Patient consent policies in XACML

Mwangi, E.W.

Award date:
2008

Patient consent policies in XACML

By
Eva Wanjiru Mwangi

Supervisors:
Dr. Milan Petkovic (Philips Research Laboratories)
Dr. Jerry den Hartog (TU/e)
Prof. Sandro Etalle (TU/e)

July 2008
Philips Research Laboratories
Technical University of Eindhoven (TU/e)

Acknowledgements

This master's thesis concludes my studies at the Technical University of Eindhoven in the Mathematics and Computer Science department. The research carried out in this thesis would not have been possible without the help of a number of people.

I carried out this thesis at the Philips Research Laboratories in the Information and System Security group. My special thanks go to Dr. Milan Petkovic for giving me the opportunity to carry out my research at Philips, and for daily supervision and guidance.

I would like to thank the Technical University of Eindhoven for their support. In particular, I thank my supervisors, Dr. Jerry den Hartog and Prof. Sandro Etalle for their vital input.

My thanks go to Dr. Berry Schoenmakers from the Technical University of Eindhoven and Ir. Paul Koster from Philips, for taking part in my assessment committee.

A special thank you goes to my family and friends for their unwavering support and constant encouragement.

Finally, to my friends, colleagues and fellow interns at Philips, thank you for your time and input.

Eva Wanjiru Mwangi
Eindhoven, July 2008

Abstract

The management of health information has shifted from the use of paper-based to interconnected Electronic Health Record (EHR) systems. The ease with which patients' sensitive health information is accessible in EHR systems, has raised concerns about the breach of data confidentiality and patient privacy.

Existing privacy laws and policies require healthcare providers to obtain patients' explicit consent for access to their health information. Currently, healthcare providers obtain written patient consent using standard consent forms, which describe the patients' general rights and obligations. Specifying patient consent using standard consent forms deviates from the trend of managing health information electronically, and implies that patient consent is static. These consent forms specify consent in natural language, which means that the EHR systems cannot factor in patient consent during access control. Furthermore, consent forms are standard; they are not customized to the individual preferences of each patient, and patients are not given the opportunity to give detailed consent.

XACML is a security policy authoring language that is increasingly used for access control in the healthcare domain. However, specifying semantically correct XACML policies is challenging due to its verbosity and complexity. The semantic correctness of security polices is important in order to prevent illicit access to the protected resource. Furthermore, there are no existing guidelines for the generation of correct XACML polices.

The goal of this thesis is to design a system that facilitates the specification of electronic, customized and detailed patient consent. The detailed consent includes information on who can access what section of a patient's health information, the actions that can be performed, and the conditions under which access is allowed. The system then translates the specified consent to XACML security policies, which are ready to use by EHR systems' access control engines, to govern access to health information according to the preferences specified by the patient.

Contents

Figures

Tables

1 Introduction

In the past, healthcare institutions used paper-based systems to handle patient medical information. Modern consumer architectures tend to be open, interconnected and flexible. In the professional medical domain, this resulted in the adoption of Electronic Health Record (EHR) systems. The aim of EHR systems is to improve the quality of care by making medical information readily available; increasing the efficiency of delivery of services in the healthcare setting, by the electronic exchange of health information; safer patient care due to increased availability and quality of health information; and saving costs associated with manual systems [46].

EHR systems are already in widespread use in healthcare institutions in many developed countries. This implies that personal health information is accessible from numerous sources, which increases the scale of risk of a security breach. These reasons have led to increased concern regarding invasion of privacy and confidentiality. Individual concerns about the breaches of privacy exist due to a number of reasons, including [2]:

- Personal medical information being used in a manner that acts against the interest of the individual (for example discrimination due to health status);
- Embarrassment if health information is accessed wrongfully;
- Unacceptable commercial use of personal medical information; and
- Potential harm to an individual if someone who poses a threat to that individual accesses the information (for example, in cases involving domestic violence).

Incorporating consent mechanisms into EHR systems has the potential to enhance confidentiality and privacy of medical information, which is important for the continued widespread use of EHR systems. Consent has its origins in the Hippocratic Oath, taken by healthcare providers to swear confidentiality to their patients. The dictionary definition of consent is permission or agreement. In [47], consent is defined as the agreement, express or implied, to an action based on knowledge of what the action involves, its likely consequences and the option of saying no.

It is the duty of healthcare providers to maintain the confidentiality of health data, and EHR systems must prohibit access to private medical data by unauthorized parties. A patient's medical data must only be revealed with the patient's explicit or implied consent; explicit consent is expressed orally or in writing, while implicit consent is inferred from a person's conduct. Explicit consent must include authorisations which specify who is able to access patient records and for what purpose. In contrast, many

organisations with access to patient health information do not obtain an individual's consent for the disclosure of this information, which is unethical and criminal [49].

To protect the privacy of patients, EHR systems must be access controlled, and these access control mechanisms must take into consideration the individual's consent when resolving access requests. Currently, individual explicit consent is obtained in written form in standard documents which the patient is required to sign, that describe the general rights and obligations of the patient and the healthcare institution.

1.1 Research goal

Written consent is static; it is not customised for each individual patient. Written consent is also not specified to sufficient detail, since only general consent is specified. The patients do not have the opportunity to specify detailed consent that includes consent for various healthcare providers, for any section or element of their medical data and the conditions within which access can be done. It also does not follow the recent trend of managing health information electronically. For patient consent to be taken into consideration in the access mechanisms of EHR systems, it must be expressed in electronic form. Furthermore, the consent needs to be expressed in a form that is recognised by the access control mechanisms.

Security policies are a means to specify authorisations that access control engines can process. To ensure that a security policy contains the intended authorisations, the resource owner needs to specify it, after which it must be translated correctly to machine-readable format [48]. Specific security policy languages exist that represent policies in machine-readable format. XACML is one such security policy language, which is an OASIS standard. Specifying correct XACML policies is challenging, since XACML has a verbose syntax that generates long policies that are cumbersome to read and write. Furthermore, access control polices are complex due to the varying authorisations they contain and the amount and complexity of the information they manage. In addition, most literature has focused on the testing of XACML policies [43, 49, 50, 51]. However, there is little or no focus on guidelines for the generation of correct XACML policies.

In this thesis, the use of the XACML policy language to represent patient consent is examined. The goal is the design of a method for the generation of correct XACML consent policies, from consent specified in structured natural language. This is achieved by first tackling the electronic specification and capture of individual, detailed patient consent in structured natural language. The consistency of the specified consent is then examined, by performing conflict detection and resolution on the authorisations. Finally, the translation of the structured natural language patient consent to XACML policies is investigated.

1.2 Report Structure

Chapter 2 gives background information on current and future initiatives for the specification of patient consent in Canada, England and The Netherlands; and provides background information on the XACML standard. Chapter 3 focuses on the specification and capture of consent; it looks into current consent initiatives, the various types and elements of consent, and presents the proposed method of specifying patient consent. Chapter 4 and 5 respectively tackle conflict detection and resolution amongst the specified authorisations that constitute patient consent. Chapter 6 focuses on the translation of the consent to XACML, while Chapter 7 presents the testing method employed to test the translation. Finally, Chapter 8 presents the conclusion.

2 Background

The shift from paper-based to interconnected electronic health records has increased concern on the invasion of patient's privacy, by inadvertent disclosure and sharing of private medical information. For this reason, a number of initiatives exist or are in the planning phase for the implementation of patient consent in electronic health records. This chapter presents the existing and future consent mechanisms in Canada, England and The Netherlands; it discusses the EHR initiatives in each country, the privacy laws in place, and the existing or envisioned patient consent mechanisms and their shortcomings. Since the goal of this thesis is to represent patient consent in XACML policies, some background information on the XACML standard is also presented.

2.1 Patient Consent

2.1.1 Canada

Canada is in the process of putting in place an interoperable electronic health record, which will provide its citizens with a secure lifetime record of patient medical information. This record will be electronically available anywhere to authorized health providers and to the patient. Infoway, a non-profit corporation comprised of the Deputy Ministers of Health, is charged with the coordination of the drive for a pan-Canadian EHR.

The envisioned EHR will not be a central nationwide database. Instead, health information will be managed at the provincial or territorial level, in regional health information networks called Electronic Health Record Solution Infostructures (EHRi). The EHRi will be interoperable and interconnected across the country. Applications at the points of service will be used to load patient data into the regional EHRi, while the EHRi will maintain patient data in regional registries and data repositories. One of the components of the EHRi is the consent directive repository, which is the component that maintains information regarding a patient's consent directives for the use, collection and disclosure of identifiable medical information.

2.1.1.1 Privacy laws and policies

In Canada, the privacy of personal health information is protected at the federal and provincial/territorial level [3]. The Pan-Canadian Health Information Privacy and Confidentiality Framework was created to harmonise differing provincial and territorial privacy requirements. This framework specifies the following, with regard to consent for disclosure of personal health information [3]:

- there is implied knowledgeable consent to share health information within the circle of care;
- the individual has the right to opt out of the sharing of health information; and
- it must be possible for a healthcare provider to override a withdrawal of consent in an emergency.

Most territories and provinces in Canada adopted this consent model.

2.1.1.2 Consent architecture and mechanisms

Infoway developed a conceptual privacy and security architecture (P&S Architecture), which presents a vision of the desired security and privacy mechanisms in the EHRi in the near future. The key consent components of the conceptual P&S Architecture include:

- the potential to mask[1] health data at the data element level;
- the potential to mask health data from specific providers or electronic health record users;
- the ability to override masking as long as the override is logged; and
- consent directive repositories which will store individuals' consent choices.

The masking functionality is currently in use in several Canadian e-health projects. In Alberta, the Physician Office System Program (POSP) is being used to migrate physicians to electronic health records and facilitate the integration of their office systems to the regional EHRi. The POSP uses role-based access control to disclose health information based on the role of the user. In POSP, patients cannot opt out of having their information recorded in the EHR. Only through their physicians can patients mask their data. The data could be the whole record, a portion of it or even a data element in the record.

In British Columbia, a province-wide network called PharmaNet links all pharmacies, emergency department physicians and medical practitioners. PharmaNet supports drug dispensing, drug monitoring and insurance claims processing. In this network, patients cannot opt out of the recording of their medical prescription information. Pharmacists and emergency department physicians can access a patient's prescription information without explicit consent. Medical practitioners however, require explicit consent to have access to a patient's information within PharmaNet. Individuals can mask their information by

[1] The data is hidden such that it appears not to exist.

asking their pharmacist to attach a keyword to all their prescription information. The patient can then share the keyword with any pharmacist, physicians or practitioners that the patient chooses. Masking however, cannot be done to a section or element of the patient's prescription information.

The Drug Profile Viewer (DPV) system is a similar prescription system that has been implemented in Ontario. This is a province-wide system that enables the Ministry of Health to share with healthcare providers the claim histories of prescription drugs for patients who receive benefits. In this system, a patient can mask and unmask their prescription information fully or partially. A patient can do so by downloading and filling in withdrawal forms from the Ministry of health's website or by making a phone call to the Ministry.

2.1.2 England

England has a universal, publicly funded healthcare system provided by the National Health Service (NHS). The National Programme for Information Technology (NPfIT), a centralized nationwide health information exchange infrastructure to support the NHS, is under development in England. The Care Records Service is the main feature of the NPfIT, which will create an electronic patient record for every patient.

Care Records will consist of:

- **The Detailed Care Record**
  It contains the detailed medical records of patients and is to be accessible locally.

- **The Summary Care Record**
  This will be a summary of essential clinical health information that is available nationwide.

- **Personal demographic data**
  This includes name, address, gender, date of birth and consent preferences

The Summary Care Records will be available across the country by 2010.

2.1.2.1 Privacy laws and policies

The Data Protection Act, Human Rights Act, common law duty of confidence, and professional ethical standards Act protect health information in England. In addition, the Care Record Development Board (CRDB) releases the Care Record Guarantee yearly, which sets out the general rules that will govern information held in the care records. Some of the key principles of the 2006 Care Record Guarantee include:

- Healthcare providers will have access to a patient's health records on a need to know basis.

- The patient can choose to limit how NHS shares the information in their electronic care record. NHS generally will only share identifiable health information if:

    - the patient requests that they do so;
    - the patient gives their specific permission;
    - it is required to share by law;
    - it has special permission for health or research purposes; or
    - it has special permission due to reasons that are in the interest of the public e.g. an epidemic.

In the future, the CRDB will make it possible for people who are concerned about some entries in their care record, to be able to request that they be kept private.

2.1.2.2 Consent architecture and mechanisms

For the detailed care records, the patients will not have the right to opt out of recording of clinical information. However, for the electronic sharing of this information it is expected that patients will be able to opt out. Furthermore, three 'levels of choice' have been proposed using which the patient will be able to grant access to their care record to healthcare providers. These levels are the treating physician, the organisation involved in their care and the entire NPfIT system.

Individuals will be able to opt out of the creation of a summary care record that will be included in the national database. Individuals, who will not provide any consent directives, will be presumed to have provided implicit consent to the creation of their summary care record. For the sharing of their record, patients will be able to opt out so that only their GP and the GP's teams can view the information. In addition, patients will be able to indicate what information is to be included to the summary care record, by making requests to, or in collaboration with, their GP.

The options 'seal' and 'seal and lock' will be provided for patients to protect their sensitive information. Sealed records will be flagged as sealed when a healthcare provider tries to access them, while sealed and locked records will seem non-existent. To make use of this option, patients will make requests to their healthcare provider, who will then seal or seal and lock the patient's information.

2.1.3 The Netherlands

A National Healthcare Information Hub (Dutch acronym LSP) is in use in the Netherlands [3]. It is a central web-based record locator service that maintains clinical data locally in the healthcare providers system or regional database. A practitioner registers patients' clinical data in the National Reference Index. Using this index, practitioners can query and find all providers who have relevant patient information in their systems. To facilitate matching data with a patient, a unique Citizen Service Number has been issued to every individual. Healthcare providers have unique provider numbers, and have been provided with a smartcard to use for authentication to the LSP.

In this way, the LSP provides access control based on the identity and role of the requester of data, and the data they are authorized to receive.

2.1.3.1 Privacy Laws and Policies

The Medical Treatments Contracts Act (Dutch acronym WGBO), the Individual Healthcare Professions Act, the Personal Data Protection Act (Dutch acronym WBP) and relevant EU directives like the 95/46/EC directive protect health information in the Netherlands. Under these laws, a health care provider generally cannot disclose health data to other parties. However, implied consent is assumed for sharing health information with persons directly involved in the patient's treatment. This is limited to the information that is essential for the particular treatment. The individual has the right to refuse the sharing of their medical data, even for treatment. In an emergency however, a healthcare provider is allowed to bypass all restrictions and obtain access to the required patient data.

2.1.3.2 Consent architecture and mechanisms

Patients are notified by letter when their data is being loaded into the National Reference Index. Once included in the NRI, the data will be available to other providers via the LSP using implied consent. Patients have the option to opt out totally from the nationwide electronic exchange of any of their medical data. They give this consent directive to their healthcare provider, who does not register the patients' information in the National Reference Index. Similarly, patients can partially opt out by selecting portions or individual data elements of their information that should not be registered in the National Reference Index. In some regional applications, individuals can limit access to their records to specific healthcare providers. However, it is not possible to do so for a section or data element in their records.

2.1.4 Limitations of existing consent mechanisms

Australia, England and The Netherlands are not the only countries that have EHR systems in place; this is a general trend in the developed countries. Medical data is being created, stored, shared and accessed in electronic form using electronic health records. In this way, the data is easily accessible and shareable amongst healthcare providers. Since medical data is sensitive and private, patients' preferences as to who can have access to their data must be taken into consideration when resolving requests for access to their medical data. This means that patient consent also needs to be expressed in electronic form, and in a manner that enables the consent to be factored in during resolution of access requests. Furthermore, patients should be able to specify consent for sections or elements of their medical data, for individual or groups of healthcare professionals.

What is currently in place is that patient consent is mostly specified and captured in writing. In Canada, for instance, patients can download and fill out consent forms to give consent [52]. Alternatively, the patient is presented with the patient consent form at the time of registration or consultation in a healthcare institution [2, 26]. The patient consent form conventionally lists the general rights and obligations of the patient and the

healthcare institution concerning the patient's medical information, and captures the details and signature of the patient. Appendix A contains an example of a patient consent form given in accordance to the Health Insurance Portability and Accountability Act (HIPPA) of the United States of America.

Using consent forms to specify and capture consent presents a number of disadvantages. Firstly, patients are not afforded the opportunity to specify different access authorisations for different healthcare providers, for various sections of their medical information or for other relevant conditions[2]. Patients have different preferences on how their medical information should be handled in different situations. However, standard patient consent forms that specify the same consent are presented to all patients. This essentially means that the consent preferences for each patient are not captured. Secondly, patient consent that is captured using patient consent forms has to be translated to a language that can be understood by the access control engine. A question arises on whether the translation will reflect the consent expressed by the patient, and how this can be ensured. Furthermore, specifying consent in writing means that the patient should be physically present at the healthcare institution. If any changes need to be done to the consent by the patient or the healthcare institution, communication has to be done in writing. This is clearly slow and ineffective.

Finally, current consent initiatives have limited functionality. It is generally not possible for consent to be provided for any section or element of a patient's medical information, and for any healthcare providers represented as individuals or in groups. Moreover, the patient cannot directly use the existing consent mechanisms to specify consent; currently this has to be done through the healthcare provider. Typically, the patient specifies consent in writing or orally to the healthcare provider, who will then use this as input to the consent mechanism. The work presented in this thesis addresses all the limitations specified above.

[2] These are given in Section 3.2.2.

2.2 XACML

The aim of this thesis is to generate patient consent policies using the XACML policy authoring language. This section provides background information on XACML, and concentrates on:

- the data flow model, which shows the major actors of the XACML domain and their functions in the process of resolving an access request;
- the language model, which discusses the composition of XACML policies and their evaluation during the resolution of access requests; and
- the new delegation profile of XACML.

XACML is the OASIS standard language for the specification of authorisation and entitlement policies. The goal of XACML is to propose a common language through which an enterprise can manage the enforcement of all the elements of its security policy in all the components of its information system. XACML uses XML syntax since XML's syntax and semantics are extensible to accommodate the requirements of XACML, and enjoys popular support from the main platform and tool vendors.

2.2.1 XACML Data flow model

The major actors in the XACML domain are:

- **Policy Administration Point (PAP)**
  The system entity that creates a policy or a policy set.

- **Policy Enforcement Point (PEP)**
  This system entity performs access control by making access requests, and enforcing the access decisions.

- **Policy Decision Point (PDP)**
  This system entity evaluates applicable policies and renders an authorisation decision.

- **Policy Information Point (PIP)**
  The system entity that is the source of attributes.

- **Context handler**
  This system entity converts the access request from the native format of the PEP, to the XACML canonical form, and converts the authorisation decisions in the XACML canonical form to the PEP's native response format. This is necessary since XACML is intended to be suitable for a variety of application environments, and therefore the core language needs to be insulated from the application environment.

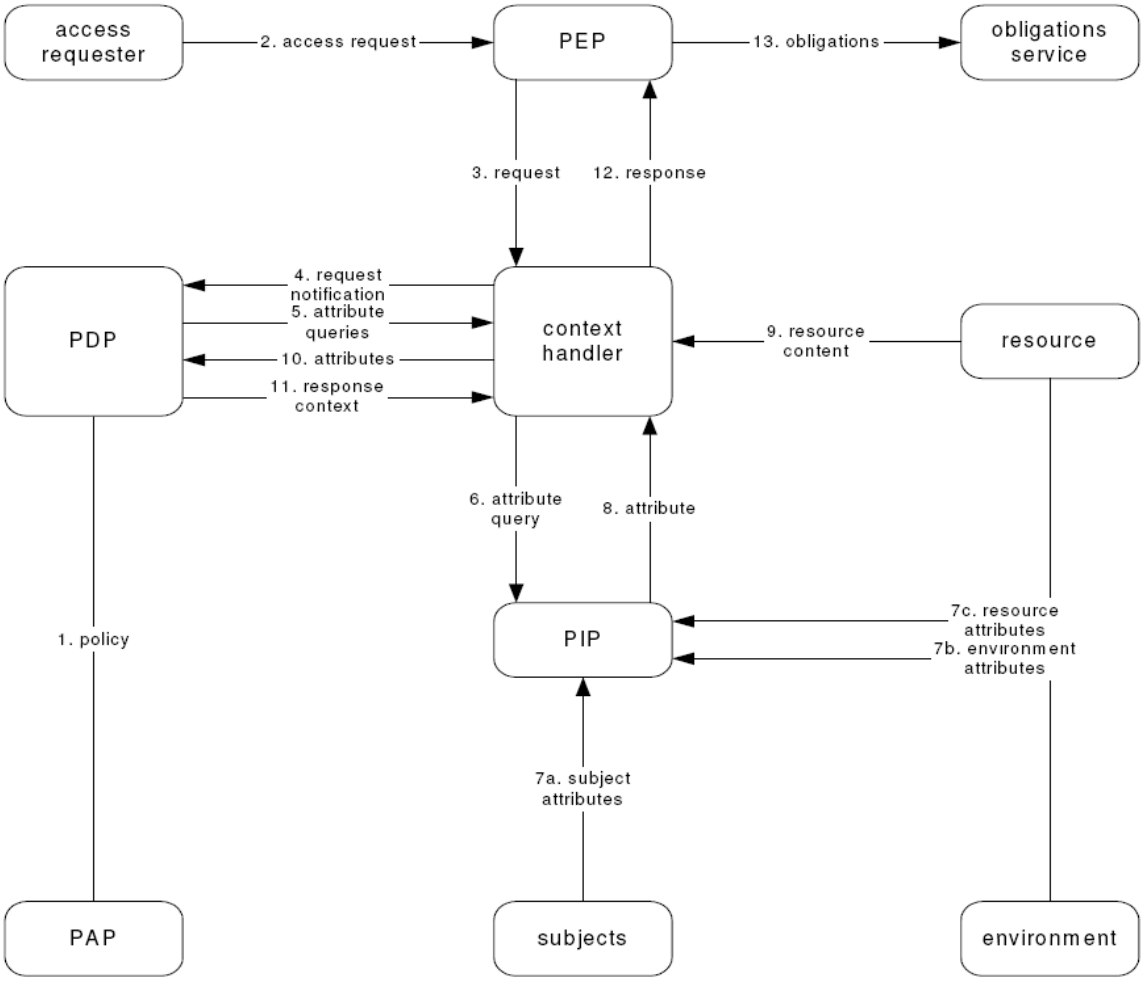

Figure 1: XACML data flow model

The PAP writes policies and makes them available to the PDP. When an access requester makes an access request to the PEP, the PEP forwards the request to the context handler, which transforms it from its native format to a XACML request context. Optionally, the context handler may add subject, resource, action or environment attributes (characteristics) to the access request. The request context is then forwarded to the PDP. The PDP may request for more attributes from the context handler, which will in turn make requests for the attributes to the PIP. Optionally, the context handler may include the resource in the request context. On making an authorisation decision, the PDP creates a response context and returns it to the context handler, which then translates the response context to the native response format of the PEP. The response is then forwarded to the PEP, which must fulfill any obligations indicated in the response. Finally, the PEP either permits or denies access to the resource, in line with the received response.

2.2.2 The policy language model

XACML defines three top-level policy elements:

- Rule;
- Policy; and
- PolicySet.

These elements are described in detail in the coming sections and are illustrated in Figure 2.

2.2.2.1 Rule

The rule is the most basic unit of a policy. A rule is made up of the following:

- **Rule target**
  The target of a rule defines the subject, action, resource and environment attributes to which the rule applies. If the rule target is the same as its containing policy's target, then the rule target is omitted and the rule will inherit its containing policy's target.

- **Rule effect**
  This is the 'Permit' or 'Deny' response indicated by the rule-writer, and given by a rule as its response to an applicable access request

- **Rule condition**
  This Boolean expression refines the applicability of the rule beyond what the rule's target specifies.

A rule can only exist in isolation within one of the major components of the XACML domain, and otherwise it must be contained in a policy.

2.2.2.2 Policy

A Policy is the most basic unit that can be exchanged between the major components of the XACML domain. A policy contains the following elements:

- **Policy target**
  The policy target contains the subject, action, resource and environment attributes that define the access requests to which the policy applies.

- **Rule-combining algorithm**
  The rule-combining algorithms specify the procedures by which the results of evaluating the component rules are combined.

- **Rules**
  Rules are described above.

- **Obligations**
  These are returned to the PEP together with the response context, and are the actions that must be enforced together with the response at the PEP.

The rule-combining algorithms are discussed in Section 2.2.3.

2.2.2.3 Policy Set

A policy set contains the following components:

- **Policy set target**
  This is similar to the rule and policy target.

- **Policy-combining algorithm**
  This specifies the procedure by which the results of evaluating the constituent policies are combined when evaluating the response of a policy set.

- **Policies**
  These are described above.

- **Obligations**
  Same as above.

The policy-combining algorithms are discussed in the next section.

2.2.3 Combining algorithms

A number of rule and policy combining algorithms have been defined in the XACML standard. A rule-combining algorithm defines a method of arriving at an authorisation decision given the individual results of the evaluation of a set of rules. Similarly, the policy-combining algorithm specifies the procedure for arriving at an authorisation decision given the individual results of evaluation of a set of policies [19]. There are four rule/policy combining algorithms:

- **Deny-overrides**
  If a <Rule> or <Policy> element evaluates to 'Deny', then regardless of the result of evaluating the other <Rule> or <Policy> elements in the applicable policy or policy set, the combined result is 'Deny'.

- **Permit-overrides**
  The permit overrides combining algorithm returns a 'permit' result, if one of the <Rule> or <Policy> elements of the applicable policy or policy set evaluates to 'Permit', regardless of the result of evaluating the other <Rule> or <Policy> elements.

- **First-applicable**
  In this case, the combined result is the same as the result of evaluating the first <Rule>, <Policy> or <PolicySet> element which is applicable to an access request.

- **Only-one-applicable**
  This combining algorithm is only applicable to policies, and the result ensures that only one policy or policy set applies to the access request. If no policy or policy set is applicable to the access request, then a 'Not applicable' result is returned. On the other hand, if more than one policy or policy set is applicable to the access request, then the result is 'Indeterminate'.

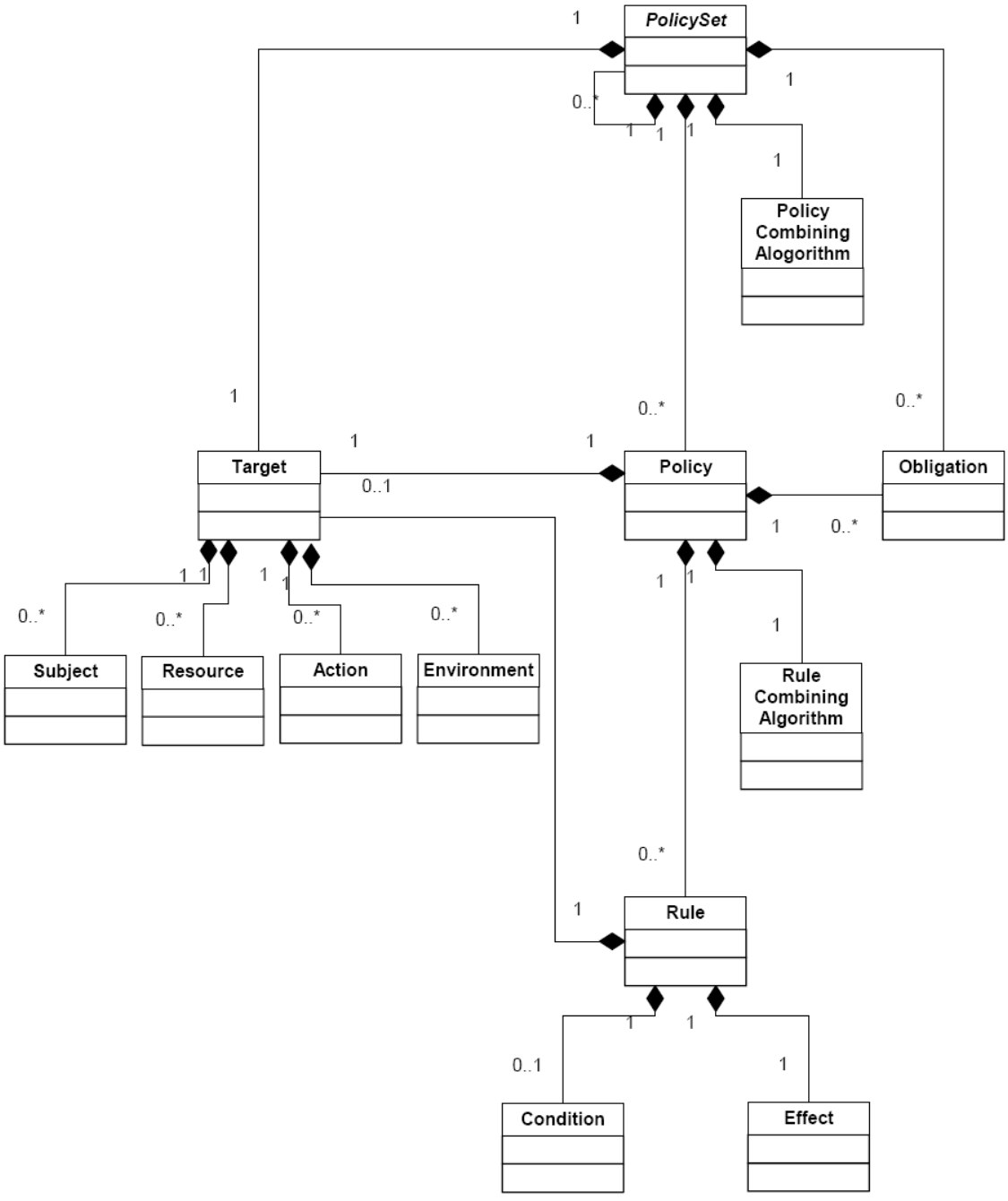

Figure 2: XACML Policy language model

2.2.4 Policy evaluation

For a policy to be applicable to an access request, all the subject, action, resource and environment attributes specified in the policy's target must match those specified in the request context. If any one of the attributes returns an 'Indeterminate' response, then the target shall return the value 'Indeterminate'. However, if any one of the attributes returns a 'No match' value, then the target shall return a 'No match' response.

The rules inside the policies are evaluated next, starting with the targets of the rules, which are evaluated in the same manner as the target of a policy. If a condition exists in the rule, it is evaluated next. The condition returns 'True' if the <Condition> element is absent, or if it evaluates to 'True'. The value of the condition returns 'False' if the condition evaluates to 'False', or 'Indeterminate' if the condition evaluates to 'Indeterminate'. Table 1 below summarises the evaluation of a rule.

Table 1: Rule truth table

Target	Condition	Rule Value
'Match'	'True'	Effect
'Match'	'False'	'NotApplicable'
'Match'	'Indeterminate'	'Indeterminate'
'No-match'	Don't care	'NotApplicable'
'Indeterminate'	Don't care	'Indeterminate'

Finally, the value of a policy will be determined by the rules it contains and the rule-combining algorithm used to combine the results of each applicable rule. This is summarised in Table 2.

Table 2: Policy truth table

Target	Rule values	Policy value
'Match'	At least one rule value is its effect	Specified by the rule-combining algorithm
'Match'	All rule values are 'NotApplicable'	'NotApplicable'
'Match'	At least one rule value is 'Indeterminate'	Specified by the rule-combining algorithm
'No-match'	Don't care	'NotApplicable'
'Indeterminate'	Don't care	'Indeterminate'

2.2.5 Delegation profile of the draft XACML 3.0

The delegation profile of the draft XACML 3.0 makes it possible to express permissions about the right to issue policies. Policies that have been issued can then be checked against these rights. This ensures that policies used by the PDP are only written by those entities that have the permission to do so.

In this profile, policies are either trusted or untrusted policies. Untrusted policies contain the 'Policy Issuer' element that shows the source of the policy, so that its source is verified by the PDP during the resolution of access requests. The verification is done by checking the trusted policies directly or through other policies that have issuers, for the existence of the authorisation that gives the author of the policy the permission to specify it. If the authority to specify a policy can be traced from an untrusted policy to a trusted policy, then the policy is used by the PDP. Otherwise, the policy is discarded as being untrusted. A policy that does not have the 'Policy Issuer' element is a trusted policy, whose origin is valid and requires no verification.

2.2.6 Limitations of XACML

XACML policies are difficult to write with respect to syntax and semantics. XACML syntax is wordy and thus generates very long policies. Simple policies generate long XML text that is not easy for a human being to write or read. A problem therefore arises when complex security policies need to be expressed in XACML. It is also difficult to ascertain the semantic correctness of a XACML policy. Semantic correctness is vital since semantic errors will lead to vulnerabilities that may facilitate illegitimate accesses to the protected resource.

A number of tools have been created to ease the work involved in writing syntactically correct XACML policies [19]. The ParthenonXACML tool [44] for example, performs syntax and type check of XACML policies and access requests. Furthermore, the UMU-XACML policy definition software [41] can be used to generate syntactically correct XACML policies. As far as semantic correctness is concerned, there are tools like Margrave [43] that test policies for semantic correctness. However, there are no tools for the generation of semantically correct XACML policies.

3 Specification of patient consent

The previous chapter presented a discussion on the capabilities of existing patient consent technologies and provided some background information on the XACML standard. This and the subsequent chapters of this thesis present the design of the system that facilitates the specification of customized, detailed patient consent, and the translation of this consent to XACML policies that can be utilized by access control engines during the resolution of access requests.

The starting point in the creation of a patient consent policy is the specification of the consent policy by the patient or a legal custodian. The first part of this chapter presents the new initiatives from the literature, followed by the proposed method for the specification and capture of the patient consent, which will be the input to the XACML policy generation process.

1.1 New initiatives

The easiest way to specify consent from the patients' point of view would be using natural language. However, a great deal of improvement needs to be done in the area of natural language processing and artificial intelligence, before patient consent or any other high level access control policies can be specified in this way [10]. Conversely, with the aid of a Graphical User Interface (GUI), natural language can be employed in the specification of consent. A GUI presents a framework that adds structure to the natural language and reduces the level of ambiguity that is inherent in natural language. The GUI achieves this by using interface objects[3] to enforce a structured way of entering textual information and selecting one or more from a number of presented options.

A number of initiatives for the specification of patient consent electronically using a GUI exist. Medicon and eMedicalBook are prototypes that have been developed at the University of Wollongong in Australia. Using these prototypes, patients can view their medical information, assign permission to different healthcare providers and choose which parts of their medical information to disclose with the help of a GUI [5]. Another example is the MedicClient demonstrator, which is a simplified clinical health information system that uses patient consent to control access to medical information [6]. Patient consent is captured using a computer-based form, with textboxes for free text and drop down boxes for presenting options. In this prototype, medical information is

[3] Interface objects include text boxes, radio buttons, check boxes and drop down lists

associated with consent, such that a patient can view their medical information with the associated consent using a GUI.

Like these prototypes, a GUI will be used by the patient directly to specify consent, will be adopted in this thesis for the specification of patient consent.

3.2 Specification of patient consent

This section presents the proposed method of specifying the patient consent that is to be used as input to the XACML policy generation process. First, the various types of electronic consent (e-consent) in healthcare presented in literature are discussed. Then, what should constitute patient consent is discussed, since before considering how patient consent should be specified, it is important to consider what constitutes patient consent and therefore what should be specified. Finally, the specification method itself is presented.

3.2.1 E-consent models

In [1], six e-consent models in the healthcare domain have been identified. These models capture the different types of consent and include:

- **Consent/delegation by default**
  It is assumed that any person can have access to the patient's medical information unless the patient gives explicit denial in the future.

- **General consent/delegation**
  The patient gives consent or gives the right to delegate or give consent to a particular individual without any exceptions.

- **General consent/delegation with specific denial**
  A patient gives consent or gives the right to delegate or give consent to a general class but with exceptions to specific subclasses of the general class. The patient can allow access or delegate the right to delegate or give consent to the medical information, but with specific denials for some particular information. Alternatively, the patient can give consent or give the right to delegate or give consent to a particular class of users but deny consent/delegation to specific users in the class.

- **General denial with specific consent/delegation**
  This is similar to the previous model with the exception that the general class is denied access while the specific class is given consent or given the right to delegate or give consent.

- **General denial**
  The patient denies access to the medical information by a particular individual without exception.

- **Denial by default**
  It is assumed that nobody can have access to the patient's medical information unless the patient gives explicit consent.

The general consent/delegation with specific denial and the general denial with specific consent/delegation models are the most commonly used consent models in the healthcare domain. This is because patient consent usually contains both positive and negative authorisations[4], such that a general positive authorisation is given followed by specific negative authorisation(s), or general negative authorisations followed by specific positive authorisation(s).

The second most commonly used models are the general consent/delegation and the general denial models. The general consent/delegation models are used to specify consent for personal-doctors, immediate family members or legal custodians, while the general denial models are often employed in specifying consent for sensitive medical information like sexual and psychological health. Finally, the consent/delegation by default and denial by default models are used as default configurations for access control engines.

1.2.2 Elements of patient consent

Patient consent expresses authorisations that specify whether an entity is granted or denied access to part or all of a patient's medical information, to perform some action on the information and the conditions in which this should be done. Therefore, the patient or a custodian of the patient has to specify these elements of consent as authorisations that will be enforced during access control. In [1, 2, 8, 26, 27, 28, 29, 30], the following elements of consent are identified:

- **Grantor**
  This person gives consent and could be the patient or a legal custodian of the patient.

- **Grantee**
  This is the individual, role or group to which consent is granted.

- **Patient**
  This is the patient in question.

[4] Positive and negative authorisations are the same as permit and deny authorisations, respectively.

- **Action**
  This is the operation that the grantor does or does not allow the grantee to perform on the data. Examples of such operations include read, write, collect, access, use, disclose, amend, or delete.

- **Data**
  An XPath expression that represents all or part of the patient's medical information.

- **Effect**
  This is either 'permit' or 'deny'.

It is important to specify the grantor since the patient is not always the grantor. In the case where the patient and the grantor elements are not equal, the new delegation profile of the draft XACML 3.0 standard will be applied when creating the XACML policy. This is because the XACML policy will be created with an 'Issuer' element, which will be used to check that the patient has delegated the grantor the permission to issue consent on the patient's behalf [20].

System-wide unique identifiers should represent the identities of the grantor and the patient. A form of authentication should be used to verify the identity, the membership to a group or the possession of a role of the grantor during the specification of consent, and the grantee during an access request. Identity authentication can be done by the use of PINs, passwords, digital signatures or biometrics [2], and the grantor and grantee active sessions can be used to extract identity, group and role information.

In the XACML 3.0 draft standard, the following five data types have been listed as valid XACML identifiers for resources: X.500 directory names, email addresses, IP addresses, DNS hostnames and XPath expressions. In this thesis, the resource being protected is structured medical information. It is imperative that the grantor be able to specify consent for any part of the patient's medical data, since the level of sensitivity of the data varies with its nature and the grantor's preferences. Therefore, the identifier used to express the data element of consent should provide the grantor the ability to navigate and select any section or element of the patient's medical information. XPath is a language for navigating and selecting nodes in an XML document [21, 22]. XML is now the dominating standard for representing data and structured documents [23], and is widely used in medicine. Standards like HL7-CDA (Health Level 7 - Clinical Document Architecture) and OpenEHR for the management, storage, retrieval and exchange of health data are based on XML [24]. From the five identifiers listed in the XACML standard, only XPath provides document browsing functionality. Furthermore, XPath is compatible with the current technology used to represent medical data, and thus will be used for expressing the data element of consent in this thesis.

In addition to the consent elements specified above, the grantor specifies the conditions under which the grantee can perform actions on the patient's data. The following three conditions are identified from the literature:

- **Purpose**
  This specifies the purpose for which the grantee can access the data. In [2, 8, 30] a number of purposes are identified. These include 'treatment', 'payment', 'operations', 'research', 'public-health', 'planning', 'quality-measures', 'health status evaluation by third parties' and 'marketing'.

- **Context**
  This specifies the context within which the grantee should be permitted to act on the data. Two contexts have been identified; the grantee can access the patient's data in 'normal' or 'emergency' situations [2], [8], [30]. An example is that a grantor might not want a particular individual to have access to some medical data in normal situations, but would allow it in a life-threatening situation.

- **Validity period**
  The authorisation specified by the grantor is valid within this period. For this element to be useful, the date of the specification of consent must be captured.

The grantor must specify all the nine elements for each authorisation, with the exception of the validity period, which can be left unspecified[5]. This is to prevent under-specification of the patient consent policy [36]. Consider an example of an authorisation that permits a particular grantee, to perform a particular action on some patient's data for the purpose of treatment, while another authorisation exists that denies the same grantee the permission to perform the same action on the same data in the normal context. If the grantee makes a request to perform the stated action on the stated data, with 'treatment' for the purpose and 'emergency' for the context, then both authorisations will apply to the request. One authorisation give a permit response, while the other will give a deny response. However, if for both authorisations, the context and purpose elements were specified, then both authorisations will only be applicable if all the elements are the same.

The grantor uses a GUI to specify consent. The grantor's unique identity can be specified in textboxes, which may be coded with rules that validate the input for the correct format. If the grantee is an individual, the grantor is presented with a textbox similar to the one used to specify the grantor's identity. It is important for the grantor to specify whether a grantee represented using a unique ID also plays healthcare roles, and the grantor must be furnished with this information during the specification process. This is the case when the grantor gives consent to an individual who also happens to be a healthcare provider. This will be useful during conflict detection among the authorisation rules that make up patient consent.

[5] If the validity period is left unspecified, this implies that the authorisation is valid indefinitely, or until the patient specifies otherwise.

If the grantee is a healthcare role or a group, the grantor is presented with a predefined list(s) of healthcare provider's roles and groups provided by the healthcare institution. Alternatively, the grantor can use a drill down menu, which is a list that contains other lists, which in turn contain other lists in a manner that represents the hierarchical ordering of the healthcare institution's roles and groups. This increases efficiency since the grantor can specify consent for a role/group that is more general or lower in the role/group hierarchy, and this consent would then be inherited by the more specific or senior roles [25].

A textbox similar to the one that is used to specify the grantor can be used to specify the patient's identity. The action can be selected by the grantor from a list of viable actions as determined by the healthcare institution. These viable actions are those that the healthcare institution can perform on a patient's medical data, and may be presented using drill down menus if a hierarchical ordering of the actions exists. Similarly, the effect, purpose, context and validity period elements are presented in viable lists. The purpose and context lists shall have an additional option 'all' to select all purposes or contexts. For the data element, an XPath expression builder may be used. An XPath expression builder facilitates the construction of XPath expressions, which select sections or individual values from an XML document, and to visualise the results in real time. An example of an XPath expression builder is included in the Liquid XML tool [31]. With this tool, the creation of an XPath expression is simplified; for example, when a user moves the mouse over an element of an XML document opened in this tool, an XPath expression showing the parent element hierarchy is displayed. All the user has to do is to add the name of the element to its parent element hierarchy to get the XPath expression for that element.

The GUI should also provide a section using which the grantor shall select whether or not the specified authorisation is a delegation or access authorisation. If the authorisation is a delegation, the grantor permits or denies the grantee specified in the authorisation the permission to delegate or give the specified authorisation. On the other hand, if the authorisation is a consent authorisation, then the grantor permits or denies the grantee specified in the authorisation the permission to perform the action on the data in the specified conditions.

A preliminary GUI that is still under development is shown in Figure 3.

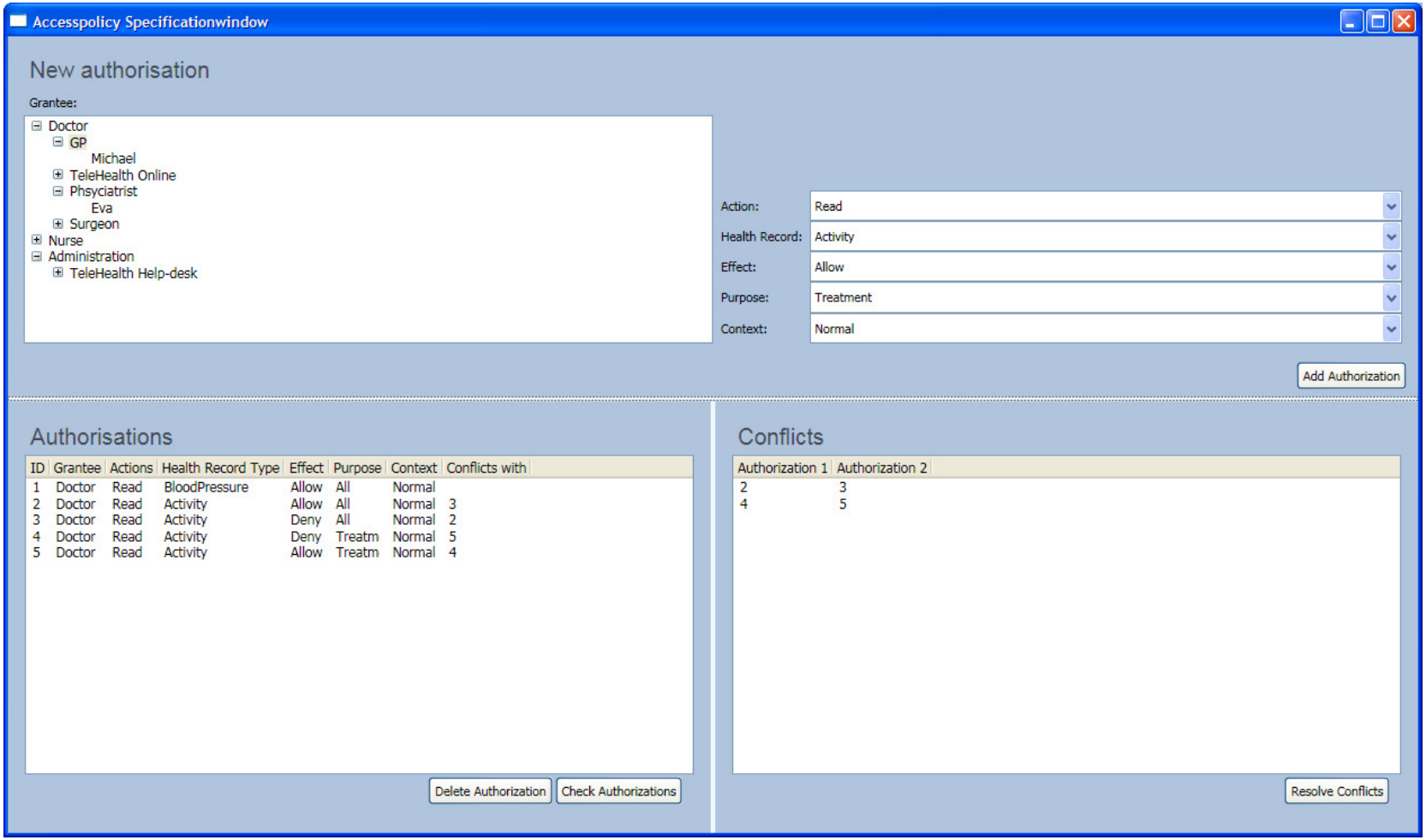

Figure 3: GUI for the specification of patient consent

3.2.3 Consent specification

As discussed in Section 2.2.1, in the XACML data flow model, the Policy Administration Point (PAP) is the entity in the XACML data flow model that writes policies, and avails them to the PDP for resolution of an access request [19]. For this reason, the specification of patient consent and the process of transforming this consent to XACML policies will be performed at the PAP.

The previous section presented the elements that make up patient consent. This section will now present the details of how exactly a grantor uses these elements to specify consent. In line with the discussion presented on the various models of consent in Section 3.2.1, the patient's consent will fall in one the following four consent models:

- General consent/delegation
- General consent/delegation with specific denial
- General denial with specific consent/delegation
- General denial

From the list above, it is clear that regardless of which of the four models the patient's consent falls in, general authorisations must be specified first. If need be, the patient can then give specific authorisations that are exceptions to the general authorisations.

3.2.3.1 General consent, delegation or denial

The grantor must select the general consent or denial that will apply to any grantees who request to perform some action on the patient's data. For this reason, general consent or denial is referred to as 'default consent' from this point. Default consent applies to the entire medical record of a patient, and is expressed using the 'grantor', 'patient', 'data', 'action' and 'effect' elements.

Default consent shall be specified for all viable actions that can be performed on the patient's medical data. Therefore, there should be as many default consent authorisations as there are viable actions. Actions can also be organised into hierarchies, such that the grantor need only to specify default consent for the actions that are higher up in the action hierarchy. An example of such an action is update, which may refer to both read and write actions [30].

The grantor has the option of selecting permit or deny as default consent for an action. If the grantor selects permit as the default consent, then this implies that the grantor adopts an open policy, such that the grantor generally allows all grantees to perform the corresponding action on the patient's medical data. However if the grantor chooses to adopt deny as the default consent, then the patient adopts a closed policy which means that no one is allowed to perform the specified action on the patient's medical data.

If the grantor would like to delegate the right to delegate or give consent to an individual, then the grantor will issue a general delegation authorisation. A general delegation authorisation is given to an individual grantee. Similar to the default consent, the general delegation is either a permit or deny authorisation, where permit allows delegation while deny prevents delegation to the specified grantee.

1.2.3.2 Specific consent, delegation or denial

The grantor gives specific authorisations as exceptions to the default consent or general delegation. These exceptions can be permit or deny exceptions, depending on the default consent or general delegation specified. For example, if a grantor chooses to adopt deny as the default consent for a particular action, the grantor could then specify an exception that allows a certain grantee to perform the action on their medical data. Furthermore, the grantor can disallow this grantee to perform the action on some section of the medical data that the grantor considers private. In summary, the grantor specifies a default deny consent, followed by specific permit consent and then an even more specific deny consent. In an alternative example, if a grantor chooses to use general delegation for a specific individual, then the grantor could express exceptions such that the individual is not delegated the right to delegate or give consent for some of the patient's medical data.

1.2.4 Storing consent

After a grantor has specified the consent made up of the default and specific authorisations, this information needs to be stored. Decision tables are a way of storing policies, which allow the specification of many authorisations in one table [9], and are adopted in this thesis. Therefore, all the authorisations specified by all grantors are kept in one decision table. The decision table's columns are labeled with the names of the consent elements. Each authorisation given by a grantor is then entered in one row of the table, while entering each element in its corresponding column in the decision table. All the elements of consent must have only one value for each authorisation or row in the decision table. This keeps the decision table simple since it ensures that every table entry has only one value.

A decision table is simple, yet rich enough to express patient consent and is easily readable by the patient, a grantor or a relevant authority of the medical institution without requiring any pre-processing. Furthermore, decision tables keep all the authorisations together as one policy, and are easy to store and retrieve using matrices, lists or databases. There is one decision table for each patient, where all authorisations from all grantors are kept.

3.3 Working example

Let us take an example of a patient who would like to specify the following consent policy. The patient's unique ID is represented by 'patient-ID', and the patient is specifying the consent in a healthcare institution that reads and writes to patients' medical information for various purposes.

I would like to give general denial for read and write access for my data. However, I permit the following:

- *I would like my doctor to read and write my medical data for all purposes and contexts.*
- *I would like all other doctors to read and write my data for treatment purposes only and for all contexts for the next one year.*
- *Dr. John Mathews can only read or write to my data for treatment in an emergency.*
- *I would like my husband to read all my data for all purposes and contexts, and to be able to delegate the same permission to others.*
- *I would also like my mother to read my data for all purposes and contexts. However, my mother should not read my Gynecological-information and the Blood-pressure measurements taken within the last 3 months.*

For the default consent, the patient selects 'deny' for the 'read' and 'write' actions. This will apply to all of the patient's data, for all grantees and for all access requests. For the specific consent authorisations listed as bullets, the patient will use either unique identifiers or the names of the healthcare roles or groups to specify the grantees. The grantor and patient elements are specified using unique identifiers.

Table 3: Decision table containing the patient consent policy for patient-ID

Auth No.	Grantor	Grantee	Action	Data	Effect	Purpose	Context	Validity Period	Auth type
1	patient-ID		read	/patient-ID/*	-				A
2	patient-ID		write	/patient-ID/*	-				A
3	patient-ID	personal-doctor	read	/patient-ID/*	+	all	all		A
4	patient-ID	personal-doctor	write	/patient-ID/*	+	all	all		A
5	patient-ID	doctor	read	/patient-ID/*	+	treatment	all	1 year	A
6	patient-ID	doctor	write	/patient-ID/*	+	treatment	all	1 year	A
7	patient-ID	Dr.-John-Mathews-ID	read	/patient-ID/*	+	treatment	emergency		A
8	patient-ID	Dr.-John-Mathews-ID	write	/patient-ID/*	+	treatment	emergency		A
9	patient-ID	husband-ID	read	/patient-ID/*	+	all	all		D
10	patient-ID	mother-ID	read	/patient-ID/*	+	all	all		A
11	patient-ID	mother-ID	read	/patient-ID/Gynecological-information/*	-	all	all		A
12	patient-ID	mother-ID	read	/patient-ID/Blood-pressure[age<=3]	-	all	all		A

Table 1 shows the decision table for the patient consent policy specified above. The decision table contains the authorisations that make up patient-ID's consent to access of her medical information. Each authorisation is either an access (A) or a delegation (D) authorisation, and this is indicated in the 'Auth type' column.

The first two authorisations in the decision table are the default consent for the read and write actions. Since default consent is applicable to all access requests, the grantee, purpose, context and validity period fields are left blank. The effect is entered in the table using the '+' and '-' signs, which mean 'Permit' and 'Deny' respectively. The column for the data has an XPath expression that represents either part or all of the patient's medical information. The '/patient-ID/*' XPath expression represents all of the patient's medical data, assuming that all the data is stored in an XML document whose root element is the patient's unique identifier.

Authorisations 3 and 4 of the decision table represent the access authorisations for the patient's personal-doctor. Any individual with this role can read and write to all of the patient's medical information for all purposes and contexts. Authorisations 5 and 6 give permission to any individual with the doctor role to read and write the patient's medical data for treatment purposes and in all contexts for 1 year. The seventh and eighth authorisations specify access permissions for Dr. John Mathews, who is only allowed to read the patient's information for treatment in an emergency. The next authorisation gives permission to the patient's husband to delegate or give consent to read the patient's medical information. The last three authorisations specify permissions for the patient's mother. The ninth authorisation gives the mother the permission to read all the medical information, but authorisations 10 and 11 specify that the mother must not read gynecological and blood pressure information taken within three months, respectively.

3.4 Discussion

The method of specifying consent described in this chapter presents a number of advantages. First, a grantor can now specify e-consent. This implies that patient consent can not only be issued at the time of registration or consultation in a healthcare institution, but can also be issued at any time using a secure connection to the healthcare institution. Therefore, consent is no longer static. Using this connection, grantors can also easily make changes to the consent policy. Due to the use of a GUI, grantors can directly give e-consent, which implies that no translation from written form is required. Consequently, the e-consent is stored and used as issued by the grantor.

Secondly, grantors can now specify their individual preferences; they can permit or deny consent, they can specify which individuals can access what sections of the patient's data, what actions can be performed and in which conditions these actions can be carried out.

The use of action, role/group and condition hierarchies have been presented as a method that makes the specification of consent easier for the grantor. However, these hierarchies can be subject to change. Actions, roles/groups or conditions can be added to the hierarchies, or their ordering in the hierarchies can change. These changes may affect the meaning of the specified consent, for example if a grantor had specified the value 'all' for the purpose element of consent, this will have a different meaning if more purposes are added to the list of possible purposes. In these cases, the grantors must be informed of the changes and their possible effects, so that the grantors can make any necessary changes to the specified consent.

It is important that if the grantor is not the patient, a check should be done to ensure that the grantor has the permission to give consent on behalf of the patient. Therefore, a policy specified by the patient must exist that delegates the necessary authority to the grantor. Before proceeding to the specification of consent, this policy must be checked and the grantor verified as having the necessary permission to give consent on the patient's behalf. This grantor verification functionality is available in the delegation profile of the draft of the XACML 3.0 standard.

It is possible that more or different situational information is required, apart from the purpose, context and validity period of consent. Other situational information like the location of access may be necessary. The situational information is not restricted to those discussed in this thesis.

During an access request, some of the elements of consent could be derived by using information that is available in the system, without necessarily requiring explicit specification by the grantee. The grantee's current role/group or identity could be detected from the system. The role could be detected from the role in the active session. Similarly, if the grantee is known to the system using an identity or as a member of the group, then this could be used as the grantee element of consent in the access request. The context could also be retrieved from the system. For example if the grantee is accessing the patient's data in the emergency room of a hospital, then the emergency

context is used. In addition, if the data is being accessed in the consultation area of the healthcare institution, then treatment is used for the purpose element.

Since the final goal is to represent patient consent in a XACML policy, the elements of consent in the decision table are essentially attributes that will be compared with those provided in an access request. As discussed in Section 2.2.2, XACML attributes are subject, action, resource or environment attributes. It is therefore important to make a mapping between these attribute types and the elements in the decision table. The grantee, grantor and patient elements are subject attributes, while the action element is the action attribute. The data element is a resource attribute and purpose, context and validity period are environment attributes.

4 Conflict detection

The previous chapter presented the method for the specification and capture of patient consent, which will be the input to the process of the creation of a XACML policy. When a grantor specifies consent, it is important to compare the various authorisations that make up the consent to check for any conflicts. Conflicts may cause an access request to elicit both permit and deny responses from the policy, a situation that may cause unwanted or unprecedented effects.

In this chapter, the issue of conflict detection among the authorisations in patient consent is addressed. The first part of the chapter defines conflict, its disadvantages and presents the types of conflict, while the next sections outline the proposed method for detecting conflict in consent authorisations that have been specified by a grantor. This chapter then concludes with a brief analysis of the methods presented.

4.1 Conflict

Conflicts in policies arise when the objectives of two or more authorisations cannot be simultaneously met, due to positive and negative authorisations applying to the same objects [34]. In general, when multiple authorisations apply to the same data, potential is created for some form of conflict [12]. Conflict should be detected and resolved, to stop one authorisation from preventing the activities of another authorisation from being performed, or to prevent the target objects from going into unwanted states [35]. In general, the goals of conflict detection [13] can be summarized as follows:

- to identify an actual conflict that has occurred;
- to predict that a potential conflict may occur in the future; and
- to communicate the actual or potential conflict to a resolution process.

To identify or predict conflict between authorisations, one must be able to identify existing or possible overlaps in the elements of the authorisations. This is because without some kind of overlap between the elements of two authorisations, there can be no conflict between them [17]. Conflict occurs when the subjects, actions, targets and conditions of authorisations overlap [13, 15, 12].

There are two main types of conflicts:

- **Modality conflicts**
  These are inconsistencies that exist in a policy specification, when authorisations with opposite modalities refer to the same subjects, actions, conditions and targets. This type of conflict can be discovered by syntactic analysis of authorisations [12, 13]. Specifically, if there are two authorisations, one that is positive and another that is a negative authorisation, applying to the same target, an actual conflict can be detected by checking to see whether they refer to the same subjects, conditions or actions.

- **Application specific conflicts**
  These types of conflict are brought about by inconsistencies between authorisations and external criteria or other organisational polices [16]. An example of this is the conflicts that arise from the principles of separation of duty. These conflicts cannot be discovered by examination of the authorisations in the policies; additional external information like the conflicting organisational policies is required.

A patient's consent policy includes a number of authorisations. Each authorisation is a positive or negative authorisation, and together these authorisations will be used to control access to the patient's medical data. This existence of multiple positive and negative authorisations creates the potential for modality conflict in a patient's consent policy. Since this thesis focuses on the patient policy itself, and is not concerned with external policies or criteria, application specific conflicts are not considered.

4.2 Conflict detection algorithm for patient consent

Conflict detection must be done on creation of the patient consent policy to ensure the consistency of the authorisations that make up the policy. In addition, conflict detection must be done every time there is an update or modification of the patient consent policy to ensure that changes do not introduce conflict. The authorisations stored in the decision table are used in succeeding runs of the conflict detection algorithm, when new authorisations are added or when some authorisations are modified.

From the previous chapter, consent was specified as either default or specific consent. Default consent is made up of the general authorisations, while specific consent is made up of the specific authorisations that are the exceptions to the default consent. For this reason, default consent will intrinsically be in conflict with the specific authorisations, and conflict detection between default and specific consent authorisations is therefore not necessary. Consequently, conflict detection is performed separately for default and specific consent. On the other hand, conflict detection is done amongst access and delegations authorisations alike, since delegation authorisations are essentially potential access authorisations.

Modality conflicts result from the overlap of the subject, resource, action and condition elements in authorisations with opposite modalities [15, 16, 17]. The following sections discuss the proposed method to detect these overlaps.

4.2.1 Subject overlap

For authorisations to be in conflict, they have to refer to the same subject. This section describes how subject overlap is detected in the authorisations.

When comparing authorisations, subject overlap is detected using the grantee element. The grantee elements of the authorisations are compared to see whether they overlap. This is straightforward if the grantee elements being compared are unique IDs. Then the IDs are compared to see if they are equal, and if indeed they are, then subject overlap exists. However, the grantee element could also contain roles or groups. Roles and groups are also compared like unique IDs, but unlike unique IDs, this comparison is not sufficient. This is because roles or groups may be arranged in hierarchies, such that one role/group inherits permissions from another [25]. This is common to roles; e.g., a 'cardiologist' role may inherit permissions from a 'doctor' role, which may in turn inherit permissions from a 'medical staff' role. In this case, subject overlap exists if the roles/groups of the authorisations being compared are related in the hierarchy, such that one of the roles/groups is a parent or an ancestor of the other.

A grantee can have any number of roles and belongs to any number of groups. It is therefore possible for a grantee to belong to two roles and/or groups that have opposite permissions for the same authorisation. It is also possible for the grantee to present both of these roles and/or groups to the Policy Enforcement Point in a request context, therefore eliciting both 'permit' and 'deny' responses and creating conflict. Therefore, there has to be a way to predict to some extent the roles or groups that grantees can possibly possess at the time of performing an access request. This is done by use of attribute tables.

4.2.1.1 Attribute tables

Attribute tables identify for each grantee specified in the decision table of a patient's consent policy, the other possible roles or groups that the grantee can have amongst those specified in the patient's policy.

As specified in Section 3.2.2, healthcare institutions should have a list of roles and groups, using which the grantor can specify consent for healthcare professionals. These institutions should also keep a 'role/group reference' at the Policy Administration Point of the XACML language model that shows possible combinations of these roles and groups, i.e. those roles and/or groups that can together be assigned to one healthcare professional. Healthcare institutions can create these 'role or group clusters' from existing hospital policies like separation of duty policies. Using this role reference and the list of roles and/or groups specified in the patient's policy, it is possible to determine those from the list that may cause a subject overlap.

Given a healthcare professional's unique identity, it is possible to retrieve the healthcare roles and groups assigned to the healthcare professional from the Policy Information Point. In this way, it is possible to identify all those authorisations that apply to that grantee, which will be those authorisations that contain any of the retrieved roles or groups in their grantee element. It is also possible to identify the authorisations that could possibly apply to that grantee, by making use of the retrieved roles or groups and the role/group reference. These will be the authorisations that have in their grantee element, any of the roles or groups that are identified to be in the same role or group cluster with those roles or groups that the healthcare provider currently possesses.

The creation of the attribute tables is done by first creating a healthcare grantee list from the decision table that is comprised of the following:

- names of healthcare roles or groups e.g. doctor or nurse,
- unique identities of individuals who are also healthcare professionals

The list of unique identities of individuals who are also healthcare professionals is sent to the Policy Information Point (PIP) from the Policy Administration Point (PAP), where the specification of the policy is being done, through the Policy Decision Point (PDP) and the context handler. All the healthcare roles of these individuals are retrieved and returned to the PAP in the same manner, through the context handler and the PDP. At the PAP, an attribute table is created that has the healthcare grantee list in one column, and another column in which the possible roles or groups of each grantee will be listed. For a role or group healthcare grantee, the role/group reference is looked up. Any role/group on the healthcare grantee list that is found to be in the same role/group cluster with the healthcare grantee role/group being looked up, is added to the 'possible roles/groups' column of the healthcare grantee being looked up. If the grantee is a unique identity, then this individual's healthcare roles and groups that were previously fetched from the PIP are compared with the roles/groups in the healthcare grantee list. If there is a match, such that the individual is found to possess some of the roles/groups in the healthcare grantee list, these role/groups and the roles/groups in their 'possible roles/groups column' are entered in the individual's 'possible roles/groups' column of the attribute table.

The attribute table shows the other possible roles or groups that can be possessed by a healthcare grantee. A (potential) subject overlap can now be detected for two authorisations being compared. There is a subject overlap if the grantee elements have the same value, if the grantee elements are related in a hierarchy, or if the grantee element of one of the authorisations has in its 'possible roles/groups column in the attribute table, the grantee element of the other authorisation.

4.2.2 Data overlap

For potential or actual conflict to exist between two authorisations, the authorisations must refer to the same data. Data overlaps are checked in the XPath expressions that represent the patient's medical information in the authorisations. In the literature, a number of algorithms that detect containment or intersection between XPath have been presented [21, 22, 32, 33, 36]. For the detection of data overlap, the preferred XPath

intersection algorithm is run on the data elements of the authorisations. If the XPath expressions intersect then a data overlap exists and conflict is possible. Otherwise, if the XPath expressions have do not intersect, then there is no data overlap, and the authorisations cannot be in conflict with each other, since they refer to completely different data items.

4.2.3 Action overlap

If the actions of two authorisations are different, no action overlap exists. However, if the actions are arranged in a hierarchy, such that some actions inherit their properties from other actions, then an action overlap exists if the actions are related. This relationship must be such that one action is a parent or an ancestor of the other action, e.g. a hierarchy can exist such that 'access' is a parent of 'read' and 'write'. Finally, if the actions are equal, action overlap exists.

4.2.4 Condition overlap

The elements of the authorisations that are compared during the condition overlap check are the purpose and context elements. The validity period is not considered during the detection of conflict between authorisations. This is because the validity period specifies the length of time that the authorisation is valid, but does not contribute to the authorisation itself. The conditions of two authorisations can have one of the following relationships:

- condition 1 and condition 2 intersect
- condition 1 and condition 2 are disjoint
- condition 1 and condition 2 are equal

Two conditions are considered to intersect when they have the same value for one of the purpose or context elements, but different values for the other one; or when one of the authorisations has the value 'all' in the purpose or context elements, since the value 'all' will include whichever value specified in the other authorisation's purpose or context element. On the other hand, two conditions are disjoint when all the purpose and context values are different and none of the purpose or context elements of both authorisations has the value 'all'. Finally, two conditions are considered equal when the conditions contain the same values, or when one condition is a subset of the other. One condition is a subset of another when:

- the purpose and context values of both conditions are the same,
- one condition has the value 'all' for the purpose element and the other has a specific purpose, while the context elements of both conditions contain the same specific value, or the other condition has the value 'all'; or vice versa;
- one condition has the value 'all' for both purpose and context elements; or
- both conditions contain 'all' for the purpose element, while the other element is such that one condition has the 'all' value, while the other has a specific value; or vice versa.

When the conditions are equal, then condition overlap exists. When two conditions intersect or are disjoint, there is no condition overlap. This is because regardless of the fact that the conditions may have the same value for one condition element, the other condition element will be different and thus the conditions in their entirety (purpose and context) are different. The consequence of this is that authorisations, whose conditions do not overlap, will never be applicable to the same access request. Since their conditions are essentially different, they can never be in conflict with each other.

4.2.5 Conflict detection algorithm

The sign, action, subject, data and condition overlap checks together constitute the conflict detection algorithm, which is summarized in Figure 4 below.

When two authorisations are being compared for conflict, the signs of the two authorisations are compared first. If they are the same, i.e. both '+' or both '-', then the conflict detection algorithm should return a 'no conflict' response. Otherwise, the actions of the two authorisations are then compared. If there is no action overlap then conflict detection algorithm should send a 'no conflict' response, else, the conditions of the two authorisations are compared. If the conditions of the two authorisations overlap, then conflict is possible, depending on whether the grantee and data elements overlap, which should be checked next. When the conditions do not overlap, then the conflict detection algorithm should return a 'no conflict' response. Next, the check for subject overlap is performed. If there are no subject overlaps then conflict detection should return a 'no conflict' response, else the data overlap is checked next. If there is no overlap in the XPath expressions that are the data elements of the two authorisations, then the conflict detection algorithm should return a 'no conflict' response. If an overlap is detected in the XPath expressions then the conflict detection algorithm should return a 'conflict' response.

The authorisations are going to be compared and the results of conflict detection stored in conflict detection matrices. These matrices are created to store the results of the comparison of each authorisation with every other authorisation, and are shown in Section 4.3 below.

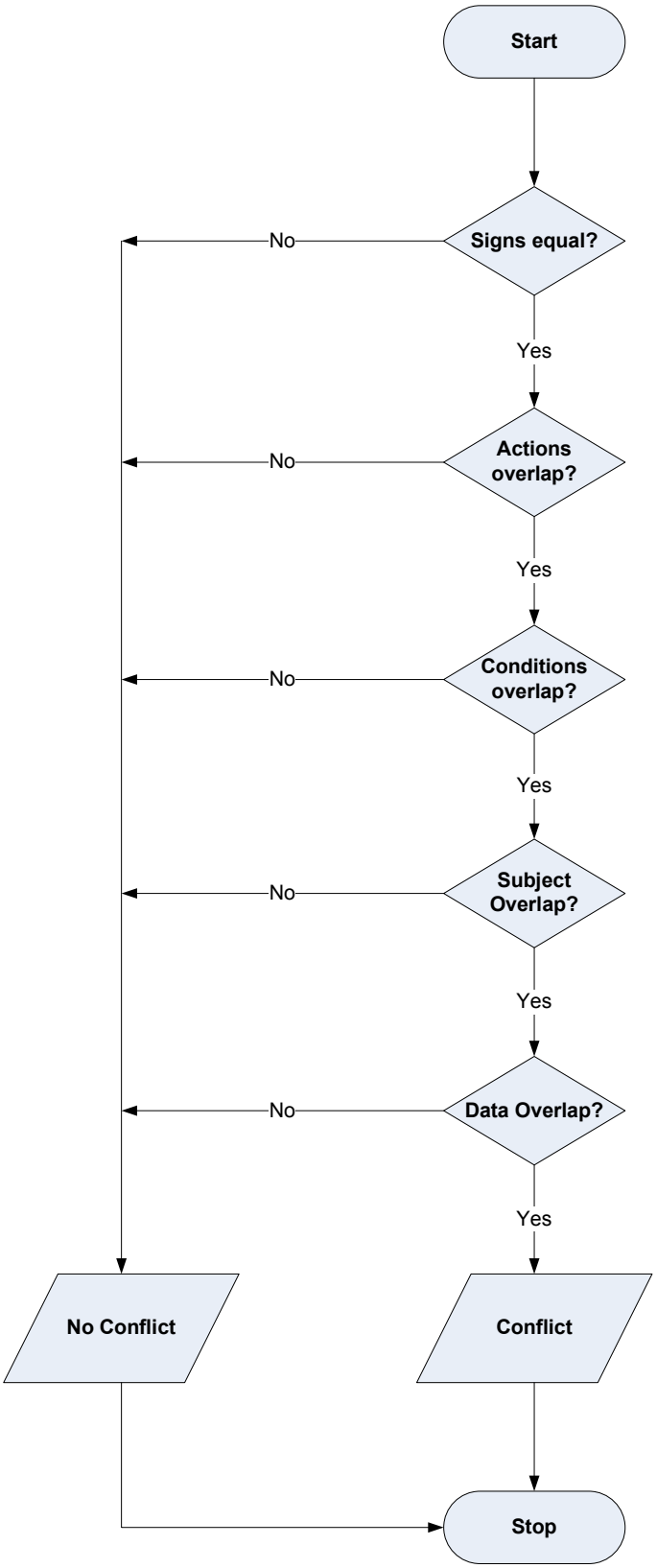

Figure 4: Conflict detection algorithm flowchart

4.3 Working example

This section shows how to detect conflict amongst the authorisations for our working example (See Section 3.3).

Separating the default and specific consent

Rows 1 and 2 of the decision table contain the default consent. These two rules will be compared with each other for conflict detection. The rest of the rows contain the specific authorisations that will be compared amongst one another for conflict.

Creating the attribute table

The grantees in the authorisations are personal-doctor, doctor, Dr.-John-Mathews-ID, husband-ID and mother-ID. The personal-doctor and doctor attributes are role grantees, while the rest are unique identifiers. The distinction between roles and identifiers can be made from the GUI.

The grantee list consists of grantees that are healthcare providers. Assuming that the patient specified that the husband and mother are not healthcare professionals, the personal-doctor, doctor and Dr.-John-Mathews-ID grantees constitute the grantee list.

Dr. John Mathews' unique identity is passed from the PAP to the PDP, from the PDP to the context handler and from the context handler to the PIP. At the PIP, Dr. John Mathews' healthcare roles or groups are looked up. Assuming that he possesses only the doctor role, this role is sent back to the PAP through the context handler and the PDP. At the PAP, the role reference is consulted. It is assumed that the role reference indicates that any healthcare provider with the role doctor can have the personal-doctor role and vice versa[6]. Therefore, the attribute table for this consent is as shown in Table 4 below.

Table 4: Attribute table for the working example at the PIP

Grantee	Possible roles and groups
personal-doctor	doctor
doctor	personal-doctor
dr.-John-Mathews-ID	doctor, personal-doctor

Conflict detection

A conflict matrix is created for each authorisation, such that each authorisation is compared with all the other authorisations for conflict detection, in consultation with the conflict detection tables and the attribute table. If conflict is detected between two

[6] Since a personal-doctor is a doctor, and anyone with the doctor role can become the patient's personal-doctor.

authorisations, then the corresponding cell in the matrix is marked with 'C' for conflict. On the other hand, 'N' will represent the lack of conflict.

Default authorisations

There are two default authorisations, and therefore one conflict matrix will be created as shown in Figure 5. Clearly, there will be no conflict created by these authorisations since they have the same sign.

Auth No.	2
1	N

Figure 5: Conflict detection matrix for the default authorisations

Specific authorisations

The rest of the authorisations in the decision table are specific authorisations, and their conflict matrices are shown in Figure 6.

Auth No.	4	5	6	7	8	9	10	11	12
3	N	N	N	N	N	N	N	N	N

Auth No.	5	6	7	8	9	10	11	12
4	N	N	N	N	N	N	N	N

Auth No.	6	7	8	9	10	11	12
5	N	N	N	N	N	N	N

Auth No.	7	8	9	10	11	12
6	N	N	N	N	N	N

Auth No.	8	9	10	11	12
7	N	N	N	N	N

Auth No.	9	10	11	12
8	N	N	N	N

Auth No.	10	11	12
9	N	N	N

Auth No.	11	12
10	C	C

Auth No.	12
11	N

Figure 6: Conflict detection matrices for the specific authorisations

The conflict matrix of authorisation 3 shows that it has no conflict with any other authorisation. Authorisation 3 does not have any conflict with authorisations 4-10 since they have the same sign. It also has no conflict with 11-12 since there is no subject overlap. The same applies to the conflict matrices for authorisations 4-9; they have the same sign with all the authorisations, apart from authorisations 11 and 12 with which there is no subject overlap. Authorisation 10 however is in conflict with 11 and 12. This is because the authorisations have opposite signs, the same action, equal conditions, the subject is the same, and the data overlaps. The data element of authorisation 11 is an XPath expression representing the patient's Gynecological-information, which is a subset of 10's data element which is an XPath expression representing all of the patient's information. Similarly, the data element of authorisation 12 is an XPath expression representing part of the patient's blood pressure information, which is also a subset or authorisation 10's data element.

4.4 Discussion

The conflict detection algorithm has been presented in this chapter. It detects actual and potential conflict amongst authorisations specified by a grantor and stored in the decision table.

The new idea of attribute tables has been proposed in this chapter, to aid the detection of potential conflicts by detecting potential subject overlaps between the grantee elements of two authorisations. This is done by looking up a healthcare institution's roles/groups reference for healthcare grantees represented as roles/groups, and looking up the healthcare roles and groups for grantees specified using unique identities. The role/group reference may change if policies in the healthcare institutions change, and the roles or groups that an individual has may also change. This implies that the conflict detection process may become inaccurate due to these changes, which may lead to an incorrect XACML policy.

Role/group references are created from institutional policies that change rarely. In case of any changes, the role/group reference can be updated with the relevant changes. On the other hand, the roles and groups that are assigned to an individual healthcare provider may be susceptible to frequent change. Those roles or groups that are related to the healthcare qualifications of an individual may change rarely, but those that are administrative or linked to other criteria may change often. A way of dealing with these changes in the roles/groups reference and in individual's roles and groups is by recreating the affected attribute tables. The affected attribute tables will be those that contain the roles/groups in that have changed in the role/group reference, or those that contain the ID of the healthcare provider whose roles/groups have changed. If the 'possible roles/groups' column of an attribute table changes, the conflict detection and subsequent stages in this process of creating a XACML policy may be executed to keep the generated policies as up to date as possible.

It is important to note that the conflict detection algorithm is dependent on the method of performing access requests that will be implemented at the PEP. The algorithm presented here assumes that an access request is presented with one value for each element of consent. If the access requester can present multiple action, subject, data or environment attributes to the PEP, these should be broken down to a number of access requests at the context handler.

The conflict detection algorithm is made of four primary operations:

- the string compare operation, that is performed when comparing the sign, action, condition and grantee elements of two authorisations;
- the XPath expression intersection detection, which is done when checking whether the data elements of two authorisations overlap; and
- the table look-up operation for consulting the attribute table.

The table look-up is the least expensive operation, with a constant time complexity, while the string compare operation is a linear operation. On the other hand, the algorithms that detect intersections between two XPath expressions run in quadratic time in the worst case [22]. Therefore, the most expensive operation during conflict detection is performed when checking for data overlaps.

Whenever a grantor adds a new default/specific authorisation in a decision table, the new authorisation must be checked for conflict with all other default/specific authorisations. The less expensive string-compare and table look-up operations are performed first, while the data overlap is checked last. In many cases therefore, conflict detection between two authorisations will cease before the data overlap check, which is the most expensive operation, is performed.

5 Conflict resolution

A discussion on how conflict can be detected amongst authorisations specified as part of a patient's consent policy, has been presented in Chapter 4. After the conflict is detected amongst authorisations, this conflict must be resolved. This chapter gives the steps that are taken to resolve conflict. It stars by presenting the conflict resolution policies in the literature, and then uses some of these policies to resolve conflicts in patient consent. This chapter also discusses the assigning of priorities to authorisations, and concludes with a discussion.

5.1 Conflict resolution policies

Conflict resolution is done to get rid of conflicts and their effects. A number of conflict resolution policies have been discussed in [18]. These are:

- **Preference policy**

A preferred authorisation is determined, such that when both authorisations i.e. positive or negative are derived for a particular access request, one is chosen over the other. Negative authorisations are preferred in restrictive systems where security is of primary importance, while positive authorisations are chosen in systems that are more open and user friendly.

- **Globality/Locality policy**

Using the locality policy, the more specific policy is preferred over the other. For example in an organisation, policies that apply to a department are followed over those specified for the whole organisation. As another example, if a subject inherits both positive and negative authorisations from an ancestor in a subject hierarchy, then the authorisation from the ancestor closest to the subject is preferred. The globality policy applies in the opposite way, such that the more general policy applies. In the example of the organisation, the organisation's policy would take precedence over the departmental policy; while in the second example, the subject would inherit the authorisation of the furthest ancestor. This policy is non-deterministic because it is possible to have authorisations that are at the same level and so none takes precedence.

- **Majority/Minority policy**

  This policy is used in situations where numbers count. In the majority policy, the authorisation that will hold is the one that has more occurrences than the other does. The reverse is the case in the minority policy, the authorisation with the least occurrences will have the highest preference. Like the globality/locality policy, this policy is also non-deterministic as it is possible to have a tie in the numbers, and so not be able to determine which authorisation should hold.

Conflict resolution for patient consent will be performed according to these policies.

5.2 Conflict resolution algorithm for patient consent

In the resolution of conflict for patient consent policies, the locality and preference policies are applied. The use of the locality policy follows from the models of e-consent for the healthcare domain, which contain general and specific authorisations. The specific authorisations are exceptions to the general authorisations and take higher precedence over the general authorisations. However, in the cases that the locality policy cannot decide which authorisation takes precedence, then using the preference policy, the grantor must make a decision on the preference of the authorisations.

The majority/minority policy is not employed since the frequency of authorisations does not count. What matters is who the grantors of the authorisations are, and what authority these grantors have; for example a number of delegated grantors may give a permit authorisation for access to a patient's medical data, but if the patient issues a deny authorisation, the patient's authorisation must take precedence.

A patient consent policy may be made up of authorisations issued by different grantors. The first step in resolving conflict between authorisations is to check the grantors of the two authorisations. If the same grantor issued the conflicting authorisations, then the next step of conflict resolution is performed. If the patient has issued one authorisation while another grantor has issued the other, then the authorisation specified by the patient has higher preference. However if the authorisations are issued by different grantors, of whom none is the patient, a hierarchy created by the patient may exist that shows which grantor is senior to the other, and the senior grantor's authorisation is assigned higher preference. This grantor hierarchy must be a total order to be deterministic. If no grantor hierarchy exists, then the grantor may not be allowed to issue the conflicting authorisation.

The second step involves checking the specificity of the authorisations. The locality resolution policy is employed to identify the more specific authorisation, which will be given higher preference. The locality policy applies:

- if the data of one authorisation is a subset of the other;
- if the actions have an action hierarchy such that one senior than the other;
- if the grantees are roles/groups that have a role/group hierarchy and one role/group is senior to another; and
- if the condition of one authorisation is a subset of the other.

The locality policy is applied when the two authorisations being compared are such that one authorisation has the data, action and role/group elements as more senior in all three hierarchies. The locality policy is also applied when one authorisation has at least one of the data, action and role/group elements being more senior than the one in the other authorisation, while the rest of the elements of the authorisations are at the same hierarchical level. In these cases, the more specific authorisation is the one with the senior elements. If the data, action and role/group elements are at the same level of hierarchy, then the condition elements are used to decide on locality, such that the authorisation with the more specific condition is the more specific authorisation.

XPath containment algorithms are used to detect whether or not one data element is a subset of the other. The action and role hierarchies can be stored at the PAP as trees and a breadth-first search algorithm used to traverse the tree and find the more senior action or role/group, which will be higher in the tree than the other authorisation's action or role/group.

When the globality or locality policy cannot be employed[7], the grantor's opinion is solicited interactively to obtain the preferred policy. This is applying the preference policy of conflict resolution, where the grantor has to choose which of the two authorisations takes precedence over the other. The conflict resolution algorithm is summarised in Figure 7.

[7] When both rules have at least one senior data, action or role/group, and therefore none of the rules is more local, or the grantor opts to assign priorities manually.

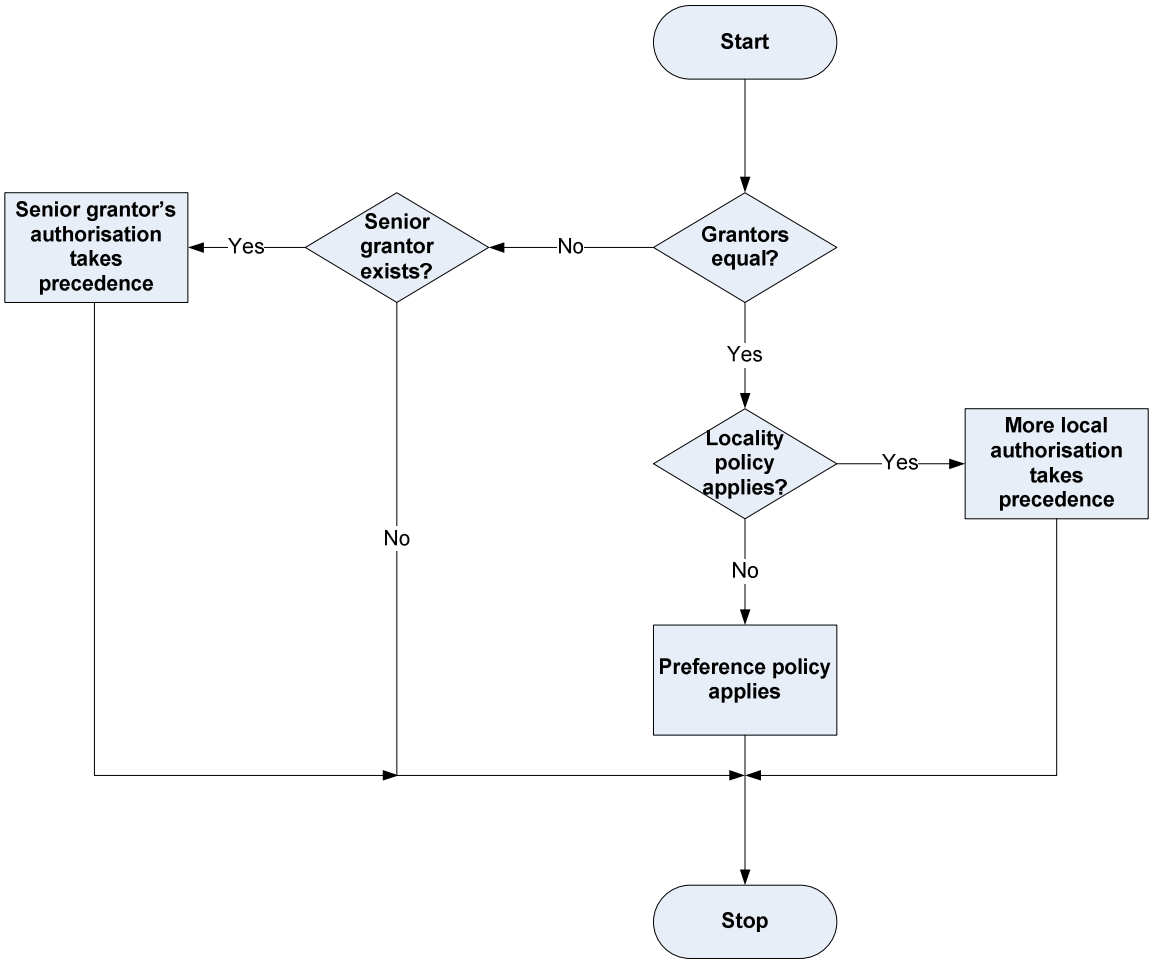

Figure 7: Conflict resolution algorithm

During conflict resolution, the grantor is given informative messages on how the existing conflicts are resolved. The grantor may be presented with the option of changing the priorities that are assigned automatically by the locality policy, especially if the grantor is the one who specified both authorisations, or when the grantor has seniority over the grantor of the other authorisation.

5.2.1 Priority

On deciding which authorisations take preference over others, this preference is indicated in the authorisations. This is done by assigning priorities to the authorisations in the case that the authorisations do not have priority values, or re-assigning priorities to those authorisations that had priorities assigned in previous conflict resolution runs.

The authorisations are divided into those that are not involved in any conflict and those that have conflict with at least one other authorisation. Furthermore, these two groups of authorisations are divided into the default and specific authorisations. All the authorisations that were not involved in conflicts with any other authorisations are given

priority values in a linear fashion, with any authorisation being assigned any priority value. However, the default authorisations in this group must be assigned the lower of these priority values, since the default authorisations are the more general authorisations, and apply only in general situations. On the other hand, the specific authorisations are assigned higher priority values. Finally, the group of conflicting authorisations is assigned higher priority values than the group of non-conflicting authorisations, with default authorisations having lower priority values than the specific authorisations, and the authorisations that have higher preference according to the conflict resolution algorithm being assigned higher priorities. This is illustrated in Figure 8 below.

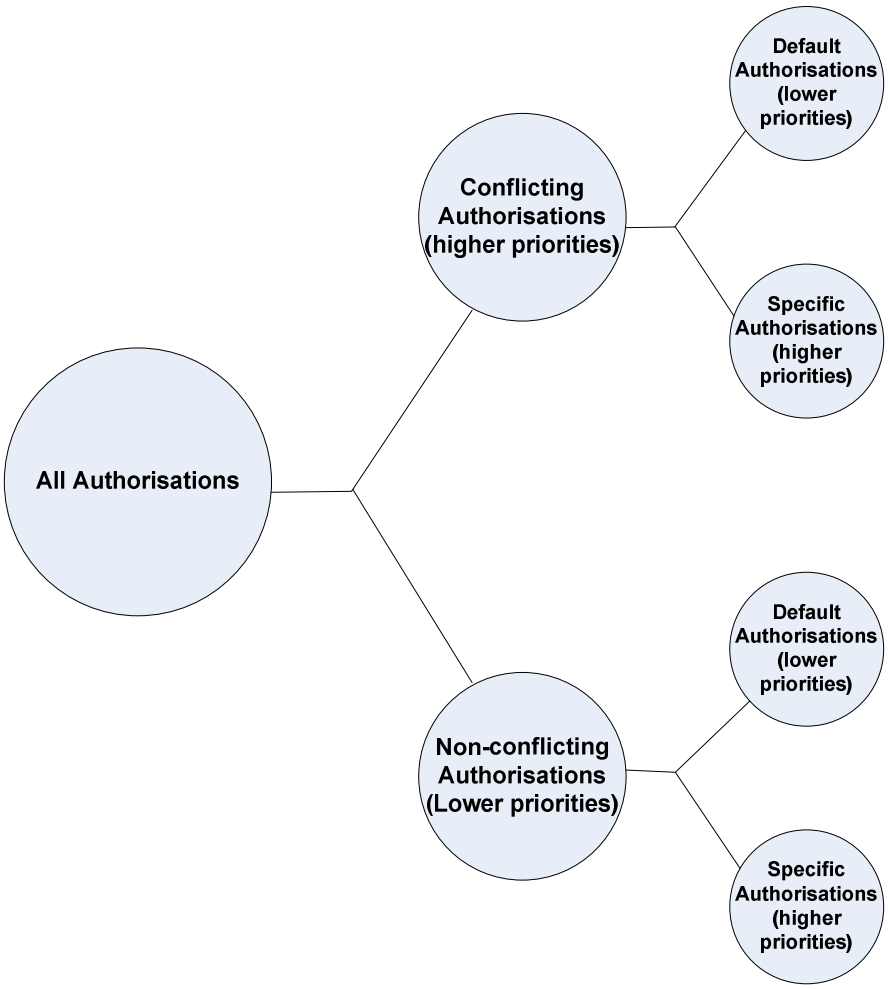

Figure 8: Priority assignment across the authorisations

5.2.2 Priority assignment to conflicting authorisations

Priority assignment to non-conflicting authorisations is straightforward since it is done in a linear fashion. However, for the conflicting authorisations, higher priority must be assigned to those rules that are assigned higher precedence during conflict resolution. When conflict is resolved using the locality policy, the more local authorisation is given higher priority. However, when conflict is resolved using the preference policy, the grantor independently assigns priorities to authorisations, during which the grantor may introduce cycles.

A cycle is introduced in the following manner. Assume there are three authorisations, ‘a’, ‘b’ and ‘c’. Further, assume that authorisation ‘a’ and ‘b’ conflict, authorisation ‘b’ and ‘c’ conflict, and authorisation ‘c’ and ‘a’ conflict. If the grantor uses the preference policy to resolve conflict, then the grantor may assign higher priority to authorisation ‘a’ to resolve the conflict between ‘a’ and ‘b’, and higher priority to authorisation ‘b’ to resolve the conflict between ‘b’ and ‘c’. The grantor may then assign a higher priority to authorisation ‘c’ when resolving conflict between authorisations ‘a’ and ‘c’, thereby creating a cycle, as shown in Figure 9.

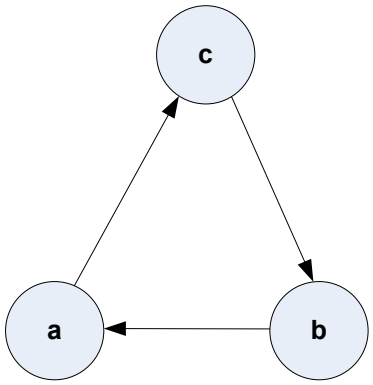

Figure 9: Cycle between authorisations ‘a’, ‘b’ and ‘c’

In graph theory, a cyclic path is a path that has the same first and final vertices [37]. Cycles are undesirable since they introduce conflict. In the example given above, authorisation ‘a’ has higher precedence that authorisation ‘c’, since authorisation ‘a’ has been assigned a higher priority that authorisation ‘b’, which has in turn been assigned a higher priority that ‘c’. At the same time, authorisation ‘a’ has lower precedence than authorisation c, since authorisation c has been assigned a priority that is higher than ‘a’.

When performing conflict resolution using the locality policy, cycles cannot be introduced. This is because when authorisation ‘a’ is more senior/specific than authorisation ‘b’, which is then more senior/specific than authorisation ‘c’, it is not possible for authorisation ‘c’ to be more senior/specific than authorisation ‘a’. Furthermore, the action and role/group hierarchies are represented as trees, which inherently do not contain cycles [38].

To prevent the introduction of cycles, a Directed Acyclic Graph (DAG) is incrementally created during conflict resolution. The DAG is first created for all the authorisations whose conflict has been resolved using the locality policy and afterwards, those that have been resolved using the preference policy are added to the DAG. This graph can be connected or can be made up of a number of disconnected components. A separate DAG must be created for the default and the specific authorisations. Each vertex of the graph will represent one authorisation, with the name of each vertex being the ID of the authorisation it represents. The edges between the vertices represent the preference assigned to the authorisations. An authorisation 'a' with a higher preference than an authorisation 'b' will have a directed edge pointed towards it from vertex 'b'.

The DAG is first checked for the existence of a directed path between the authorisations' vertices. If a directed path exists, this indicates that the authorisations were in conflict with other authorisations, and conflict has already been resolved between the two authorisations, albeit indirectly. Therefore, conflict resolution does not need to be done again. The grantor is simply informed of the existing priorities. This is important since it keeps cycles from being created in the DAG. The opportunity to disagree with the currently assigned priorities may be afforded to the grantor. Using purely the preference resolution policy, the grantor could reassign priorities to the authorisations. This could be done by displaying the authorisations as structured in the graph for the grantor to change while continuously checking for conflict. Alternatively, the grantor could be allowed to assign all the priorities independently.

If no directed path exists between the authorisations, conflict resolution is done and the vertices are added to the graph if need be[8]. A directed edge is added between them accordingly; from the vertex of the authorisation that was assigned the lower priority, pointing towards the authorisation that was assigned the higher priority.

After all the conflicting authorisations have been added to the DAG in this way, the authorisations have to be assigned priority values that reflect the structure of the DAG. The graph should therefore be traversed in a manner that will output the list of vertices that maintains the precedence relationships in the graph. The topological sort [38] is an operation that processes the vertices of the graph such that no vertex is processed before any vertex that points to it. This implies that no vertex is processed before any of its predecessors. In the DAG, all the predecessors of a vertex have a lower precedence than the vertex. Therefore, a topological sorting of the graph will produce a list that maintains the precedence order coded in the DAG.

If the DAG that is produced is disconnected, then during the topological sort all the nodes in a connected component are visited before moving to the next connected component. Since the vertices with the higher preferences are deeper in the graph, a topological sort for one connected component will produce a list of authorisation IDs arranged in ascending order of preference. Priority values are then assigned to the list in ascending order. It is important to note that a topological sort does not produce a unique list, and any legal topological ordering will do.

[8] One or both vertices may already exist in the graph if the authorisations were already in other conflicts.

In summary, conflict resolution is performed in the following manner:

- the conflicting authorisations are divided into default and specific authorisations, and the following steps are then performed separately for both groups:
    - the DAG is checked for the existence of a directed path between the authorisations, if none exists conflict resolution can go on, otherwise the grantor can change the existing precedence;
    - a decision is made on whether the locality or the preference policy will be used to resolve conflict;
    - all the authorisations whose conflict is resolved using the locality policy are added to a DAG; and
    - for those authorisations whose conflict is resolved using the preference policy, the grantor indicates preference; they are added to the same DAG that has already been created for those authorisations whose conflict was resolved using the locality policy.

- a topological sort is then performed on both DAGs, and priority values assigned to the resulting lists, such that the list from the default authorisations' DAG is assigned lower priority values than the list from the specific authorisations' DAG.

The priority values are added to a new 'priority' column of the decision table, in the corresponding row of each authorisation.

5.3 Working example

The conflict detection matrices generated in Section 4.3 will be the input to the conflict resolution algorithm. Each step is outlined below.

Separating the conflicting and non-conflicting authorisations

In our example, the conflict detection matrix that has conflict is for authorisation 10, which is found to be in conflict with authorisation 11 and 12. These authorisations are separated from the rest of the authorisations for conflict resolution.

Assigning priorities to non-conflicting authorisations

The non-conflicting authorisations are divided into default (1 and 2) and specific (3, 4, 5, 6, 7, 8, 9) authorisations. The default authorisations are assigned priority values in a linear fashion, while the specific authorisations are assigned higher priority values than the default authorisations, also in a linear fashion. Authorisation 1 and 2 are assigned priority values '1' and '2' respectively, while Authorisations 3 – 9 are assigned priority values '3' – '9' respectively.

Creating the DAGs

The conflicting authorisations are divided into default and specific authorisations. In our example, there are no default rules amongst the conflicting rules, and therefore only one DAG will be created for the specific authorisations. A directed path, between the first pair of conflicting authorisations, is searched for in the DAG. Since no DAG exists yet, searching the graph will not yield any results, and conflict resolution can be done for these two authorisations.

To decide on which conflict resolution policy will be used to resolve conflict, the elements of the authorisations are compared. The grantors of the two authorisations are compared and since the grantors are the same, the authorisations are compared for specificity. The conditions of the two authorisations are equal, and the actions and grantee elements are the same. The XPath expressions that are the data elements of the two authorisations are taken through the XPath containment algorithm. Authorisation 10's XPath expression selects all element nodes that are children of the patient's ID. This is essentially the entire patient's medical information. However, authorisation 11's XPath expression selects all the children of the Gynecological-information node, which is a child of the patient-ID node. Clearly, the nodes that are represented by authorisation 11's XPath expression are a subset of those represented by authorisation 10's XPath expression. Therefore, the XPath expression that belongs to authorisation number 11 is contained in the XPath expression in authorisation 10. Since the action, grantee and condition elements of the two authorisations are equal, and the data elements are related by containment, then the locality policy is used to perform conflict resolution. Authorisation 11 is local or specific and therefore takes precedence to authorisation 10.

The Directed Acyclic Graph can now be updated with two vertices that are named the IDs of the authorisations and one directed edge. Since authorisation 11 is given higher preference to authorisation 10, the directed edge shall point to authorisation 11. This is shown in Figure 10.

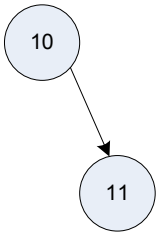

Figure 10: Directed acyclic graph for authorisation 10 and 11

This process is repeated to resolve the conflict between authorisations 10 and 12. Since authorisation 12 is more specific than authorisation 10, the locality policy is also used to resolve this conflict. In this example, there is no conflict that required to be resolved using the preference policy. If this was the case, all the conflicts that could be resolved using the locality policy would have been resolved and added to the DAG first, before proceeding to those that would have been resolved using the preference policy.

The DAG that is created for the resolved conflicts is shown in Figure 11.

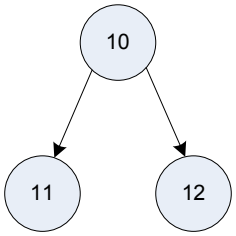

Figure 11: Complete DAG for the patient policy specified in the decision table

Assigning priorities to conflicting authorisations

A topological sort will be performed on the dag to obtain a total order from the partial order represented by the graph. Priority is then assigned in ascending order in the list as shown in Figure 12.

Topological sort	**10**	**12**	**11**
Priority	10	11	12

Figure 12: Topological sort and assigned priorities

Finally, these assigned priorities are added to the authorisations in the decision table.

Table 5: Decision table containing the patient consent policy for patient-ID

Auth No.	Grantor	Grantee	Action	Data	Effect	Purpose	Context	Validity Period	Auth type	Priority
1	patient-ID		read	/patient-ID/*	-				A	1
2	patient-ID		write	/patient-ID/*	-				A	2
3	patient-ID	personal-doctor	read	/patient-ID/*	+	all	all		A	3
4	patient-ID	personal-doctor	write	/patient-ID/*	+	all	all		A	4
5	patient-ID	doctor	read	/patient-ID/*	+	treatment	all	1 year	A	5
6	patient-ID	doctor	write	/patient-ID/*	+	treatment	all	1 year	A	6
7	patient-ID	Dr.-John-Mathews-ID	read	/patient-ID/*	+	treatment	emergency		A	7
8	patient-ID	Dr.-John-Mathews-ID	write	/patient-ID/*	+	treatment	emergency		A	8
9	patient-ID	husband-ID	read	/patient-ID/*	+	all	all		D	9
10	patient-ID	mother-ID	read	/patient-ID/*	+	all	all		A	10
11	patient-ID	mother-ID	read	/patient-ID/Gynecological-information/*	-	all	all		A	12
12	patient-ID	mother-ID	read	/patient-ID/Blood-pressure[age<=3]	-	all	all		A	11

5.4 Discussion

This chapter has presented a method of resolving conflict amongst authorisation specified in a patient consent policy, with the use of the locality and preference policies that have been discussed in the literature. Furthermore, the idea of applying a topological sort on a Directed Acyclic Graph, to aid in the assigning of priorities to the authorisations without introducing cycles has been presented.

Three main operations are carried out during the conflict resolution:

- String manipulation operations like string compare operations;
- breadth-first search to search the hierarchies;
- table look-ups for the conflict detection matrices; and
- the XPath containment algorithm that determines whether the data elements have a containment relationship.

String manipulation operations have a linear time complexity, while table look-ups have a constant time complexity. However, depending on the XPath syntax used to represent the data element of the authorisations, and the containment detection algorithm chosen, the time complexity can be polynomial for simple XPath expressions, to undecideable for expressive XPath. It is therefore important to keep the XPath expressions in the data elements of the authorisations as simple as possible by restricting the syntax to the basic elements such as the child (/), descendant (//), attribute ([]) and the wildcard (*) elements, which in practice is sufficient to browse patient medical data.

6 Translation to XACML Policy

The authorisations that make up the patient consent policy have been specified, analyzed for conflict, and existing conflicts have been resolved. Essentially, these authorisations compose a complete policy that can be used by an access control engine. However, our goal is to represent patient consent policies using the standardized XACML policy language. Therefore, the translation of the authorisations to XACML policies needs to be done. This chapter outlines the proposed method to perform this translation, which is performed in two steps. First, a XACML skeleton policy is created from the authorisations that exist in the decision table, and then these skeletons are used to generate full XACML policies.

6.1 XACML Skeleton policy

A XACML policy consists of a target and rules, and each rule contains a target and a condition. A decision has to be made on what the target of the policy and the targets and conditions of the rules are going to contain. This is important since the targets and conditions determine the applicability of the policy and its rules to an access request, and must thus be composed in a manner that ensures that they are evaluated during the resolution of relevant access requests. The XACML policies that will be created in this chapter will constitute the authorisations in the decision tables that were generated after conflict resolution was performed. Therefore, a mapping has to be made from the authorisations in the decision tables, to the constituent parts of a XACML policy. This mapping is done with the use of a XACML skeleton policy.

XACML skeleton policies show the mapping from the elements that make up the authorisations to the subject, action, resource and environment attributes that constitute the targets and conditions of a XACML policy. XACML skeleton policies do not contain the syntax that makes up a complete XACML policy, and therefore cannot be processed by a XACML policy engine. Instead, a skeleton policy depicts the core components of a XACML policy, which do not change regardless of any syntax changes[9]. Furthermore, the skeleton policies are applicable across the various profiles of the XACML standard. This is because the syntax changes across the profiles, but the core components of the policies and rules do not change.

[9] For example, the draft XACML 3.0 core profile has a different syntax from the 2.0 standard, which does not affect the skeleton policy.

The skeleton policy follows the XACML policy language model discussed in Section 2.2.2, and the following section describes how each element of the model is created.

6.2 Translation to XACML Skeleton policy

The rows of the decision table will be processed one after the other, and their elements added to the skeleton policy and its rules. The translation is done in the following manner.

6.2.1 Skeleton policy set creation

The skeleton policy set is translated from the authorisations in the decision table in the following manner:

- the skeleton policy set is named using the unique ID of the patient;
- since each access request will be to perform some action on the patient's medical information, the patient's unique ID is contained in the skeleton policy set's target as the subject attribute; and
- if a grantor hierarchy exists, the first-applicable policy-combing algorithm is used, else the priority-overrides policy-combining algorithm is used.

There is only one policy set for each patient.

6.2.2 Skeleton policy creation

All the authorisations issued by the patient are kept in one skeleton policy in the skeleton policy set. If any other grantors have issued authorisations on the patient's behalf, these authorisations are kept in separate skeleton policies but in the same skeleton policy set, with one skeleton policy for each grantor. Each skeleton policy is named after its grantor.

The patient's policy is the only trusted policy. Any authorisation given by other grantors are untrusted. All untrusted skeleton policies must contain an 'Issuer' element, which specifies the identity of the grantor.

During the translation, each authorisation's grantor element is examined, and the authorisation is added to its grantor's policy.

6.2.3 Skeleton policy targets

The target of the skeleton policy that belongs to the patient must apply to all access requests related to the patient's health data, and therefore its target must remain empty. However, the skeleton policies issued by other grantors are translated in the following manner:

- all the grantees in the authorisations issued by the grantor are included in the subject attribute of the target; and

- all the actions in the authorisation issued by the grantor are included in the action attribute of the target.

No environment attributes are included in the target of the skeleton policies. This is because the environment attributes represent the conditions necessary for access, but are not the main components of an access authorisation or request.

6.2.4 Skeleton rules

Each authorisation in the decision table will be a skeleton rule. If the authorisation is a delegation authorisation then an administrative skeleton rule is created for that authorisation. The administrative skeleton rules' subject, action, resource and environment attributes will be marked with the word 'delegated'. This is to signify that when creating the full XACML policy from the skeleton policy, the syntax from the XACML delegation profile is used.

6.2.5 Skeleton rule targets

The skeleton rule targets are constituted in the following manner:

- The resource attribute of the skeleton's rule will be an authorisation's data element.

- The skeleton rule's target has to be specific to the grantee to which the rule applies. Therefore, the subject attribute of the skeleton rule target is the grantee of the authorisation that the skeleton rule represents.

- The skeleton rule target has to specify the particular action, which is applicable to the rule. Therefore, the action attribute of the skeleton rule target is the action of the authorisation that the skeleton rule represents.

A skeleton rule represents an authorisation in the decision table, which implies that its target must constitute the authorisation's core components, and as such, the environment attributes are not included.

6.2.6 Skeleton rule conditions

The environment attributes of the authorisations are kept in the skeleton rule's condition elements. Rule conditions contain the conditions of the authorisation rules, which consist of the purpose, context and validity period elements.

6.3 Priority combining algorithms

In the XACML standard, a way to represent priority values for the rules of a policy is not specified, and no algorithm exists to combine rules with regard to priority values. The simplest way to address this problem is to translate the priority values specified in the rules of the policy to the physical ordering of the rules in the policy. The rules are arranged in the policy in descending order of priority; the rule with the highest priority is written at the top of the policy, and the rest of the rules follow in descending order of priority. Any rules with equal priorities are listed in random order. This enables the use of the first-applicable rule-combining algorithm. As discussed in Section 2.2.3, this rule-combining algorithm evaluates the rules in the order in which they have been written in the policy. In this fashion, the rule with the highest priority is evaluated for an access request. This applies to skeleton rules as well. Similarly, the first-applicable policy-combining algorithm evaluates the policies in a policy set in the order of appearance. The policies should therefore be arranged in the policy set according to the grantor hierarchy. This is also the case for skeleton policies.

However, it is not always the case that policies in a policy set are arranged in order of priority in a policy set. An example is the situation where a grantor hierarchy does not exist, and all the policies have to be evaluated and the priorities need to be compared across the policies. Clearly, the first-applicable combining algorithms cannot be used. It is therefore recommended that priority values are explicitly contained in rules, and combining algorithms that process priority values are created.

To cater for priorities in XACML, the XACML standard should include a priority attribute for the rule element. This attribute would be in addition to the existing '*ruleid'* and '*effect'* attributes for the rule element, and would hold the priority value of the rule. If the priority values of rules are encoded using an attribute, then a rule combining algorithm that processes this attribute is required. A priority policy-combining algorithm is also needed, to select the policy whose decision has the highest priority across policies in the same policy set. The proposed priority combining algorithms are discussed in the following sections.

6.3.1 Priority-overrides policy-combining Algorithm

This algorithm combines the decisions of the policies in a policy set, to provide one decision to an access request based on the priorities of the policies' decisions. The policy-combining algorithm takes a policy set as an argument. For each policy in the policy set, the decision and priority of the policy is evaluated. If an applicable policy's priority is the highest so far, the decision of the policy is noted as the potential decision. When all the policies have been evaluated, the decision of the applicable policy with the highest priority is returned as the decision of the policy set. If no policy was applicable, a decision of 'Not applicable' is returned. However, if there was an error in the evaluation of one of the policies, then the 'Indeterminate' decision is returned.

The priority-overrides rule-combining algorithm is almost the same as the priority-overrides policy-combining algorithm, with the only difference being that rules are being evaluated instead of policies. The priority-overrides rule-combining algorithm is shown in Appendix D.

```
Decision priorityOverridesRuleCombiningAlgorithm(Policy policy[])
{
      Boolean atLeastOneError  = false;
      Boolean potentialDeny    = false;
      Boolean potentialPermit  = false;
      Int perviousPriority = 0;
      Int currentPriority = 0;

      for( i=0 ; i < lengthOf(policy) ; i++ )
      {
            Decision decision = evaluate(policy[i]);
            Priority priority = evaluatePriority(policy[i]);
            previousPriority = currentPriority;
            currentPriority = priority;
            if (previousPriority > currentPriority)
            {
                  currentPriority = previousPriority;
            }
            if (decision == Deny)
            {
                  potentialDeny = true;
                  potentialPermit = false;
                  continue;
            }
            if (decision == Permit)
            {
                  potentialDeny = false;
                  potentialPermit = true;
                  continue;
            }
            if (decision == NotApplicable)
            {
                  continue;
            }
            if (decision == Indeterminate)
            {
                  atLeastOneError = true;
            }
      }
      if (potentialDeny)
      {
            return Deny;
      }
      if (potentialPermit)
      {
            return Permit;
      }
      if (atLeastOneError)
      {
            return Indeterminate;
      }
      return NotApplicable;
}
```

6.4 From skeleton to full XACML policy

Skeleton policies can be translated to full XACML policies by deciding beforehand which XACML elements and attributes will be used to represent the subject, action, resource and environment attributes in a patient consent policy. This is possible since all the elements in the authorisations and their types are known beforehand. Furthermore, a XACML policy is structured. A policy is made up of a target and its rules, which in turn are made up of a target and a condition. The XACML elements and attributes that constitute XACML rule and policy targets and rule conditions are clearly defined in the XACML standard. Owing to this structure, automatic generation of a full XACML policy from the skeleton policy is possible.

The automatic generation of a full XACML policy has not been implemented in this thesis. What is required is the generation of templates that will have all the XACML syntax necessary to represent the elements of consent in the skeleton policy set. Since all the elements are predefined, these templates are fixed and will only require maintenance if the elements of consent change.

The generation of the syntax for the XACML skeleton policies in this thesis was done with the use of the UMU XACML editor [41] that was developed at the University of Murcia (UMU). It provides a structured way of creating a policy set, policies within the policy set and rules within the policies. This is done by use of a GUI that allows you to add XACML elements to the XACML policy and select the relevant attributes of the elements from lists or to type them in. Once the decision is made on which XACML elements and attributes will be used, then creating the policy is simple and only requires entry of the subject, action, resource and environment elements from the skeleton policy and rules. A caption of the editor is shown in Figure 13 below. Appendix B shows the XACML policy generated in this way. Note that since neither the XACML engines nor the UMU-XACML editor support the delegation profile of XACML 3.0, the XACML policy is generated according to the core-profile of XACML.

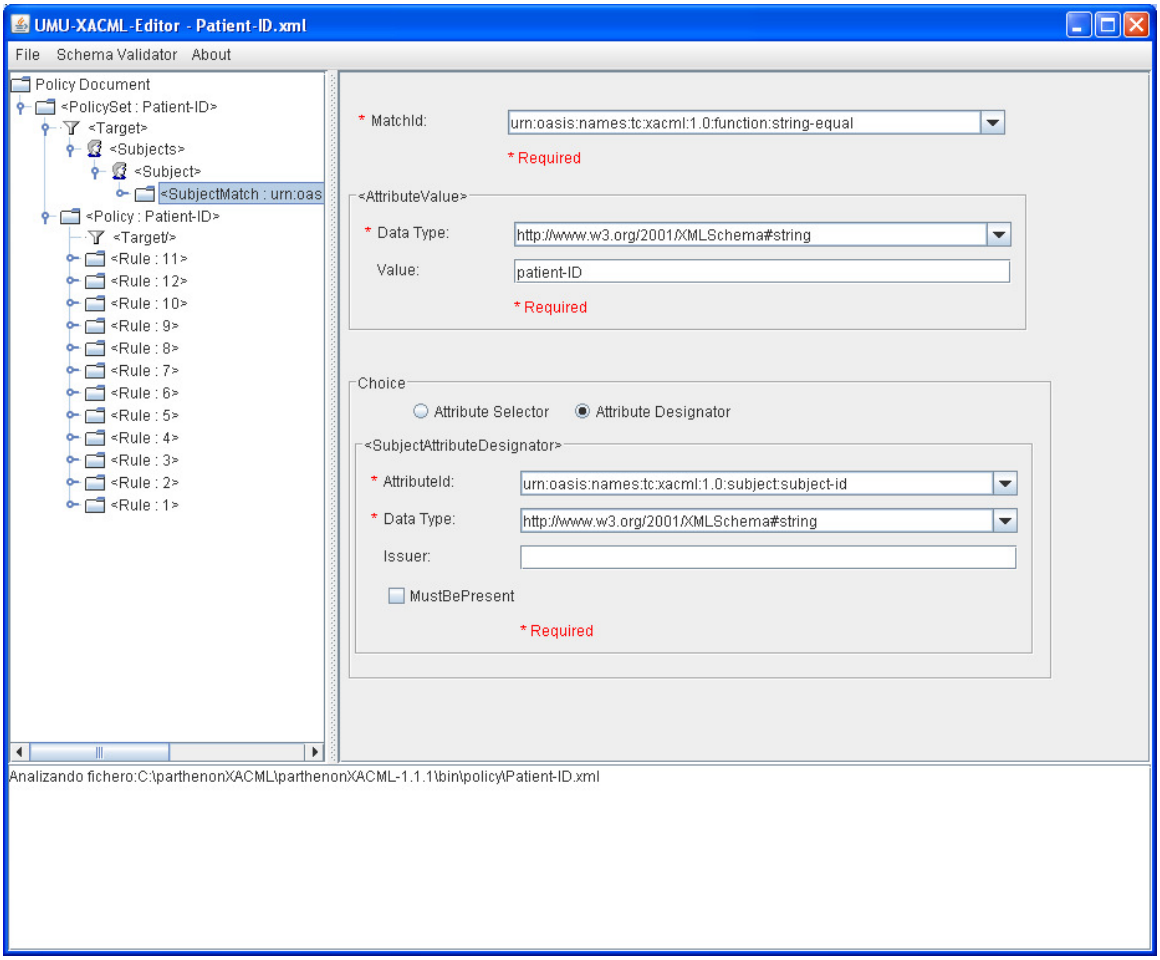

Figure 13: UMU-XACML editor

6.5 Working example

The transformation algorithm processes each of the authorisations specified in the decision table in Section 5.3. The main steps are outlined next.

Skeleton policy set

A policy set is created for the patient consent policy. The policy set must be assigned a name and the policy-combining algorithm selected. The name assigned to this policy set is Patient-ID-policy-set, and the First-applicable policy-combining algorithm is used, assuming a grantor hierarchy exists and the patient's policy has higher authority than the husband's (future) policy.

```
PolicySetId = patient-ID
PolicyCombiningAlgId = First-applicable
• Policy set target
      o Subjects:
            ▪ Subject:
                  * subject-id = patient-ID
```

The target of the skeleton policy set contains the patient's ID.

Skeleton policy

A policy is created for each grantor. The grantor of all the authorisation rules is the patient and so all the authorisations are kept in one policy in the policy set. In the future, if the patient's husband[10] delegates permissions, then a separate policy will be created for these authorisations. The policy named after the grantor, and the first-applicable rule-combining algorithm is chosen, which means that the rules inside the policy must be arranged in descending order of priority.

```
PolicySetId = patient-ID
PolicyCombiningAlgId = First-applicable
• Policy set target
      o Subjects:
            ▪ Subject:
                  * subject-id = patient-ID

PolicyId = patient-ID
RuleCombiningAlgId = First-applicable
• Policy target
```

The skeleton policy target is empty since this is the patient's policy, which must apply to every access request about the patient.

[10] The patient delegated the authority to the husband to delegate or give consent on the patient's behalf.

Skeleton rules

The skeleton rules are generated according to the instructions in Section 6.2.4, 6.2.5 and 6.2.6. They are shown below.

PolicySetId = patient-ID
PolicyCombiningAlgId = First-applicable
- Policy set target
  - Subjects:
    - Subject:
      - subject-id = patient-ID

PolicyId = patient-ID
RuleCombiningAlgId = First-applicable
- Policy target

RuleId = 11 Effect = deny
Rule target
- Resources:
  - Resource:
    - xpath = /patient-ID/Gynecological-information/*
- Subjects:
  - Subject:
    - subject-id = mother-ID
- Actions:
  - Action:
    - action-id = read

Rule condition
- Purpose = all
- Context = all

RuleId = 12 Effect = deny
Rule target
- Resources:
  - Resource:
    - xpath = /patient-ID/Blood-pressure[age<=3]
- Subjects:
  - Subject:
    - subject-id = mother-ID
- Actions:
  - Action:
    - action-id = read

Rule condition
- Purpose = all
- Context = all

RuleId = 10 Effect = permit

Rule target
- Resources:
  - Resource:
    - xpath = /patient-ID/*
- Subjects:
  - Subject:
    - subject-id = mother-ID
- Actions:
  - Action:
    - action-id = read

Rule condition
- Purpose = all
- Context = all

RuleId = 9 Effect = permit

Rule target
- Resources:
  - Resource:
    - xpath = /patient-ID/*
- Subjects:
  - Subject:
    - subject-id = husband-ID
- Actions:
  - Action:
    - action-id = read

Rule condition
- Purpose = all
- Context = all

RuleId = 8 Effect = permit

Rule target
- Resources:
  - Resource:
    - xpath = /patient-ID/*
- Subjects:
  - Subject:
    - subject-id = dr.-John-Mathews-ID
- Actions:
  - Action:
    - action-id = write

Rule condition
- Purpose = treatment
- Context = emergency

RuleId = 7 Effect = permit

Rule target
- Resources:
  - Resource:
    - xpath = /patient-ID/*
- Subjects:
  - Subject:
    - subject-id = dr.-John-Mathews-ID
- Actions:
  - Action:
    - action-id = read

Rule condition
- Purpose = treatment
- Context = emergency

RuleId = 6 Effect = permit

Rule target
- Resources:
  - Resource:
    - xpath = /patient-ID/*
- Subjects:
  - Subject:
    - role = doctor
- Actions:
  - Action:
    - action-id = write

Rule condition
- Purpose = treatment
- Context = all
- Validity period = 1 year

RuleId = 5 Effect = permit

Rule target
- Resources:
  - Resource:
    - xpath = /patient-ID/*
- Subjects:
  - Subject:
    - role = doctor
- Actions:
  - Action:
    - action-id = read

Rule condition
- Purpose = treatment
- Context = emergency
- Validity period = 1 year

RuleId = 4 Effect = permit

Rule target
- Resources:
  - Resource:
    - xpath = /patient-ID/*
- Subjects:
  - Subject:
    - role = personal-doctor
- Actions:
  - Action:
    - action-id = write

Rule condition
- Purpose = all
- Context = all

RuleId = 3 Effect = permit

Rule target
- Resources:
  - Resource:
    - xpath = /patient-ID/*
- Subjects:
  - Subject:
    - role = personal-doctor
- Actions:
  - Action:
    - action-id = read

Rule condition
- Purpose = all
- Context = all

RuleId = 2 Effect = permit

Rule target
- Resources:
  - Resource:
    - xpath = /patient-ID/*
- Actions:
  - Action:
    - action-id = write

RuleId = 1 Effect = permit

Rule target
- Resources:
  - Resource:
    - xpath = /patient-ID/*
- Actions:
  - Action:
    - action-id = read

The skeleton rules shown above are arranged in descending order of priority.

Full XACML policy

A full XACML policy was generated by pre-selecting the XACML elements and attributes, which will be used to represent the various elements of the authorisations in the XACML policy. Together with the skeleton policies, these elements and attributes are used to create a full XACML policy.

Figure 13, which shows a screenshot of the UMU-XACML editor, illustrates an example of how the full XACML policy was generated. This screenshot contains three main windows, one on the left, one on the right and one below. The window on the left shows the policy set and its component policy and rules. The 'SubjectMatch' of the policy set's target has been selected and highlighted and its contents displayed in the window on the right. The 'Match' elements are used to compare the values specified in the policy, with those presented in the request context. In this case, the match element is comparing subject attributes, and thus it is called a 'SubjectMatch' element. As shown in the window on the right, the 'SubjectMatch' is comparing a string with the value 'patient-ID' with a value contained in the request context's 'patient' attribute, which is also a string. This comparison is done using the 'string-equal' function. In short, this match is checking whether the identity of the patient is equal to the one provided by the access requester in the request context. The 'AttributeId' and 'Value' values are obtained from the skeleton policy, while the data types and the match function are selected from those available in the drop down lists.

6.6 Discussion

This chapter has presented a new method for the translation of a patient consent policy represented in a decision table, to a XACML policy. The translation algorithm takes the authorisations from the conflict resolution algorithm as input. It is done in two steps. The first step involves the creation of skeleton policies, which is a new idea to represent only the core components of the XACML policy. The second step involves adding the necessary syntax to create a full XACML policy. In addition to the proposed translation, a new XACML policy/rule-combining algorithm has been presented here, that evaluates rules and policies according to their priorities.

A question arises on the efficiency of the new priority-overrides combining algorithm, compared to XACML's first-applicable algorithm. The first-applicable algorithm is more efficient since it does not process all the rules or policies. Rather, it returns the decision of the first applicable rule or policy. The priority-overrides algorithm on the other hand must process all the applicable rules or policies, in order to return the decision of the rule or policy that has the highest priority. The main advantage of the priority-overrides algorithm is evident during policy combination; it allows the priority values of the decisions of policies to be taken into consideration during their combination. The first-applicable policy-combining algorithm assumes that the policies are arranged in order of priority in the policy set. This however, is not always the case, and the decision with the highest priority across the policies may be required, which can only be done by the priority-overrides policy-combining algorithm.

7 Testing the translation

Chapters 3 – 6 have presented the method to translate a patient consent policy specified in structured natural language form by the use of a GUI, to a XACML policy. The method includes four main steps; the specification of the policy, conflict detection amongst the authorisations in the policy, conflict resolution to remove any detected conflicts and finally the translation to XACML. This chapter now presents the methodology used to test the policies generated using this process. It starts with a discussion on the testing approach employed, the generation of test cases and the tools that were chosen for the testing. The testing process is then presented and the chapter concludes with a discussion.

7.1 Testing approach

The aim of the testing is to check whether the translation of the patient consent policy to XACML is successful. The translation is deemed successful if the policy as specified by the patient contains the same authorisations as the XACML policy. This will be tested by creating access requests that will be used to query the XACML policy that was generated by the translation process, and checking whether the generated decision conforms to the patient consent policy in the decision table that was generated after the conflict resolution. As discussed in Chapter 5, the resolution of conflict is done interactively, and the grantor of consent has the opportunity to specify the order of preference of the authorisations, or to change the preference of authorisations that are assigned automatically. In this regard, the final patient consent policy is the one generated after the stage of conflict resolution; it contains the authorisations and the preferences as specified by the grantor.

In addition, a tool called Margrave was used to test the XACML policies by applying it to generate counterexamples of the rules in the policies. The testing approach is summarised in Figure 14 below.

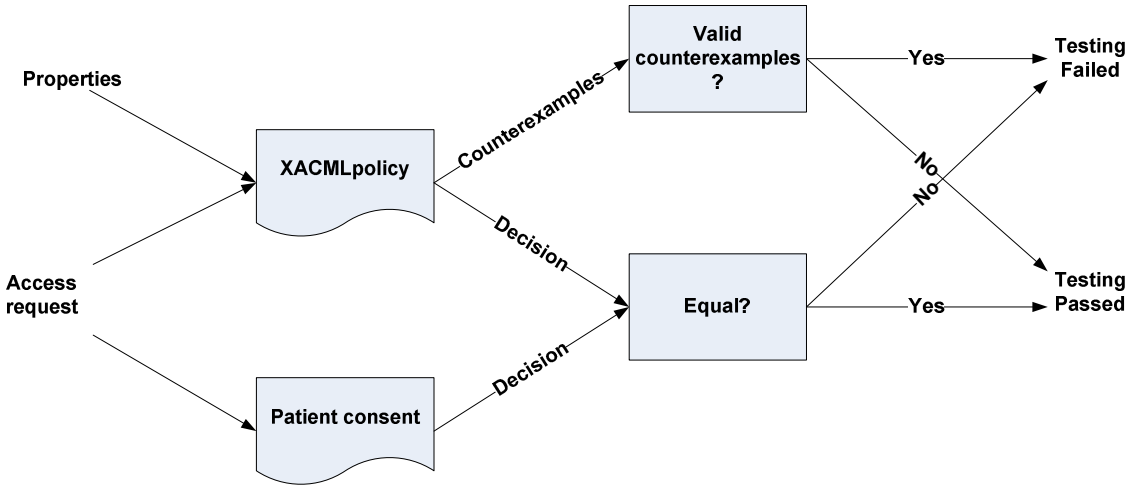

Figure 14: Testing approach

7.2 Generation of test cases

The generation of test cases was done according to the principles of structured white based testing. Structured white based testing is the testing technique for testing software based on the logic relation between the input presented to the software and the output produced by the software. This technique is used when the software to be tested is available and is done for finding functional mistakes. The goal of the testing is to get as much coverage of the software as possible in the most efficient manner [42]. There are four levels of coverage and these are statement, branch/decision, condition and full coverage. Statement coverage tests each program statement, while branch/decision coverage tests the branching or decision statements. Condition coverage exists when each predicate of the condition is evaluated to both truth-values, or when the predicates of the conditions are evaluated to all possible combination of truth-values. Full coverage is a combination of all the coverage criteria.

In this case, the software under test is the XACML policy and the patient consent policy specified in the decision table. Both policies contain authorisations, which will either apply or not apply to an access request. If any of the subject, action, resource or environment elements presented in the access request are not compatible with a rule or an authorisation, that rule/authorisation is not applicable to the access request and will not be factored in the decision of the policy. For this reason, it suffices to use statement coverage criterion for the tests, where a statement is a XACML rule or an authorisation. Consequently, each rule or authorisation is a test case; the same elements in the rule or authorisation are included in an access request, which is used to query both the patient consent policy and the XACML policy.

7.3 Testing tools

A number of tools were employed to test the XACML policy and the authorisations in the decision table. This section discusses these tools.

7.3.1 Sun's XACML Implementation

This is an open source implementation of the XACML standard, created by Sun Microsystems and written in the Java programming language. It provides support for the following features:

- It provides full support for parsing policy and request/response documents in XACML,
- It determines the applicability of policies to requests and,
- It can evaluate requests against policies.

This tool was used to query the XACML policies with access requests. Figure 15 shows the phase in the testing process this tool was applied, which is represented with broken lines.

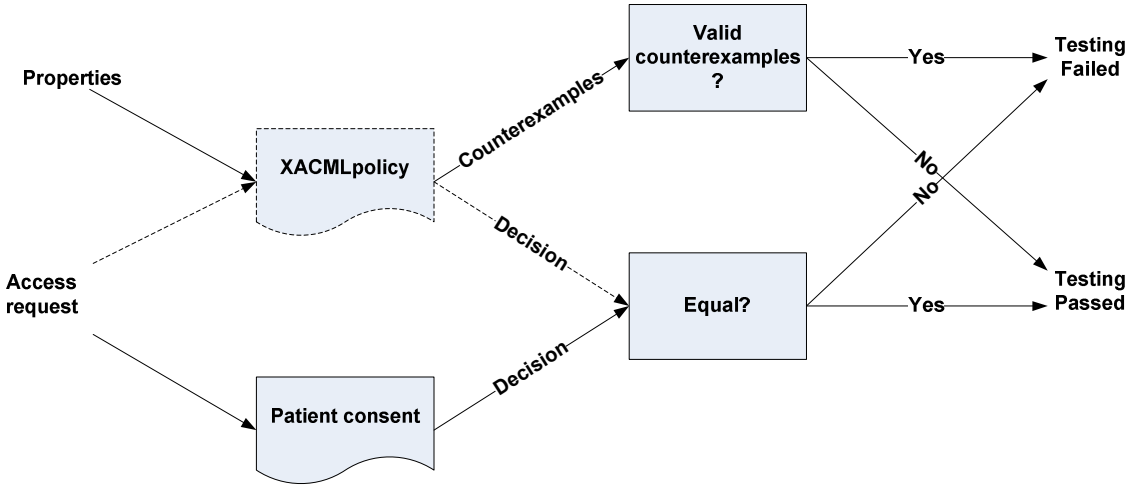

Figure 15: SunXACML's application in testing

7.3.2 Margrave

Margrave is a Scheme API for Policy Verification and Change Analysis of access control policies, written in a subset of XACML. Margrave's functionality is summarised as follows:

- Margrave can verify properties about a policy,
- it can give an explanation about a policy,
- it can perform a change impact analysis between two policies and,

- it can verify and explain the result of the change impact analysis done between two policies.

In the testing that was carried out, Margrave was used to verify properties about a XACML policy. Each rule in the XACML policy was coded into a property and Margrave was then used to produce counterexamples to the rule. These counterexamples highlight the cases when a policy may respond with decisions that are opposite of what its rules propose; or those that give the expected decision, but to an access request that contains elements that together do not constitute any rule of the policy. Figure 16 below shows the phase of testing in which this tool was used, which is represented with broken lines.

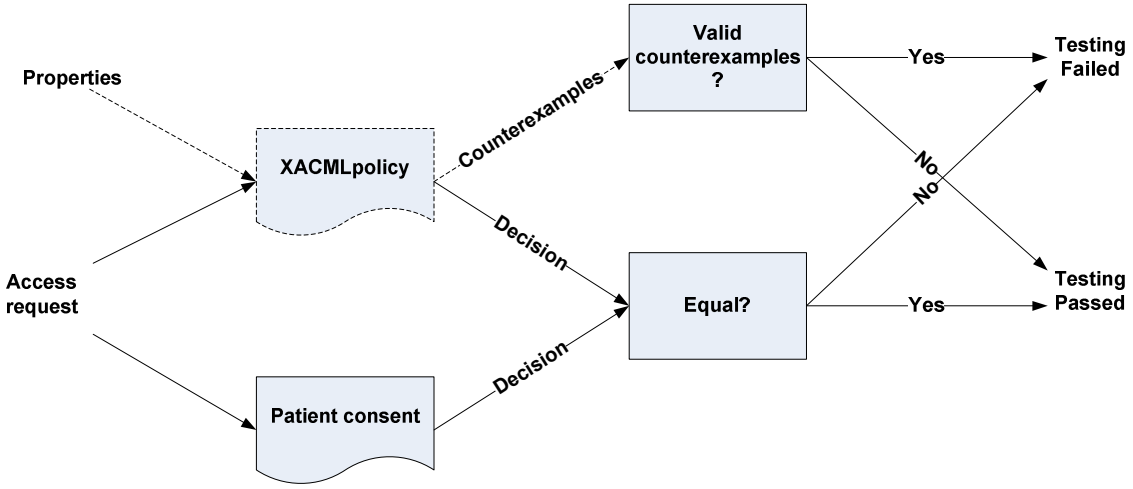

Figure 16: Margrave's application in testing

7.3.2.1 The Supported Subset of XACML

Margrave supports a subset of the XACML 2.0 standard. It supports the <Rule>, <Policy> and <PolicySet> elements and most of the features of the <Target> and <Condition> elements found within them. Margrave supports the following rule/policy combining algorithms:

- First Applicable;
- Deny-Override; and
- Permit-Override.

The latest version of margrave offers limited support for the <Condition> element, which may be found in the <Rule> element, and some of the elements and attributes found within this element. Margrave can only compare attributes by the use of string-equality testing, and therefore cannot deal with matching XPath expressions. Furthermore, it supports only three logical functions: 'Not', 'And' and 'Or'. Margrave also cannot handle requests with multiple subjects or resources, does not handle obligations and does not deal with hierarchical data. Finally, the delegation profile is not supported by margrave.

Due to these constraints, the XACML policies were altered to conform to the subset of XACML supported by Margrave.

7.3.3 Prolog

Prolog is a declarative logic programming language, where the program's logic is expressed using relations. These relations can be queried, and both relations and queries are expressed using terms. Given a query, the prolog engine tries to find a resolution refutation of the negated query [45]. If the negated query can be refuted, it follows that the original query is a logical consequence of the program.

In the testing carried out, prolog was used to test the specified patient consent policy, as shown in Figure 17.

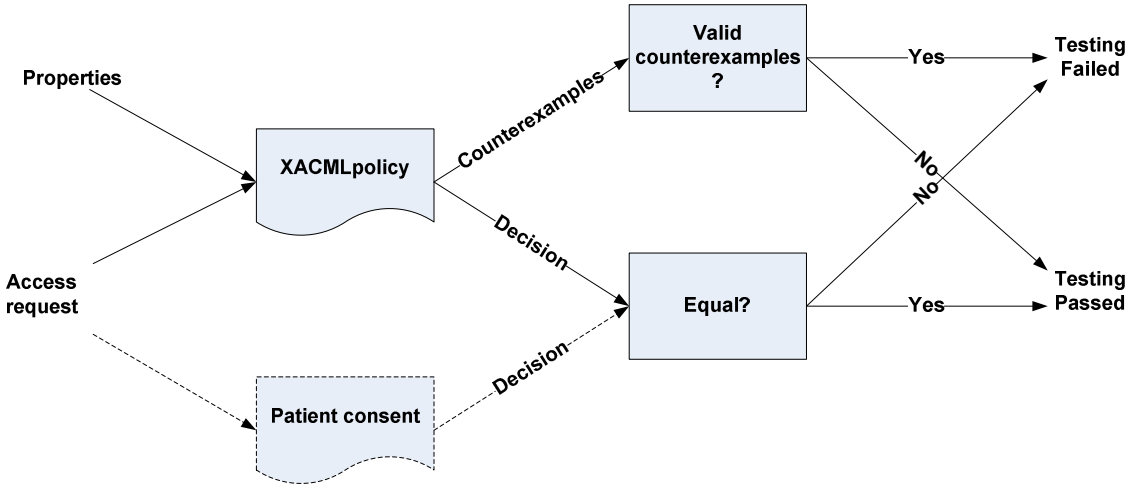

Figure 17: Prolog's application in testing

7.4 Testing the XACML policy

The XACML policies were tested using the SunXACML tool. The XACML policies were loaded into the SunXACML access control engine and queried with access requests. Using the first-applicable rule/policy-combining algorithms, the effect of the rule with the highest priority was returned as the decision to an access request. Figure 18 shows the screenshot of the testing of a XACML policy, while Appendix C contains the access request used to generate the response in the screenshot. The screenshot shows some information messages generated by the SunXACML engine and the response of the policy to the access request. The information messages were generated since the identifier 'role', which is not one of the identifiers in the XACML standard, was used. The standard has not defined an identifier to use for roles, and a user-defined identifier was used. The response is shown next between the '<Response>' elements. The decision of the policy is contained in the '<Decision>' element; the decision of the policy to the access request in Appendix C is 'permit'.

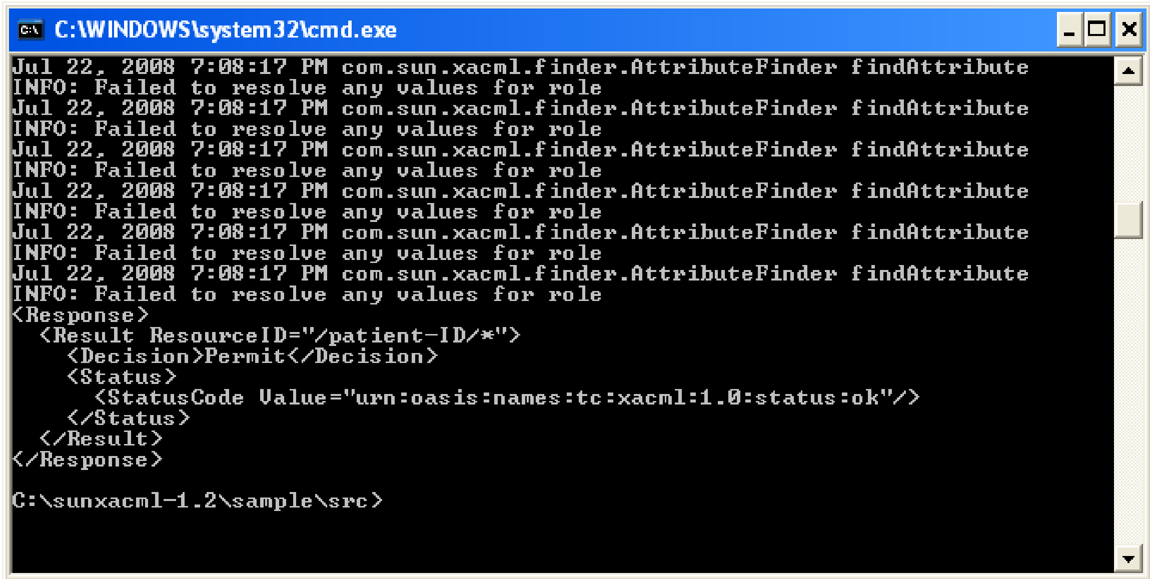

Figure 18: Sample XACML response.

7.4.1 Generating XACML counterexamples

Using the verification functionality of margrave, counterexamples to the rules of the XACML policies were generated. Each XACML policy was loaded and each rule of the policy coded as a property. The property was then verified against the XACML policy and any counterexamples produced were then examined for their validity. An example of the property formulated to generate counterexamples for rule 9j, which says that a husband is permitted to read the patient's medical data for treatment purpose in an emergency context, is shown below.

```
1. (require "../../Margrave/analysis/margrave.scm")
2. (define Pol_1 (load-xacml-policy "../../Desktop/Working-Example-
   Finalest.xml"))
3. (define (find-Pr_1-counter-example Pol_1)
4. (avc-and (restrict-policy-to-dec 'P Pol_1)
5. (avc-and  (make-avc  'Subject  'urn:oasis:names:tc:xacml:1.0:subject-
   category:access-subject 'husband-ID)
6. (make-avc  'Resource  'urn:oasis:names:tc:xacml:1.0:resource:resource-
   id '/patient-ID/*)
7. (make-avc    'Action    'urn:oasis:names:tc:xacml:1.0:action:action-id
   'read)
8. (make-avc 'Environment 'purpose 'quality-measures)
9. (make-avc 'Environment 'context 'emergency))))
```

In the property shown above, line 1 enables the use of Margrave in a Scheme file. This is necessary because Margrave is used within the Scheme functional programming language. Line 2 loads the XACML policy, while line 3 begins the property and specifies that the aim is to find counterexamples. Line 5 restricts the decision of the property to permit using 'P'. During testing, each property was restricted to both permit and deny.

Line 6, 7, 8 and 9 define the subject, resource, action and environment attributes specified in rule 9 of the XACML policy shown in Appendix B. This property will return the empty set if no counter-example can be found, and thus this property holds. Otherwise, the set of valid counterexamples is returned.

On running the property, the following output was generated by margrave.

```
 1:/Action, urn:oasis:names:tc:xacml:1.0:action:action-id, read/
 2:/Action, urn:oasis:names:tc:xacml:1.0:action:action-id, OTHER/
 3:/Resource, urn:oasis:names:tc:xacml:1.0:resource:resource-id,
/patient-ID/*/
 4:/Resource, urn:oasis:names:tc:xacml:1.0:resource:resource-id, OTHER/
 5:/Action, urn:oasis:names:tc:xacml:1.0:action:action-id, write/
 6:/Environment, context, normal/
 7:/Environment, context, OTHER/
 8:/Environment, purpose, treatment/
 9:/Environment, purpose, OTHER/
 10:/Subject, role, personal-doctor/
 11:/Subject, role, OTHER/
 12:/Environment, context, emergency/
 13:/Environment, purpose, payment/
 14:/Environment, purpose, operations/
 15:/Environment, purpose, public-health/
 16:/Environment, purpose, quality-measures/
 17:/Environment, validity period, 1 year/
 18:/Environment, validity period, OTHER/
 19:/Subject, role, doctor/
 20:/Subject, urn:oasis:names:tc:xacml:1.0:subject-category:access-
subject, dr.-John-Mathews-ID/
 21:/Subject, urn:oasis:names:tc:xacml:1.0:subject-category:access-
subject, OTHER/
 22:/Subject, urn:oasis:names:tc:xacml:1.0:subject-category:access-
subject, husband-ID/
 23:/Subject, urn:oasis:names:tc:xacml:1.0:subject-category:access-
subject, mother-ID/
 24:/Resource, urn:oasis:names:tc:xacml:1.0:resource:resource-id,
/patient-ID/Gynecological-information/*/
 25:/Resource, urn:oasis:names:tc:xacml:1.0:resource:resource-id,
/patient-ID/Blood-pressure/*
 26:/Subject, patient, patient-ID/
 27:/Subject, patient, OTHER/

          1         2
12345678901234567890123456
{
1-1--------1---1-----10----
1-1--------1---1-----1100--
}
```

The output consists of two parts; the first part shown above by a listing of 27 items is the key and represents all the variables created for the policy. The representation of the counterexample(s) is given below it. It shows the numbers 1-27 which represent the key shown above, and the values ‘1’, ‘0’ or ‘-’ for each number. If a number has the value ‘1’, then the variable represented by the number in the key is present in the counterexample,

while ‘0’ means that the variable must not be present, while ‘-’ implies that the value of the variable represented by the number is irrelevant in the counterexample.

Two counterexamples are generated in this example, which is depicted by the existence of two rows in the representation of the counterexample. The first row shows that this counterexample has the variables read (1), /patient-ID/* (3), emergency (12), quality-measures (16) and husband-ID (22), but must not have the mother-ID (23) while the rest of the variables do not matter. In summary, this counterexample says that the husband is allowed to read the patient’s medical information for quality-measures purpose and in an emergency context, if only he is not the patient’s mother.

The second counterexample contains the variables read (1), /patient-ID/* (3), emergency (12), quality-measures (16), husband-ID (22), mother-ID (23), and must not have /patient-ID/Gynecological-information/* (24) and /patient-ID/Blood-pressure/* (25). This counterexample says that the husband is allowed to read the patient’s medical information for quality-measures purpose and in an emergency context, if he is the mother, on condition that the data must not be blood pressure or gynecological information.

In the XACML policy, the mother and husband are different subjects. Furthermore, these counterexamples do not go against the patient consent policy in our example, Appendix E contains all the counterexamples that were generated by the margrave tool for the XACML policy similar[11] to the one in Appendix B.

7.5 Testing the patient consent

The testing of the patient consent policy was done by loading the authorisations in the decision table into the prolog engine, by using a ‘cando’ predicate. Appendix F shows the authorisations as loaded into prolog’s rule engine. Queries were made using the ‘cando’ predicate at the prompt, with the effect and the rule number left unspecified and replaced with variables. The XPath intersection algorithm was not implemented in prolog, and therefore the intersection check of the data elements of the authorisation rules was done manually. This was achieved by also leaving the data elements of the queries unspecified and replacing them with variables. Consequently, the responses to the queries also returned the corresponding data elements, and the intersection algorithm was manually used to check data overlap. Furthermore, the XPath expressions could not be used due to a syntax clash with prolog’s commenting syntax (/*). Figure 19 shows a screenshot of how the queries were done.

Prolog returns all viable responses to a query. It can then be seen from the responses given, which response has the highest priority, which is considered as the response of the patient consent policy to the access request.

[11] The syntax of the policy shown in Appendix B was altered to the subset that Margrave recognises.

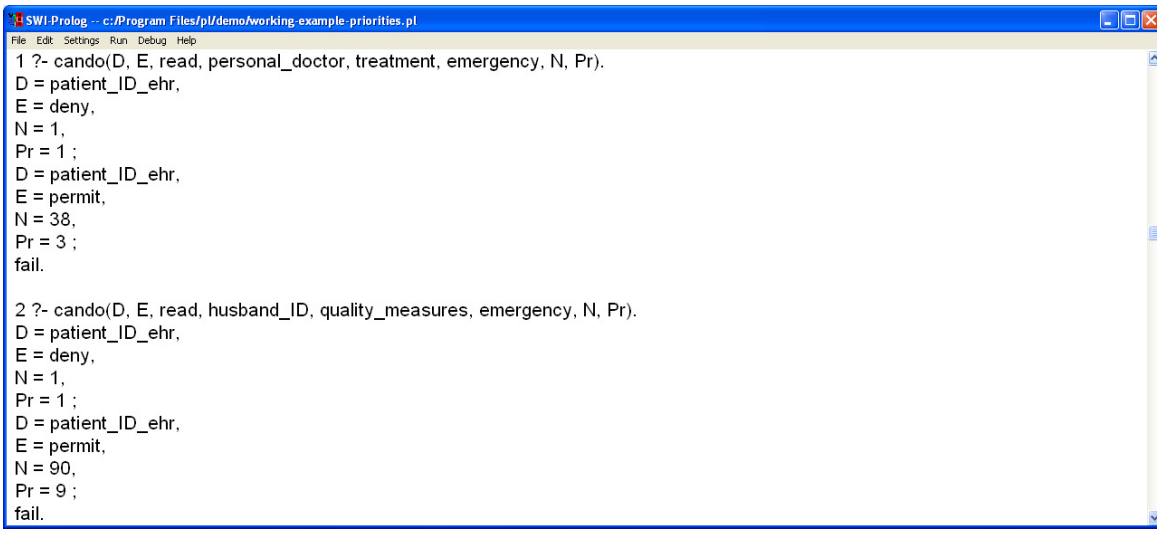

Figure 19: Sample prolog queries and response

7.6 Testing results

The testing was successful, such that the decisions that were generated by the XACML policies were equal to the decisions obtained from the authorisations in the patient consent policy. The XACML policies always returned the expected responses to access requests, which implies that the translation from the specified patient consent to the XACML policy is correct. Furthermore, no counterexamples that Margrave produced demonstrated vulnerabilities or errors in the XACML policies. This testing was done for three different policies.

7.7 Discussion

A number of challenges presented themselves during this testing phase.

- **Tool limitations**
  Margrave recognizes a limited subset of XACML, which means that the generated policy can only contain these recognized XACML elements and attributes

- **Tool incompatibility**
  The Sun XACML and Margrave tools do not implement the same set of the XACML standard, which implies that a compromise has to be done on which set of XACML will be used to generate the policies.

- **Tool errors**
  The Sun XACML tool has incorrectly implemented the <Condition> element of a rule. In the XACML 2.0 standard, the condition element contains one <Expression> element. However, the tool implements the <Condition> element to expect more than one <Expression> element, and therefore generates errors when loaded with an otherwise correct XACML policy.

- **Length of XACML policies**
  This XACML policies generated for testing were long due to the use of the limited syntax of Margrave, the lengthy syntax of XACML and the rich policies that were specified. Since the various XACML tools recognise different syntax, these policies required manual configuration which was a long and tedious process. Consequently, the testing was not extensively carried out.

- **Testing XPath intersection**
  XACML 2.0 contains an XPath-node-match function that takes XPath expressions, and evaluates to true if an intersection exists between the nodes and/or children of the nodes represented by the two XPath expressions. This function however has not been implemented in any of the XACML tools, and therefore this intersection test had to be undertaken without the aid of tools.

Some of the testing presented in this chapter may be included in the system for the grantors to use to check their policies for unprecedented effects. The functionality of Margrave that generates counterexamples may be useful to show the grantors the possible overlaps in their authorisations. However, the output must be more user friendly than what is generated by Margrave.

8 Conclusion

The focus in healthcare has shifted from the healthcare providers' paternalistic approach, to a patient-focused approach. It is now important for patients to be able to give independent and specific consent for the use of their health information. The management of medical information using Electronic Health Records (EHR) necessitates a shift in the method used to represent patient consent. For patient consent to be enforceable in EHR systems, it must be specified in electronic form, and in a format that can be processed by access control mechanisms. Consent must reflect each patient's preferences, and contain all the relevant elements that constitute consent.

This thesis presented the design of a system for the generation of customized electronic patient consent policies using the XACML policy language. The system facilitates the specification of patient consent policies using structured natural language presented in a Graphical User Interface (GUI). Using this GUI, grantors can specify the general and specific authorisations that make up patient consent. To ensure the consistency of the specified authorisations, the system performs conflict detection amongst the default and specific authorisations. With the use of the locality and preference conflict resolution policies, existing conflict is resolved and priorities are accordingly assigned to the authorisations. Directed Acyclic Graphs that depict the precedence allocated during conflict resolution are created, to ensure the assigning of priorities does not introduce cycles. The authorisations are then translated into XACML skeleton polices and rules, which show the mapping of the elements of the authorisations to the XACML policy language model. Finally, the selected XACML elements and attributes are added to the XACML skeleton policies and rules, to create full XACML policies.

The main contributions of this thesis are as follows.

- This thesis presents the idea of the generation of XACML policies automatically to enhance the creation of syntactically and semantically correct XACML policies. The literature presents a number of methods and tools for the testing of already generated XACML policies. No literature was found that treats the generation of XACML policies.
- The idea of attribute tables has been created to facilitate the detection of potential modality conflicts amongst authorisations. In the literature, modality conflicts are only detected by performing syntactical analysis of the authorisations.
- The idea of using Directed Acyclic Graphs and the topological sort to ensure that the priority assignment is done without the introduction of cycles has been presented.

- The translation algorithm has been created in this thesis to translate patient consent policies to XACML.
- A new policy and rule-combining algorithm that performs combination based on priorities has been created and presented in this thesis.

This system is currently under implementation at Philips Research Laboratories. The specification process is almost in place, and the detection of actual conflict has already been implemented.

8.1 Related work

In [53], a policy workbench called SPARCLE that is used for the generation of machine-readable security policies is presented. It allows users to enter policies in constrained natural language and translates this to XML. In this workbench, emphasis has been placed on the parsing of the constrained natural language. Unlike the work presented in this thesis, no conflict detection and resolution amongst the authorisation of the policies is done. The MotOrBAC prototype [54] is another tool that can be used in the creation of a security policy, also in XML. In MotOrBAC, conflict detection and resolution has been implemented to deal with actual conflicts in the authorisations in the policies. However, unlike the work presented here, no attempt has been made to predict and resolve potential conflicts. Both SPARCLE and MotOrBAC generate XML policies that have no defined way of evaluation, combination or checking for applicability of a policy to an access request. The work described here produces XACML policies, whose evaluation, combination, applicability and other factors are described in the XACML standard.

8.2 Future Work

There are a number of unresolved issues that require further work:

- a formal proof of the translation process by making use of the semantics of XACML is needed; and
- there is the need for a full implementation of a XACML 2.0 access control engine; existing implementations are mainly for version 1.1 of XACML, while those tools that implement version 2.0 are not yet complete.

Bibliography

[1] Chun Ruan and Vijay Varadharajan. Supporting E-consent on Health Data by Logic. *Foundations of Intelligent Systems,* 2871/2003:392-396, 2003.

[2] T. Rooney and J. Aitken. Consent and Electronic Health Records. Discussion Paper. http://www.health.gov.au/internet/hconnect/publishing.nsf/Content/E250BD83358D3A56CA257128007B7EC9/$File/cons_dp.pdf, July 2002.

[3] Pritts, J. & Connor, K. The Implementation of E-consent Mechanisms in Three Countries: Canada, England and the Netherlands. http://ihcrp.georgetown.edu/pdfs/prittse-consent.pdf, February 2007.

[4] HIPAA Patient consent form. http://www.paincarerehab.com/pdf/patient-consent.pdf.

[5] Khin T. Win and John A. Fulcher. Consent Mechanisms for Electronic Health Record Systems: A Simple yet unresolved issue. *Journal of Medical Systems*, 31(2), April 2007.

[6] C.M. O'Keefe, P.A. Greenfield and A.D. Goodchild. A Decentralised Approach to Electronic Consent and Health Information Access Control. *Journal of Research and Practice in Information Technology*, 34, 137-154, 2004.

[7] N. Damianou, A. Bandara, M. Sloman, and E. Lupu. A survey of policy specification approaches. April 2002.

[8] Security and privacy technical committee. HITSP Security and Privacy Technical Note. http://library.ahima.org/xpedio/groups/public/documents/external/bok1_036592.pdf. October, 2007.

[9] K. Verlaenen, B. DeWin and W. Joosen. Towards simplified specification of policies in different domains. *Proceedings of Integrated Management 2007,* May 2007.

[10] D. C. Verma. Simplifying network administration using policy-based management. Network, IEEE, Volume: 16, Issue: 2, April 2002.

[11] F. Cuppens, N. Cuppens-Boulahia, and M. B. Ghorbel. High-level conflict management strategies in advanced access control models. In *Workshop on Information and Computer Security (ICS)*, Timisoara, Romania, September 2006.

[12] E. C. Lupu and M. Sloman. Conflicts in policy-based distributed systems management. *IEEE Transactions on Software Engineering*, 25(6):852–869, November/December 1999.

[13] E. Lupu and M. Sloman. “Conflict Analysis for Management Policies.” Proceedings of IFIP/IEEE *International Symposium on Integrated Network Management (IM’1997)*, May 1997.

[14] Jajodia, S., Samarati, P. and Subrahmanian, V. S. A logical language for expressing authorizations. In *Proceedings of the 1997 IEEE Computer Society Symposium on Reasearch in Security and Privacy* (Oakland, CA, May), IEEE Computer Society Press, Los Alamitos, CA, 31–42, 1997.

[15] H. Moungla and F. Krief. A probabilistic heuristic for conflict detection in policy based management of Diffserv networks. *Proceedings of IEEE/IFIP MATA 2005 : 2nd International Workshop on Mobility Aware Technologies and Applications*, Montréal, Canada, 2005.

[16] Taner Dursun. A generic policy-conflict handling model. Computer and Information Sciences – ISCIS, 2005.

[17] J.D. Moffett and M.S. Sloman. Policy conflict analysis in distributed systems management. *Ablex Publishing Journal of Organizational Computing*, 4(1), 1-22, 1994.

[18] Amir H. Chinaei, Hamid R. Chinaei, and Frank Wm Tompa. A unified conflict resolution algorithm. In *VLDB Workshop on Secure Data Management*, pp. 1-17, September 2007.

[19] OASIS eXtensible Access Control Markup Language (XACML). http://www.oasis-open.org/committees/xacml/, 2005.

[20] OASIS XACML 3.0 Administration and Delegation Profile. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.

[21] Gerome Miklau and Dan Suciu. Containment and equivalence for a fragment of XPath. *ACM* 51(1), 2-45, January 2003.

[22] Beda Christoph Hammerschmidt, Martin Kempa and Volker Linnemann. On the Intersection of XPath Expressions. In: *IDEAS, IEEE Computer Society*, 49–57, 2005.

[23] Pierre Genevès. Improving Efficiency of XPath-Based XML Querying. *IFIP Student Forum,* 143-154, 2004.

[24] Reguläre Beiträge. Why is medical software so hard? *Computer Science - Research and Development*, 22(3), March, 2008.

[25] Ravi S. Sandhu, Edward J. Coyne, H. L. Feinstein and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2): 38-47, February 1996.

[26] J. Bergmann, O. J. Bott, D. P. Pretschner, R. Haux. An e-consent-based shared EHR system architecture for integrated healthcare networks. *International journal of medical informatics*, 76(2-3):130–136, 2007.

[27] P. A. B. Galpottage and A. C. Norris. Patient consent principles and guidelines for e-consent: A New Zealand perspective. *Health Informatics Journal,* 11: 5-18, 2005.

[28] R. Clarke. eConsent: A Critical Element of Trust in eBusiness. *In 15th Bled Electronic Commerce Conference: eReality: Constructing the eEconomy*, June 2002.

[29] K. T. Win, H. Song, P. Croll and J. Cooper. Implementing patient's consent in electronic health record systems. In *Proceedings of CollECTeR,* December 2002.

[30] C. Ruan and V. Varadharajan. An authorisation model for e-consent requirement in a health care application. In *Applied Cryptography and Network Security*, 2846, 191-205, 2003.

[31] XML Schema tools. http://www.w3.org/XML/Schema#Tools.

[32] S. Gançarski, C. Le Pape and A. L. Gançarski. Freshness control of XML documents for query load balancing. In *Proceedings of the 18th International Conference on Database and Expert Systems Applications*, 35-39, 2007.

[33] Stefan Böttcher: Testing Intersection of XPath Expressions under DTDs. *International Database Engineering & Applications Symposium,* 2004.

[34] N. Dunlop, J. Indulska and K. Raymond. Methods for Conflict Resolution in Policy-Based Management Systems. In *Proceedings of the 7th IEEE International Enterprise Distributed Object Computing Conference*, 98-109, September 2003.

[35] J. D. Moffett and M. S. Sloman. Policy Hierarchies for Distributed Systems Management. *IEEE JSAC Special Issue on Network Management*, 11(9), 1404-1414, December 1993.

[36] Thomas Schwentick. Xpath query containment. *SIGMOD Record*, 33(1), 101–109, 2004.

[37] R. Sedgewick and C. J. Van Wijk. Algorithms in C++. Addison Wesley, 2002.

[38] R. Sedgewick. Algorithms in C. Addison Wesley, 1990.

[39] Mahdi Mankai and Luigi Logrippo. Access control policies: Modeling and validation. In *Proceedings of the 5th NOTERE Conference*, 85–91, August 2005.

[40] J. H. Kingston. Algorithms and data structures: design, correctness, analysis. Addison-Wesley, 1998.

[41] UMU XACML editor. http://xacml.dif.um.es/.

[42] Judi Romijn. Lecture 2: Structure-based white box testing. http://www.win.tue.nl/~jromijn/2IW30/.

[43] Margrave. http://www.cs.brown.edu/research/plt/software/margrave/.

[44] Parthenon XACML. http://www.parthcomp.com/xacml_toolkit.html.

[45] Wikipedia. http://en.wikipedia.org/wiki/Main_Page.

[46] J. Pritts. Patient consent in the electronic exchange of health information. http://www.health.state.mn.us/e-health/summit/pritts07.pdf.

[47] Cambridge Health Informatics Limited. Gaining patient consent to disclosure. http://www.iccs.informatics.ed.ac.uk/~mjh/chameleon/NHSDocs/gpcdrep.pdf, 2001.

[48] D.Chadwick and A. Sasse. The virtuous circle of expressing authorization policies. *In: Semantic Web Policy Workshop*, Georgia, 2006.

[49] V. C. Hu, E. Martin, J. Hwang, and T. Xie. Conformancechecking of access control policies specified in XACML. *In Proc. IWSSE*, 275–280, July 2007.

[50] G. Hughes and T. Bultan. Automated verification of access control policies. Technical Report 2004-22, University of California, Santa Barbara, 2004.

[51] E. Martin and T. Xie. A fault model and mutation testing of access control policies. *In Proc. 11th International Conference on World Wide Web*, 2007.

[52] Ontario ministry of health full withdrawal of consent form. http://www.health.gov.on.ca/english/public/forms/form_menus/eda_fm.html.

[53] Brodie, C. A., Karat, C., and Karat, J. An empirical study of natural language parsing of privacy policy rules using the SPARCLE policy workbench. *In Proceedings of the Second Symposium on Usable Privacy and Security*, 2006.

Index

Appendices

A. Patient Consent Form

I understand that, under the Health Insurance Portability & Accountability Act of 1996 (HIPAA), I have certain rights to privacy regarding my protected health information. I understand that this information can and will be used, but is not mandatory for me to sign in order to:

- Conduct, plan and direct my treatment and follow-up among the multiple healthcare providers who may be involved in that treatment directly and indirectly.
- Obtain payment from third party payers
- Conduct normal healthcare operations such as quality assessments and physician certifications

I have been informed by you of your *Notice of Privacy Practices* containing a more complete description of the uses and disclosures of my health information. I have been given a copy of your *Notice of Privacy Practices* prior to signing this consent. I understand that this organization has the right to change its *Notice of Privacy Practices* from time to time and that I may contact this organization at any time at the address above to obtain a current copy of the *Notice of Privacy Practices.*

I understand that I may request in writing that you restrict how my private information is used or disclosed to carry out treatment, payment or healthcare operations. I also understand you are not required to agree to my requested restriction, but if you do agree then you are bound to abide by such restrictions.

I understand that I may revoke this consent in writing at any time, except to the extent that you have taken action relying on this consent.

Patient Name: _____

Signature: _____

Relationship to Patient: _____

Date: _____

B. XACML Policy for the working example

This appendix contains the XACML policy for the working example, which was generated using the UMU-XACML editor.

```
<?xml version="1.0" encoding="UTF-8"?>
<PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:cd:04"
PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable"
PolicySetId="Patient-ID">
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">patient-ID</AttributeValue>
          <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
        </SubjectMatch>
      </Subject>
    </Subjects>
  </Target>
  <Policy PolicyId="Patient-ID" RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
      algorithm:first-applicable">
    <Target />
    <Rule Effect="Deny" RuleId="11">
      <Target>
        <Resources>
          <Resource>
            <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-equal">
              <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">/patient-ID/Gynecological-
              information/*</AttributeValue>
              <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
              DataType="http://www.w3.org/2001/XMLSchema#string" />
            </ResourceMatch>
          </Resource>
        </Resources>
        <Subjects>
```

```
    <Subject>
      <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">mother-ID</AttributeValue>
        <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
      </SubjectMatch>
    </Subject>
  </Subjects>
  <Actions>
    <Action>
      <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
        <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
      </ActionMatch>
    </Action>
  </Actions>
</Target>
<Condition>
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
      <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal" />
      <EnvironmentAttributeDesignator AttributeId="purpose"
      DataType="http://www.w3.org/2001/XMLSchema#string" />
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">quality-
        measures</AttributeValue>
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">public-
        health</AttributeValue>
        <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">operations</AttributeValue>
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">payment</AttributeValue>
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">treatment</AttributeValue>
      </Apply>
    </Apply>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
      <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal" />
```

```
        <EnvironmentAttributeDesignator AttributeId="context"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">emergency</AttributeValue>
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">normal</AttributeValue>
        </Apply>
      </Apply>
    </Apply>
  </Condition>
</Rule>
<Rule Effect="Deny" RuleId="12">
  <Target>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">/patient-ID/Blood-
          pressure[age<=3]</AttributeValue>
          <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
        </ResourceMatch>
      </Resource>
    </Resources>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">mother-ID</AttributeValue>
          <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
        </SubjectMatch>
      </Subject>
    </Subjects>
    <Actions>
      <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
          <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
        </ActionMatch>
```

```
      </Action>
    </Actions>
  </Target>
  <Condition>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
        <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal" />
        <EnvironmentAttributeDesignator AttributeId="purpose"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">quality-
          measures</AttributeValue>
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">public-
          health</AttributeValue>
          <AttributeValue
          DataType="http://www.w3.org/2001/XMLSchema#string">operations</AttributeValue>
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">payment</AttributeValue>
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">treatment</AttributeValue>
        </Apply>
      </Apply>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
        <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal" />
        <EnvironmentAttributeDesignator AttributeId="context"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">emergency</AttributeValue>
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">normal</AttributeValue>
        </Apply>
      </Apply>
    </Apply>
  </Condition>
</Rule>
<Rule Effect="permit" RuleId="10">
  <Target>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-equal">
```

```
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">/patient-
        ID/*</AttributeValue>
        <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
      </ResourceMatch>
    </Resource>
  </Resources>
  <Subjects>
    <Subject>
      <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">mother-ID</AttributeValue>
        <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
      </SubjectMatch>
    </Subject>
  </Subjects>
  <Actions>
    <Action>
      <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
        <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
      </ActionMatch>
    </Action>
  </Actions>
</Target>
<Condition>
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
      <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal" />
      <EnvironmentAttributeDesignator AttributeId="purpose"
      DataType="http://www.w3.org/2001/XMLSchema#string" />
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">quality-
        measures</AttributeValue>
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">public-
        health</AttributeValue>
```

```
          <AttributeValue
          DataType="http://www.w3.org/2001/XMLSchema#string">operations</AttributeValue>
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">payment</AttributeValue>
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">treatment</AttributeValue>
        </Apply>
      </Apply>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
        <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal" />
        <EnvironmentAttributeDesignator AttributeId="context"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">emergency</AttributeValue>
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">normal</AttributeValue>
        </Apply>
      </Apply>
    </Apply>
  </Condition>
</Rule>
<Rule Effect="Permit" RuleId="9">
  <Target>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">/patient-
          ID/*</AttributeValue>
          <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
        </ResourceMatch>
      </Resource>
    </Resources>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">husband-
          ID</AttributeValue>
          <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
        </SubjectMatch>
```

```
    </Subject>
  </Subjects>
  <Actions>
    <Action>
      <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
        <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
      </ActionMatch>
    </Action>
  </Actions>
</Target>
<Condition>
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
      <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal" />
      <EnvironmentAttributeDesignator AttributeId="purpose"
      DataType="http://www.w3.org/2001/XMLSchema#string" />
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">quality-
        measures</AttributeValue>
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">public-
        health</AttributeValue>
        <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">operations</AttributeValue>
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">payment</AttributeValue>
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">treatment</AttributeValue>
      </Apply>
    </Apply>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
      <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal" />
      <EnvironmentAttributeDesignator AttributeId="context"
      DataType="http://www.w3.org/2001/XMLSchema#string" />
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">emergency</AttributeValue>
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">normal</AttributeValue>
      </Apply>
    </Apply>
```

```
    </Apply>
  </Condition>
</Rule>
<Rule Effect="Permit" RuleId="8">
  <Target>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">/patient-
          ID/*</AttributeValue>
          <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
        </ResourceMatch>
      </Resource>
    </Resources>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">dr.-John-Mathews-
          ID</AttributeValue>
          <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
        </SubjectMatch>
      </Subject>
    </Subjects>
    <Actions>
      <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">write</AttributeValue>
          <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
  <Condition>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
```

```
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">treatment</AttributeValue>
        <EnvironmentAttributeDesignator AttributeId="purpose"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
      </Apply>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">emergency</AttributeValue>
        <EnvironmentAttributeDesignator AttributeId="context"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
      </Apply>
    </Apply>
  </Condition>
</Rule>
<Rule Effect="Permit" RuleId="7">
  <Target>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">/patient-
          ID/*</AttributeValue>
          <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
        </ResourceMatch>
      </Resource>
    </Resources>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">dr.-John-Mathews-
          ID</AttributeValue>
          <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
        </SubjectMatch>
      </Subject>
    </Subjects>
    <Actions>
      <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
```

```
          <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
  <Condition>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">treatment</AttributeValue>
        <EnvironmentAttributeDesignator AttributeId="purpose"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
      </Apply>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">emergency</AttributeValue>
        <EnvironmentAttributeDesignator AttributeId="context"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
      </Apply>
    </Apply>
  </Condition>
</Rule>
<Rule Effect="Permit" RuleId="6">
  <Target>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">/patient-
          ID/*</AttributeValue>
          <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
        </ResourceMatch>
      </Resource>
    </Resources>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">doctor</AttributeValue>
```

```
        <SubjectAttributeDesignator AttributeId="role"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
      </SubjectMatch>
    </Subject>
  </Subjects>
  <Actions>
    <Action>
      <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">write</AttributeValue>
        <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
      </ActionMatch>
    </Action>
  </Actions>
</Target>
<Condition>
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">treatment</AttributeValue>
      <EnvironmentAttributeDesignator AttributeId="context"
      DataType="http://www.w3.org/2001/XMLSchema#string" />
    </Apply>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
      <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal" />
      <EnvironmentAttributeDesignator AttributeId="context"
      DataType="http://www.w3.org/2001/XMLSchema#string" />
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">emergency</AttributeValue>
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">normal</AttributeValue>
      </Apply>
    </Apply>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-less-than-or-equal">
      <EnvironmentAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-
      date" DataType="http://www.w3.org/2001/XMLSchema#date" />
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#date">2007-08-14</AttributeValue>
        <AttributeValue DataType="http://www.w3.org/TR/2002/WD-xquery-operators-
        20020816#yearMonthDuration">P1Y</AttributeValue>
```

```
        </Apply>
      </Apply>
    </Apply>
  </Condition>
</Rule>
<Rule Effect="Permit" RuleId="5">
  <Target>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">/patient-
          ID/*</AttributeValue>
          <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
        </ResourceMatch>
      </Resource>
    </Resources>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">doctor</AttributeValue>
          <SubjectAttributeDesignator AttributeId="role"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
        </SubjectMatch>
      </Subject>
    </Subjects>
    <Actions>
      <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
          <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
  <Condition>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
```

```
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">treatment</AttributeValue>
        <EnvironmentAttributeDesignator AttributeId="purpose"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
      </Apply>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
        <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal" />
        <EnvironmentAttributeDesignator AttributeId="context"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">emergency</AttributeValue>
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">normal</AttributeValue>
        </Apply>
      </Apply>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-less-than-or-equal">
        <EnvironmentAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-
        date" DataType="http://www.w3.org/2001/XMLSchema#date" />
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#date">2007-08-14</AttributeValue>
          <AttributeValue DataType="http://www.w3.org/TR/2002/WD-xquery-operators-
          20020816#yearMonthDuration">P1Y</AttributeValue>
        </Apply>
      </Apply>
    </Apply>
  </Condition>
</Rule>
<Rule Effect="Permit" RuleId="4">
  <Target>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">/patient-
          ID/*</AttributeValue>
          <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
        </ResourceMatch>
      </Resource>
    </Resources>
```

```
  <Subjects>
    <Subject>
      <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">personal-
        doctor</AttributeValue>
        <SubjectAttributeDesignator AttributeId="role"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
      </SubjectMatch>
    </Subject>
  </Subjects>
  <Actions>
    <Action>
      <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">write</AttributeValue>
        <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
      </ActionMatch>
    </Action>
  </Actions>
</Target>
<Condition>
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
      <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal" />
      <EnvironmentAttributeDesignator AttributeId="purpose"
      DataType="http://www.w3.org/2001/XMLSchema#string" />
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">quality-
        measures</AttributeValue>
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">public-
        health</AttributeValue>
        <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">operations</AttributeValue>
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">payment</AttributeValue>
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">treatment</AttributeValue>
      </Apply>
    </Apply>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
```

```
        <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal" />
        <EnvironmentAttributeDesignator AttributeId="context"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">emergency</AttributeValue>
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">normal</AttributeValue>
        </Apply>
      </Apply>
    </Apply>
  </Condition>
</Rule>
<Rule Effect="Permit" RuleId="3">
  <Target>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:any-of">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">/patient-
          ID/*</AttributeValue>
          <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
        </ResourceMatch>
      </Resource>
    </Resources>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">personal-
          doctor</AttributeValue>
          <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
        </SubjectMatch>
      </Subject>
    </Subjects>
    <Actions>
      <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
```

```
          <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
  <Condition>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
        <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal" />
        <EnvironmentAttributeDesignator AttributeId="purpose"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">quality-
          measures</AttributeValue>
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">public-
          health</AttributeValue>
          <AttributeValue
          DataType="http://www.w3.org/2001/XMLSchema#string">operations</AttributeValue>
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">payment</AttributeValue>
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">treatment</AttributeValue>
        </Apply>
      </Apply>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
        <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal" />
        <EnvironmentAttributeDesignator AttributeId="context"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">emergency</AttributeValue>
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">normal</AttributeValue>
        </Apply>
      </Apply>
    </Apply>
  </Condition>
</Rule>
<Rule Effect="Deny" RuleId="2">
  <Target>
    <Resources>
```

```
      <Resource>
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">/patient-
          ID/*</AttributeValue>
          <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
        </ResourceMatch>
      </Resource>
    </Resources>
    <Actions>
      <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">write</AttributeValue>
          <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
</Rule>
<Rule Effect="Deny" RuleId="1">
  <Target>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">/patient-
          ID/*</AttributeValue>
          <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
        </ResourceMatch>
      </Resource>
    </Resources>
    <Actions>
      <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
          <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
```

```
            </ActionMatch>
          </Action>
        </Actions>
      </Target>
    </Rule>
  </Policy>
</PolicySet>
```

C. Sample XACML request

```
<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="urn:oasis:names:tc:xacml:1.0:policy" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Subject>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>husband-ID</AttributeValue>
    </Attribute>
  </Subject>
  <Resource>
    <Attribute AttributeId=" urn:oasis:names:tc:xacml:1.0:resource:xpath"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>/patient-ID/*</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="purpose" DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>treatment</AttributeValue>
    </Attribute>
  </Environment>
  <Environment>
    <Attribute AttributeId="context" DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>emergency</AttributeValue>
    </Attribute>
  </Environment>
```

D. Priority-Overrides Rule-Combining Algorithm

This algorithm combines the effects of the rules of a policy to produce one policy response by taking into account the priorities of the rules. It evaluates all the rules in a policy, and for each rule, the algorithm evaluates the decision of the rule and its priority. If the rule is applicable to an access request, and the priority of the rule is the highest so far, then the decision of the rule is noted as the potential final decision of the rule-combining algorithm. The same is repeated for all the rules in the policy. If no applicable rule is found the algorithm returns a 'Not applicable' decision. If at least one of the rules was not correctly evaluated due to existing errors, then an 'Indeterminate' response is generated by the algorithm. If the decision of the applicable rule that had the highest priority is deny, then the algorithm returns 'Deny' as the result of the rule combination. On the other hand, if the decision of the applicable rule with the highest priority is permit, then the algorithm will return 'Permit' as the result of the rule combination. The final priority can be made accessible by keeping the priority variable as a global variable.

```
Decision priorityOverridesRuleCombiningAlgorithm(Rule rule[])
{
      Boolean atLeastOneError  = false;
      Boolean potentialDeny    = false;
      Boolean potentialPermit  = false;
      Int perviousPriority = 0;
      Int currentPriority = 0;

      for( i=0 ; i < lengthOf(rules) ; i++ )
      {
            Decision decision = evaluate(rule[i]);
            Priority priority = evaluatePriority(rule[i]);
            previousPriority = currentPriority;
            currentPriority = priority;
            if (previousPriority > currentPriority)
            {
                  currentPriority = previousPriority;
            }
            if (decision == Deny)
            {
                  potentialDeny = true;
                  potentialPermit = false;
                  continue;
            }
            if (decision == Permit)
            {
                  potentialDeny = false;
                  potentialPermit = true;
                  continue;
            }
            if (decision == NotApplicable)
            {
                  continue;
            }
            if (decision == Indeterminate)
            {
                  atLeastOneError = true;
            }
      }
      if (potentialDeny)
      {
            return Deny;
      }
      if (potentialPermit)
      {
            return Permit;
      }
      if (atLeastOneError)
      {
            return Indeterminate;
      }
      return NotApplicable;
}
```

E. Counterexamples generated by Margrave

This appendix shows the counterexamples that were generated by the Margrave tool during testing for the working example's XACML policy. As described in Chapter 7, the generation of counterexamples included coding each rule in a XACML policy into a property, and verifying each property against the policy. Each property was verified twice by restricting the decision of the rule to both 'permit' and 'deny'.

The key is presented first, and all the properties and the counterexamples they generated follow. Chapter 7 contains the details on how to interpret the properties and the counterexamples.

```
 1:/Action, urn:oasis:names:tc:xacml:1.0:action:action-id, read/
 2:/Action, urn:oasis:names:tc:xacml:1.0:action:action-id, OTHER/
 3:/Resource, urn:oasis:names:tc:xacml:1.0:resource:resource-id,
/patient-ID/*/
 4:/Resource, urn:oasis:names:tc:xacml:1.0:resource:resource-id, OTHER/
 5:/Action, urn:oasis:names:tc:xacml:1.0:action:action-id, write/
 6:/Environment, context, normal/
 7:/Environment, context, OTHER/
 8:/Environment, purpose, treatment/
 9:/Environment, purpose, OTHER/
 10:/Subject, role, personal-doctor/
 11:/Subject, role, OTHER/
 12:/Environment, context, emergency/
 13:/Environment, purpose, payment/
 14:/Environment, purpose, operations/
 15:/Environment, purpose, public-health/
 16:/Environment, purpose, quality-measures/
 17:/Environment, validity period, 1 year/
 18:/Environment, validity period, OTHER/
 19:/Subject, role, doctor/
 20:/Subject, urn:oasis:names:tc:xacml:1.0:subject-category:access-
subject, dr.-John-Mathews-ID/
 21:/Subject, urn:oasis:names:tc:xacml:1.0:subject-category:access-
subject, OTHER/
 22:/Subject, urn:oasis:names:tc:xacml:1.0:subject-category:access-
subject, husband-ID/
 23:/Subject, urn:oasis:names:tc:xacml:1.0:subject-category:access-
subject, mother-ID/
 24:/Resource, urn:oasis:names:tc:xacml:1.0:resource:resource-id,
/patient-ID/Gynecological-information/*/
 25:/Resource, urn:oasis:names:tc:xacml:1.0:resource:resource-id,
/patient-ID/Blood-pressure/*/
 26:/Subject, patient, patient-ID/
 27:/Subject, patient, OTHER/

======================================================================
(define Pol_1 (load-xacml-policy "../../Desktop/Patient-ID-policy-
set.xml"))
(define (find-Pr_1-counter-example Pol_1)
(avc-and (restrict-policy-to-dec 'D Pol_1)
(avc-and (make-avc 'Subject 'urn:oasis:names:tc:xacml:1.0:subject-
category:access-subject 'mother-ID)
```

```
(make-avc 'Resource 'urn:oasis:names:tc:xacml:1.0:resource:resource-id
'/patient-ID/Blood-pressure/*)
(make-avc 'Action 'urn:oasis:names:tc:xacml:1.0:action:action-id 'read)
(make-avc 'Environment 'purpose 'quality-measures)
(make-avc 'Environment 'context 'emergency))))

         1         2
12345678901234567890123456
{
1----------1---1------1-1--

}
============================================================================

(define Pol_1 (load-xacml-policy "../../Desktop/Patient-ID-policy-
set.xml"))
(define (find-Pr_1-counter-example Pol_1)
(avc-and (restrict-policy-to-dec 'P Pol_1)
(avc-and (make-avc 'Subject 'urn:oasis:names:tc:xacml:1.0:subject-
category:access-subject 'mother-ID)
(make-avc 'Resource 'urn:oasis:names:tc:xacml:1.0:resource:resource-id
'/patient-ID/Blood-pressure/*)
(make-avc 'Action 'urn:oasis:names:tc:xacml:1.0:action:action-id 'read)
(make-avc 'Environment 'purpose 'quality-measures)
(make-avc 'Environment 'context 'emergency))))

         1         2
12345678901234567890123456
{

}

============================================================================
(define Pol_1 (load-xacml-policy "../../Desktop/Patient-ID-policy-
set.xml"))
(define (find-Pr_1-counter-example Pol_1)
(avc-and (restrict-policy-to-dec 'D Pol_1)
(avc-and (make-avc 'Subject 'urn:oasis:names:tc:xacml:1.0:subject-
category:access-subject 'mother-ID)
(make-avc 'Resource 'urn:oasis:names:tc:xacml:1.0:resource:resource-id
'/patient-ID/Gynecological-information/*)
(make-avc 'Action 'urn:oasis:names:tc:xacml:1.0:action:action-id 'read)
(make-avc 'Environment 'purpose 'quality-measures)
(make-avc 'Environment 'context 'emergency))))

         1         2
12345678901234567890123456
{
1----------1---1------11---

}
============================================================================

(define Pol_1 (load-xacml-policy "../../Desktop/Patient-ID-policy-
set.xml"))
(define (find-Pr_1-counter-example Pol_1)
(avc-and (restrict-policy-to-dec 'P Pol_1)
```

```
(avc-and (make-avc 'Subject 'urn:oasis:names:tc:xacml:1.0:subject-
category:access-subject 'mother-ID)
(make-avc 'Resource 'urn:oasis:names:tc:xacml:1.0:resource:resource-id
'/patient-ID/Gynecological-information/*)
(make-avc 'Action 'urn:oasis:names:tc:xacml:1.0:action:action-id 'read)
(make-avc 'Environment 'purpose 'quality-measures)
(make-avc 'Environment 'context 'emergency))))

         1         2
12345678901234567890123456
{

}

==========================================================================

(define Pol_1 (load-xacml-policy "../../Desktop/Patient-ID-policy-
set.xml"))
(define (find-Pr_1-counter-example Pol_1)
(avc-and (restrict-policy-to-dec 'P Pol_1)
(avc-and (make-avc 'Subject 'urn:oasis:names:tc:xacml:1.0:subject-
category:access-subject 'mother-ID)
(make-avc 'Resource 'urn:oasis:names:tc:xacml:1.0:resource:resource-id
'/patient-ID/*)
(make-avc 'Action 'urn:oasis:names:tc:xacml:1.0:action:action-id 'read)
(make-avc 'Environment 'purpose 'quality-measures)
(make-avc 'Environment 'context 'emergency))))

         1         2
12345678901234567890123456
{
1-1--------1---1------100--

}
==========================================================================
(define Pol_1 (load-xacml-policy "../../Desktop/Patient-ID-policy-
set.xml"))
(define (find-Pr_1-counter-example Pol_1)
(avc-and (restrict-policy-to-dec 'D Pol_1)
(avc-and (make-avc 'Subject 'urn:oasis:names:tc:xacml:1.0:subject-
category:access-subject 'mother-ID)
(make-avc 'Resource 'urn:oasis:names:tc:xacml:1.0:resource:resource-id
'/patient-ID/*)
(make-avc 'Action 'urn:oasis:names:tc:xacml:1.0:action:action-id 'read)
(make-avc 'Environment 'purpose 'quality-measures)
(make-avc 'Environment 'context 'emergency))))

         1         2
12345678901234567890123456
{
1-1--------1---1------101--
1-1--------1---1------11---

}

==========================================================================
```

```
(define Pol_1 (load-xacml-policy "../../Desktop/Patient-ID-policy-
set.xml"))
(define (find-Pr_1-counter-example Pol_1)
(avc-and (restrict-policy-to-dec 'P Pol_1)
(avc-and (make-avc 'Subject 'urn:oasis:names:tc:xacml:1.0:subject-
category:access-subject 'husband-ID)
(make-avc 'Resource 'urn:oasis:names:tc:xacml:1.0:resource:resource-id
'/patient-ID/*)
(make-avc 'Action 'urn:oasis:names:tc:xacml:1.0:action:action-id 'read)
(make-avc 'Environment 'purpose 'quality-measures)
(make-avc 'Environment 'context 'emergency))))

         1         2
12345678901234567890123456
{
1-1--------1---1-----10----
1-1--------1---1-----1100--

}
```

==========

```
(define Pol_1 (load-xacml-policy "../../Desktop/Patient-ID-policy-
set.xml"))
(define (find-Pr_1-counter-example Pol_1)
(avc-and (restrict-policy-to-dec 'D Pol_1)
(avc-and (make-avc 'Subject 'urn:oasis:names:tc:xacml:1.0:subject-
category:access-subject 'husband-ID)
(make-avc 'Resource 'urn:oasis:names:tc:xacml:1.0:resource:resource-id
'/patient-ID/*)
(make-avc 'Action 'urn:oasis:names:tc:xacml:1.0:action:action-id 'read)
(make-avc 'Environment 'purpose 'quality-measures)
(make-avc 'Environment 'context 'emergency))))

         1         2
12345678901234567890123456
{
1-1--------1---1-----1101--
1-1--------1---1-----111---

}
```

==========

```
(define Pol_1 (load-xacml-policy "../../Desktop/Patient-ID-policy-
set.xml"))
(define (find-Pr_1-counter-example Pol_1)
(avc-and (restrict-policy-to-dec 'P Pol_1)
(avc-and (make-avc 'Subject 'urn:oasis:names:tc:xacml:1.0:subject-
category:access-subject 'dr.-John-Mathews-ID)
(make-avc 'Resource 'urn:oasis:names:tc:xacml:1.0:resource:resource-id
'/patient-ID/*)
(make-avc 'Action 'urn:oasis:names:tc:xacml:1.0:action:action-id 'read)
(make-avc 'Environment 'purpose 'treatment)
(make-avc 'Environment 'context 'emergency))))

         1         2
```

```
12345678901234567890123456 7
{
1-1----1---1-------1--0----
1-1----1---1-------1--100--

}

==========================================================================
(define Pol_1 (load-xacml-policy "../../Desktop/Patient-ID-policy-
set.xml"))
(define (find-Pr_1-counter-example Pol_1)
(avc-and (restrict-policy-to-dec 'D Pol_1)
(avc-and (make-avc 'Subject 'urn:oasis:names:tc:xacml:1.0:subject-
category:access-subject 'dr.-John-Mathews-ID)
(make-avc 'Resource 'urn:oasis:names:tc:xacml:1.0:resource:resource-id
'/patient-ID/*)
(make-avc 'Action 'urn:oasis:names:tc:xacml:1.0:action:action-id 'read)
(make-avc 'Environment 'purpose 'treatment)
(make-avc 'Environment 'context 'emergency))))

         1         2
12345678901234567890123456 7
{
1-1----1---1-------1--101--
1-1----1---1-------1--11---

}
==========================================================================
(define Pol_1 (load-xacml-policy "../../Desktop/Patient-ID-policy-
set.xml"))
(define (find-Pr_1-counter-example Pol_1)
(avc-and (restrict-policy-to-dec 'P Pol_1)
(avc-and (make-avc 'Subject 'urn:oasis:names:tc:xacml:1.0:subject-
category:access-subject 'dr.-John-Mathews-ID)
(make-avc 'Resource 'urn:oasis:names:tc:xacml:1.0:resource:resource-id
'/patient-ID/*)
(make-avc 'Action 'urn:oasis:names:tc:xacml:1.0:action:action-id 'write)
(make-avc 'Environment 'purpose 'treatment)
(make-avc 'Environment 'context 'emergency))))

         1         2
12345678901234567890123456 7
{
0-1-1--1---1-------1-------
1-1-1--1---1-------1--0----
1-1-1--1---1-------1--100--

}

==========================================================================
(define Pol_1 (load-xacml-policy "../../Desktop/Patient-ID-policy-
set.xml"))
(define (find-Pr_1-counter-example Pol_1)
(avc-and (restrict-policy-to-dec 'D Pol_1)
(avc-and (make-avc 'Subject 'urn:oasis:names:tc:xacml:1.0:subject-
category:access-subject 'dr.-John-Mathews-ID)
```

```
(make-avc 'Resource 'urn:oasis:names:tc:xacml:1.0:resource:resource-id
'/patient-ID/*)
(make-avc 'Action 'urn:oasis:names:tc:xacml:1.0:action:action-id 'write)
(make-avc 'Environment 'purpose 'treatment)
(make-avc 'Environment 'context 'emergency))))

         1         2
12345678901234567890123456
{
1-1-1--1---1-------1--101--
1-1-1--1---1-------1--11---

}

============================================================================
(define Pol_1 (load-xacml-policy "../../Desktop/Patient-ID-policy-
set.xml"))
(define (find-Pr_1-counter-example Pol_1)
(avc-and (restrict-policy-to-dec 'P Pol_1)
(avc-and (make-avc 'Subject 'role 'doctor)
(make-avc 'Resource 'urn:oasis:names:tc:xacml:1.0:resource:resource-id
'/patient-ID/*)
(make-avc 'Action 'urn:oasis:names:tc:xacml:1.0:action:action-id 'write)
(make-avc 'Environment 'purpose 'treatment)
(make-avc 'Environment 'context 'emergency))))

         1         2
12345678901234567890123456
{
0-1-1--1---1------1--------
1-1-1--1---1------1---0----
1-1-1--1---1------1---100--

}

============================================================================

(define Pol_1 (load-xacml-policy "../../Desktop/Patient-ID-policy-
set.xml"))
(define (find-Pr_1-counter-example Pol_1)
(avc-and (restrict-policy-to-dec 'D Pol_1)
(avc-and (make-avc 'Subject 'role 'doctor)
(make-avc 'Resource 'urn:oasis:names:tc:xacml:1.0:resource:resource-id
'/patient-ID/*)
(make-avc 'Action 'urn:oasis:names:tc:xacml:1.0:action:action-id 'write)
(make-avc 'Environment 'purpose 'treatment)
(make-avc 'Environment 'context 'emergency))))

         1         2
12345678901234567890123456
{
1-1-1--1---1------1---101--
1-1-1--1---1------1---11---

}

============================================================================
```

```
(define Pol_1 (load-xacml-policy "../../Desktop/Patient-ID-policy-
set.xml"))
(define (find-Pr_1-counter-example Pol_1)
(avc-and (restrict-policy-to-dec 'P Pol_1)
(avc-and (make-avc 'Subject 'role 'doctor)
(make-avc 'Resource 'urn:oasis:names:tc:xacml:1.0:resource:resource-id
'/patient-ID/*)
(make-avc 'Action 'urn:oasis:names:tc:xacml:1.0:action:action-id 'read)
(make-avc 'Environment 'purpose 'treatment)
(make-avc 'Environment 'context 'emergency))))

         1         2
12345678901234567890123456
{
1-1----1---1------1---0----
1-1----1---1------1---100--

}
============================================================================

(define Pol_1 (load-xacml-policy "../../Desktop/Patient-ID-policy-
set.xml"))
(define (find-Pr_1-counter-example Pol_1)
(avc-and (restrict-policy-to-dec 'D Pol_1)
(avc-and (make-avc 'Subject 'role 'doctor)
(make-avc 'Resource 'urn:oasis:names:tc:xacml:1.0:resource:resource-id
'/patient-ID/*)
(make-avc 'Action 'urn:oasis:names:tc:xacml:1.0:action:action-id 'read)
(make-avc 'Environment 'purpose 'treatment)
(make-avc 'Environment 'context 'emergency))))

         1         2
12345678901234567890123456
{
1-1----1---1------1---101--
1-1----1---1------1---11---

}

============================================================================
(define Pol_1 (load-xacml-policy "../../Desktop/Patient-ID-policy-
set.xml"))
(define (find-Pr_1-counter-example Pol_1)
(avc-and (restrict-policy-to-dec 'P Pol_1)
(avc-and (make-avc 'Subject 'role 'personal-doctor)
(make-avc 'Resource 'urn:oasis:names:tc:xacml:1.0:resource:resource-id
'/patient-ID/*)
(make-avc 'Action 'urn:oasis:names:tc:xacml:1.0:action:action-id 'write)
(make-avc 'Environment 'purpose 'quality-measures)
(make-avc 'Environment 'context 'emergency))))

         1         2
12345678901234567890123456
{
0-1-1----1-1---1-----------
1-1-1----1-1---1------0----
1-1-1----1-1---1------100--
```

```
}

================================================================================
(define Pol_1 (load-xacml-policy "../../Desktop/Patient-ID-policy-
set.xml"))
(define (find-Pr_1-counter-example Pol_1)
(avc-and (restrict-policy-to-dec 'D Pol_1)
(avc-and (make-avc 'Subject 'role 'personal-doctor)
(make-avc 'Resource 'urn:oasis:names:tc:xacml:1.0:resource:resource-id
'/patient-ID/*)
(make-avc 'Action 'urn:oasis:names:tc:xacml:1.0:action:action-id 'write)
(make-avc 'Environment 'purpose 'quality-measures)
(make-avc 'Environment 'context 'emergency))))

         1         2
12345678901234567890123456
{
1-1-1----1-1---1------101--
1-1-1----1-1---1------11---

}

================================================================================
(define Pol_1 (load-xacml-policy "../../Desktop/Patient-ID-policy-
set.xml"))
(define (find-Pr_1-counter-example Pol_1)
(avc-and (restrict-policy-to-dec 'P Pol_1)
(avc-and (make-avc 'Subject 'role 'personal-doctor)
(make-avc 'Resource 'urn:oasis:names:tc:xacml:1.0:resource:resource-id
'/patient-ID/*)
(make-avc 'Action 'urn:oasis:names:tc:xacml:1.0:action:action-id 'read)
(make-avc 'Environment 'purpose 'quality-measures)
(make-avc 'Environment 'context 'emergency))))
         1         2
12345678901234567890123456
{
1-1------1-1---1------0----
1-1------1-1---1------100--

}

================================================================================
```

```
(define Pol_1 (load-xacml-policy "../../Desktop/Patient-ID-policy-
set.xml"))
(define (find-Pr_1-counter-example Pol_1)
(avc-and (restrict-policy-to-dec 'D Pol_1)
(avc-and (make-avc 'Subject 'role 'personal-doctor)
(make-avc 'Resource 'urn:oasis:names:tc:xacml:1.0:resource:resource-id
'/patient-ID/*)
(make-avc 'Action 'urn:oasis:names:tc:xacml:1.0:action:action-id 'read)
(make-avc 'Environment 'purpose 'quality-measures)
(make-avc 'Environment 'context 'emergency))))

         1         2
12345678901234567890123456
{
1-1------1-1---1------101--
1-1------1-1---1------11---

}
```

F. Patient consent policy in Prolog

This appendix contains the authorisations in the decision table, as was loaded in the Prolog engine for testing.

```
cando(patient_ID_ehr, deny, read, S, P, C, N, Pr):-N=1,Pr=1.

cando(patient_ID_ehr, deny, write, S, P, C, N, Pr):-N=2,Pr=2.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=personal_doctor,P=treatment,C=normal,N=39,Pr=3.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=personal_doctor,P=treatment,C=emergency,N=38,Pr=3.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=personal_doctor,P=payment,C=normal,N=37,Pr=3.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=personal_doctor,P=payment,C=emergency,N=36,Pr=3.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=personal_doctor,P=operations,C=normal,N=35,Pr=3.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=personal_doctor,P=operations,C=emergency,N=34,Pr=3.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=personal_doctor,P=public_health,C=normal,N=33,Pr=3.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-
      S=personal_doctor,P=public_health,C=emergency,N=32,Pr=3.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-
      S=personal_doctor,P=quality_measures,C=normal,N=31,Pr=3.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr) :-
      S=personal_doctor,P=quality_measures,C=emergency,N=30,Pr=3.

cando(patient_ID_ehr, permit, write, S, P, C, N, Pr):-S=personal_doctor,P=treatment,C=normal,N=49,Pr=4.

cando(patient_ID_ehr, permit, write, S, P, C, N, Pr):-S=personal_doctor,P=treatment,C=emergency,N=48,Pr=4.

cando(patient_ID_ehr, permit, write, S, P, C, N, Pr):-S=personal_doctor,P=payment,C=normal,N=47,Pr=4.

cando(patient_ID_ehr, permit, write, S, P, C, N, Pr):-S=personal_doctor,P=payment,C=emergency,N=46,Pr=4.

cando(patient_ID_ehr, permit, write, S, P, C, N, Pr):-S=personal_doctor,P=operations,C=normal,N=45,Pr=4.

cando(patient_ID_ehr, permit, write, S, P, C, N, Pr):-S=personal_doctor,P=operations,C=emergency,N=44,Pr=4.

cando(patient_ID_ehr, permit, write, S, P, C, N, Pr):-S=personal_doctor,P=public_health,C=normal,N=43,Pr=4.
```

```
cando(patient_ID_ehr, permit, write, S, P, C, N, Pr):-
      S=personal_doctor,P=public_health,C=emergency,N=42,Pr=4.

cando(patient_ID_ehr, permit, write, S, P, C, N, Pr):-
      S=personal_doctor,P=quality_measures,C=normal,N=41,Pr=4.

cando(patient_ID_ehr, permit, write, S, P, C, N, Pr):-
      S=personal_doctor,P=quality_measures,C=emergency,N=40,Pr=4.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=doctor,P=treatment,C=normal,N=51,Pr=5.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=doctor,P=treatment,C=emergency,N=50,Pr=5.

cando(patient_ID_ehr, permit, write, S, P, C, N, Pr):-S=doctor,P=treatment,C=normal,N=61,Pr=6.

cando(patient_ID_ehr, permit, write, S, P, C, N, Pr):-S=doctor,P=treatment,C=emergency,N=60,Pr=6.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=dr_john_mathews,P=treatment,C=emergency,N=7,Pr=7.

cando(patient_ID_ehr, permit, write, S, P, C, N, Pr):-S=dr_john_mathews,P=treatment,C=emergency,N=8,Pr=8.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=husband_ID,P=treatment,C=normal,N=99,Pr=9.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=husband_ID,P=treatment,C=emergency,N=98,Pr=9.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=husband_ID,P=payment,C=normal,N=97,Pr=9.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=husband_ID,P=payment,C=emergency,N=96,Pr=9.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=husband_ID,P=operations,C=normal,N=95,Pr=9.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=husband_ID,P=operations,C=emergency,N=94,Pr=9.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=husband_ID,P=public_health,C=normal,N=93,Pr=9.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=husband_ID,P=public_health,C=emergency,N=92,Pr=9.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=husband_ID,P=quality_measures,C=normal,N=91,Pr=9.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=husband_ID,P=quality_measures,C=emergency,N=90,Pr=9.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=mother_ID,P=treatment,C=normal,N=109,Pr=10.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=mother_ID,P=treatment,C=emergency,N=108,Pr=10.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=mother_ID,P=payment,C=normal,N=107,Pr=10.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=mother_ID,P=payment,C=emergency,N=106,Pr=10.
```

```
cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=mother_ID,P=operations,C=normal,N=105,Pr=10.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=mother_ID,P=operations,C=emergency,N=104,Pr=10.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=mother_ID,P=public_health,C=normal,N=103,Pr=10.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=mother_ID,P=public_health,C=emergency,N=102,Pr=10.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-S=mother_ID,P=quality_measures,C=normal,N=101,Pr=10.

cando(patient_ID_ehr, permit, read, S, P, C, N, Pr):-
      S=mother_ID,P=quality_measures,C=emergency,N=100,Pr=10.

cando(patient_ID_gynecological_information, deny, read, S, P, C, N, Pr):-
      S=mother_ID,P=treatment,C=normal,N=119,Pr=12.

cando(patient_ID_gynecological_information, deny, read, S, P, C, N, Pr):-
      S=mother_ID,P=treatment,C=emergency,N=118,Pr=12.

cando(patient_ID_gynecological_information, deny, read, S, P, C, N, Pr):-
      S=mother_ID,P=payment,C=normal,N=117,Pr=12.

cando(patient_ID_gynecological_information, deny, read, S, P, C, N, Pr):-
      S=mother_ID,P=payment,C=emergency,N=116,Pr=12.

cando(patient_ID_gynecological_information, deny, read, S, P, C, N, Pr):-
      S=mother_ID,P=operations,C=normal,N=115,Pr=12.

cando(patient_ID_gynecological_information, deny, read, S, P, C, N, Pr):-
      S=mother_ID,P=operations,C=emergency,N=114,Pr=12.

cando( patient_ID_gynecological_information, deny, read, S, P, C, N, Pr) :-
      S=mother_ID,P=public_health,C=normal,N=113,Pr=12.

cando( patient_ID_gynecological_information, deny, read, S, P, C, N, Pr) :-
      S=mother_ID,P=public_health,C=emergency,N=112,Pr=12.

cando( patient_ID_gynecological_information, deny, read, S, P, C, N, Pr) :-
      S=mother_ID,P=quality_measures,C=normal,N=111,Pr=12.

cando( patient_ID_gynecological_information, deny, read, S, P, C, N, Pr) :-
      S=mother_ID,P=quality_measures,C=emergency,N=110,Pr=12.

cando( patient_ID_blood_pressure, deny, read, S, P, C, N, Pr) :-
      S=mother_ID,P=treatment,C=normal,N=129,Pr=11.
```

```
cando( patient_ID_blood_pressure, deny, read, S, P, C, N, Pr) :-
      S=mother_ID,P=treatment,C=emergency,N=128,Pr=11.

cando( patient_ID_blood_pressure, deny, read, S, P, C, N, Pr) :-
      S=mother_ID,P=payment,C=normal,N=127,Pr=11.

cando( patient_ID_blood_pressure, deny, read, S, P, C, N, Pr) :-
      S=mother_ID,P=payment,C=emergency,N=126,Pr=11.

cando( patient_ID_blood_pressure, deny, read, S, P, C, N, Pr) :-
      S=mother_ID,P=operations,C=normal,N=125,Pr=11.

cando( patient_ID_blood_pressure, deny, read, S, P, C, N, Pr) :-
      S=mother_ID,P=operations,C=emergency,N=124,Pr=11.

cando( patient_ID_blood_pressure, deny, read, S, P, C, N, Pr) :-
      S=mother_ID,P=public_health,C=normal,N=123,Pr=11.

cando( patient_ID_blood_pressure, deny, read, S, P, C, N, Pr) :-
      S=mother_ID,P=public_health,C=emergency,N=122,Pr=11.

cando(patient_ID_blood_pressure, deny, read, S, P, C, N, Pr):-
      S=mother_ID,P=quality_measures,C=normal,N=121,Pr=11.

cando(patient_ID_blood_pressure, deny, read, S, P, C, N, Pr):-
      S=mother_ID,P=quality_measures,C=emergency,N=120,Pr=1.
```