectural designs such as the *openEHR* Reference Model.

2.2.3 ISO/DTR 20514 – Electronic Health Record Definition, Scope and Context

ISO/DTR 20514[12][13] proposes a classification of EHRs, introduces some definitions for the main EHR categories, and describes some EHR system concepts. This standard approached an EHR definition making a distinction between contents and structure. It starts by describing the EHR structure, therefore ensuring applicability to a wider range of users and systems. It also supports compliance with the law and access control requirements. The context of care, which cannot be expressed in a supplementary definition or as part of this technical specification, is kept to a minimum.

Complementarily to this, there is a more specialized definition regarding the possibility of sharing patient health information and the role of EHR as an efficient and high quality method for delivering health care information.

2.2.4 CEN Standard: EN13606 – a Standard for EHR System Communication

EN13606[14] is a European standard and recently became a full ISO standard. A practical implementation of this standard is openEHR. Systems based in this specification are able to document, exchange, archive, and re-use information in any database in a standardized way.

As part of this standard, there is a distinction between shareable and non-shareable data, this facilitates keeping some sensitive information securely stored. This standard defines an Integrated Care Electronic Health Record (ICEHR).

Healthcare providers can download predefined archetypes and produce templates for national, regional, and local use. Additionally, it is possible to create both input and output forms, generate application and messaging formats, etc. giving a healthcare organization complete freedom for documenting, exchanging, archiving and reusing medical information. More concretely, archetypes define what can be documented. While templates are specifically used for defining documented, exchanged, stored, and reused data.

This standard is based on two important ISO specifications. It has its foundation in the ISO 18308 standard (this defines quality criteria in EHR systems) and ISO 22600 (this identifies privilege

management and access control aspects). Additionally, several European projects have contributed in this standard.

There is also a clear separation between domain content and technology. This means that changes in technology will not affect healthcare record contents in the system and vice versa.

2.2.5 CEN 'Standard Architecture for Healthcare Information Systems' (ENV 12967, aka "HISA")

HISA[15] is focused on a modular and open approach to health-care systems. The main underlying idea is that HIS are organized as a federation of applications that can cooperate and interact using a middleware layer. Furthermore, HISA specifies interfaces to the common services layer, allowing developers to modify them according to their needs.

There is a clear separation of the middleware services layer into two parts. The first part, called Healthcare Common Services (HCS), covers requirements of users in the health-care domain. While the second part, the Generic Common Services (GSC) can be used for generic purposes in any business domain.

2.2.6 ANSI Health Level 7 (HL7)

ANSI Health Level Seven is a non-profit organization, accredited by ANSI, with the mission to develop standards for health-based systems. [16] HL7 is a set of messaging formats together with an exchange reference vocabulary for data in health contexts[17].

HL7 was developed because there was a major lack of standardization in health care information systems. Before HL7 v2, systems were highly customized and expensive because each one was specially created. Moreover, software companies were unwilling to share their application programming interfaces with each other, which complicated writing standardized applications. HL7 was created in order to facilitate software compatibility[17].

HL7 provides a generous suite of standards for exchange, integration, sharing, and retrieval of electronic health data. These standards are classified into several different categories, as shown in Table 2.1.

HL7 is used generally used to send encoded information across applications. Currently, there are two different versions of HL7. The first is HL7 v.2x, which is used by commercially available off-the-shelf (COTS) applications. HL7 v.3 incorporates features to represent complex relationships. Additionally, a Certification Commission for Healthcare Information (CCHIT) was created to certify that companies appropriately implemented HL7 in their products[8]. Table 2.1 gives an overview of the different groups of standards that are part of HL7[16].

Table 2.1: HL7 Standards suite

Category	Description
Primary Standards	Main standards for system integration, interoperability, and compliance
Foundational Standards	Set of standards that specify the tools employed to build the standards and technological framework that HL7 adopters need to manage

Clinical and Administrative Domains	Standards in this group are made for messaging and health-based documentation
EHR Profiles	Functional models that makes possible to manage EHR are found in this group
Implementation Guides	Support documentation for existing standards
Rules and References	Standards defining technical specifications, data structures and criteria for software and standards development
Education & Awareness	Draft Standards for Trial Use (DSTUs), ongoing projects and support resources for HL7 standards.

2.2.7 HL7 Clinical Document Architecture (CDA)

HL7 CDA is a XML-based standard used for encoding clinical documents for exchange. A CDA document is a collection of information that exists outside a message. In addition to text this information can contain images, sounds, and other multimedia content. HL7 CDA divides this content into human interpretable information and optional structures for meta-information. CDA is quite flexible with regard to information representation and therefore it supports both simple documents and complex information[18][19]. CDA documents are encoded in XML and are formally specified using the HL7 v3 Reference Information Model (RIM).

CDA Level 1 describes the most general details of the document. CDA Level 2 provides information about document constraints through the use of different section levels. A more detailed description of the information contained in each of the section levels can be found in Figure 2.4. More concretely, sections represent text blocks, whereas entries encode the content of a narrative block in the same section[18]. Depending on the type of document these constraints might define an Emergency Department Discharge Summary or a Diagnostic Imaging Report. Finally, CDA Level 3 specifies additional constraints at the Entry level. The relations between the different levels, sections, and entries are illustrated in Figure 2.4.

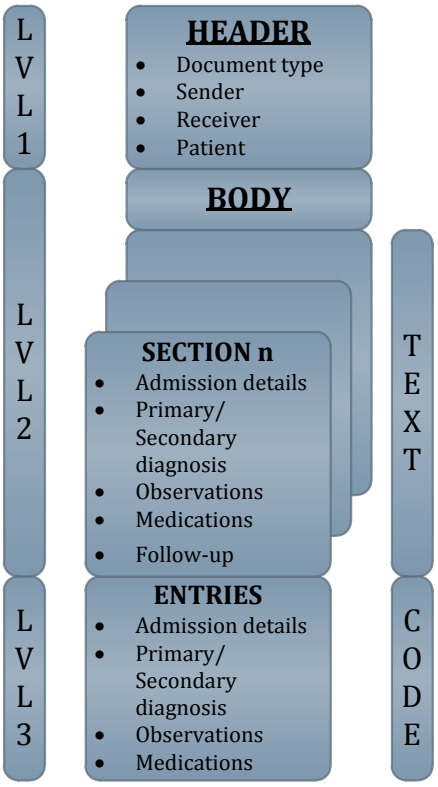


Figure 2.4: HL7 CDA Overview

2.3 Access control for electronic health-care records

Access control is commonly encountered in daily life. When keys are used to open a door or when someone logs into a webpage using their credentials an access control policy is being employed. Access to information is just another situation where authorization schemas need to be applied in order to appropriately ensure user privacy and prevent unauthorized accesses to data. More formally, access control is a mechanism that allows enforcing a chosen security policy that defines who is authorized to perform a specific action on an object.

In the case of a HIS, it is important to keep in mind that the HIS is trying to ensure the best health care through the use of an information system. Therefore, restricting access to information could in some situations determine life and death. Consequently, it is important to be flexible so that the patient's privacy does not prevent authorized health-care, but does not permit *inappropriate* access to a patient's data.

As an example, of the balance between privacy and patient case we consider the Swedish Personal Data Act (PUL)[†]: §18: Health and hospital care. This section of the law states:

> *"Sensitive personal data may be processed for health and hospital care purposes, provided the processing is necessary for*
>
> *a) preventive medicine and health care,*
>
> *b) medical diagnosis,*
>
> *c) health care or treatment, or*
>
> *d) management of health and hospital care services.*
>
> *A person who is **professionally operational** within the health-care sector **and** is **subject to a duty of confidentiality** may also process sensitive personal data that is subject to the duty of confidentiality. This also applies to the person who is subject to a similar duty of confidentiality and who has received sensitive personal data from the operation within the health care sector."*[‡]

Experience to date indicates that is **not** feasible to determine in advance which medical staff members will need to access a specific patient's data. Furthermore, systems that restrict access to a defined group of caregivers for a patient have been found to obstruct the health-care process. One approach is to extend authorization to further information that can be important in an emergency situation. Additionally, logging accesses in emergency situations helps avoid misuse of private information[20].

There are a number of requirements that an appropriate HIS access control system must conform to:

- Permissions must be consistent and therefore avoid conflicts.
- Permissions could have temporal constraints.
- Access control must support "Break the glass" situations.
- In case of a "Break the glass" situation, actions shall be tracked for future auditing.
- In case of a "Break the glass" situation, users shall be notified about subsequent audits.
- Permissions must allow or deny certain actions.
- Subjects could be human users or automated agents.
- A hierarchy of administrators will be in charge of authorization management.
- Access control must conform to privacy legislation.

Before introducing access control models, it is important to describe what the principle of least privilege is. When assigning access rights to system users, it is preferable that these permissions are assigned in accordance with the minimum privileges they need for carrying out their normal functions. This security foundation is also called the principle of least privilege (POLP) or principle of least authority (POLA). [21]

In the following subsections, different access control models will be explained and discussed in relation to a HIS.

[†] Personal Data Act (1998:204); issued 29 April 1998, http://www.coe.int/t/dghl/standardsetting/dataprotection/national%2520laws/SWEDEN_PDAct.pdf .

[‡] Emphasis as per the lecture notes of Gerald Q. Maguire Jr. for the course II2202 at KTH during Fall 2011.

2.3.1 Access Control Matrix Model (ACMM)

ACMM is one of the simplest approaches that can be used in order to define an authorization policy in a computer-based system. Its structure is based on a column representing processes or objects and a row for each subject (user). Each matrix entry is an array of access rights that a subject has to an object. For example, Table 2.2 shows that user P1 has read (R), write (W), execute (X) and own (O) access to object F3, while user P2 has no access to object F3.

Table 2.2: Matrix Model

	F1	F2	F3	P1
P1	RWO	RO	RWXO	RWX
P2	-	RO	-	RWXO

Unfortunately, the simplest approaches are often insufficient to deal with real security problems. For this reason some additional modifications have been proposed for the ACMM. One example is to add an extra field for each one of the access rights in order to specify a temporal permission variable (see slide 9 of [22]).

This simple access control system obviously does not suit a complex HIS. First of all, it would be costly, in time, to resolve if a specific user is allowed to perform a specific action on an object, as a HIS typically contains a large number of patients' data each entry of which would be considered a different object. Additionally, there will be many different users allowed to use the system, and there is a high probability that there are many with similar or equal access control matrixes (for example, if they are in the same position or have similar responsibilities within the organization). Therefore, the access control matrix will have a large number of rows, many of which have the same access permissions to a set of objects (represented by columns). This matrix would take a large amount of storage if represented explicitly. Hence one can think of a denser representation that exploits equivalence classes. We will explore some of these groupings in the following subsections.

2.3.2 Discretionary Access Control (DAC)

Discretionary access control (DAC) is a model that restricts access based on the subjects' identity or membership. This schema is usually contrasted with mandatory access control, also called non-discretionary access control (see section 2.3.3). DAC is often used when system objects are owned by subjects, where usually these users are the ones who created the object, and others acquire new privileges directly from the object's owner(s)[23].

2.3.3 Mandatory Access Control (MAC)

Mandatory access control (MAC) allows classifying information depending on its level of sensitivity into a multi-level security model. This is a good model to implement in organizations with a strong hierarchical structure. In this model more powerful users always have greater access than less powerful users. This model is typically used by the military. A typically pattern is that users are allowed to read from any lower level, but they are unauthorized to read from higher levels. Conversely, they can write to higher security layers, but at the same time they are forbidden to write to lower security levels[23]. In this manner information can propagate freely up the security hierarchy, but access to this information become more and more limited as it moves up the hierarchy. This scheme allows users with high level access to see all of the details and prevents those with lower levels of access from disclosing information that they do not have access to.

2.3.4 Role-based Access Control (RBAC)

Usually in organizations, users do not own the information they access. Instead, the corporation is the owner of the systems and the information that is produced by applications. Therefore access control is most frequently a matter of job function rather than ownership.

Access control can be determined by the roles that each individual has in the organization. That is why role-based access control (RBAC) structures users in terms of the responsibilities they have within the organization. For example, some roles in a hospital are: doctor, nurse, pharmacist, filing clerk, etc.

A role is in fact a collection of actions that users can perform. Administrators are responsible for assigning privileges to roles and also for assigning users to roles (i.e., a user is a member in a group of people who have the same role). Note that an individual many have multiple roles.

Security policies are often defined in concordance with organizational policies. To appropriately support these policies, it is necessary to have centralized control of access rights. Security administrators are in charge of enforcing policies, hence they are responsible for assigning appropriate access rights[24].

As permissions are associated with roles, it is possible that there will be conflicts of interests when assigning different roles to a user, therefore it is necessary to introduce constraints regarding the separation of duties[25].

The NIST RBAC model [25] aims to unify existing RBAC designs. In the specification of this model, different RBAC levels are identified. These levels are mainly characterized by adding new features in a cumulative way to the well-known RBAC model. In the next subsections, these different variations are explained in more detail.

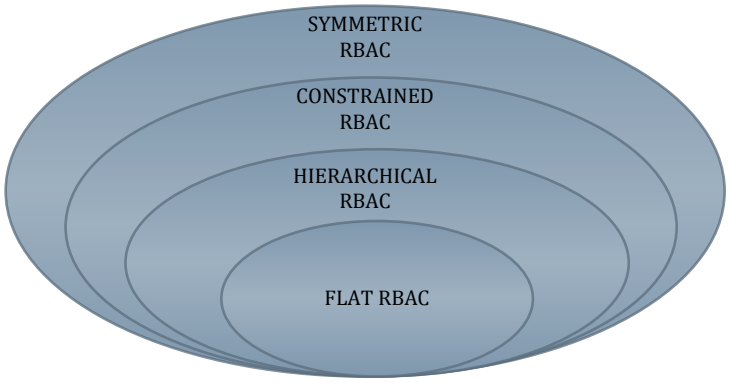


Figure 2.5: RBAC sets

2.3.4.1 Flat RBAC (F-RBAC)

Flat RBAC (F-RBAC) encompasses the basic characteristics of RBAC. Permissions are assigned to roles, and users obtain permissions by being members of roles in a many-to-many relationship between roles and users.

As shown in Figure 2.6, there are a total of three sets corresponding to users that could either be human individuals or automated agents. These roles are usually associated with job titles within the organization, while permissions are actions that can be performed over particular system objects.

This model supports user-role review, which means it is possible to determine to which roles a user belongs to and which users a role is assigned. However, role-permission is not included in this level as this can be difficult to implement in large-scale distributed systems, therefore role-permission is left as a fourth level feature.

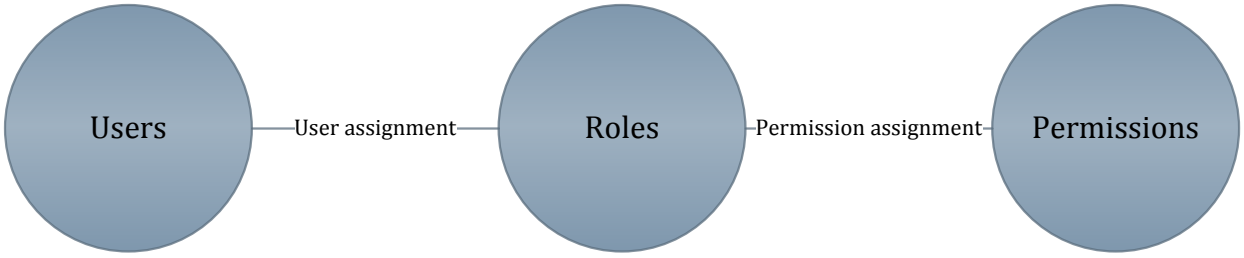


Figure 2.6: Flat RBAC

2.3.4.2 Hierarchical RBAC (H-RBAC)

Hierarchical RBAC (H-RBAC) adds a new feature of hierarchy, as is easily to understand from its name. Role hierarchies are introduced in order for roles to inherit permissions directly from other existing roles. In the case of hierarchical organizational structures this is a practical approach, where more powerful roles share some permission(s) with junior roles. This is illustrated in Figure 2.7. However, in some situations it is a dangerous practice to give too much power to a certain role, which is why it is possible to introduce additional constraints to limit inheritance. Examples of reasons to introduce such limits are users' mistakes and software viruses.

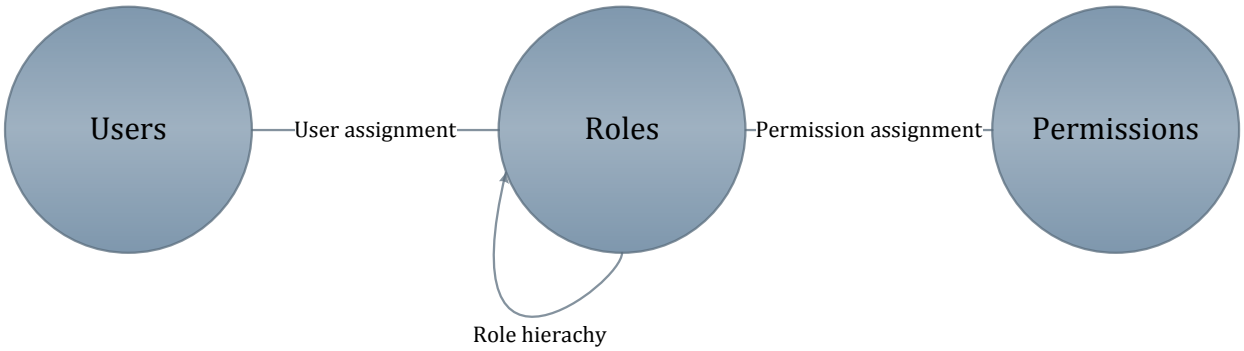


Figure 2.7: Hierarchical RBAC

2.3.4.3 Constrained RBAC (C-RBAC)

Constrained RBAC (C-RBAC) incorporates new features to enforce separation of duties (SOD), which seeks to reduce the risk of fraud or potential damages in the system. SOD spreads authority for an action over multiple users, because it is essential to ensure that these responsibilities are assigned to *different* users, rather than to a single user who could gain too much authority. An example that illustrates that two (specific) permissions cannot be assigned to the same user would be "Give a loan" and "Approve a loan" – as otherwise one person could both make and approve loans (even to themselves).

This model allows both static and dynamic SOD and permits a policy to determine which one should be used within the system.

- Static Separation of Duty (SSD): This feature prevents that users that belong to a certain role can enroll to other concrete one. Additionally, constraints are naturally inherited within role hierarchy. This is illustrated in Figure 2.8.

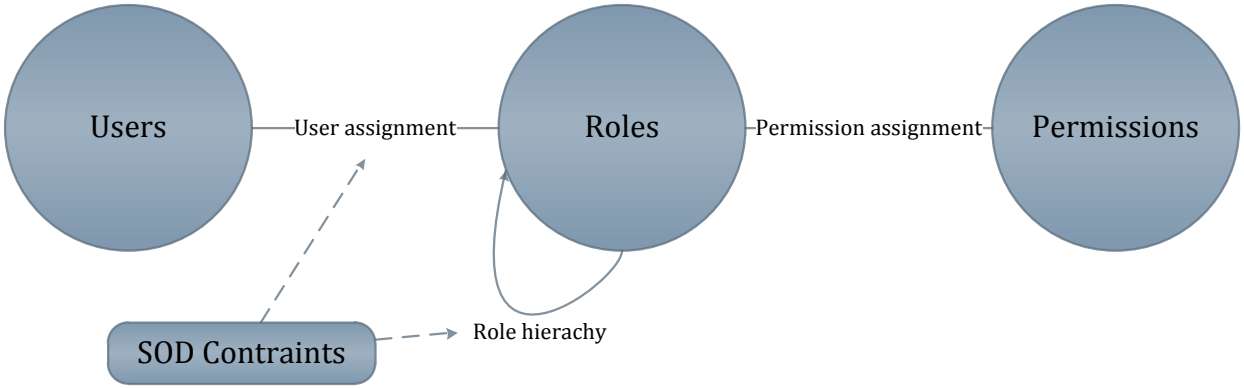


Figure 2.8: Constrained RBAC with SSD

- Dynamic Separation of Duty (DSD): DSD allows a user to be enrolled in a set of roles that do not represent a conflict of interest when performed independently, but when acting simultaneously constitute a conflict of interest. This is illustrated in Figure 2.9.

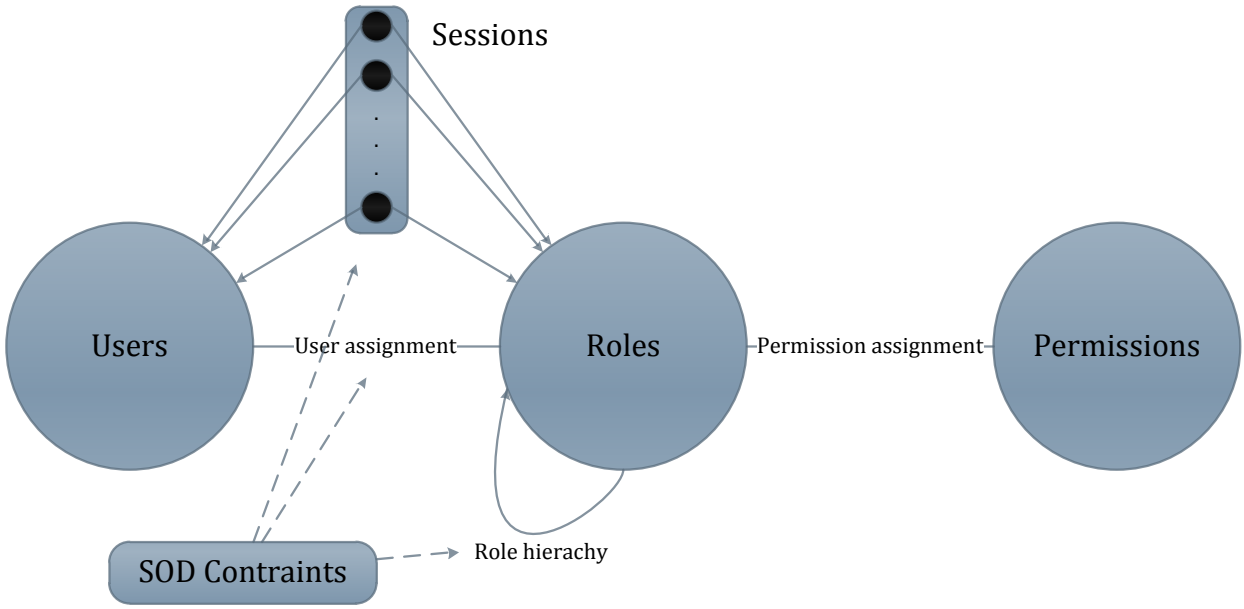


Figure 2.9: Constrained RBAC with DSD

2.3.4.4 *Symmetric RBAC (S-RBAC)*

Finally, Symmetric RBAC (S-RBAC) adds a permission-role assignment interface. This interface makes it possible to find who within the organizational context will be affected by a permission-role assignment. Note that these assignments may change throughout time, possibly based on new organizational policies. However, these changes can be difficult to manage if these permissions are present in a distributed system and multiple IT administrators need to take part in the modification.

2.3.4.5 *Other uncovered issues*

The NIST RBAC model is a good start to standardize RBAC. However, there remain matters that are still unresolved or only partially covered by the NIST model. Scalability is not really discussed within the NIST model. Additionally, although a security manager can include them, this model does not initially

include **negative** permissions, because in a system with large hierarchies negative permissions can add a lot of complexity. Also, the nature of the permissions is not specified, that means that access rights can be applied either to system objects or to entire systems. Other points that are not covered are which roles are assigned to users, guidelines on how to design roles and assign permissions to the users, and how the authorization policy will be administered

2.3.5 *A posteriori* access control

An EHR may be composed of information from various tests and notes of attending physicians, nurses, and specialists. There are two major requirements for these records within a HIS. The first requirement is that the HIS must provide high availability of a patient's EHR to authorized users. The second requirement is to ensure the privacy and integrity of this information. A particular case can be found in both The Netherlands and the United States of America (USA) because in both countries the private sector is a major provider of health care, therefore medical data is extremely valuable to these firms. Not only is this information valuable for providing the health-care services that they provide and to bill the patient (or their insurance company) for any care which is provided, but also because improper disclosure of this data could lead to civil and criminal penalties against the hospital (or in the case of the USA under The Health Insurance Portability and Accountability Act of 1996 (HIPAA) the individuals employed by the hospital or other medical facility can be personally subject to penalties).

The main advantage of *a posteriori* access control approach is that medical staff is not constrained by security policies or failures regarding the security infrastructure when they need to treat a patient. This method takes the view that these kinds of administrative issues can be resolved after the patient has been treated in the case of an emergency, for example. In practice, this system initially ignores *a priori* the current security policy. For this reason, auditors needed to confirm that the actions of the authorized staff conformed to the security policy. Consequently, details of all actions undertaken need to be logged by the medical staff. This approach is in contrast to an *a priori* approach, where authorization requests are checked for validity at the time of an access.

To achieve the maximum level of privacy, auditors examine the system logs to see that all actions were in keeping with the relevant access policy. Examples of these auditors are internal auditors, governmental authorities, and patient union representatives. When any action is taken, an audit trail is generated to keep track of who performed the action, when they performed it, and perhaps why they performed this action.

So, we can conclude that *a posteriori* access control speeds the process of retrieving information that needs to be immediately available for medical purposes and enables access to medical information when the authorization policy is not available (a might happen in a distributed system), but does **not** prevent users from misusing the information which they access.

An important point about a well-defined *a posteriori* access control is that there must be some kind of system that legally or administratively punishes misbehavior. Another approach that can be used is trust management systems. Using a trust management system would enable a complete overview of how users are trusted in the system. Such a system could decrease the trustworthiness of users that have a history of misbehaving[26].

2.3.6 Attribute-based Access Control (ABAC)

Attribute-based access control (ABAC) consists in using different attributes that characterize the system entities for access control purposes. These attributes are certain properties associated with subjects, actions, resources, or the environment. Common entities that can be identified as sources of attributes are [27]:

- **Subject** who is initiating the access request. Subject attributes would be role memberships, position within the organization, assigned responsibilities, etc.
- **Action** which is to be performed. In this case, the type of action is the most characteristic attribute that could be identified, but other data may be present. For example, print (type of action) a piece of information 2 (amount to print) times.
- **Object** that is to be accessed. Object attributes are mainly characteristics of the object in question. Therefore these attributes will differ among the set of objects.
- **Environment** of the access request. These attributes are usually related to time and physical location. Additionally, device type, communication type, and other criteria could be relevant indicators to determine access request decisions.

2.3.7 Purpose-based Access Control (PBAC)

While many of the methods proposed for access control of EHR are based on RBAC, other models have been proposed to conform to the medical information privacy regulations. One of these alternative models is purpose-based access control models (PBAC) [28]. These models are designed to grant access when the purpose for access is the same as the data's intended purpose. For example, in [29] Peng, Gu, and Ye present a PBAC-based approach that dynamically computes the purpose during the access decision based on different attributes such as subject, context, and authorization policies.

PBAC [29] is based on a set of purposes for which data is collected and for data access. For example, data could be collected for diagnosis purposes. Purposes that are related to data are called Intended Purposes, and purposed for data access are Access Purposes. Furthermore, Intended Purposes are composed of Allowed Intended Purposes and Prohibited Intended Purposes. When a data access is requested, both purposes must match to permit the access. Moreover, purposes are organized in tree structures, where each node represents a purpose and each edge represents a hierarchical relation. [25]

Additional extensions, that include features of other access control models, have been proposed for a PBAC design [30]. The use of roles as RBAC does is a common extension that eases the management of purposes. Furthermore, decisions based on attributes, as ABAC does, offers an extra feature that enhances flexibility.

In PBAC it could be relevant to identify states of the system where some permission should be activated or deactivated. For instance, we could allow certain users or roles to use the system during specific time intervals or when they are using the application from certain physical machines. This time and location data are consider being system attributes.

PBAC approach, and therefore access purposes, can be easily mapped to security policies within an organization. However, the key issue for is how to determine the purposes that each user action concerns, and consequently, if these purposes conform to current regulations.

Peng, Gu, and Ye [29] propose three different ways of determining the purpose(s) of access. First, users will be required to state their access purpose(s) along with their access requests. Although this is a simple way of learning the user's stated purpose, it offers only limited security (although these purposes can be logged for an *a posteriori* access control audit). Secondly, the system registers these access purposes for each procedure or application. In this way security is obviously improved, but there is a substantial amount of work that may be necessary in order to relate the operations or applications to the intended purposes. Thirdly, purposes can be determined based on user attributes and system attributes. This last approach is the most secure of the three proposed, although it implies a more complex computation over the different attributes.

2.4 Health-care system requirements

There are many requirements involved in health care information systems: guaranteeing care delivery, keeping data secure, and maintaining privacy of sensitive data[31]. As medical data is one of the most sensitive types of information, it is essential to ensure that access to this data by the different health-care staff *only* occurs if they have been authorized to access this specific data. In many legal jurisdictions each individual patient should manage the access to *their* private information, hence they need to approve or disapprove of their information being visible to specific medical (and even administrative) staff members.

Since in some jurisdictions a patient's health is more important than his or her privacy, the principle called Care Comes First (CCF) is often applied. Therefore in the case of a medical emergency, the patient's medical data may be accessed by a health-care professional whom is responsible for delivering care, overriding the implemented access control policy. For this reason, a doctor may require some medical data in an emergency that under normal conditions he or she would **not** be allowed to access. That is a good example illustrating a phenomenon called in the event of an emergency "break the glass". As noted previously, when this policy violation happens, log files should be analyzed after the fact to determine if there has been access misuse or abuse[31].

The access control model should support the definition of different types of roles (such as a nurse or doctor) inside a health-care organization. Moreover, the subjects included in these defined roles can be structured in a hierarchy. This does not mean that access control is simply based upon roles, but that it is essential to consider roles as the user's specific role(s) is an important piece of information that has been used as one of the factors when making an access control decision in a number of systems.

Another important aspect to take into account for a health care information system emergency is that the required health information access can vary as a function of the emergency's seriousness. Therefore, we need to distinguish between the access that would be granted in the event of a car crash or a natural disaster, and consequently show *different* information based upon the severity of the specific emergency.

With regard to the "Break the glass" phenomenon, it is important to introduce logging mechanisms that can ensure auditing *a posteriori*. Additionally, these measures will to some extent discourage health-care staff from misusing personal data and detect accesses to a patient's information that is in violation of the organization's policies.

While allowing access which violates the access control policy undoubtedly facilitates the medical task in some concrete situations when rapid access is the highest priority, allowing such violations represents a security hole that could be misused. This problem mainly occurs because it may be possible to access information that is not actually relevant to the patient's *current* care (such as which drugs were administered to the patient two years ago). However, we cannot *a priori* say that earlier information is definitely irrelevant, but rather this will have to be addressed in the *a posteriori* audit. Additionally, it is apparent that other access purposes can exist in addition to the usual access and emergency access cases. For example, medical researchers may be interested in analyzing health-care records or inspectors may be allowed to examine health-care information in case of negligence. In both cases, the purpose of accessing the data has to be consistent with the purposes for which the data was acquired or in some jurisdictions have an overwhelming importance to society[§].

[§] PUL §19: Research and statistics regulations states: "Sensitive personal data may be processed for research and statistics purposes, provided the processing is necessary in the manner stated in Section 10 and provided the interest of society in the research or statistics project within which the processing is included is manifestly greater than the risk of improper violation of the personal integrity of the individual that the processing may involve."

An aspect that is not usually covered in role-based access methods is the environmental of the request; however, this is often an important characteristic in health-care. Therefore, environmental conditions should be treated separately and are **not** implicit in the subjects or objects.

2.4.1 Authorization infrastructure

A set of components [32][33] are necessary to build a security infrastructure that satisfactorily answers data access requests from the users, while allowing administrators to manage security policies. These components include:

- Policy Enforcement Point (PEP): The PEP represents the front-end where client requests for data are captured. The PEP acts as a proxy by forwarding these requests to the Policy Decision Point (PDP). This means that when the PEP receives messages asking for data access, it constructs a request asking for a policy decision and sends this request to the PDP. If the PDP allows this access, then the PEP forwards the request to the resource, otherwise it simply denies access to the data and returns this decision to the PEP.
- Policy Decision Point (PDP): The PDP is in charge of decision making regarding access requests. When an access request arrives, the PDP queries the policy server in order to retrieve the policies associated with the resource. By applying these policies the PDP decides whether the requested operation is allowed or not. It returns its decision to the PEP who forwarded the access request.
- Policy Information Point (PIP): The PIP offers additional information that could be relevant for decision making. This data is usually obtained from other backend systems that can provide this information. Furthermore, these systems are managed by other agents and the purpose of PIP is to retrieve relevant information for authorization purposes.
- Policy Administration Point (PAP): The PAP provides a control panel through which authorization policies are managed by administrators.

2.4.1.1 Access control workflow

The overall workflow of a request to access a resource involves the following steps (as illustrated in Figure 2.10):

1. An access request is sent either by a computerized agent or a human user.
2. The PEP sends a XACML request to the PDP. (Details of XACML will be presented in section 2.4.2.1.)
3. The PDP sends a XACML request to the policy server to retrieve relevant policies.
4. An XACML response containing the policies is sent by the policy server to the PDP.
5. The PDP sends an attribute request to the PIP.
6. The PIP replies with an attribute response message to the PDP.
7. After processing the access request using the available information, the PDP sends a XACML response to the PEP.
8. Finally, based upon the PDP's response the PEP either allows access or denies access to the requested information.

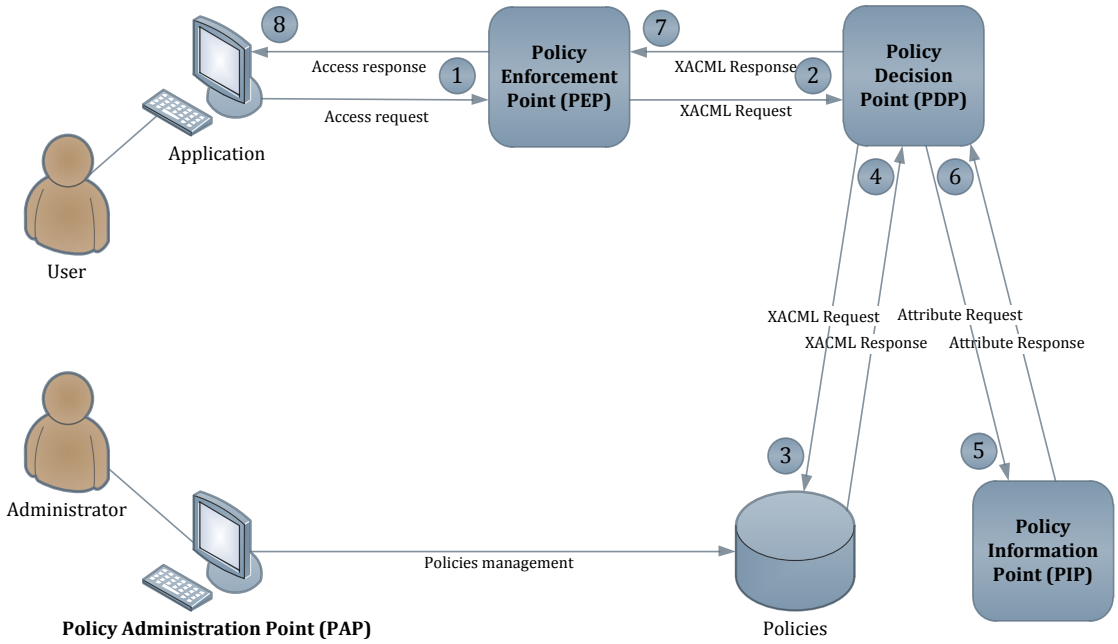


Figure 2.10: Authorization infrastructure

2.4.2 Access control languages

Different components have been identified as part of the security infrastructure and it is likely that some of these elements are in separate physical locations; therefore these components constitute a distributed system. Furthermore, a HIS can use completely independent subsystems and consequently it needs a standard for communications between these subsystems. For this reason, the different messages that are exchanged between them need to use information exchange standards. The most relevant of these standards is XACML, which is presented in section 2.4.2.1.

2.4.2.1 eXtensible Access Control Markup Language 3.0 (XACML)

The eXtensible Access Control Markup Language 3.0 (XACML) is a solution created by the OASIS organization for exchanging security information in a standardized way[34]. It was originally created by Sun Microsystems (now part of Oracle) in order to replace existing proprietary access control mechanisms.

XACML is composed of two XML schemas. *XACML Policy* provides a XML-based syntax for developing access control policies. While *XACML Context*, is a communication language that defines access request and response messages. An example of XACML messages can be found in appendix A.

Among the advantages of XACML 3.0[35] over other access control mechanisms is the reusability of policy definitions across different applications, ease in creating and managing policies, support of emerging system requirements, and self-referral of different XACML policies, for example, global policies and country- specific policies.

2.4.2.2 Security Assertion Markup Language 2.0 (SAML)

The Security Assertion Markup Language 2.0 (SAML) is a standard created by the OASIS organization for creating and exchanging security information among networked systems[36]. It has been developed by the Security Technical Committee of OASIS.

SAML is also a XML-based framework used for transmitting authentication, authorization, and attribute information. The main difference between XACML and SAML 2.0 is that SAML provides additional details of the user's authorization.

SAML 2.0 aims to solve a problem regarding the multiplicity of different incompatible protocols: SAML 1.0 & SAML 1.1, ID-FF from Liberty Alliance, and the Shibboleth OpenSAML.

2.4.3 Information security policies

Information security policies [37] are rules established by an organization regarding protection and control of institutional data. These policies should protect the organization's sensitive information, give a number of rules that define the expected behavior for the different system users, enable administrators to manage policies, and comply with current legislation. Note that although we referred to the "organization's sensitive data", this includes sensitive information for which the organization is responsible, even if they do not "own" this information.

Policy analysis tools have been created to discover conflicts between policies. The Firewall Policy Advisor[38] is an example of a tool used to detect policy conflicts in firewall policies. This software can examine filtering rules in a multi-firewall enterprise environment for rules that might cause network vulnerabilities. Additionally, the Firewall Policy Adviser can discover anomalies in both centralized and distributed firewalls.

Policy anomalies [28] usually appear when Electronic Health-care Record (EHCR) instances from different sources but associated with the same subject's EHCR are inconsistent with each other. Policy anomalies can also arise when a policy is added or updated during the policy specification stage. When inconsistencies exist between different EHCR instances, it can be difficult to deduce which one is correct. Therefore, it is necessary to examine the conflicting policies in order to choose the appropriate instance.

Several types of policy anomalies can occur:

Contradictory	Two policies that have different effects over the same subject, target objects, and intended purposes.
Exception	Two policies that have different effects, where one policy is a subset of the other.
Correlation	Two policies that have different effects, where one policy intersects with the other.

2.4.3.1 Policy evaluation steps

Policy evaluation occurs through the following five steps:

1. **Policy filtration**: Relevant authorization policies are chosen from the pool of policies.
2. **Authorization zone segmentation**: The authorization zone is divided into different disjoint groups.
3. **Conflicting zone identification**: Deduce possible conflicts between disjoint segments.
4. **Strategy-based conflict resolution**: Resolve conflicts and produce an evaluation output.
5. **Permitted zone aggregation**: Authorized segments are aggregated and output.

2.4.3.2 *Policy conflicts strategy workflow*

Strategies to resolve policy conflicts include:

Recency-overrides	Newer policies have higher priority than older ones. However, two policies may have the same creation date. Therefore, this strategy is non-deterministic, as it does not resolve conflicts in all cases. A finer granularity clock could solve this issue and in some cases enable a deterministic choice.
Specificity-overrides	More specific policies have higher priority than generic authorizations. However, this may also be non-deterministic when multiple policies have the same priority and specificity.
Deny-overrides	Access requests are denied if one of the conflicting policies has a denial effect.

Figure 2.11 illustrates how the different policy conflicts are resolved within the policy conflicts strategy workflow.

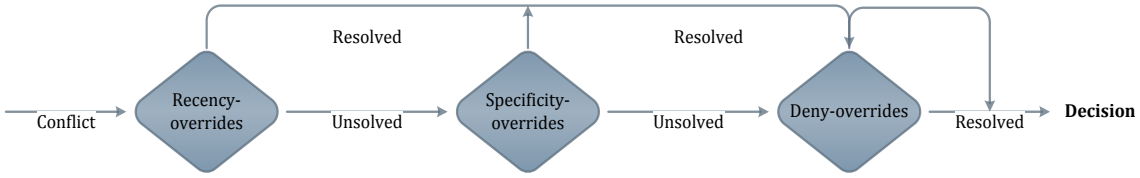


Figure 2.11: Policy conflicts strategy flowchart

2.5 Legal issues

Legislation regarding personal data privacy in Sweden is quite strict, but the system that is to be implemented is being designed for use in Uganda, where legislation regarding data privacy is not yet well-defined. The reason for this is that computerization is only present in the urban areas of the country.

Privacy is recognized as a human right by the Republic of Uganda. However, it is not an absolute right because in cases when individual interests can prejudice public interests it is possible to override privacy rights. Additionally, Uganda is currently a developing country; therefore many persons believe there are more important matters than individual privacy that need to be solved for the benefit of the Ugandans[39].

As a result we will design our implementation keeping in mind the Swedish legislation, but where necessary will adapt the system to the local (target country's) requirements. The goal is to achieve a practical implementation that could be used in Uganda, but with a change in policies could be used in Sweden and other countries with strong privacy legislation. It is an essential requirement of this thesis project to develop a system that would be applicable in Uganda. However, it is also clear that developing a system that *only* meets these requirements will have little general utility (and hence have few users – resulting in a higher total system operations and maintenance costs than a system that is very widely used). Additionally, as Uganda develops it is very likely that strong privacy protection laws will be enacted.

3 Method

This section gives an overview of a method for building a suitable authorization service for a HIS. For this work, there are different sub problems involved that will be described in this chapter. Section 3.1 discusses how to represent the context-relevant information. Section 3.2 explains how to model the different authorization policies. Section 3.3 examines how to decide about authorization requests. Section 3.4 gives an overview of different tools for logging the authorization process. Section 3.5 introduces the possibility of integrating the proposed authorization service with an existing EHR. Finally, section 3.6 summarizes the chapter.

3.1 Representing the authorization context

As discussed in the previous chapter, a variety of attributes from different entities are needed for access control. This implies that we must explicitly consider attribute representation, because it is important to organize this information in order to easily manage it.

An ontology is an excellent framework for representing knowledge, especially within a concrete domain. According to Tom Gruber, an ontology is a "explicit specification of a conceptualization" [40].

To continue, a good point is to think about the information that can be represented in the ontology. Fortunately, ABAC introduces a few entities that in fact are the most important factors for authorization. However, it is possible to find more attributes and characteristics derived from these concepts that are relevant as well.

Several advantages of ontologies are discussed by Dersingh, Liscano, and Jost in [41]. Among these advantages are two that are relevant to our problem: (1) the possibility of openly sharing knowledge in distributed environments and (2) for reasoning about collected information.

In order to proficiently work with ontologies, there are formal languages available for constructing and manipulating ontological structures. Particularly, a set of these languages were specifically built for semantic web applications. Therefore it is more important to consider these concrete technologies, as they can interact and integrate the contents available on the Internet.

Figure 3.1 illustrates the structure of the Semantic Web components that have mainly been developed by the World Wide Web Consortium (W3C)[42] and are considered W3C Recommendations. The most important elements of this picture are explained below.

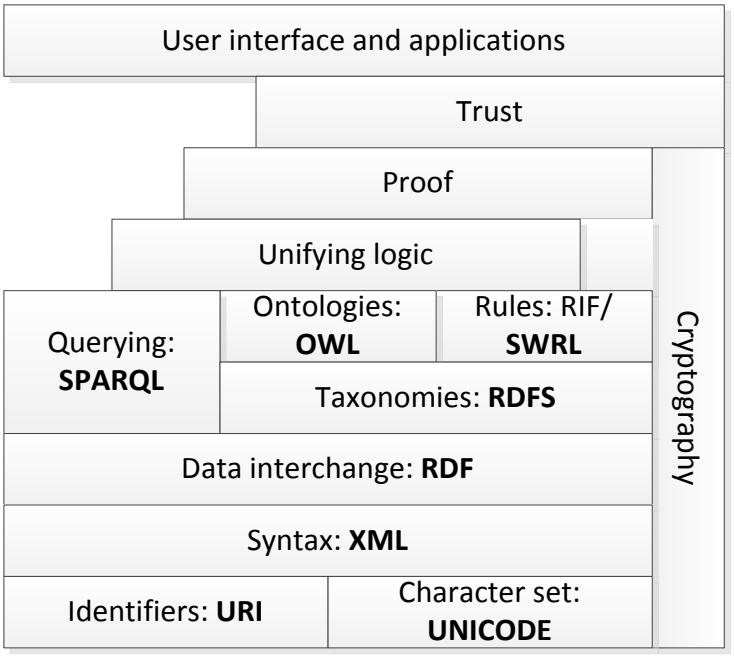


Figure 3.1: Semantic Web Stack

A **Uniform Resource Identifier (URI)** is a method for identifying a resource in the World Wide Web using a string of characters.

The **eXtensible Markup Language (XML)**[43] is a simple and very flexible language for encoding information. Today, XML is widely used for data exchange purposes. Different working groups are responsible for developing and maintaining the different subcomponents of the XML technology.

The **eXtensible Markup Language Schema (XMLS)**[44] introduces mechanisms for defining the syntax and semantics of XML documents. Conformity rules can be introduced in order to validate documents according to a specific schema.

The **Resource Description Framework (RDF)**[45][46] is a standard for exchanging information on the Internet. Among its main characteristics are data merging from different models and ease of model evolution. RDF has become the foundational framework for expressing knowledge on the Internet, where each device makes a use of distributed data, possibly with a particular structure, across the network. RDF uses URI's to identify relationships between different concepts and at the same time to recognize these pair of elements. In RDF is possible to represent simple semantics such as classes, sub-classes, properties, and sub-properties.

The **Resource Description Framework Schema (RDFS)**[47] is language for representing data in the Semantic Web. For this purpose, RDFS defines a vocabulary and other built-in features that may be used to describe resources, properties, and so on.

The **Web Ontology Language (OWL)**[48], originally built on RDF, is a Semantic Web language used to represent complex knowledge about elements and about the different relationships among them. The current version is OWL 2 (an extension to OWL 1). In OWL it is possible to constrain classes and relationships, expressing disjoint relationships between classes, for example.

Basically, OWL offers scenarios where three concepts are present. Firstly, classes are representative entities within the domain of interest. Secondly, OWL encodes the relationships among these classes. Thirdly, OWL encodes the properties that characterize these classes.

Although the main syntax used is RDF/XML, different serializations such as Turtle[49], XML Serialization[50], the Manchester Syntax[51], and the functional-style syntax[52] can be used in OWL.

OWL is composed of three sublanguages [53]:

- OWL Lite: Syntactically the simplest sublanguage. It is adequate when a simple class hierarchy and constraints are required.
- OWL DL: More expressive than OWL Lite and based on Description Logics. Therefore it allows automated reasoning and inconsistency checking.
- OWL Full: Most expressive sublanguage used when expressiveness is more relevant than decidability. So, automated reasoning is consequently not supported by the OWL Full.

Before working with OWL it is important to argue which one of the proposed sublanguages is the most appropriate. To decide between the different alternatives, we will look at the main characteristics for each one of the sublanguages.

In context, it is well-known that health-care environments are complex systems and therefore a certain level of expressiveness is necessary in order to build and structure a consistent base of knowledge. However, this knowledge base must support reasoning because one of the purposes of this representation is to reason about the knowledge expressed in the representation. Therefore we have chosen OWL DL to design our ontology for representing authorization concepts.

There are additional technologies that can be found upper in the stack that may be of interest. Some of these may be relevant to us, in particular: SPARQL[54] is a query language to retrieve and manipulate RDF-based models; SWRL[55] is an extension to OWL that integrates Horn-like rules: and RIF[56] is a XML-based framework used for rule interchange between different rule-based systems.

A widely used and efficient tool for knowledge-based systems and consequently a tool that is useful in the semantic web area is Protégé[57]. Protégé is an open source Java-based development environment, created by researchers at Stanford University. Protégé is used for developing ontologies and knowledge-based systems and it supports a variety of formats such as XMLS, RDF, RDFS, and OWL. Furthermore, a committed user and developer community exists behind Protégé, including professionals from a wide variety of expertise areas.

As part of this thesis project, a sample authorization context terminology has been developed. This is in fact an ontology. The complete ontology is given in Appendix B. The idea behind this schema is simple. There are different entities which are at the same time characterized by properties. Therefore, this model represents the different properties that can be related to the system entities and provides a reference for working with a common conceptual structure. These different concepts are used for specifying all the relevant data for authorization purposes that will compound SWRL rules.

3.2 Modeling the authorization policies

SWRL extends the possibilities of manipulating and expressing information in OWL ontologies. Specifically, SWRL rules are composed of two main components called antecedent and consequent. Where both components are a set of atoms united by the conjunction operation '∧' and variables start with a question mark symbol '?'.

SWRL provides a set of built-ins, which includes XQuery[58] and XPath[59] built-ins from XML Schema. Additionally, there are built-ins for comparisons, mathematical operations, strings manipulation, date and time, and lists. A complete list of SWRL built-ins can be found in [55].

In the SWRL examples given in Figure 3.2 the first rule expresses the following: If 'x' has a father 'y' and 'y' has a brother 'z' then 'x' has a uncle 'z'. While the second rules expresses that If person 'p'

has age ‘a’ and age ‘a’ is greater than 17 then ‘p’ is an adult. Note that *swrlb* is the namespace tag for SWRL built-in functions.

Antecedent ⇒ Consequent

```
hasParent(?x,?y) ∧ hasBrother(?y,?z) ⇒ hasUncle(?x,?z)

Person(?p) ∧ hasAge(?p,?a) ∧ swrlb:greaterThan(?a, 17)⇒ Adult(?p)
```

Figure 3.2: SWRL examples

Going deeper into this topic, in section 2.4 some HIS requirements were discussed. Some of these are the “Break the glass” phenomenon, role hierarchy, and static separation of duty. In the paragraphs below we will see how the proposed approach can be used to represent these requirements.

Firstly, the Break the glass (BTG) phenomenon can be easily translated into a fact in the knowledge base: **isBTG(?b)**. If we consider the following rule: **Subject(?s) ∧ Patient(?p) ∧ shareLocation(?s, ?p) ∧ hasRole(?s, Doctor) → canViewPatientDetails(?s, ?p)**, which indicates that a doctor who shares the same location as a patient can view the patient´s details. We can adapt this rule to support the BTG phenomenon by adding the rule: **Subject(?s) ∧ Patient(?p) ∧ isBTG(?b) → canViewPatientDetails(?s, ?p)**, which allows a subject to view patient´s details in case of a BTG situation. It is important to note that this is the least restrictive case possible. In other circumstances it would be possible to restrict access even in emergency situations.

Secondly, a role hierarchy is easily designed using semantic rules. The concrete rule for enabling hierarchies is: **hasRole(?s, ?r2) ∧ isSubrole(?r1, ?r2) → hasRole(?s, ?r1)**, which simply relates a role with all its subroles.

Thirdly, due to the introduction of roles hierarchy, conflicts among roles may appear. That is the reason why enforcing a static separation of duty can resolve this conflict of interest. The rule that corresponds to this requirement is**: hasRole(?s, ?r2) ∧ isIncompatible(?r1, ?r2) → canAssignRole(?s, ?r1)**, which evaluates if a new role is incompatible with the current assigned roles.

These rules have been an example of how to represent system requirements using SWRL. Although a collection of rules has not been provided as part of this thesis, this section illustrated how it is possible to do create a repertory.

3.3 Deciding about authorization requests

Rule-based languages are perfectly complemented by rule engines or semantic reasoners because they are in charge of inferring logical consequences from a set of asserted facts. Therefore, they are needed to infer new knowledge and consequently decide whether the authorization requested complies with the policies.

In the semantic web area, good open source semantic reasoners are available. This is because the semantic web plays an important role in many organizations from different sectors. Real life examples can be found in [60], including small and large companies that have created both research and industrial products based on semantic web technologies.

We have considered five different semantic reasoners out of the many open source rule engine alternatives. These five are described briefly below.

Jena[61] is a Java framework that provides tools for building semantic web applications. It offers an API for working with RDF data in different formats, an API for managing OWL and RDFS ontologies, a

rule-based inference engine for RDF and OWL data, an efficient approach for storing RDF information on disk, a query engine based on SPARQL, and publication of data to external applications.

Jess[62] is a rule engine for the Java language. Jess is an extension of the CLIPS[63] programming language and provides rule-based programming for reasoning in expert systems. It accepts two different rule formats, the Jess rule language and XML. It provides an API that makes it easy to integrate Jess inside a custom Java application.

Pellet[64][65] is a Java-based open source reasoner for OWL 2. It offers an API that features ontology analysis and repair, entailment, query, data type reasoning, user-defined data types, multi-ontology reasoning, and ontology debugging. It is possible to use Pellet together with both Jena and OWL APIs. Additionally Pellet has a built-in DIG interface [66], which is a XML-based standard interface for Description Logics.

Bossam[67] is another inference engine designed for the semantic web. It is based on the Rete algorithm [68] and supports different types of ontologies. In practice, Bossam can be used for inferencing and querying different types of documents, such as Bossam rule, RDFS, OWL, and SWRL documents.

SWRLTab[69] is an environment for developing SWRL rules in Protégé. Additionally, it offers a set of libraries for using rules, working with XML-based files, spreadsheets, and a variety of other libraries. It supports a query language called SQWRL [70], for SWRL rules included in OWL models.

When comparing one reasoning engine with others, it is important to consider the implementation language, portability, reasoning logic, and performance. However, performance is rather difficult to compare as each of these reasoners behaves differently depending on the specifics of the knowledge they are reasoning with. Table 3.1 shows a basic comparison of the five different rule engines that have been introduced above.

Table 3.1: Semantic reasoners

Reasoner	Licensing	Implementation language	Rule support	Documentation availability
Jena	Free/open source	Java	Jena rules	User manual/Java docs
Jess	Free/closed source	Java	Jess rules/XML	User manual/Java doc
Pellet	Free/open source	Java	SWRL	Java docs
Bossam	Free/closed source	Java	SWRL/Bossam rules	User manual/Java docs
SWRLTab	Free/open source	Java	SWRL	User manual/Java docs

The table illustrates the prevalence of Java when building rule engines for the semantic web. However, some of these projects have developed their own rule language, instead of implementing a complete support system for the SWRL rules specification. That is obviously not desirable as it does not motivate standardization for this kind of system.

After considering these five different alternatives, it was concluded that SWRLTab is the best option, because it offers full support for SWRL rules and it is also completely embedded in the Protégé development environment.

SWRLTab implements a number of different features to work with SWRL rules. Table 3.2 shows the different components included in SWRLTab.

Table 3.2: SWRLTab software components

Component	Details
SWRL Editor	Edition of SWRL rules in an OWL ontology
SWRL APIs	Libraries to work with SWRL rules
SWRL Built-ins	Defined libraries used to work with SWRL and OWL
SQWRL Query Tab	Graphical interface for SQWRL
SQWRL Query API	Interface to a data access technology used to retrieve SQWRL results of queries
SWRL Temporal Ontology and Library	Tools for modeling and manipulating temporal information
SWRL Built-in Bridge	Mechanism for defining Java-based implementations of SWRL built-ins
SWRL Bridge	Infrastructure that facilitates integration with external rule engines.
SWRL Jess Bridge	Structure for integrating Jess rule engine
SWRL Factory	APIs that allow developers to work with SWRL rules

3.4 Logging the authorization requests

Implementing appropriate logging is an important part of the software development process as it allows an application to report the inner context of its execution. This context information can stored permanently and can be analyzed afterwards. In the context of this thesis project this logging is important for two main reasons: (1) to facilitate the development of the system and (2) to provide an audit trail which could be examined *a posteriori* to find miss-use of access to patient information.

Java Logging [71] is a technology natively introduced in the Java *util* package. It mainly offers logging services to capture different kinds of information within an application. It supports multiple output channels for delivering log reports. Furthermore, its API is capable of working with other logging frameworks that are already functioning in the host system.

Logback[72] is the successor of the Log4J[73] project. Logback is divided into three components: *logback-core, logback-classic and logback-access*. The first of these, *logback-core*, is the foundation for the rest of modules. The second, *logback-classic* is an improved version of Log4J[73] and supports Simple Logging Facade 4 Java (SLF4J) – see the description below. Therefore, it is possible to use

alternative logging implementations such as the native Log4J[73] and the Java Logging[71]. Finally, *logback-access* provides integration with Servlet containers.

Commons Logging (ACL)[74] is a package that provides a bridge for different popular logging frameworks, such as Log4J[73] and Avalon LogKit[75]. Additionally, it provides an interface for writing adapters for other logging frameworks.

Simple Logging Facade 4 Java (SLF4J)[76] is an abstraction tool for several logging frameworks, such as Java Logging[71], Log4J[73], or Logback[72] It offers a pluggable interface for a variety of logging tools at deployment time. An extraordinary number of projects depend on SLF4J. A list of these projects can be found at the project's webpage [76].

A comparison of these different logging technologies is given in Table 3.3. The first impression when analyzing the different logging alternatives is that these tools offer quite similar features. These are all open source technologies, written in the Java programming language, and provide quite complete documentation for developers. However, there are a couple of important differences among them. The first one, log levels, is the number of different priorities that can be assigned to a log message and the second one, standard *appenders*, is the number of already implemented listeners or output channels for the logging messages.

Table 3.3: Logging technologies

	Tool	Licensing	Implementation language	Number of Log levels	Number of Standard *appenders*	Documentation availability
Framework	Java Logging	Free/open source	Java	7	5	User manual/Java docs
	Logback	Free/open source	Java	5	8	User manual/Java docs
Wrappers	ACL	Free/open source	Java	6	Depends	User manual/Java docs
	SLF4J	Free/open source	Java	5	Depends	User manual/Java docs

Additionally, there is even a more interesting distinction between the entries in Table 3.3 as two of these technologies are logging frameworks, while the two other are wrappers for other logging technologies. These wrappers provide logging interfaces that can switch to different underlying logging APIs. Each of these wrappers already implements support for some of these logging technologies. Therefore, an implementation of an application should use a logging wrapper (bridge) as this is more flexible and offers a polymorphic alternative if there is a need to change the core logging technology. Both ACL and SLF4J are potential alternatives. We have chosen to use ACL as it is a project of the Apache Foundation and therefore supported by the Apache community, while SLF4J is developed by a company called Quality Open Software.

There are two basic logging methods useful in conjunction with ACL. A log that is a simple and basic logger and a *LogFactory* which offers more advanced features for using customized implementations of logging tools. ACL is simply a wrapper for using another logging framework and by default it uses the Log4J tool.

Log4J is a very popular logging tool that is appropriate for the logging needs of this project. However, as it will be used through the ACL wrapper, it is always possible to switch to a different logging framework.

In Log4J, there are different levels that may be assigned to log messages. A list of these levels can be seen in Table 3.4.

Table 3.4: Message levels in Log4J

Message levels	Level explanation
FATAL	Important errors that cause execution termination
ERROR	Other runtime errors and unexpected failures
WARN	Use of deprecated or non-recommendable APIs, and other undesirable situations
INFO	Important runtime events such as startup and shutdown
DEBUG	Detailed information about runtime flow
TRACE	Even more detailed information about runtime flow

The following three rules give a good overview of the Log4J functioning:

- Named hierarchy: A logger is a *child* of another logger if its name followed by a dot is a prefix of the *parent* logger's name. For example, *java* is a parent of *java.util* and an ancestor of *java.util.List*.
- Level inheritance: If a logger does not receive a level, it inherits one from the closest ancestor that has a level assigned.
- Basic selection rule: Considering the hierarchy of standard levels *DEBUG* < *INFO* < *WARN* < *ERROR* < *FATAL*. A log request of level p is enabled in a q level logger, only if $p \geq q$.

Furthermore, Log4J allows sending log requests to multiple destinations. Technically, an output or a print request is an *appender*.

Appenders are inherited additively from the logger hierarchy. Therefore, an output of a logger L will go to the *appenders* associated with L and its ancestors. However, this additive property can be disabled by setting the additive flag to false. So, if an ancestor of L, named P, deactivates the additive property an output request to L will be redirected to the *appenders* in L but **not** to the *appenders* or any of the ancestors of P.

In Log4J, it is possible not only to customize the output destination but also to customize the output format. This is accomplished by using layouts. The *PatternLayout*[77] class is responsible for this task.

PatternLayout offers conversion patterns. A conversion pattern is composed of format control expressions named *conversion specifiers*. A *conversion specifier* starts with a percent sign (%), it optionally includes a format modifier that manages characteristics such as field width, padding, and justification. Format modifiers are followed by conversion characters, in charge of type of data, priority, date, thread name, and so on.

Log4J is completely configurable. Configuration files can be written in XML or in Java properties format. The *BasicConfigurator* class, included in the Log4J library, sets the default configuration. More concretely, it assigns a DEBUG level to the root logger, *ConsoleAppender* is initialized to be the root

logger, and a default output format using the *PatternLayout* class is set for the calling logger. If you wish to set a different configuration, more details and examples about how to indicate parameters (such as logger levels, *appenders*, and pattern layouts) can be found in [78].

To give a more real-world approach for dealing with multiple clients and to provide a light method to uniquely catch all log requests from the same client, Log4J implements a pattern called Nested Diagnostic Contexts (NDC)[79].

NDC manages different connections with a stack of contextual information on a per thread basis. In other words, NDC keeps a stack of different contexts, which are uniquely identified by information such as a client host name or by the use cookies. A basic overview of the NDC source code is shown in Figure 3.3: NDC class. Note that these operations only affect the NDC of the current thread.

```
public class NDC {

    // Used when printing the diagnostic
    public static String get();
    // Remove the top of the context from the NDC.
    public static String pop();
    // Add diagnostic context for the current thread.
    public static void push(String message);
    // Remove the diagnostic context for this thread.
    public static void remove();
}
```

Figure 3.3: NDC class

3.5 Integration with an EHR system

Fortunately, there are several tools available on the Internet that aim to offer EHR support. So, an obvious decision was to try to integrate the authorization system proposed in this work with some of the open source EHR projects. In the next subsection we will examine three alternative open source HER systems. Based upon a comparison between these three systems we selected one of them. Further details of this system are given in section 3.5.2.

3.5.1 Three open source EHR systems

Although there is a long list of such systems, there are only three projects described in this section. These three were chosen based upon their popularity and the availability of supporting documentation. Below we introduce three of the most relevant projects in the medical open source community.

Open Electronic Medical Record (OpenEMR)[80][81] is open source medical software developed by the non-profit organization OEMR. The purpose is to ensure good universal medical care. It provides mechanisms for managing patient demographics, patient scheduling, prescriptions, billing, and reporting. Additionally, it supports multiple languages, offers online documentation, and is maintained by a dedicated community. OpenEMR is coded using PHP and supports RBAC. The privileges that can be assigned to the user roles are predefined and can be found in [82].

Patient Open Source (PatientOS)[83] is a HIS. Its architecture has been built to cope with the complexities of an Enterprise Information System (EIS). It supports scheduling, clinical and administrative forms, medications, prescriptions, laboratory reporting, and orders. From the technical point of view [84], PatientOS is a Java-based web application, driven by the Model-View-Controller

(MVC) pattern[85], uses Hibernate[86] as object relational mapper, is database independent, and according to the list of features [87] it implements Role-based security.

Open Medical Record System (OpenMRS)[88] is a project that aims to improve the information management tools through an open source medical record system platform for developing countries. It is mostly used in Africa and supported by many and different organizations. In contrast to the two previous systems, OpenMRS provides a more flexible implementation of the medical domain. To support this, it implements tools for creating reports, extending personal details, entering and exporting data, searching in a dictionary for health-care terminology, supporting complex data, and ultimately multiple languages. It provides a base that can be adapted to the local needs of the system. OpenMRS is a web application written in Java, follows a Model-View-Controller (MVC) pattern[85] through the Spring framework[89], uses Hibernate[86] as object relational mapper, MySQL[90] as a database engine, and implements RBAC.

Table 3.5: Open source EHR

EHR	Licensing	Implementation language	Access control	Documentation availability
OpenEMR	Free/open source	PHP	RBAC	User manual/API docs
PatientOS	Free/open source	Java	RBAC	-
OpenMRS	Free/open source	Java	RBAC	User manual/Java docs

After analyzing these different projects, OpenMRS was selected as an EHR system that could serve as a valid scenario. The main reasons for selecting it were that it is implemented in Java and it provides quite a lot of useful documentation. While examining these different systems we learned that most open source EHR systems use RBAC because health-care environments are commonly characterized by a hierarchy of employees (as are most organizations). Although roles matter within a health-care context, access control in health-care scenarios frequently involves more complex factors, thus simply considering roles is not sufficient for authorizing access to sensitive medical information. The method proposed in this thesis project would extend the existing authorization scheme can could potentially provide a more accurate approach to making authorization decisions.

3.5.2 A deeper look into OpenMRS

OpenMRS has a quite abstract internal structure that could be difficult to understand. However, as previously mentioned, this allows a lot of flexibility - even to the point of creating an almost completely customized EHR. Internally, OpenMRS is a very scalable and flexible system. It is divided into a number of different domains. These domains are listed in Table 3.6: OpenMRS domains.

For each one of these elements there is a service interface class responsible for managing the domain from the application's backbone. In addition, there is a service interface implementation class that provides a default implementation of the service class. Furthermore, there exists a Data Access Object (DAO) for each one of the services. The DAO is a design pattern that offers an interface to a database system. Therefore, there is also a concrete implementation class for each DAO interface. For example, *PatientService* is a service interface class, while *PatientServiceImpl* would be the default implementation of the service class, with a *PatientDAO* interface class and its implementation class named *HibernatePatientDAO*.

Table 3.6: OpenMRS domains

Domain	Details
Concept	Data used to support system's mission
Encounter	Data obtained when a health-care provider intervenes with a patient
Form	User-Interface element s of multiple components
Observation	Health care information obtained from encounters
Order	Requested things or actions
Patient	Basic patients data
User	Basic users data
Person	Basic persons data
Business	Non-medical administrative information
Group/Workflow	Basic groups data

Regarding access control, OpenMRS uses a hierarchical variant of RBAC, where roles, assigned to system users, simply contain privileges. Some examples of privileges would be *AddPatient* or *AddConcept*. In this case, privileges are associated with services combining Aspect Oriented Programming (AOP) and the Spring framework [91]. Further details of how AOP is used for authentication and authorization purposes in the OpenMRS project on the webpage [92].

In AOP, annotations add meta-data about applications. More specifically, annotations can be applied to declarations of classes, fields, methods, among other program elements. Some examples of Java built-in annotations applied to code are: **Override** (indicating that a method is overridden) or **Deprecated (**indicating a method is obsolete). Other examples of built-in Java annotations applied to annotations are: **Retention** (indicates if the annotation is available in code, in the compiled class, or at runtime) or **Target** (that restricts which kind of elements the annotation may be applied to).

Regarding access control in OpenMRS, there are three classes directly involved in the authorization process. Firstly, **Authorized** is an interface corresponding to the annotation and defines a list of privileges and a Boolean attribute called *requireAll* that when true requires all privileges to be assigned to the user roles. Some examples of the use of this annotation can be found in Appendix 63□A. Secondly, *AuthorizedAnnotationAttributes* is a supportive class that includes different methods for getting information present in an Authorized annotation. Finally, the core of the authorization decision process in OpenMRS can be found in the *AuthorizationAdvice* class. This class implements an AOP advice, which is an operation executed at a particular join point. More concretely, *AuthorizationAdvice* is a "before" type of advice, which means that is executed before every service layer method. An overview of these classes and an example of the Authorized interface can be found in the Appendix 63□A.

The different OpenMRS webpages require that the user has specific privileges to access them. This is managed by two controllers, namely **require** tag and **privilege** tag. These are implemented in the *RequireTag* and *PrivilegeTag* classes, respectively.

The require tag checks user privileges for viewing a webpage. Figure 3.4 gives an example of this tag in action. In this case, it specifies that the user must have the *AddPatient* privilege to access the *addPatients.form* resource, otherwise the request will be redirected to the login.htm resource. Similarly the privilege tag checks user privileges for viewing webpage sections. An example of the privilege tag is shown in Figure 3.5.

```
<openmrs:require privilege="Add Patient" otherwise="/login.htm"
        redirect="/admin/patients/addPatients.form" />
```

Figure 3.4: Require tag example

```
<openmrs:hasPrivilege privilege="View Navigation Menu" />
```

Figure 3.5: Privilege tag example

3.6 Designing the authorization service

It has been already seen that authorization in an HIS is rather a complex task due to the context of the system. Therefore methods and techniques that can deal with this level of complexity are the most suitable approaches to implement a solution.

Figure 3.6 illustrates the overall system structure. Section 2.4.1 introduced a basic system architecture provided by the XACML specification. As can be seen, this adaptation has kept all of the components and mainly extends the capabilities of some of them. The following are the necessary steps for the new workflow when processing an authorization request:

1. An access request is sent by OpenMRS using an integrated interface.
2. The PEP sends a XACML request to the PDP.
3. The PDP sends a XACML request to retrieve semantic policy information.
4. The PDP receives the requested semantic policy data.
5. The PDP sends an attribute request to the PIP.
6. Supplementary sources of information might be available for feeding the PIP.
7. The PIP replies with an attribute response message back to the PDP.
8. The PDP passes the collected knowledge to the reasoner for evaluation, i.e., to reach a decision.
9. The reasoner responds with an output regarding its access decision.
10. The PDP sends relevant data to the logging tool.
11. The PDP sends a XACML response back to the PEP.
12. Finally based upon the PDP's response, the PEP either allows access or denies access to the requested information.

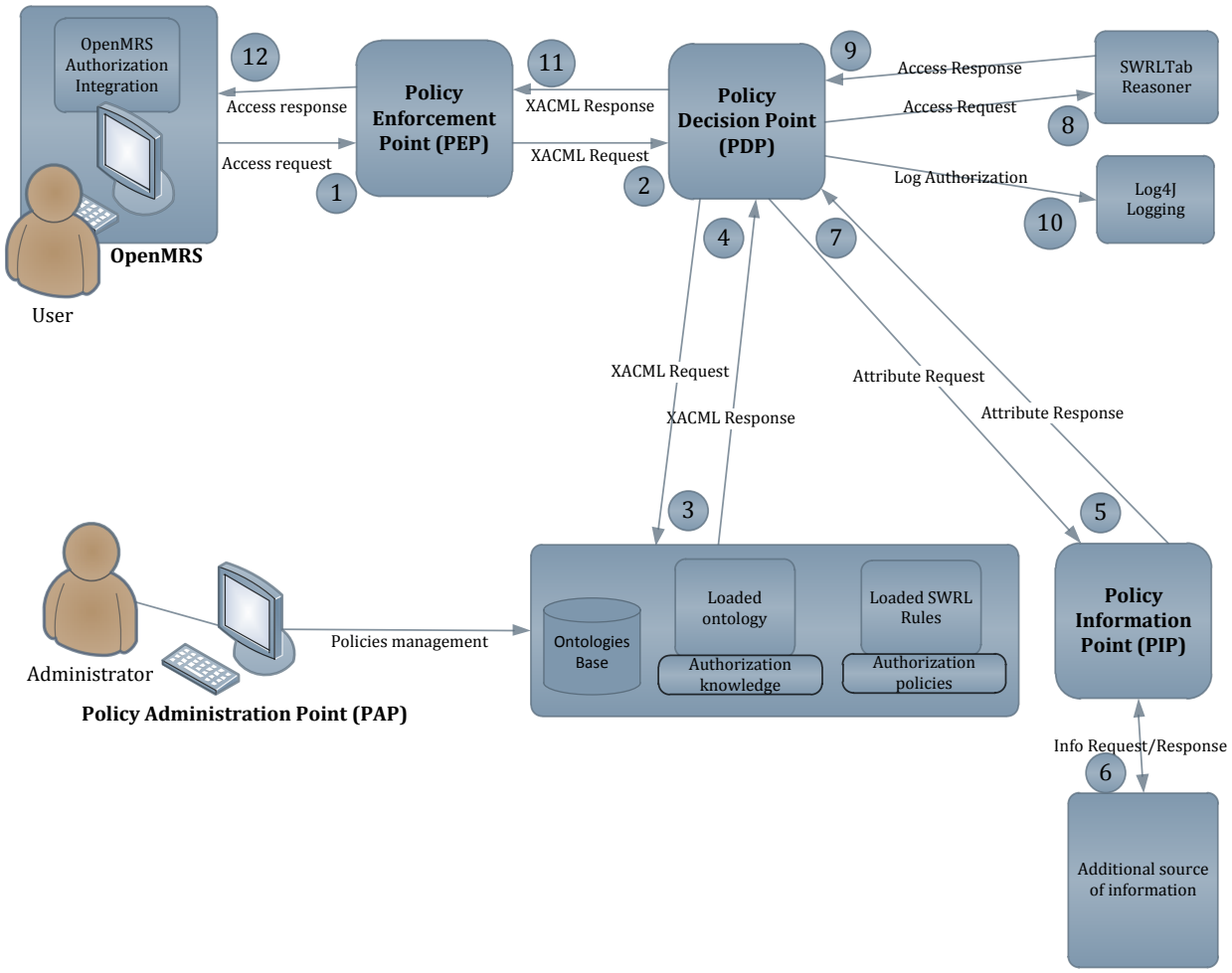


Figure 3.6: Proposed authorization infrastructure

4 Analysis

This chapter describes what has been accomplished with regard to the previously identified problems. It gives a practical explanation of how to build the proposed authorization mechanism with a special emphasis on the underlying software. This analysis is structured based upon the different policy components that are part of the authorization system as a whole. Section 4.1 introduces the PAP. Section 4.2 continues with the PEP's architecture. Section 4.3 discusses building the PDP. Finally, section 4.5 consolidates all these components into a single system.

4.1 Building the Policy Administration Point

This section introduces a practical solution that begins with building the PAP component. The PAP allows administrators to manage the different data and policies directly involved in the authorization process. The most important elements of the PAP are ontologies, rules, and logs. Each of these will be described in its own subsection.

As Java is one of the most widely used technologies for semantic web projects we have chosen to use this programming language for creating all of our new components. We expect that Java offers a scalable base for the implementation of the final system.

4.1.1 Ontologies administration

Ontologies are a great technique for representing important information that may be relevant for the authorization process. The Protégé OWL API is used to manage ontologies in the PAP component.

The Protégé OWL API [93] is an open source Java library for OWL and RDF(S). Its main features are creating, loading, manipulating, storing, and reasoning on OWL models. An online programmer's guide [94] explains how to install and configure the framework and gives a few examples of the most relevant topics.

The OntologiesController class is responsible for managing the existing ontologies in the system. Figure 4.1 shown an overview of its structure with the most important attributes and methods.

This class is in charge of maintaining a list of the available ontologies, a default ontology that will be initially loaded, and storing the current ontology in use in the attribute *loaded_ontology*. Furthermore, this class offers basic management functions (such as *addOntology*, *updateOntology*, and *deleteOntology*) and more advanced functions for querying the loaded ontology. These are *getClasses*, *getIndividuals*, and *getProperties* methods.

```
public class OntologiesController {

	// Default ontology
	private static Ontology default_ontology;
	// Ontology currently loaded, initially the default ontology
	private static Ontology loaded_ontology;
	// OWL Model currently loaded
	private static OWLModel loaded_model;
	// List of existing ontologies
	private static ArrayList<Ontology> ontologies;

	// Add a new ontology
	public static String addOntology(String name, String path);
	// Update an existing ontology
	public static String updateOntology(String old_name, String old_path,
		String name, String path);
	// Delete an existing ontology
	public static String deleteOntology(String name);
	// Gets all classes and instances from an ontology
	public static String viewOntology(String name);
	// Load an existing ontology
	public static String loadOntology(String name);
	// Obtain classes of the loaded model
	public static ArrayList<String> getClasses();
	// Obtain individuals corresponding to a classes of the loaded model
	public static ArrayList<String> getIndividuals(String cls_name);
	// Obtain properties corresponding to a class of the loaded model
	public static SortedMap<String, String> getProperties(String cls_name);
}
```

Figure 4.1: OntologiesController class

4.1.2 Rules administration

A key component of this system is the rules that represent and comply with the authorization policy. Rules are related to an ontology that serves as the knowledge representation. Additionally, rules are a tool for interpreting this structured information.

For the concrete task of managing SWRL rules we use the SWRL Factory [95]. The *RulesController* is the class in charge of managing the SWRL rules in the loaded ontology. An outline of this class can be found in Figure 4.2. This class uses an instance of the SWRLFactory class for performing its different assignments. These are basic rule manipulating methods, such as *parseSWRLRule*, *createSWRLRule*, *updateSWRLRule*, and *deleteSWRLRule*. Additionally, it is possible to retrieve a complete set of SWRL rules using the *getSWRLRules* call and a list of the SWRL built-ins by calling the *getBuiltins* method.

```
public class RulesController {

	private static SWRLFactory factory;

	//Load a new OWL model
	public static void loadModel();
	// Parse an SWRL rule
	public static String parseSWRLRule(String name, String rule);
	// Create an SWRL rule
	public static String createSWRLRule(String name, String rule);
	// Update an SWRL rule
	public static String updateSWRLRule(String old_name, String old_rule,
		String name, String rule);
	// Delete an SWRL rule
	public static String deleteSWRLRule(String name);
	// Obtain SWRL Built-in functions
	public static ArrayList<String> getBuiltins();
	// Obtain SWRL rules present in the loaded model
	public static ArrayList<String[]> getSWRLRules();

	public static SWRLFactory getRuleEngine();
}
```

Figure 4.2: RulesController class

4.1.3 Logging administration

The final component, logging, is responsible for tracking authorization decisions in the system. Fortunately, OpenMRS uses ACL. This was the same logging tool identified in section 3.4 as the most appropriate choice for logging purposes in the authorization system. Our main interest is in logging authorization decisions. An access decision is simply a piece interesting information and according to the ACL best practices, the INFO, DEBUG, and TRACE levels should be used for this purpose.

Logging is mostly needed in exceptional cases, such as the BTG phenomenon. In this situation, it is crucial to track authorization requests as there may be subsequent charges of negligence in the use of sensitive information. To support *a posteriori* analysis all authorization requests must be logged and stored for future audits. Note that in these exceptional cases, the response need *not* be logged as the overriding policy is to always authorize requests in exceptional cases.

In a normal situation, which basically means absence of medical emergencies, both authorization requests and responses could be logged depending on the authorization's policies. Authorization responses need **not** be logged, as the response could be regenerated from the request and the current set of SWRL rules. Note that this implies that any changes to these rules must be logged.

In normal situations the proper functioning of the system need not be tracked because the authorization policies shall be complied with at all times and logging these requests and responses would not be useful for detecting misuse of the information (assuming that the correct set of policies have been implemented. However, activating logging of all authorization requests and responses could be interesting for analytical purposes. One of the uses for this logging data could be to improve the set of policies and

their encoding as rules. On the other hand, deactivating logging of normal requests and responses could positively affect the performance of the authorization decision process.

Requests and responses will be captured in the PDP because it is the point where the decision is made and therefore all the required information is already available. While capturing the relevant information is quite important for security reasons. It is even more critical to have a dashboard, where it is possible to analyze and detect accesses during BTG and ultimately to assess the necessity of that action, therefore this information is essential is a subsequent audit.

OpenMRS currently only captures audit information regarding changes in the different database tables and does not implement a means for BTG. Thus adding a means of handling BTG could be considered as a substantial improvement for this EHR system.

4.2 Building the Policy Enforcement Point

This section presents how to construct the PEP with the use of available open source tools. To achieve this, OpenAz[96][97] was selected as a good starting point as it provides a robust enforcement component. More specifically, the OpenAz project (a joint effort between Oracle and Cisco) offers a set of instruments for implementing access control in software applications.

Java AzApi is an interface responsible for requesting authorization decisions. This API is also based in the specifications given by the XACML for request and response of authorization decisions. Furthermore, the OpenAz implementation includes Sun's XACML library[98], originally developed by Sun Microsystems (now part of Oracle). This implementation completely supports all of XACML. Two additional advantages of Java AzApi that are worth highlighting are: its support for packing multiple authorization decisions into a single authorization request and scope limitation of authorization queries.

DecisionService is the class that receives an authorization request and makes a decision. This class is shown in Figure 4.3. *decide* is the key method that analyzes the request in terms of its parameters and associated values. *decide* calls the *getAzDecisionValue* method that delegates the decision to the PDP and processes its response. Finally, both *getStatusMessage* and *getStatusCode* are supporting methods that can give more information about a request. This information is useful, for example, to obtain more details when a request has been denied or unresolved.

```
public class DecisionService extends AbstractService {
	//Main authorization decision flow
	public AzResponseContext decide(AzRequestContext azReqCtx);
	//PDP decision delegation and response treatment
	public AzDecision getAzDecisionValue();
	//Message corresponding to the response
	public String getStatusMessage();
	//Code corresponding to the request status
	public AzStatusCode getStatusCode();
}
```

Figure 4.3: DecisionService class

The *EnforcementService* class builds an XACML request from different parameters. These parameters can be either obtained from the user´s session information or directly receiving from the requestor. Figure 4.4 shows a sample of this class in action. In this figure, the different types of attributes are hash maps, and for each hash map's entry there is a specific parameter type and associated value. For example, environment attribute is characterized by the current time parameter and its value. As well as

subject attribute that contains the username parameter and its value. This example is meant to be simple and understandable, but it is possible to add as many parameters as needed. Some of these parameters could be those that appear declared such as authentication method, authentication time, DNS name, IP address and so on.

```
public class EnforcementService implements AzXacmlStrings{

	//Session parameters
	public final static String SESSION_USER_NAME;
	public final static String SESSION_AUTH_METHOD;
	public final static Date SESSION_AUTH_TIME;
	public final static String SESSION_DNS_NAME;
	public final static String SESSION_IP_ADDRESS;

	public static void main(String[] args) throws Exception {
		//Service definition
		AzService azService = new DecisionService();
		PepRequestFactoryImpl pep = new PepRequestFactoryImpl(CONTAINER,
		azService);

		HashMap<String,String> environment = new HashMap<String,String>();
		//Environment attributes
		environment.put(AzXacmlStrings.X_ATTR_ENV_CURRENT_DATE_TIME,
		String.valueOf(new Date()));

		 HashMap<String,String> subject = new HashMap<String,String>();
		 //Subject attributes
		 subject.put(AzXacmlStrings.X_ATTR_SUBJECT_ID, SESSION_USER_NAME);

		HashMap<String,String> resource = new HashMap<String,String>();
		//Resource attributes
		resource.put(AzXacmlStrings.X_ATTR_RESOURCE_ID, "PatientDetails");

		HashMap<String,String> action = new HashMap<String,String>();
		//Action attributes
		action.put(AzXacmlStrings.X_ATTR_ACTION_ID, "View");

		//Build Request
		PepRequest req = pep.newPepRequest(subject, action, resource,
		environment);
		//Action attributes
		PepResponse resp = req.decide();
}
```

Figure 4.4: EnforcementService class

4.3 Building the Policy Decision Point

Probably the most significant part of this system is the PDP, as this is where decisions are made. In this component, requests are received and though the query of SWRL rules (which could retrieve additional data from the PIP), a response is ultimately generated. In order to generate this response, a semantic reasoner is indispensable, as our authorization policies are written in a semantic rule language.

As explained in section 3.3, we have chosen to use SWRLTab. SWRLTab is composed of different parts and allows the programmer to integrate external reasoners due to the incorporated bridge libraries. In particular, it builds in the Jess reasoner through the SWRLJessTab component. Moreover, this library is used in Protégé as a tab, thus it is possible to use Protégé to edit SWRL rules.

Figure 4.5 illustrates how to initialize a Jess rule engine object. At this point, the rule engine is created and ready to execute SWRL rules and infer knowledge from the OWL model. In the SWRLRule Engine API [99] there is a brief description about this class. The methods in the *SWRLRuleEngine* class a listed in Table 4.1.

```
SWRLRuleEngine ruleEngine = SWRLRuleEngineFactory.create("SWRLJessBridge", owlModel);
```

Figure 4.5: Jess rule engine creation

Table 4.1: SWRLRuleEngine methods

Method	Description
Reset	Clears the rule engine knowledge
importSWRLRulesAndOWLKnowledge	Calls *reset* method and imports SWRL rules and OWL knowledge into the rule engine
Run	Executes the rule engine
writeInferredKnowledge2OWL	Transfers inferred information to the OWL model
Infer	Wrapper for the previous methods. It basically clears, runs the rule engine and writes back to OWL model the inferred knowledge

Logging authorization requests and responses is another of the responsibilities of the PDP component. This is because the relevant data can be immediately captured after the access decision has been made. It is quite important to ensure the availability of this data; hence this element is embedded in the PDP and not placed somewhere else. Note that this logging has to be done in a reliable manner.

4.4 Building the Policy Information Point

The policy information point (PIP) is the critical component that improves the PDP's decision making. The PIP is responsible for providing authorization information. In particularly the PIP contains the ontology database. If there is a need for different authorization ontologies, the PIP is responsible for loads the appropriate ontology and a set of SWRL rules related to that ontology. Consequently, PIP is responsible for resolving requests regarding the information contained in the loaded ontology and the different policies, represented by SWRL rules, which are applied.

Furthermore, as ontologies might not provide sufficient information in certain cases, the PIP can integrate other external systems that might provide relevant information for the authorization in context. In order to interact with these external providers, it is essential to define an interface for connecting the authorization and external system. It is also important to emphasize that when referring to the external system this component could be remotely located and even operated by another organization. This aspect of the PIP is not part of this thesis project and is proposed as future work.

4.5 Integrating the authorization system

After defining the different components of the authorization system, it is time to think about how to embed it into an EHR. For this task, OpenMRS will be the candidate EHR and its authorization components have been identified and previously discussed.

In order to integrate our new authorization mechanism with OpenMRS we need to consider whether we need to modify the three components involved in the authorization process in OpenMRS. These are the Authorized, *AuthorizedAnnotationAttributes*, and *AuthorizationAdvice* classes.

Authorized is an interface that identifies which privileges are necessary for a given access. In the case of OpenMRS the access will be made by Java methods. This element will remain unchanged because it already contains the necessary information and the rest of data can be captured in *a posterior* step of the authorization process. *AuthorizedAnnotationAttributes* is a class that supports the extraction of contextual information for the Authorized annotation. This class will be also kept in its original form.

Only the *AuthorizationAdvice* class will require major changes, as it implements the RBAC. These modifications will include attribute collection and implementation of the proposed access control system.

At this point, we have to consider the different privileges that the user must have in order to access to a given method. In RBAC, this would be checked by looking at the user's roles. However, in our proposed approach it is important to be aware of which are the different and necessary attributes for a given privilege authorization decision. This raises the need of an auxiliary service that specifies the attributes that depend on each one of the privilege authorization decisions. Fortunately, the OpenAz developers thought about the same issue and consequently they defined the Attribute Manifest File (AMF)[100]. There is an available sample of such a file in Appendix A.

AMF is simply an XML-based format for exchanging metadata about attributes which may be part of an access control decision. This information enables the PIP to know what attributes should be provided, where this information is located, and how it can be retrieved. A number of examples of AMF in action extracting attribute information from different sources, such as LDAP directories, SQL databases, and SAML attribute authority can be found in [100].

5 Conclusions and Future Work

The purpose of this chapter is to gather the results achieved as part of this thesis project. To begin a set of final conclusions will be drawn in section 5.1. Following this, section 5.2 discusses the problems found when working on this thesis project. Finally, section 5.3 gives some ideas that could be taken into consideration to improve the system presented in this thesis.

5.1 Conclusions

The initial idea behind this project was to introduce an appropriate authorization system in a health-care environment. As a result of this master's thesis project, an advanced access control model has been proposed for an authorization mechanism in a health-care information system. Although the system has not been completely implemented and tested, most of the relevant concepts needed for implementing a functional prototype can be found in this thesis. We presume that from these ideas and our experiments with parts of the system that it would be possible to build the system and integrate it with OpenMRS.

Authorization is one of the most common issues to found in an information system. As with other complex problems, solving the problem of authorization in a health-care system usually requires advanced techniques. Therefore we have approached our solution by utilizing advanced techniques that have already been developed and are in wide use. Semantic web ontologies proved to be a key component to achieve this objective. However, such advanced techniques are likely to have their own advantages and drawbacks. So, it is imperative to assess the matching the problem's requirements with the method's features. We have attempted to do this as described in Chapter 4.

Moreover, there has been an important emphasis throughout this project on the target users of this proposal. These target users are medical centers in developing countries that are in need of a low-cost and efficient EHR and HIS. I strongly believe that the open source community has provided an excellent set of components that can be used to realize such a health-care system.

Due to working on this project, I gained knowledge about computer security, specifically user authorization techniques. Additionally, I dealt with different medical terminology and how a hospital's flowcharts are designed. As this project was initially carried out in a multidisciplinary team of professionals I learnt how to model systems based upon the stakeholders' requirements. Furthermore, I applied my technical knowledge to identify a system that complies with these initial user requirements.

If I had to do this project again, I would have invested more time in identifying advanced techniques for information systems, rather than focusing on methods currently used by existing health-care information systems. While I acquired a lot of knowledge about how these existing systems work, I think I should have dedicate more time to explore more advanced and innovative solutions. I believe that these solutions would allow me to realized the proposed solution, rather than simply giving a theoretical proposal for a solution. Additionally, I would have proposed more details about alternative frameworks to be integrated, as some of them might come with bugs that need to be fixes as exposed by [101].

5.2 Discussions

Although the tool proposed in this thesis introduces an innovative approach for authorization in EHR systems, some of the parts have only been briefly explained. One of the main contributions of this thesis is its proposal for a structure for solving the problem of authorization in EHR systems, but there remains room for further innovation in most aspects of this proposal.

In the earliest development phase, different existing techniques were considered for solving the problem. After measuring the inaccuracy of these methods, an alternative and considerably more modern approach was taken. Due to the lack of a specific working EHR system, this project took place in somewhat of an information vacuum. However, this did enable me to consider a number of different EHR systems and find their similarities, thus my proposed solution is perhaps stronger as it is not specialized for one specific EHR system.

One of the major open questions concerning the proposed solution is how it will perform in reality. Any real-world solution will need to offer its users a good response time, otherwise the system will be of no practical use. This is one of the most likely reasons why most of the current EHR systems use simpler (but inaccurate) methods for authorization. It remains to be seeing if a real implementation will meet the required response time requirements of a potentially large number of users. However, one of the advantages of the proposed solution is that the processing of authorization requests is parallelizable (i.e., different user's requests can be processed independently of other user's requests) – if the policies do not have temporal and cross user dependencies.

5.3 Future work

The solution proposed in this thesis is quite well aligned with an innovative approach to authorization in health-care information systems. There is plenty of room for future improvements and research in this specific topic. Some of the suggested future work will be introduced below.

An initial idea comes from working with ontologies as a knowledge base. A clear restriction of this kind of model is its static nature. That means that the data represented is not expected to change over time. Any changes require the ontology model to be reloaded. Therefore a "dynamic ontology" would be an interesting concept to introduce. In this way an up to date knowledge base would be available and consequently this could improve the quality of the authorization response. However, ontology and dynamic may be incompatible terms; hence it is necessary to research to what extent it is possible to add a time-variable behavior to this knowledge representation.

Another interesting issue concerns how authorization requests are built. This issue mainly concerns the different information that is needed for resolving an authorization request. More concretely, it is possible to have a great variety of information available for the different SWRL rules, but there is not a standard procedure to capture this data from the different information sources. For example, it would be interesting to define a standard working procedure of how to resolve conflicts if the same information can be retrieved from multiple sources and how to assemble these different instance into a single piece of information.

Furthermore, some of the information that might be useful for an authorization decision might be retrieved from remote systems or even requires that certain calculations be performed to obtaining it. Therefore, it would be quite interesting to investigate to what extent caching techniques can be used to provide faster availability of the relevant information and consequently enable quicker authorization responses. Although caching is a good mechanism for providing fast information availability, it would be necessary to understand what kind of cache and where to place the cache in the system's workflow. Additionally, how the cache entries should be invalidated must be addressed.

One of the greatest future steps for this work would be a complete implementation in a real health care environment. Undoubtedly, that would determine the effectiveness of this approach and also identify the parts that may need to be changed or improved. It is obvious that this proposal is costly in terms of time compared to the typically used access control models, as it implements a more complex structure. However, it is as important to note the continuing improvements in hardware and the use of cloud computing as a future trend. Both of these will allow running more complex applications while keeping the response time with in an acceptable bound.

Moreover, as part of this thesis, relevant indications have been given for completely building the proposed authorization system. Therefore, it is mainly a matter of setting up all the discussed technical components and performing both functional and performance testing to assess the effectiveness and appropriateness of the proposed access control system.

A further great step would be making the distinction between different levels of policies, such as local and regional policies. This would allow the introduction of more homogenous authorization policies among different medical centers that operate under the same organization. An Additional point would be deciding about policies in case of conflicts between two policies at different levels. In fact, the semantic reasoning already implements a conflict resolution algorithm, thus this feature would not be difficult to add.

5.4 Required reflections

This thesis project represents an important step forward to providing better protection for patient's records; hence it can potentially have a large impact on the personal integrity of these patients. As part of the ethical considerations of this thesis the author continually had to examine each design decision with regard to how it would improve the protection of each patient's records, while balancing the need to carryout potentially lifesaving procedures and meet public health needs.

As open source software the results of this thesis project can help reduce the cost of providing access control to an electronic health-care records system. This could have a potentially large economic impact by enabling the deployment of these systems, while still providing protection of the patient's health-care records.

As part of the *ICT4MPOWER* the project is expected to have a positive impact on health care in Uganda and potentially if adopted elsewhere too. Electronic health-care records are expected to have a very large social impact in detecting contagious disease and in developing effective strategies for preventing and treating various diseases. However, a perceived impediment to such systems being able to realize their potential is the potentially negative effects of such records if they are not adequately protected from misuse. Having an access control system as proposed in this thesis is one possible means for improving the protection of these records.

References

[1] "Biohealthmatics.com - Hospital Information Systems." [Online]. Available: http://www.biohealthmatics.com/technologies/intsys.aspx. [Accessed: 08-Feb-2012].

[2] F. Fedele and G. Srl, "Healthcare and Distributed Systems Technology," *Cambridge-UK: ANSAworks*, vol. 95, 1995.

[3] Sonja Venter, V. P. Shaw, and S. Muyambo, "Design and Implementation of Health Management Systems: Experiences from HISP-SA," Durban, South Africa, Jun-2008.

[4] M. A. Malik and H. R. Khan, "Understanding the implementation of an electronic hospital information system in a developing country: a case study from Pakistan," in *Proceedings of the Third Australasian Workshop on Health Informatics and Knowledge Management-Volume 97*, 2009, pp. 31–36.

[5] F. T. Igira, O. H. Titlestad, J. H. Lungo, A. Makungu, M. M. Khamis, Y. Sheikh, M. Mahundi, M. J. Ngeni, O. Suleiman, and J. Braa, "Designing and implementing hospital management information systems in developing countries: case studies from Tanzania-Zanzibar," *Health Informatics in Africa (HELINA)*, 2007.

[6] "Health Information Systems Programme." [Online]. Available: http://www.hisp.org/. [Accessed: 22-Feb-2012].

[7] Paul C. Webster, "The rise of open-source electronic health records," *The Lancet*, vol. 377, no. 9778, pp. 1641–1642, May 2011.

[8] The MITRE Corporation, Center for Enterprise Modernization, "Electronic Health Records Overview," The MITRE Corporation, McLean, Virginia, Apr. 2006.

[9] Centers for Medicare & Medicaid Services, "Electronic Health Records," 26-Mar-2012. [Online]. Available: https://www.cms.gov/Medicare/E-Health/EHealthRecords/index.html?redirect=/EHealthRecords/. [Accessed: 24-Jul-2012].

[10] Edward P. Ambinder, "Electronic health records," *Journal of oncology practice / American Society of Clinical Oncology*, vol. 1, no. 2, pp. 57–63, Jul. 2005.

[11] Centers for Medicare & Medicaid Services, "HIPAA - General Information," 12-Apr-2012. [Online]. Available: http://www.cms.gov/Regulations-and-Guidance/HIPAA-Administrative-Simplification/HIPAAGenInfo/index.html?redirect=/HIPAAGenInfo. [Accessed: 24-Jul-2012].

[12] "ISO EHR Standards - openEHR." [Online]. Available: http://www.openehr.org/standards/iso.html. [Accessed: 21-Jul-2012].

[13] "ISO/DTR 20514 – Electronic Health Record Definition, Scope and Context." [Online]. Available: http://www.openehr.org/downloads/standards/iso/isotc215wg3_N202_ISO-TR_20514_Final_%5B2005-01-31%5D.pdf. [Accessed: 21-Jul-2012].

[14] "ERS EN 13606 Product&Services Suite." [Online]. Available: http://ec.europa.eu/idabc/servlets/Doc0832.pdf?id=31920. [Accessed: 19-Jul-2012].

[15] "CEN Standards - openEHR :: future proof and flexible EHR specifications." [Online]. Available: http://www.openehr.org/standards/cen.html. [Accessed: 21-Jul-2012].

[16] "Health Level Seven International - Homepage." [Online]. Available: http://www.hl7.org/. [Accessed: 23-Feb-2012].

[17] W. A. Al-Hamdani, "XML security in healthcare web systems," in *2010 Information Security Curriculum Development Conference*, 2010, pp. 80–93.

[18] "A basic view of CDA v3.doc." [Online]. Available: http://www.hl7.org.uk/repository/uploads/565/1/A%20basic%20view%20of%20CDA%20v3.doc. [Accessed: 22-Mar-2012].

[19] "CDA - Hl7book." [Online]. Available: http://hl7book.net/index.php?title=CDA. [Accessed: 22-Mar-2012].

[20] "HL7 Emergency Access." [Online]. Available: http://www.docstoc.com/docs/79381628/HL7-Emergency-Access. [Accessed: 06-Mar-2012].
[21] "What is principle of least privilege (POLP)? - Definition from Whatis.com." [Online]. Available: http://searchsecurity.techtarget.com/definition/principle-of-least-privilege-POLP. [Accessed: 06-Mar-2012].
[22] Steven M. Bellovin, "Lecture 3:Complex access control," Columbia University, Computer Science Department, New York, NY, USA, 14-Sep-2005.
[23] Bjørn-Erik Stenbakk and Gunnar Rene Øie, "Role models in healthcare," Department Of Computer And Information Science, Norwegian University of Science And Technology, Tromsø, Norway, Specialization assignment TDT4700 Healthcare informatics, Nov. 2004.
[24] D. F. Ferraiolo and D. R. Kuhn, "Role-based access controls," *Arxiv preprint arXiv:0903.2171*, 2009.
[25] Ravi Sandhu, David Ferraiolo, and Richard Kuhn, "The NIST model for role-based access control: towards a unified standard," in *Proceedings of the fifth ACM workshop on Role-based access control*, New York, NY, USA, 2000, pp. 47–63.
[26] M. A. C. Dekker and S. Etalle, "Audit-Based Access Control for Electronic Health Records," *Electronic Notes in Theoretical Computer Science*, vol. 168, no. 0, pp. 221–236, Feb. 2007.
[27] "Attribute Based Access Control." [Online]. Available: http://www.axiomatics.com/attribute-based-access-control.html. [Accessed: 02-Apr-2012].
[28] J. Jin, G.-J. Ahn, H. Hu, M. J. Covington, and X. Zhang, "Patient-centric authorization framework for electronic healthcare services," *Computers & Security*, vol. 30, no. 2–3, pp. 116–127, May 2011.
[29] Huanchun Peng, Jun Gu, and Xiaojun Ye, "Dynamic Purpose-Based Access Control," in *Parallel and Distributed Processing with Applications, 2008. ISPA '08. International Symposium on*, 2008, pp. 695 –700.
[30] J. W. Byun, E. Bertino, and N. Li, "Purpose based access control of complex data for privacy protection," in *Proceedings of the tenth ACM symposium on Access control models and technologies*, 2005, pp. 102–110.
[31] C. A. Ardagna, S. De Capitani di Vimercati, S. Foresti, T. W. Grandison, S. Jajodia, and P. Samarati, "Access control for smarter healthcare using policy spaces," *Computers & Security*, vol. 29, no. 8, pp. 848–858, Nov. 2010.
[32] V. Kapsalis, L. Hadellis, D. Karelis, and S. Koubias, "A dynamic context-aware access control architecture for e-services," *Computers & Security*, vol. 25, no. 7, pp. 507–521, 2006.
[33] "eXtensible Access Control Markup Language (XACML) what is it and why is it important? : Coding Bliss." [Online]. Available: http://codingbliss.com/?p=161. [Accessed: 01-Apr-2012].
[34] "OASIS | Advancing open standards for the information society." [Online]. Available: http://www.oasis-open.org/. [Accessed: 01-Apr-2012].
[35] "Analysis: XACML - Computerworld." [Online]. Available: http://www.computerworld.com.au/article/40716/analysis_xacml/. [Accessed: 05-Apr-2012].
[36] "OASIS Security Services (SAML) TC | OASIS." [Online]. Available: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security. [Accessed: 05-Jun-2012].
[37] Sorcha Diver (previously Canavan), "Information Security Policy - A Development Guide for Large and Small Companies," SANS Institute InfoSec Reading Room, Jul. 2006.

[38] “Firewall Policy Advisor Project Homepage.” [Online]. Available: http://www.mnlab.cs.depaul.edu/projects/SPA/index.htm. [Accessed: 27-Mar-2012].
[39] “PHR2006 - Republic of Uganda | Privacy International.” [Online]. Available: https://www.privacyinternational.org/article/phr2006-republic-uganda. [Accessed: 02-Feb-2012].
[40] Thomas R. Gruber, “A translation approach to portable ontology specifications,” *Knowledge Acquisition*, vol. 5, no. 2, pp. 199–220, Jun. 1993.
[41] A. Dersingh, R. Liscano, and A. Jost, “Context-aware access control using semantic policies,” *Ubiquitous Computing And Communication Journal (UBICC) Special Issue on Autonomic Computing Systems and Applications*, vol. 3, pp. 19–32, 2008.
[42] “World Wide Web Consortium (W3C).” [Online]. Available: http://www.w3.org/. [Accessed: 08-Jun-2012].
[43] “Extensible Markup Language (XML).” [Online]. Available: http://www.w3.org/XML/. [Accessed: 08-Jun-2012].
[44] “W3C XML Schema.” [Online]. Available: http://www.w3.org/XML/Schema. [Accessed: 08-Jun-2012].
[45] “RDF - Semantic Web Standards.” [Online]. Available: http://www.w3.org/RDF/. [Accessed: 08-Jun-2012].
[46] “Quick Intro to RDF.” [Online]. Available: http://www.rdfabout.com/quickintro.xpd. [Accessed: 08-Jun-2012].
[47] “RDF Vocabulary Description Language 1.0: RDF Schema.” [Online]. Available: http://www.w3.org/TR/rdf-schema/. [Accessed: 08-Jun-2012].
[48] “OWL - Semantic Web Standards.” [Online]. Available: http://www.w3.org/2001/sw/wiki/OWL. [Accessed: 08-Jun-2012].
[49] “Turtle - Terse RDF Triple Language.” [Online]. Available: http://www.w3.org/TeamSubmission/turtle/. [Accessed: 08-Jun-2012].
[50] “OWL 2 Web Ontology Language XML Serialization.” [Online]. Available: http://www.w3.org/TR/2009/REC-owl2-xml-serialization-20091027/. [Accessed: 08-Jun-2012].
[51] “OWL 2 Web Ontology Language Manchester Syntax.” [Online]. Available: http://www.w3.org/TR/2009/NOTE-owl2-manchester-syntax-20091027/. [Accessed: 08-Jun-2012].
[52] “OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax.” [Online]. Available: http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/. [Accessed: 08-Jun-2012].
[53] “OWL Web Ontology Language Overview.” [Online]. Available: http://www.w3.org/TR/2004/REC-owl-features-20040210/#s1.3. [Accessed: 05-Aug-2012].
[54] “SPARQL Query Language for RDF.” [Online]. Available: http://www.w3.org/TR/rdf-sparql-query/. [Accessed: 08-Jun-2012].
[55] “SWRL: A Semantic Web Rule Language Combining OWL and RuleML.” [Online]. Available: http://www.w3.org/Submission/SWRL/. [Accessed: 09-Jun-2012].
[56] “RIF FAQ - RIF.” [Online]. Available: http://www.w3.org/2005/rules/wiki/RIF_FAQ#About_RIF. [Accessed: 08-Jun-2012].
[57] “The Protégé Ontology Editor and Knowledge Acquisition System.” [Online]. Available: http://protege.stanford.edu/. [Accessed: 10-Jun-2012].
[58] “XQuery 1.0: An XML Query Language (Second Edition).” [Online]. Available: http://www.w3.org/TR/xquery/. [Accessed: 09-Jun-2012].
[59] “XML Path Language (XPath).” [Online]. Available: http://www.w3.org/TR/xpath/. [Accessed: 09-Jun-2012].

[60] “W3C Semantic Web FAQ.” [Online]. Available: http://www.w3.org/RDF/FAQ. [Accessed: 10-Jun-2012].
[61] “Apache Jena - Apache Jena.” [Online]. Available: http://jena.apache.org/. [Accessed: 09-Jun-2012].
[62] “Jess, the Rule Engine for the Java Platform.” [Online]. Available: http://www.jessrules.com/. [Accessed: 09-Jun-2012].
[63] “CLIPS: A Tool for Building Expert Systems.” [Online]. Available: http://clipsrules.sourceforge.net/. [Accessed: 16-Jun-2012].
[64] “Pellet: OWL 2 Reasoner for Java.” [Online]. Available: http://clarkparsia.com/pellet. [Accessed: 09-Jun-2012].
[65] “Pellet OWL Reasoner.” [Online]. Available: http://www.mindswap.org/2003/pellet/. [Accessed: 09-Jun-2012].
[66] “DIG Interface.” [Online]. Available: http://dig.sourceforge.net/. [Accessed: 09-Jun-2012].
[67] “About Bossam « Bossam Rule/OWL Reasoner.” [Online]. Available: http://bossam.wordpress.com/about-bossam/. [Accessed: 09-Jun-2012].
[68] “CIS587: The Rete Algorithm.” [Online]. Available: http://www.cis.temple.edu/~giorgio/cis587/readings/rete.html. [Accessed: 09-Jun-2012].
[69] “ProtegeWiki: SWRLTab.” [Online]. Available: http://protege.cim3.net/cgi-bin/wiki.pl?SWRLTab. [Accessed: 13-Jun-2012].
[70] “ProtegeWiki: SQWRL.” [Online]. Available: http://protege.cim3.net/cgi-bin/wiki.pl?SQWRL. [Accessed: 13-Jun-2012].
[71] “Java Logging Technology.” [Online]. Available: http://docs.oracle.com/javase/6/docs/technotes/guides/logging/. [Accessed: 18-Jun-2012].
[72] “Logback Home.” [Online]. Available: http://logback.qos.ch/. [Accessed: 16-Jun-2012].
[73] “Apache log4j 1.2 -.” [Online]. Available: http://logging.apache.org/log4j/1.2/index.html. [Accessed: 10-Jun-2012].
[74] “Commons Logging - Overview.” [Online]. Available: http://commons.apache.org/logging/. [Accessed: 16-Jun-2012].
[75] “Apache Avalon.” [Online]. Available: http://avalon.apache.org/closed.html. [Accessed: 10-Jun-2012].
[76] “SLF4J.” [Online]. Available: http://www.slf4j.org/. [Accessed: 16-Jun-2012].
[77] “PatternLayout (Apache Log4j 1.2.17 API).” [Online]. Available: http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/PatternLayout.html. [Accessed: 11-Jun-2012].
[78] “Apache log4j 1.2 - Short introduction to log4j.” [Online]. Available: http://logging.apache.org/log4j/1.2/manual.html. [Accessed: 12-Jun-2012].
[79] “NDC (Apache Log4j 1.2.17 API).” [Online]. Available: http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/NDC.html. [Accessed: 12-Jun-2012].
[80] “OEMR and OpenEMR.” [Online]. Available: http://www.oemr.org/. [Accessed: 06-Jun-2012].
[81] “OpenEMR Project.” [Online]. Available: http://www.open-emr.org/. [Accessed: 06-Jun-2012].
[82] “Access Controls Listing - OpenEMR Project Wiki.” [Online]. Available: http://www.open-emr.org/wiki/index.php/Access_Controls_Listing. [Accessed: 06-Jun-2012].
[83] Chris Bloom, “PatientOS, Inc. — Welcome.” [Online]. Available: http://www.patientos.com/. [Accessed: 28-Jul-2012].
[84] “PatientOS Architecture.” [Online]. Available: http://www.patientos.org/merits.pdf. [Accessed: 06-Jun-2012].

[85] “Java BluePrints - J2EE Patterns - MVC.” [Online]. Available: http://java.sun.com/blueprints/patterns/MVC-detailed.html. [Accessed: 20-Jun-2012].
[86] “Hibernate - JBoss Community.” [Online]. Available: http://www.hibernate.org/. [Accessed: 20-Jun-2012].
[87] Chris Bloom, “PatientOS Hospital Features.” [Online]. Available: http://www.patientos.com/emr/hospitals. [Accessed: 28-Jul-2012].
[88] “OpenMRS » Open source health IT for the planet.” [Online]. Available: http://openmrs.org/. [Accessed: 06-Jun-2012].
[89] “SpringSource.org |.” [Online]. Available: http://www.springsource.org/. [Accessed: 20-Jun-2012].
[90] “MySQL :: The world’s most popular open source database.” [Online]. Available: http://www.mysql.com/. [Accessed: 20-Jun-2012].
[91] “Chapter 6. Aspect Oriented Programming with Spring.” [Online]. Available: http://static.springsource.org/spring/docs/2.0.x/reference/aop.html. [Accessed: 20-Jun-2012].
[92] “OpenMRS AOP - Documentation - OpenMRS Wiki.” [Online]. Available: https://wiki.openmrs.org/display/docs/OpenMRS+AOP. [Accessed: 19-Jun-2012].
[93] “Protégé-OWL API.” [Online]. Available: http://protege.stanford.edu/plugins/owl/api/. [Accessed: 12-Jun-2012].
[94] “ProtegeOWL API Programmers Guide - Protege Wiki.” [Online]. Available: http://protegewiki.stanford.edu/wiki/ProtegeOWL_API_Programmers_Guide. [Accessed: 12-Jun-2012].
[95] “ProtegeWiki: SWRLFactory FAQ.” [Online]. Available: http://protege.cim3.net/cgi-bin/wiki.pl?SWRLFactoryFAQ. [Accessed: 13-Jun-2012].
[96] “OpenAz Main Page - OpenLiberty.org Wiki.” [Online]. Available: http://www.openliberty.org/wiki/index.php/OpenAz_Main_Page. [Accessed: 14-Jun-2012].
[97] “Az FAQ - OpenLiberty.org Wiki.” [Online]. Available: http://www.openliberty.org/wiki/index.php/Az_FAQ. [Accessed: 14-Jun-2012].
[98] “Sun’s XACML Implementation.” [Online]. Available: http://sunxacml.sourceforge.net/. [Accessed: 14-Jun-2012].
[99] “ProtegeWiki: SWRLRule Engine API.” [Online]. Available: http://protege.cim3.net/cgi-bin/wiki.pl?SWRLRuleEngineAPI. [Accessed: 21-Jul-2012].
[100] “Lockhart-Attribute Manifest File.” [Online]. Available: http://xml.coverpages.org/Lockhart-AMF-Format-V08.pdf. [Accessed: 23-Jun-2012].
[101] “Think again before adopting the commons-logging API.” [Online]. Available: http://articles.qos.ch/thinkAgain.html. [Accessed: 05-Aug-2012].

Appendices

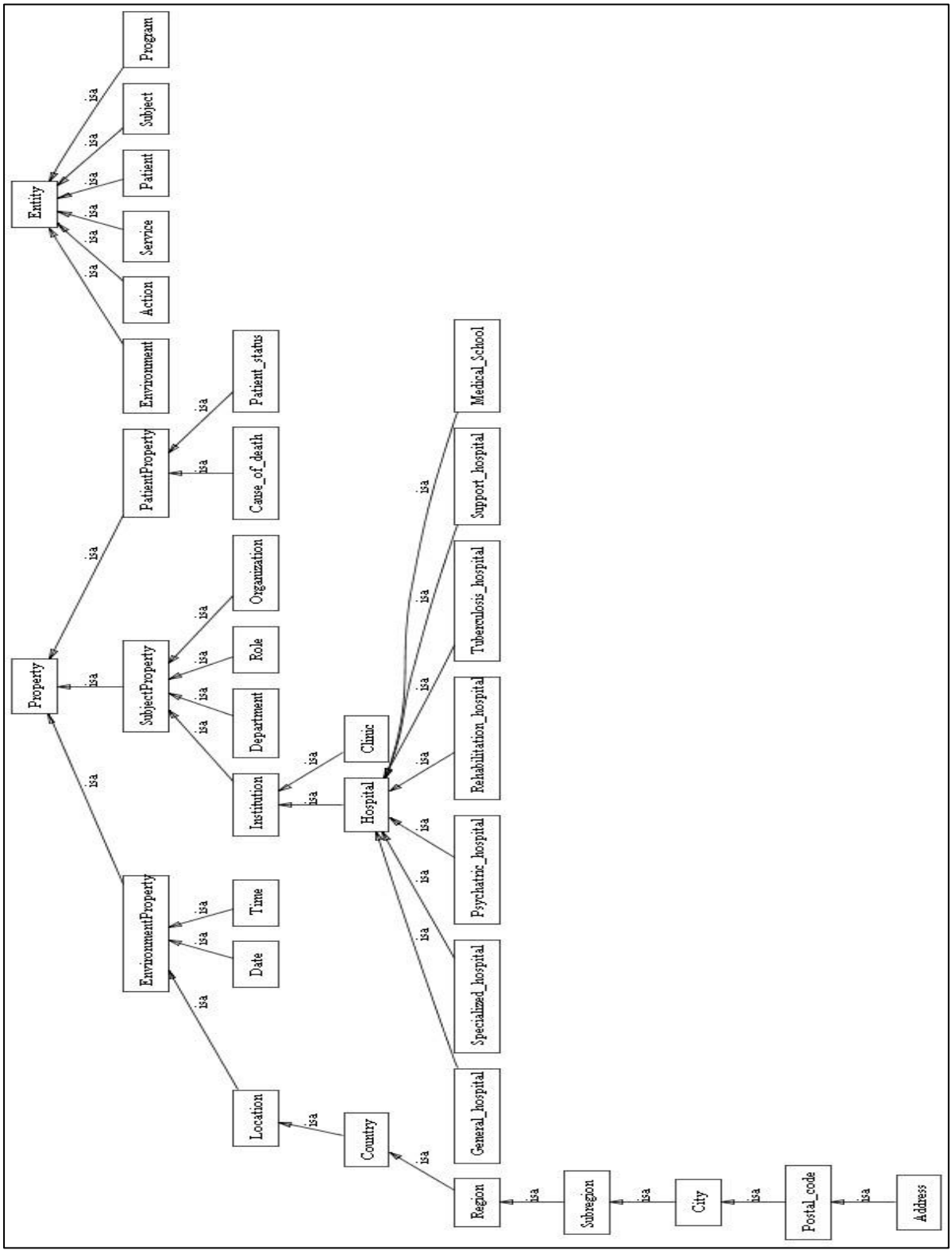
A. XACML messages

This appendix shows a pair of XACML messages corresponding to a XACML request and a XACML response, respectively.

```
<Request xmlns="urn:oasis:names:tc:xacml:1.0:context">
          <Subject>
                    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
                              DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">
                              <AttributeValue>user@example.org</AttributeValue>
                    </Attribute>
                    <Attribute AttributeId="urn:example:age"
                              DataType="http://www.w3.org/2001/XMLSchema#integer">
                              <AttributeValue>30</AttributeValue>
                    </Attribute>
                              <Attribute AttributeId="urn:example:fullAge"
                              DataType="http://www.w3.org/2001/XMLSchema#boolean">
                    <AttributeValue>true</AttributeValue>
                    </Attribute>
          </Subject>
          <Resource>
                    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
                              DataType="http://www.w3.org/2001/XMLSchema#string">
                              <AttributeValue>http://www.restricted.se/index.html</AttributeValue>
                    </Attribute>
          </Resource>
          <Action>
                    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
                              DataType="http://www.w3.org/2001/XMLSchema#string">
                              <AttributeValue>read</AttributeValue>
                    </Attribute>
          </Action>
          <Environment>
          <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"
                              DataType="http://www.w3.org/2001/XMLSchema#time">
          </Attribute>
          </Environment>
</Request>
```

```
<Response>
    <Result>
      <Decision>Permit</Decision>
      <Status>
        <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
      </Status>
    </Result>
</Response>
```

B. Authorization ontology



C.Ontologies administration

```
public static String getClassesAndInstances(String ontName) {
	String result = " ";
	Collection<?> classes;
	…
	OWLModel onto = ProtegeOWL.createJenaOWLModelFromURI(
	getByName(ontName).getPath());
	classes = onto.getUserDefinedOWLNamedClasses();

	for (Iterator<?> it = classes.iterator(); it.hasNext();) {
		OWLNamedClass cls = (OWLNamedClass) it.next();
		Collection<?> instances = cls.getInstances(false);
		result += "Class " + cls.getBrowserText();
		result += " (" + instances.size() + ")<br>";
		for (Iterator<?> jt = instances.iterator(); jt.hasNext();) {
			OWLIndividual individual = (OWLIndividual) jt.next();
			result += " - " + individual.getBrowserText()+"<br>";
		}
	}
	…
	return result;
}
```

```
public static ArrayList<String> getClasses() {

	ArrayList<String> result = new ArrayList<String>();

	Collection<?> classes = loaded_model.getUserDefinedOWLNamedClasses();
	for (Iterator<?> it = classes.iterator(); it.hasNext();) {
		OWLNamedClass cls = (OWLNamedClass) it.next();
		result.add(cls.getBrowserText());
	}
	Collections.sort(result);
	return result;
}
```

```
public static ArrayList<String> getIndividuals(String cls_name) {

	ArrayList<String> result = new ArrayList<String>();

	if (cls_name != null) {
		OWLNamedClass cls = loaded_model.getOWLNamedClass(cls_name);
		Collection<?> instances = cls.getInstances(false);
		for (Iterator<?> jt = instances.iterator(); jt.hasNext();) {
			OWLIndividual individual = (OWLIndividual) jt.next();
			result.add(individual.getBrowserText());
		}
	}
	Collections.sort(result);
	return result;
}
```

```
public static SortedMap<String, String> getProperties(String cls_name) {

	SortedMap<String, String> result = new TreeMap<String, String>();
	OWLNamedClass cls = loaded_model.getOWLNamedClass(cls_name);
	Collection<?> props = cls.getAssociatedProperties();
	RDFProperty prop;

	for (Iterator<?> jt = props.iterator(); jt.hasNext();) {
		prop = (RDFProperty) jt.next();
		String desc = prop.getBrowserText()+"("+cls_name+", ";
		RDFResource range = prop.getRange(false);
		desc += range.getBrowserText()+")";
		result.put(prop.getBrowserText(), desc);
	}
	return result;
}
```

D.OpenMRS Authorization

```
@Target( { ElementType.METHOD, ElementType.TYPE })
@Retention(RetentionPolicy.RUNTIME)
@Inherited
@Documented
public @interface Authorized {

/**
* Returns the list of privileges needed to access a method. (i.e. "View Users").
* Multiple privileges are compared with an "or" unless <code>requireAll</code> is set
*to true
*/
	public String[] value() default {};

/**
 * If set to true, will require that the user have <i>all</i> privileges listed in
* <code>value</code>. if false, user only has to have one of the privileges. Defaults
* to false
*/
	public boolean requireAll() default false;

}
```

The Authorized annotation has two fields *value* and *requireAll*. *Value* takes one or more privileges, and *requireAll* can be set to true (but is false by default).

```
To require that the user have either 'View Users' or 'Add Users' privileges:
@Authorized ({"View Users", "Add User"})
public void getUsersByName(String name);

To require that the user have all privileges
@Authorized (value = {"Add Users", "Edit Users"}, requireAll = true)
public void getUsersByName(String name);

To just require that the user is authenticated
@Authorized ()
public void getUsersByName(String name);
```

```java
public class AuthorizedAnnotationAttributes {

	/** Get the <code>Secured</code> attributes for a given target class */
	public Collection getAttributes(Class target);

	/** Get the <code>Secured</code> attributes for a given target method */
	public Collection getAttributes(Method method);

	/** Returns whether or not to require that the user have all of the privileges
	* in order to be "authorized" for this class */
	public boolean getRequireAll(Class target);

	/** Returns whether or not to require that the user have all of the privileges
	* in order to be "authorized" for this method */
	public boolean getRequireAll(Method method);

	/** Determine if this method has the @Authorized annotation even on it */
	public boolean hasAuthorizedAnnotation(Method method);
}
```

```java
public class AuthorizationAdvice implements MethodBeforeAdvice {

/** Allows us to check whether a user is authorized to access a particular method */
	public void before(Method method, Object[] args, Object target) throws Throwable {

		// Obtain authenticated user
		User user = Context.getAuthenticatedUser();

		AuthorizedAnnotationAttributes attributes = new
		AuthorizedAnnotationAttributes();

		// Obtain authorization attributes
		Collection<String> attrs = attributes.getAttributes(method);
		boolean requireAll = attributes.getRequireAll(method);

		// Iterate through required privileges
		if (attrs.size() > 0) {
			for (String privilege : attrs) {
				if (Context.hasPrivilege(privilege)) {
					// Only a privilege is needed and present
					if (requireAll == false) {
						return;
					}
				} else if (requireAll == true) {
					// Not all privileges are present
					throwUnauthorized(user, method, privilege);
				}
			}
			if (requireAll == false) {
				// The user is not authorized to access the method
				throwUnauthorized(user, method, attrs);
			}
		} else if (attributes.hasAuthorizedAnnotation(method)) {
			// Just require user authentication
			if (Context.isAuthenticated() == false)
				throwUnauthorized(user, method);
		}
	}
```

E. Attribute Manifest File

```
<amf:NeededAttributes>
      <amf:NeededAttribute amf:Autoload=”true” >
      <amf:AttributeId> … </amf:AttributeId>
      <amf:Issuer> … </amf:Issuer>
      <amf:Category> … </amf:Category>
      <amf:Datatype> … </amf:Datatype>
      <amf:Scopes> … </amf:Scopes>
      <amf:EnumeratedValues> … </amf:EnumeratedValues>
      <amf:Range>
            <amf:Minimum> … </amf:Minimum>
            <amf:Maximum> … </amf:Maximum>
      <amf:/Range>
      <amf:RetrievalInfo>
            <amf:Location> … </amf:Location>
            <amf:Method> … </amf:Method>
            <amf:RetrievalURI> … </amf:RetrievalURI>
            <amf:Key>
                  <amf:KeyXACMLName> … </amf:KeyXACMLName>
                  <amf:KeyRepositoryName> … <amf:KeyRepositoryName>
            </amf:Key>
            <amf:Field> … <amf:Field>
      <amf:/RetrevialInfo>
<amf:/NeededAttribute>
<amf:NeededAttribute>
      …
<amf:/NeededAttribute>
      …
<amf:/NeededAttributes>
```

TRITA-ICT-EX-2012:176

www.kth.se