Privacy Enforcement with Data Owner-defined Policies

Dissertation

eingereicht von

Diplom-Informatiker Thomas Scheffler

vorgelegt der

Mathematisch-Naturwissenschaftlichen Fakultät
der Universität Potsdam
zur Erlangung des Akademischen Grades

Doktor der Naturwissenschaften
– Dr. rer. nat. –

angefertigt am
Institut für Informatik der Universität Potsdam
Professur Betriebssysteme und Verteilte Systeme

Potsdam, den 2. September 2013

Abstract

Data privacy continues to be a very important topic, as our dependency on electronic communication maintains its current growth and private data is shared between multiple devices, users and locations. The growing amount and the ubiquitous availability of personal private data increases the likelihood of data misuse, where private data may be used against the privacy preferences of the person that is identified by it and personal information might be revealed.

Documented cases of privacy breaches show that misuse of data has multiple causes, malicious intent is only one of them. A substantial number of privacy breaches also occur due to carelessness of data users and disregard of the original privacy requirements for the data. Privacy advocates like Goldberg [2003] and Stytz [2005] have long requested that technical measures should be used for the privacy protection of data in applications and data exchange protocols. Documented data breaches, such as the illegitimate sharing of airline passenger data in open conflict with the agreed privacy policy [Anton, He, and Baumer, 2004], have raised the demand for effective privacy protection.

Early privacy protection techniques, such as anonymous email and payment systems that have been developed by Chaum [1981, 1985], focused on data avoidance and anonymous use of services. They did not take into account that data sharing can not be avoided if people want to participate in electronic communication scenarios that involve social interactions. Many data privacy protagonists still focus their efforts on data avoidance and limiting data collection, while society has moved on. People use social networking platforms, store personal private data online and make private data available to ‘friends’ that participate in the same ‘network’. This leads to a situation where data is shared widely and uncontrollably and in most cases the data owner has no control over the further distribution and use of data that has been submitted to such services.

Previous efforts to integrate privacy awareness into data processing workflows have focused on the extension of existing access control frameworks with privacy aware functions [Park and Sandhu, 2004; Sevinç and Basin, 2006] or have analysed specific individual problems such as the expressiveness of policy languages [Karjoth, Schunter, and Herreweghen, 2003; Ashley and Karjoth, 2003]. So far very few implementations of overarching privacy protection mechanisms exist and can be studied to prove their effectiveness for privacy protection. Second level issues that stem from practical application of the implemented mechanisms, such as usability, life-time data management and changes in trustworthiness have received very little attention so far, mainly because they require actual implementations to be studied.

This thesis proposes a mechanism for the controlled distribution and use of personal private data that combines existing concepts for the specification, distribution and enforcement of access control policies with novel ideas to build a privacy protection framework with unique properties.

Most existing privacy protection schemes silently assume that it is the privilege of the *data user* to define the contract under which personal private data is released. Such an approach simplifies policy management and policy enforcement for the data user, but leaves the *data owner* with a binary decision to submit or withhold his or her personal data based on the provided policy. It is also far from clear to the people that are providing personal data, what happens when the data user changes the rules of this contract at a later time. In many cases it was shown that the stated

privacy policy amounts to a simple privacy promise, because the legal text of the declaration is not directly tied to reliable practical enforcement.

Our framework changes this assumption. We argue that granted rights must be automatically enforceable in order to be effective against carelessness and oversight on the side of the data user. If a privacy policy is agreed, this policy should be valid for all further access to the data. Furthermore, we want to empower the *data owner* to express his or her privacy preferences though privacy policies that follow the so called Owner-Retained Access Control (ORAC) mechanism. ORAC has been proposed by McCollum, Messing, and Notargiacomo [1990] as an alternate access control mechanism that offers the originator of the data, in our case the data owner, a way to express their own access control policies. A data access rule in our framework is bound to a particular subject, which could be an individual person or an organisation. The permission that is expressed in this rule is not transitive. This means that access rules strictly apply to the named subject in the policy and can not be delegated.

The data owner is given control over the release policy for his or her personal data and he or she can set permissions or restrictions according to individually perceived trust values. Such a policy needs to be expressed in a coherent way and must allow the deterministic policy evaluation by different entities. We compared different privacy policy languages and came to the conclusion that the eXtensible Access Control Markup Language (XACML) [XACML-2.0, 2005] offers a rich set of features that can be used for the expression of privacy preferences. Our privacy policies are therefore written as a set of rules in the XACML policy description language.

The privacy policy has to be communicated from the data owner to the data user, so that the data user can act accordingly. Our data protection framework augments private data with an explicit privacy policy in order to fulfil this requirement. Data and policy are stored together as a *Protected Data Object* that follows the *Sticky Policy* model as defined by Mont, Pearson, and Bramhall [2003] and Karjoth, Schunter, and Waidner [2003]. Data access policies can be referenced whenever data access is about to happen – independent of time and location of the access.

We developed a unique policy combination approach that takes usability aspects for the creation and maintenance of policies into consideration. Our privacy policy consists of three parts:
A *default policy* provides basic privacy protection if no specific rules have been entered by the data owner. An *owner policy* part allows the customisation of the default policy by the data owner. A third part of the policy, the so called *safety policy*, guarantees that the data owner can not specify disadvantageous policies, which, for example, exclude him or her from further access to the private data. We believe that this precaution is necessary, because the creators of the privacy policy are ordinary computer users and not trained privacy policy experts and giving the policy administrator complete control over the policy rule base may also lead to cases where the created rules are erroneous or harmful. The combined evaluation of these three policy-parts yields the necessary access decision.

The automatic enforcement of privacy policies is another important building block in our protection framework. We started our work with the development of a client-side protection mechanism that allows the enforcement of data-use restrictions *after* private data has been released to the data user. The client-side enforcement component for data-use policies is based on a modified Java Security Framework [Scheffler, Geiß, and Schnor, 2008], where XACML privacy policies are translated into corresponding Java permissions that can be automatically enforced by the Java Security Manager. This approach allows the privacy-aware usage of existing Java applications without implementing policy checks in the application itself. Our reference monitor implementation uses a modified Java class loader to bind the policy-derived permissions to a loaded class and thereby allows privacy enforcement for individual instances of an application class.

After evaluating benefits and drawbacks of the client-side solution we extended our work to also offer data privacy protection for scenarios that require server-side protection mechanisms. A number of usage scenarios today require the processing of sensitive private data by service providers. Prominent examples of such a use case are location-based services.

Our approach of policy enforcement through Java permissions requires the re-load of application classes for different data-sets, because once a Java class is loaded by the class loader, the set of permissions is fixed and can not be adapted. We found that server-side protection mechanisms can not be easily based on the enforcement of Java permissions by the Java Security Framework, because business applications usually follow a tiered architecture that separates different functions such as business logic, data access and data representation. Data access in a tiered business application is handled by Data Access Objects (DAOs) that might be shared by different services. Reloading a DAO for every data access is not an option.

We solved this problem by extending our reference monitor design to use Aspect-oriented Programming (AOP) and the Java Reflection API to intercept data accesses in existing applications and provide a way to enforce data owner-defined privacy policies for business applications [Scheffler, Schindler, and Schnor, 2012].

Contents

1 Introduction

After many years of research in *Privacy Enhancing Technologies* there still hardly passes a day when no loss of sensitive personal data needs to be reported: networked computer systems are compromised, laptops or physical storage media containing private data are lost, personal data are copied and used against stated privacy policies. The rise of Internet-connected smart-phones and social media platforms only extends the problem further. Personal data is collected, analysed and stored in computer systems throughout the world.

Non-profit consumer organisation such as "The Privacy Rights Clearinghouse"[1], as well as the "Open Security Foundation"[2], have compiled lists of known cases of intended or unintended releases of personal data for the US, the EU and other parts of the world. These organisations started collecting privacy related incidents systematically in 2005. As of April 2012, the Privacy Rights Clearinghouse lists 3602 events of data breaches, involving more than 500 million data-records. These statistics show no sign that the number and consequences of data breaches affecting sensitive personal data, also called data spills or data leaks, is reducing.

A high number of the data release cases in these lists can be classified as consequences of direct or indirect attacks from third parties that want to monetize sensitive data, such as credit card numbers (cf. Figure 1.1). However, there is also a significant number of cases where private data was released as a consequence of carelessness and neglect by the custodian of the data. Simple examples include the misconfiguration of databases and web servers that lead to disclosure of personal private data to the public, the misplacement of backup-tapes, as well as unauthorised data access by insiders.

This raises the question how this problematic situation can be resolved. A naïve proposal would ask for the installation and enforcement of privacy policies by the affected organisations. But again, a close inspection of the data release cases in these lists shows that such incidents also occur within organisations that have installed privacy and security policies.

For example the privacy policy of the University of Texas states:

> The University of Texas at Austin (U. T. Austin) is committed to ensuring the privacy and accuracy of your confidential information.
>
> ...the U. T. Austin has deployed extensive security measures to protect against the loss, misuse, or alteration of the information under our control. These security measures and our systems are audited by certified independent security specialists.
>
> [University of Texas, 2008]

[1] http://www.privacyrights.org/data-breach

[2] http://datalossdb.org

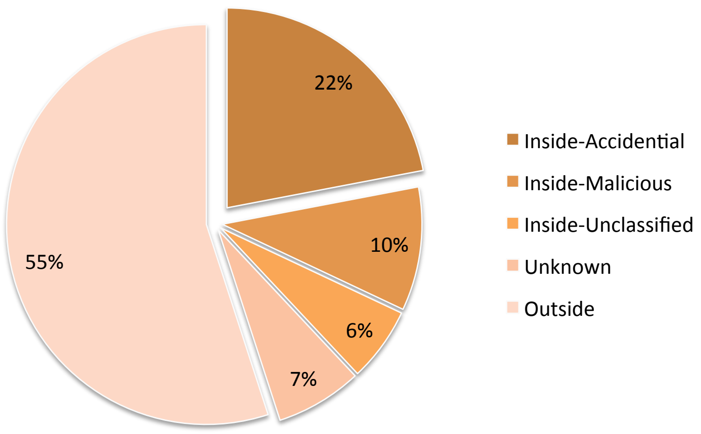


Figure 1.1: Incident Vectors of Data Loss Events - The DataLossDB collects information about known data breaches. In addition to scope and technical details such as type of data and data breach, it also classifies events according to the incident vector. The statistics show that 22% all events are caused through carelessness or neglect. [DataLossDB, 2012]

Nevertheless, the privacy policy of the University failed to prevent the public disclosure of 2500 data records containing personal information of students and faculty members in mid 2008 [Privacy Rights Clearinghouse, 2012].

Anton, He, and Baumer [2004] have documented and analysed an exemplary case where 5 million private customer travel records collected by the JetBlue airline company were shared between different private companies in full breach of the stated privacy policy and against governmental regulations.

A more recent, well published privacy breach is the excessive collection and potential distribution of geo-location data within mobile phones as described by Allan and Warden [2011].

These published cases of privacy breaches are only the tip of an iceberg of similar incidents happening all over the world and have been made publicly known because of their severity or the high profile of the involved persons or organisations. They show, however, that there exists a mismatch between stated and implemented privacy practices, because they should never have happened if the stated privacy policy had been followed.

Service-providers regularly define *Privacy Policies* that are part of their terms of services. These policies are intended to safeguard the private data of their clients by restricting their use. This is usually done by defining a number of purposes for which the data can be used and by restricting the entities that may access this data.

When we examine the privacy policy of the University of Texas more closely, we find two characteristic features that are shared by many installed privacy policies:

1. This privacy statement is a human-readable policy that makes broad promises to the person that leaves confidential data with the University: The policy states that everything possible is done to make collected private data secure.

   Not only is this claim difficult to prove or disprove, it is also difficult for the University to keep the stated promise. The claim in the policy would have to be made concrete in order to enforce it: Who has access to the data? What protection mechanisms are to be deployed? How can illegitimate data access be detected and denied?

The textual representation of the privacy policy gives no answers to this question to the person that entrusts their personal data with the University. Equally, a system designer that would need to implement this policy in the IT-systems of the University receives only vague guidance from such a policy description.

2. The scope and authority of this policy is set by the University, the entity that receives the private data, and not by the person who's data is collected. Furthermore all persons and data are treated equally by the installed policy - the policy does not enforce individual protection preferences.

   A common privacy policy for all customers does not allow to take individual preferences and protection requirements into considerations. It also raises questions for the person that entrusts his or her personal data: What happens if the privacy policy is changed once I have submitted my data? Do I have to accept different protection policies and protection standards for my personal data when I need to use different services from different organisations?

   Nevertheless, it is a convenient and well established practice to install a common privacy policy for personal data and leave the authority over this policy by the receiver of the data. Many privacy policies by large companies such as Google[3] and Amazon[4] implement their privacy policies in this way. This practice is so common that it becomes difficult to envision a different procedure for the creation of privacy policies.

Existing implemented privacy protection mechanisms seem to be not fully effective in the safeguarding of private data, as accidental data spills also affect companies and organisations with installed privacy policies.

These questions and observed deficits in existing privacy protection approaches were the motivation to research new and alternative ways for the specification and automatised enforcement of personal privacy preferences. In this thesis we analyse existing methods and assumptions for privacy protection and develop an integrated privacy protection mechanism that enforces individual privacy preferences. It is our goal to deliver a better level of protection for sensitive private data than existing policy-based mechanisms.

In order to aid the discussion and analysis of existing and new solutions for privacy protection we will use the next section to define some key terms and principles of protection properties typically found in such systems.

1.1 Defining Private Data Releases

Personal private data differs from other forms of sensitive and valuable data, because private data is inherently connected to a particular human being and might serve to identify and qualify the originator of the data. Such data is commonly described as *Personal Information*, or more precisely as *Personally Identifiable Information* (PII) and can be defined as follows:

[3]http://www.google.com/policies/privacy/

[4]http://www.amazon.com/gp/help/customer/display.html?nodeId=468496

Definition 1: Personally Identifiable Information (PII)

Information about a particular person, especially information of an intimate or critical nature, that could cause harm or pain to that person if disclosed to unauthorised parties.

Examples include medical record, arrest record, credit report, academic transcript, training report, job application, credit card number, Social Security number. [Shirey, 2007]

The definition given by Shirey is unnecessarily strict, because not only data that causes harm or pain constitutes as PII, but rather any identifying data. The special nature of this data and its protection requirements has been recognised very early during the development of electronic communication media and manifests itself in a number of legal requirements for the handling and safekeeping of private personal data, such as the Universal Declaration of Human Rights from 1948, where Article 12 states:

> No-one should be subjected to arbitrary interference with his privacy, family, home or correspondence, nor to attacks on his honour or reputation. Everyone has the right to the protection of the law against such interferences or attacks. [UDHR, 1948]

While it seems that the best protection for PII would be to keep personal data strictly confidential, there are cases when such data needs to be collected, handled and exchanged with third parties, such as government agencies, service providers, employers and others. This does not mean that this data now becomes publicly known, but rather, that a certain person or organisation is trusted to receive and use PII in accordance with agreed principles for the safekeeping and protection of the data.

Definition 2: Data Release

We define a *data release* as the restricted distribution of personal data. A data release differs from a publication, in that there exists a controlled set of trusted recipients and constrains (implicitly or explicitly defined) guard the release of the data.

The following definitions name and identify the principals that are involved in these data release cases:

Definition 3: Data Owner

The *data owner* is the subject of personally-identifiable data and usually has an interest to control the release and use of this data.

The term *owner* was chosen to reflect the fact that, even when usage is granted to third parties, this data contains personal facts that are legally still under the control of the person that is identified by this data. The literature here also uses the term *data subject*.

Definition 4: Data User

The *data user* is the subject that accesses and uses personal data of the data owner. It should honour the privacy requirements set by the data owner and follow its own stated privacy policies. In dealing with organisations the term *data user* usually represents a collection of users inside this organisation or the organisation itself.

Sometimes the data user is also called *custodian* - this would be the entity to which personally-identifiable information is entrusted. During a *data release*, private data passes to a data user that is trusted by the data owner. Whenever private data is passed to untrusted parties or made public against the interest or desire of the data owner the same act is called a *data breach*.

Definition 5: Data Breach

The act of distributing personal data beyond the set of trusted data users. A data breach can be the result of a malicious act of information access, a disregard of the defined usage constrains or simple neglect by the custodian of the data that makes PII available to untrusted data users.

We already mentioned that a data release is usually guarded by constrains on the usage and further distribution of PII. These constrains do not necessarily have to be specifically defined and many social contexts define implicit rules and conventions. If we want to respect these constraints in an automatic enforcement scheme, these rules and constraints need to be made explicit.

Definition 6: Privacy Policy

The *privacy policy* defines the constraints that guard the use of personally-identifiable data. Potential data uses might be the accessing, storing, processing and forwarding of data. A privacy policy could be set by the data owner, but most commonly it is defined by the data user and authorised by the data owner.
A privacy policy can be explicitly or implicitly defined and should be enforced by relevant security mechanisms.

The privacy policy defines the rights and permissions that the data owner grants the data user and is an integral part in any privacy enforcement approach.

1.2 Data Release Taxonomy

Data releases can be separated in cases where data is released *actively* by the data owner, such as filling in forms on web-sites and enabling access to private documents and *passive* data release, where the private data is released as by-product of other activities (see Figure 1.2).

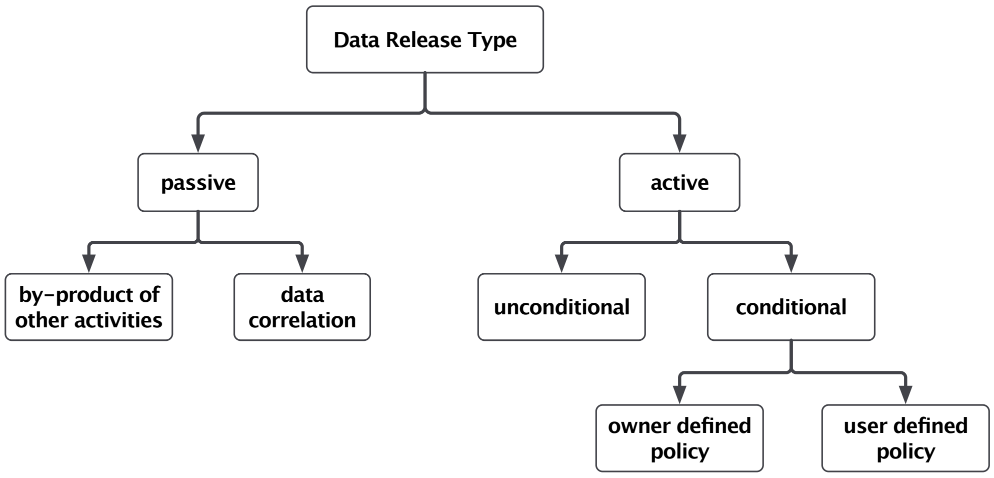


Figure 1.2: Data Release Taxonomy - data release cases can be broadly classified into active and passive release cases, as seen from the perspective of the data owner. The focus of this work lies with conditional release cases under an owner-defined policy

Passive data releases occur when personal data is gathered, collected and used without the active involvement of the data owner. These types of data release are called *passive* because the data release is not the main intent of the data owner, but is happening while pursuing other activities. We distinguish between:

- **Data release as by-product,** where personal private data is released as a side-effect of other activities such as Web-browsing, usage of personalised services, Web-searches, the use of loyalty cards, Geo-location requests, etc. Often there is no malicious intent behind the collection of this data. It is just common that web-server generate log-files and network access for most networks is closely monitored.
- **Data correlation** occurs when private data is further analysed and correlated with other events, as well as previously released or publicly available data in order to gain enhanced information about the data owner.

Active data releases take place when people release data knowingly and intentionally to a data user such as a service provider. It is assumed that the data owner is consciously initiating this data release. The data owner trusts the data user to receive his or her private data and use the data sensibly.
Example activities include the posting CVs on online job sites, the sending of confidential email messages and the presentation of medical examination records.

The release of private data can be further divided into release scenarios where private data is released conditionally or unconditionally.

- **Unconditional release** - Data is released unconditionally by the data owner if he or she does not care about special privacy protection or no adequate privacy protection mechanism is available and the releasing partner is forced to release the data anyway. There still exists a basic form of privacy protection from the legal framework that has been put in place by different countries.

Conditional release - In the conditional case the data owner releases private data under a specific privacy policy that is known to the data user. The data user can employ available privacy protection measures for the safeguard of the data, such as protection against unintended release.

We can distinguish two sub-cases, depending on the origin of the privacy policy:

User defined policy - In most current systems the policy is set by the data user. This makes it easy to tailor the policy to the privacy enforcement mechanisms of the data user. The data owner usually has very limited means to influence such a policy through the expression of choices and can usually only approve or disprove the policy.

Owner defined policy - In cases where the data owner has very specific privacy requirements he or she might want to implement their own privacy policy to which the data user should be bound. The data user must be willing to accept this policy and needs to follow and enforce it. The challenge lies in the fact that the data user needs to maintain a strong link between the policy and the protected data. Otherwise no adequate protection could be guaranteed.

1.3 Motivation

The work in this thesis has been motivated by the desire to enable privacy-aware data sharing between communication partners that generally trust each other, but might have different ideas about protection requirements for shared private data.

Most existing privacy protection schemes let the data user define the privacy policy and give the data owner little choice, to either accept this policy or refrain from data sharing. We want to empower the data owner by providing him or her with the ability to express personal privacy preferences that are closely bound to the data and can be enforced at every data access location. It was our desire to build a policy protection framework that offers strict enforcement for data owner-defined privacy policies.

Our study of the existing literature and solutions (see Chapter 2: *Background and State-of-the-Art*) showed that different building blocks for such a system existed, however, an integrated approach was missing. For example, the UCON_{ABC} usage control model defines a suitable system architecture, but leaves the important area of policy administration and management undefined (cf. Section 2.3.4).

Without an integrated privacy protection scheme it becomes difficult for the data user to honour stated privacy policies during normal data processing. This, in turn, leads to privacy breaches that are not intentional and violate stated policies.

It was our goal to make the privacy policy automatically enforceable, so that no conscious effort from the data user was required to comply with the policy. There exist several challenges to reach that goal. We needed to find a mechanism for the expression of policies that is specific enough to be automatically enforceable, offers consistent policy evaluation and can be safely used by non-experts to formulate adequate privacy statements. Other challenges lie in the design of the enforcement system itself.

We needed to find out if it is possible to re-use and adapt existing access-control and rights-enforcement schemes for the protection of personal private data or if a complete new enforcement approach is needed. If at all possible, we favoured the usage and adaptation of proven mechanisms because this approach reduces development time and minimises security issues that arise from completely new and untested designs.

We like to stress that our data release model assumes that a data user is a legitimate receiver of private data. Each data user under this model thus represents a *trusted data user* that has been granted the right to use private personal data for a certain purpose. We further assume that any data use outside of this grant is regarded a data breach and must be prevented. Our privacy enforcement framework should have the following characteristics:

Owner-controlled privacy policies Starting with the ruling of the Federal Constitutional Court of Germany in 1984 [BVerfG 65,1, 1984], it is now the general consensus in Germany that the data owner has the right to control data release. This right includes the disclosure and usage of data and is usually called the right for *informational self-determination*.

The data owner must be able to specify and revoke the necessary authorisations for data access and use. The authorisations reflect the trust relationship between the data owner and the data user, e.g. patient and general practitioner, and give greater power to the data user.

Consistent policy evaluation The private data will be presented to different parties and the privacy policy for the data must be evaluated consistently across the different parties. This means that access control decisions derived from the privacy policy must be deterministic and should capture the intentions of the data owner, regardless of the specific operations of the different data users.

Guaranteed policy enforcement The privacy policies for data repositories must be directly enforceable in order to restrict data access and provide data flow control. Policies must be enforceable equally against all involved parties. A distributed enforcement architecture might be needed to implement this requirement.

We evaluated three distinct usage scenarios that implement different typical workflow scenarios concerning personal private data and that enabled us to develop and validate our design. We started to study the development of a client-side enforcement mechanism for the protection of Personal Health Record (PHR) and later extended this work to also offer server-side privacy protection for the processing of location data. The differences between client-side and server-side enforcement are discussed next, based on the requirements of their individual use cases.

1.3.1 Client-Side Mechanisms: Privacy Protection at the Network Edge

Data stored in medical health records belongs to the most privacy sensitive types of data. Historically, health records have been created, stored and accessed locally by practitioners or hospital staff. Data access was restricted by the fact that patient records were only locally available and accessible only by authorised personnel. With the ongoing deployment of IT-centric solutions and workflows, there is now a strong move to store medical data as a *Personal Health Record* (PHR), an electronic repository, that allows collaborative access to vital patient information.

When health and treatment data will be stored in such a repository, a similar level of separation between the different data sources, as offered by the current paper-based solutions, needs to be maintained. It is therefore necessary that a privacy solution for PHRs supports fine grained data access control below the document level in order to provide adequate views on the data. Furthermore, the patient expects the execution of the same privacy policy, independently of diverging practices at individual practitioners or health care organisations.

Modern patient health cards have the ability to store personal health record data, so that it can be accessed and analysed by different practitioners participating in the treatment process and act as a repository for future diagnosis[5]. A person could choose to store their PHR on a patient smart card to have important health record data readily available and facilitate data sharing between different practitioners and episodes of illness. It is easy to understand that in this usage scenario the data owner (the patient) has a direct interest to make highly sensitive private data available to a treating medical person. However, it is equally plain to understand that data access and data sharing needs to be controllable by the patient in order to communicate and enforce legitimate privacy requirements. These privacy requirements may differ during periods of illness and health and the level of trust awarded towards a visited primary care practitioner on a vacation trip might be different than the trust in the long-time family doctor at home, so any privacy protection system needs to reflect these changes in trust and dependency.

This use case requires decentralised, data owner-based privacy policy management, especially when the data needs to be accessed and used by many different principals and organisations in a distributed manner. Data access should be limited on a *need-to-know* basis – it is usually not necessary for a visited practitioner to have access to the complete medical history of the patient.

We developed a privacy enforcement scheme and a corresponding reference monitor implementation based on the Java Security Framework [Gong, Mueller, Prafullchandra et al., 1997]. The approach uses so called *sticky policies* [Mont, Pearson, and Bramhall, 2003] written in the eXtensible Access Control Markup Language (XACML) [XACML-2.0, 2005] to formulate access rules for protected resources. The policies are *data owner-defined privacy policies* because they represent the privacy preferences of the data owner. They are called sticky, because they stay attached to the protected resource and are distributed alongside it to the data user, where they will be referenced and enforced.

1.3.2 Server-Side Mechanisms: Privacy Protection by Service Providers

Location data is another type of privacy sensitive data that is used within business and private applications and has seen phenomenal growth rates [Manyika, Chui, Brown et al., 2011]. The continuously growing smartphone market, as well as the emerging 'Internet of Things' enable service providers to very accurately track and map persons and their devices through a variety of different localisation mechanisms. The unbroken interest in social networks entices users to make their private data available to various services and it is not uncommon that devices and applications

[5]Note: The currently proposed German health card 'Gesundheitskarte' [GKV 2003, 2003] stores the personal health record data on a server infrastructure and only links to it via the patient smart card.
Similarly is the proposed Google Health service [Google Health, 2008] a purely server-based solutions that requires online access for the retrieval of patient records. Server-based solutions can offer a centralised access control service but fail to provide data-flow and usage control for data that has been released to distributed clients (decentralised policy enforcement).

indiscriminately collect location data [Allan and Warden, 2011].

Most location services are server-based solutions in which a user sends location information to a service provider, where it is processed and correlated with other available data. Location data might also be forwarded to third parties for further processing and storage. A user cannot be sure that collected data is only used for the fulfilment of a particular service and that no data is used for further purposes like market analysis, targeted advertising or others.

In order to remedy this situation, it has been our approach to extend our work on data owner-defined privacy policies to server-based solutions that can be automatically enforced by the requested service. We re-use our sticky policy approach to communicate privacy policies for location data to the service provider. Policies and data will be stored and processed together as a single data object.

We developed two different enforcement solutions based on two independent usage scenarios. Our first use case is a privacy-aware localisation service which has been developed to fulfil the privacy requirements of an assistance system for elderly people. The use of information technology to enhance the safety, well-being and independence of elderly patients under care is known as *Ambient Assisted Living* (AAL). A recent study by Schneider and Häusler [2011] with 33 participants highlighted every-day situations, in which participants could imagine or would wish for assistance. However, the study also showed that the interviewed participants were afraid of being monitored.

A research team at Potsdam University has developed the KopAL mobile orientation system [Fudickar and Schnor, 2009; Fudickar, Schnor, Felber et al., 2011], which is aimed to support elderly patients and patients suffering from mild forms of dementia in their daily activities. The patient is assisted by a mobile device, the *KopAL Assistant*, which provides a localisation function, for cases where patients are in distress and can not identify their current location, or where patients suffering from dementia simply walk away from the nursing home.

When we tried to adapt our existing client-side enforcement solution to this server-based scenario, we found that direct privacy policy enforcement through the Java Security Framework has some serious limitations. We needed to develop another enforcement mechanism that allowed us to execute policy decisions within existing application frameworks.

The *Aspect-oriented Programming* (AOP) paradigm and its ability to intercept method-calls led us to the idea of intercepting and monitor data access to protected resources within existing applications. We built a hypothetical 'theme-park location service' that allowed us to highlight problem areas and develop concrete solutions for privacy enforcement in server applications. AOP was used to design a privacy architecture that combines the flexibility of the AOP method interception with a generic policy evaluation component and lead to the construction of an AOP-based reference monitor. This approach makes it possible that different people can have their individual privacy preferences enforced, while still being able to use a common service.

1.4 Summary and Thesis Overview

This thesis researches active data release cases under a data owner-defined policy. It is our goal to control the sharing of personally identifiable data and protect data from unintended passive release at the site of the data user. Electronic data sharing and processing has become the norm and is also found in very private areas that involve a high level of personal trust. A number of well documented high profile privacy breaches in recent years have shown that there is still a mismatch between privacy requirements and adequate protection technologies, especially for cases that require the active release of private data. Our analysis of these privacy breaches has shown that for a number of cases it was not malicious intend that lead to the release of private data, but rather a mixture of carelessness, lack of communication and the not existing enforcement of stated privacy policies.

Human-readable privacy policies, which are defined by the data user, make protection statements that are not strongly tied with actual privacy enforcement practices. It therefore happens that the stated privacy policy is actively or passively ignored and private data is used for other purposes than originally intended and announced by the data user. Human readable privacy policies are expressing a promise to the data owner that is hard to validate by the data owner, but also hard to keep for the data user, because the published privacy policy has to be reliably translated into enforceable rules for the access and usage control system of the data user.

We therefore propose an integrated approach, where enforceable privacy policies are strongly tied to the private data and where a privacy protection mechanism takes care of the automatic enforcement of the policy. Keeping privacy policies and data together as a single data object greatly improves flexibility of data storage and processing. We further challenge the existing practice that privacy policies are defined by the data user. We want to empower the data owner to state their own privacy requirements and have them automatically enforced by the access control systems of the data user. We propose to use a generic policy language that can be coherently evaluated by all parties and which allows us to form access decisions independently from implementation aspects.

Policy administration and management by non-experts is also a challenge that needed to be solved within our system. We therefore propose the use of a privacy policy scheme that uses separate sub-policies with a clearly defined *Privacy Policy Precedence Relation* (P3R) to protect the data owner from the formulation of harmful policies and provide basic protection in standard cases, where the data owner has not specified an explicit policy.

The results of our work have convinced us that our privacy enforcement framework offers better privacy protection than similar existing approaches. However, the following limitations exist and must be taken into consideration when personal privacy data is released within the system:

Data flow protection - The distributed enforcement of access restrictions requires the implementation of effective data flow protection. Current computer operating systems are not well suited to automatically enforce such restrictions and general limits exist. For example, if data is displayed on a computer screen, the screen content can be photographed and displayed data can be memorised or noted down.

Trust in enforcement mechanisms - The data owner has to trust the implementation of the enforcement function to protect data from unauthorised access. Ascertaining the correct functioning of the enforcement mechanism could be performed by independent audit, however this would only be feasible for large organisational installations.

Policy referencing - Policies and data have to be stored and referenced simultaneously by the privacy enforcement scheme and have to be equally protected. Privacy protection breaks down, if the policy can be altered or the link between privacy policy and data can get separated.

1.4.1 Thesis Structure

The remainder of the thesis is structured as follows:

Chapter 2: *Background and State-of-the-Art* surveys the different protection technologies and access control models that are relevant for the protection of private electronic data. This chapter provides the scientific and technological base on which we develop our privacy enforcement architecture.

Chapter 3: *Owner-Retained Access Control Policies* discusses policy design and usability issues that are particularly important when privacy policies are developed and maintained by the data owner. We develop a prioritised policy model based on a Privacy Policy Precedence Relation that divides a policy into separate sub-policies which can provide default protection and safety rules for data owner-defined policies.

Chapter 4: *Privacy Enforcement Framework* defines and describes the elements of our enforcement framework for the specification and enforcement of owner-controlled privacy policies. It develops the architectural building blocks and explains which properties enable the controlled exchange of private data under a data owner-defined policy.

Chapter 5: *Java PrivMon - Privacy Protection for Personal Health Records* explains how the Java Security Architecture can be adapted so that it provides client-side policy enforcement and data-flow protection within our privacy enforcement framework. Personal Health Records (PHRs) are used as an example for highly sensitive personal data that is distributed and used within information systems.

Chapter 6: *Privacy Protection for Server-based Information Systems* extends the work from the previous chapter and develops enforcement solutions for situations, where owner-defined privacy policies need to be enforced in server-based scenarios. We develop the concept of a privacy-aware localisation server and implement two different solutions that make use of this concept. The first is the localisation component of the KopAL ambient assisted living system, the other solution uses aspect-oriented programming (AOP) techniques for the privacy enforcement in tiered business applications.

Chapter 7: *Conclusion and Future Work* provides a discussion of the key results and remaining issues that can serve as the base for further work. It summarises and concludes the findings of this thesis.

1.4.2 Preliminary Presentations and Publications

Important key elements of this thesis have been presented at international conferences:

- A protection model for health records using semi-structured XML-data was presented and discussed at *HEALTHINF 2008*, Funchal, Portugal [Apitzsch, Liske, Scheffler et al., 2008].
- The implementation of client-side privacy protection mechanism using a modified Java class-loading mechanism was presented at the *23rd International Information Security Conference (SEC 2008)*, Milano, Italy [Scheffler, Geiß, and Schnor, 2008].

- A secondary use-case that discusses privacy protection for location data in an Ambient Assistant Living scenario was presented at the *3rd Workshop on "Privacy and Security in Pervasive Environments"* at the PETRA2011, Crete, Greece [Scheffler, Schindler, Lewerenz et al., 2011].
- We finally presented a server-based enforcement solution using Aspect-Oriented Programming for the enforcement of privacy policies in an tiered business application at the World Congress on Internet Security 2012 in Guelph, Canada [Scheffler, Schindler, and Schnor, 2012].

A working prototype of the privacy protection system for Personal Health Records was presented at *CeBIT 2008* in Hannover, Germany, at the 3rd *Telematik-Konferenz 2009* in Potsdam, Germany and *conhIT 2009* in Berlin, Germany.

2 Background and State-of-the-Art

Protection of information through the controlling of data access is not an altogether new paradigm. Research into access control mechanisms reaches back into the very first commercial and governmental uses of computer systems. Different protection schemes have been proposed and implemented in the past that aid in the task of protecting access to electronic computer systems and safeguard the sensitive data stored on these systems.

This chapter surveys different paths that have been taken in the protection of electronic data and system security. It starts with an examination of privacy threats and gives an overview over established Privacy Enhancing Technologies. We further examine data protection approaches, as well as access control and information security models that can be used for the reliable definition and enforcement of privacy protection. The chapter than provides a background on related concepts such as authorisation, access-control frameworks and policy languages. It identifies building blocks and potential problem areas for the implementation of data owner-defined privacy policies.

2.1 Defining Data Privacy

The need for personal privacy has its roots in the concept of individual liberty as a political and philosophical concept and as such is not a new phenomenon. Controversies about data privacy even predate the computer age, as can be seen in the famous article by Warren and Brandeis [December 15, 1890] that was influenced by the new technical abilities to record and reproduce pictures and sound. A good overview of the different aspects of privacy and computer technology can be found in Agre and Rotenberg [1997].

One of the guiding principles for privacy protection and the development of privacy enabling technologies has been in the past the principle of *data sparseness* and *limitation of data collection*, targeted at the preventing of a *Big Brother scenario* [Fair Information Principles, 1973; OECD, 1980].

During the 1970s and 1980s, when these privacy principles were developed, the computerisation of households and private lives had not yet begun and electronic data collection and processing by large corporations or governmental agencies were seen as the main privacy threat. The reasoning behind these recommendations was the following: if data is not collected electronically, it can not be abused and collecting and storing less information minimises the potential for misuse.

This primary assumption, while still valid, can no longer guide the development of new privacy protection systems. With the current integration of electronic communication into almost every social and economic domain it is no longer sensible to focus on the data sparseness principle for the protection of private data. There exists now a wide range of social interactions conducted via electronic media that involve the direct exchange of personal data, such as personal electronic mail, processing of tax refunds, personal electronic health records, geo-location services and the like, that can only function if people actively make personal private data available to service providers and people sharing their personal network. Governments issue electronic ID cards[1] and e-Government initiatives are launched worldwide to offer Internet-based access to citizen data.

The focus for privacy protection has shifted from anonymous service-use to the provisioning of privacy policies and trust building. The current guiding principles for the collection and use of personal data under a privacy policy are *Notice* and *Consent*. The data owner must be given the opportunity to agree to a published privacy policy before data can be collected and used. Policy-based privacy protection schemes have been developed to reflect this practice. Well known examples are the Platform for Privacy Preferences [P3P v1.0, 2002; P3P v1.1, 2006] and the Enterprise Privacy Authorization Language [EPAL 1.2, 2003] that will be discussed later in this chapter. Service provider can publish their privacy policy and give the data owner the ability to evaluate this policy before agreeing to release private data.

The acceptance of the reality of electronic processing of personal private data by trusted service providers does not mean that the need for data privacy has vanished. However, the focus for data privacy protection has moved from data avoidance to a type of data release that provides the data owner with better control over the release and usage of personal private data. Several definitions of data privacy focus on the aspect of control over the data release:

Definition 7: Data Privacy

(i) To control who has access to information about the individual, to be able to estimate what is known about the individual by others. [Rössler, 2001]

(ii) Privacy is the right of individuals to determine for themselves when, how and to what extent information about them is communicated to others. [Ashley and Karjoth, 2003]

2.2 Privacy Enhancing Technologies

There exist a number of technical protection mechanisms that were specifically developed to protect and control the dissemination of personal information. Such mechanisms are generally known as *Privacy Enhancing Technologies* or PET.

Development and application of Privacy Enhancing Technologies is extensively discussed by different authors (e.g. Tavani [1999], Fischer-Hübner [2001]). The following sections provide an overview of existing technologies. The properties of the techniques are briefly discussed in respect to the requirements of our proposed protection system.

Not all the mentioned techniques described here implement software mechanisms. A number of proposed PET rely also on external mechanisms, such as legislative frameworks and reputation mechanisms that correspond to implemented protection schemes, but are also useful without them. We will include these techniques in our discussion because they often have had a formative influence on research and development of protection techniques, as well as the general discussion about the problem space.

[1]The German government has mandated the use of patient smart cards for general health care in 2003 [GKV 2003, 2003]

2.2.1 Data Collection Principles

Early discussion about protection of electronically stored data ensued from centralised data collections of censuses and other governmental bodies. The debate started with the goal to establish privacy guidelines that allowed the beneficial use of electronic data under a framework of principles that guard the privacy of the data.

In 1973 a task force of the U.S. Department of Health Education and Welfare formulated the *Fair Information Practice Principles* that developed several now well-accepted principles concerning the sharing and collecting of personal information, such as *Collection Limitation* and prevention of *Secondary Usage* [Fair Information Principles, 1973]. Other countries enacted similar data protection laws. In 1980 these principles were incorporated into an international privacy code by the Organization of Economic Cooperation and Development: the *OECD Guidelines on the Protection of Privacy* [OECD, 1980].

The OECD guidelines have been influential for the development of data protection techniques and it can be shown that many protection goals for *Privacy Enhancing Technologies* can be linked back to one or more principles from the guidelines. From the standpoint of data protection the following principles are especially important:

1. **Collection Limitation Principle** There should be limits to the collection of personal data and any such data should be obtained by lawful and fair means and, where appropriate, with the knowledge or consent of the data subject.

3. **Purpose Specification Principle** The purposes for which personal data are collected should be specified not later than at the time of data collection and the subsequent use limited to the fulfilment of those purposes or such others as are not incompatible with those purposes and as are specified on each occasion of change of purpose.

4. **Use Limitation Principle** Personal data should not be disclosed, made available or otherwise used for purposes other than those specified in accordance with Paragraph 3 except:

   (a) with the consent of the data subject; or
   (b) by the authority of law.

5. **Security Safeguards Principle** Personal data should be protected by reasonable security safeguards against such risks as loss or unauthorised access, destruction, use, modification or disclosure of data.

[OECD, 1980] (See Appendix *OECD Privacy Guidelines* for an overview over all principles in the guideline.)

The remaining principles *Data Quality*, *Individual Participation*, *Openness* and *Accountability* are not directly related to the topic of data protection and will therefore not be researched in this thesis.

2.2.2 Anonymisation and Pseudonymisation Techniques

Data privacy can be strengthened substantially by following a simple observation: Private data that has never been disclosed can not be compromised and used against the intentions of the data owner.

This idea has found its way into the *Fair Information Practice Principles* that states several well-accepted principles concerning the sharing and collecting of personal information, such as *Collection Limitation* and *Purpose Specification* [Fair Information Principles, 1973], [OECD, 1980] and was implemented in privacy law [Directive 95/46/EC, 1995]. Early research into privacy enhancing technologies concentrated on the following reasoning:

> If an activity can not be traced back to an individual - no personal data about the individual is revealed and privacy is not breached.

It is therefore still possible to collect and use Personally Identifiable Information (PII), if the link between originating individual and the data can be broken. Techniques that hide the user identity against service providers and snoopers can be implemented and preserve privacy (cf. Chaum [1981]). This observation has lead to the development of anonymisation and pseudonymisation tools such as anonymous remailers[2], mix networks[3] and onion-routers[4] that protect the true identity of the data owner from being disclosed to service providers and external attackers.

Anonymisation

Anonymisation is a technique that hides the true identity of the data owner. Any information that the data user might gain must be void of directly identifiable data. Privacy protection is reached through the fact that any data that is gathered can not be attributed to a particular individual and therefore the privacy principle of *Collection Limitation* is observed.

Anonymous use of services is widely accepted in real world scenarios, where it is normal to pay for goods and services in cash and to make anonymous phone calls through a public payphone. There exist different applications that might be used to send anonymous E-mail or anonymously browse the information content of web-servers. Goldberg, Wagner, and Brewer [1997] and Goldberg [2003] give a good overview of different anonymisation techniques, their different development phases and potential problem areas for such services.

The user of anonymisation services can hide its identity against the service provider and any potential eavesdropper on the communication link. It should also not be possible to link individual communication acts together. Anonymity is lost, when through the course of the transaction the communicating parties need to reveal their identity. Anonymisation provides no protection if the transmitted information itself is of identifying nature or if enough communication acts can be linked together to form an identifying profile that can later be used to identify the individual.

Effective anonymisation can be surprisingly difficult to achieve and must also look at recurring and identifying data patterns such as IP-addresses and geographic location information, to eliminate the risk of re-identification.

Pseudonymisation

Pseudonymisation techniques aim to provide the same level of identity protection as anonymisation techniques while addressing some of the weaknesses of anonymisation. The complete hiding or randomisation of user identity through anonymisation makes it easy to abuse or vandalise these

[2]http://mixminion.net/
[3]http://www.jondos.de/
[4]http://www.torproject.org/

services. Since the users can not be re-identified they run little risk to be caught in unfair or criminal behaviour. Several of the early anonymous E-mail servers (remailers) had to shut down because of Spam abuse.

The use of digital pseudonyms has been first suggested by Chaum [1981,1985] to provide unlinkability between different actions of the individual in different contexts while maintaining the ability to authenticate the individual and attribute their actions to them. The users are issued with one ore more persistent identities, called *nym*, that can not be directly linked back to their true identity. Under a pseudonym an individual is able to exert control over the release of information about themselves. Individual actions under different pseudonyms can not be linked, however, the provider of the pseudonymisation service has the knowledge to re-map the identity with a particular pseudonym.

Pseudonym systems overcome the weaknesses of anonymous systems, where system abuses can not be detected and attributed to the individual. Pseudonyms are most useful in situations where authentication is required, but not identification. Because the provider of the pseudonymous system can attest the true identity of a user, there can be extended trust between the service provider and the pseudonymous user. If clear identification is necessary, such as in the case where a postal address is needed to deliver goods, pseudonyms can only work if there is an independent clearinghouse in operation which re-translates pseudonyms into postal addresses.

The use of pseudonymous services requires a strict management of *nyms*. Otherwise there exists the danger that through the prolonged use of a pseudonym the user can be re-identified. Pseudonyms must be periodically changed and the usage scope for individual pseudonyms must be limited (different *nyms* for different services). The prime project [PRIME, 2006-2008] has developed some solutions for privacy enabled identity management.

Conclusion

Anonymisation and pseudonymisation technologies share a common attacker model. Their threat model assumes that the service provider itself is an untrusted recipient and should only be able to see anonymised or pseudonymised user data.

Anonymisation and pseudonymisation systems are generally not suited for the exchange and protection of sensitive, identifiable data, such as health care or financial transaction data. The privacy protection offered by these systems breaks down when the data owners need to reveal their true identity and other identifying data to the service provider.

2.2.3 Privacy Recognition Schemes

Privacy Recognition Schemes such as eTrust [Benassi, 1999] are an initiative from the industry to enhance user trust through the creation of privacy standards, independent control and attestation of adherence to these standards. Trust building is usually done through the display of a recognisable privacy seal on websites. These initiatives collect feedback from users and conduct audits to ensure that privacy standards are met.

Participation in these initiatives is voluntary for an organisation. Most of these schemes are targeted at larger organisation and are not suitable for small businesses and private data exchanges. Recognition schemes do not offer any privacy protection themselves. Their main focus is the independent verification of privacy protection measures and thus supports the *Openness* principle of the OECD guidelines.

2.2.4 Data Encryption

Encrypting the communication channel or encrypting the private data itself is standard practise for information security on the Internet today. A wide range of different encryption technologies, such as IPsec [Kent and Seo, 2005], Transport Layer Security [Dierks and Rescorla, 2008] and secure E-mail encryption such as OpenPGP [Callas, Donnerhacke, Finney et al., 2007] and S/MIME [Ramsdell and Turner, 2010] exist and are in active use.

Encryption offers privacy protection to the extent that an eavesdropper can not gain knowledge about the transmitted encrypted private information. Effective data encryption can therefore realise the implementation of the *Security Safeguard* principle of the OECD guideline.

Privacy protection requires protection at the time of data transport and during storage of the data. Many data encryption standards only specify a secure communication channel because data transport over a network is seen as the main security threat. Data that has passed through a protected communication channel is secure, but is usually decrypted as it leaves the secure channel and is stored for further use.

In order to preserve data privacy it is necessary to protect stored data. However, even if we assume that data is stored in encrypted form, it needs to be decrypted, at least temporarily, for further processing. Encryption schemes for E-mail or file systems usually have a single encryption key for the file system or E-mail message-base of a user and do not provide fine grained control over access to individual private files. Any data encryption scheme that is used for privacy protection must therefore be augmented to derive the authorisation for the decrypting operation directly from the corresponding privacy policy and must provide fine grained access to the data.

2.2.5 Privacy Policy Schemes

The ability to communicate privacy requirements and formulate a meaningful privacy policy is an important prerequisite for the protection of data privacy. The *Openness* and *Purpose Specification* principles of the OECD guidelines require the information of the data owner about the purpose of data collection and the data privacy practices of the data user. Privacy policies are an ideal tool for this purpose and can be specified in different forms:

Human Language Policies Privacy policies can be stated as human readable text on websites, where they describe the privacy practices of the targeted organisation. Their character is mainly informational and can have a collection limiting effect if the data owner evaluates and understands the policy prior to data release and refrains from using services with questionable privacy policies.

Human readable privacy policies have several weaknesses. First, they must be translated into security measures and authorisation rules by the data user in order to offer any enforceable protection. Jensen and Potts [2004] showed in their study that many websites now display privacy policies, however, it is still very difficult for the data owner to derive meaningful value from this practice. The stated policies are difficult to understand and compare because they lack standardisation. Furthermore, there exists a mismatch between the information needs of the data owner and the policy data provided by the data user, which limits the usefulness of such policies.

Privacy Policy Languages are formal languages that are specifically designed to facilitate the expression of privacy policies, practices and requirements. Privacy Policy Languages have the potential for automatic policy enforcement of the *Use Limitation* and *Collection Limitation* principle, if it becomes possible to derive access control decisions directly from the formal policy description. Privacy Policy Languages also offer the potential to ease privacy decisions for the data owner, because they also offer greater standardisation and automatic policy evaluation, something which is not possible with policies expressed in human language.

Specific Privacy Policy Languages will be discussed in greater detail in their own sections because they are highly important for the protection of privacy in cases where personal private data is actively released by the data owner. The following languages will be evaluated in Section 2.6:

- Platform for Privacy Preferences (P3P)
- Enterprise Privacy Authorization Language (EPAL)
- eXtensible Access Control Markup Language (XACML)
- GEOPRIV Common Policy

2.2.6 Summary

The previous section has shown that existing Privacy Enhancing Technology implement several principles from the OECD Privacy Guidelines and can be helpful tools for privacy protection. However, the section has also shown that many existing techniques focus on the principle of data sparseness and identity hiding that offer no protection in cases, where private personal data needs to be distributed. Table 2.1 shows an implementation summary for the main principles from the OECD Privacy Guidelines.

Table 2.1: Privacy enhancing technologies implementing the principles from the OECD Privacy Guidelines

	Collection Limitation	Purpose Specification	Use Limitation	Security Safeguard	Openness Principle
Encryption	(x) against outside attacker			x	
Anonymisation/ Pseudonymisation	x			(x) owner not identifiable	
Recognition Schemes					x
Privacy Policies	(x) release limiting	x	(x) if enforceable		x

Privacy policies, in the form of policy languages, offer the widest support for the principles from the OECD privacy guidelines. Directly enforceable privacy policy schemes derive access decisions from the formalised privacy policy. When they are bundled with adequate security mechanisms, they can enforce the execution of the privacy policy on protected data and provide protection even in cases, where data needs to be actively released by the data owner.

The next section will discuss some general aspects of *access control* and *access control languages* before different *privacy policy languages* and their properties are analysed in detail.

2.3 Access Control Models

Access control can be described as the prevention of unauthorised use of a resource, through the process of controlling access to resources within a system [ISO 7498-2, 1991]. The entities that perform the access activities in the system are called *Subjects*, while the resources that are accessed are called *Objects*. The controlled activities are called *Actions*. An access control scheme determines if an access request is granted on the base of *Authorisations*. Access is allowed to subjects that have permission to use the resource and denied otherwise. The access control decision is enforced by a security mechanism that mediates access to the resources in the system. A comprehensive overview of existing mechanisms and models can be found in Samarati and di Vimercati [2001]. A selection of the most important mechanisms will be discussed in the following.

2.3.1 Mandatory Access Control

Mandatory Access Control (MAC) systems enforce a central access control policy equally for all subjects in the system. The system security policy is controlled by a security policy administrator and individual users have no ability to override this policy.

> To support a MAC policy, all subjects are assigned security attributes, as are all objects which are controlled by the policy. Every security policy decision is a function of the attributes of the subjects and objects involved in a particular attempted access. The definition of these policy functions and the assignment of security attributes are defined into the system, or when configurable, are tightly controlled by a security administrator. An ordinary user may be allowed to make changes, but only if they further restrict the policy. [DTOS, 1997]

Multilevel Security Systems are security systems where some information in the system has a higher classification than certain users of the system and are prominent examples of the Mandatory Access Control paradigm.

> ”Mandatory Access Control: A means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorisation (i.e., clearance) of subjects to access information of such sensitivity.” [DoD 5200.28-STD, 1985]

A well known example is the Bell-LaPadula model for the control of information flow [Bell and LaPadula, 1973], [Landwehr, 1981]. Labels are used to classify system users according to their security clearance, objects are assigned a sensitivity level. Two principles guarantee the security of the system:

a) The *simple security property* states that no process may read data at a higher level
b) The **-property* allows no process to write data at a lower level

These properties ensure that sensitive data does not get accessed by users with a low security clearing and will not be written to objects that are readable by users with lower security clearing.

Critique

Mandatory access control schemes have their origin in the protection of military data and property. The access control system can protect data and information flow, since the assigned labels can not be changed by the system subjects and can always be referenced. However, the mandatory enforcement of simple system rules leads to inflexible schemes that require frequent administrator override in order to be usable. These systems usually employ the notion of a *Trusted Subject* that has special privileges to act outside of the system rules.

2.3.2 Discretionary Access Control

For many real world systems the restrictions imposed by the Mandatory Access Control model are to rigid. They implement the so called Discretionary Access Control (DAC) model, where the subjects themselves have the authority to generate permissions for objects under their control. Discretionary Access Control is characterised by the ability of subjects with access permission to pass that permission on to other subjects. Assuming that a subject is authorised to access a specific object in the system, it can create new objects based on this object or modify the existing object.

> "Discretionary Access Control: A means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject."
>
> [DoD 5200.28-STD, 1985]

Typical systems implementing DAC therefore do implement a number of discrete access policies defined by the authorised system users and not a single, centrally-defined policy. It thus becomes the responsibility of the respective object owner to implement an adequate access policy.

Critique

Systems implementing DAC do not attempt to control information flow. UNIX and BSD operating systems, for example, allow a file to be copied, as soon as the user has been granted *read*-access to the file. The *copy* operation does not even preserve the attributes of the original file, instead the copy has a new owner (the subject that carried out the copy operation), who in turn has full control over the operating systems permissions for this file.

Example: Data owner *Owner* wants to limit data sharing with data user $User_0$ but continues to share data with data user $User_1 - User_N$:

> *Owner* maintains a data release policy that grants no access permissions to $User_0$, but various permissions to $User_1 - User_N$.

As data is distributed between the users it is possible for $User_0$ to get access to this data indirectly via some $User_1 - User_N$, since no information is maintained by the access control systems about the fact that *Owner* had originally intended not to share data with $User_0$.

It therefore seems that discretionary access control schemes are not suited for cases, where the original data owner wants to maintain control over the further distribution of private data.

2.3.3 Owner-Retained Access Control

McCollum, Messing, and Notargiacomo demonstrated in 1990 the existence of several access control models that could not be represented through the well-known Mandatory and Discretionary Access Control mechanisms [McCollum, Messing, and Notargiacomo, 1990].

One of these access control models is the so called Owner-Retained Access Control (ORAC), where the access policy for a resource is determined by its owner or creator and not by a system rule or the security administrator. The creator of an object sets the access policy, which can not be revoked or changed later by other users that have access to the object.

This policy model combines features from the Mandatory as well as the Discretionary Access Control model. It looks to the resource owner similar to a DAC model, because he or she has complete freedom to define any permissions for this object. To subsequent users of the resource it behaves similar to a MAC model, because these users are bound by the owners policy and can not modify or override it.

> ORAC provides a stringent, label-based alternative to DAC for user communities, where the original owners of data need to retain control of the data as they propagate through copying, merging, or being read by a subject that may later write the data into other objects.
> …The user who creates a data object is considered its owner and has the right to create an ACL *(Access Control List)* on the object."
>
> [McCollum, Messing, and Notargiacomo, 1990]

Critique

The ORAC model is a very attractive access control model for the protection of sensitive private data. It is, however, not as widely known and implemented as the other access control models.

Sevinç and Basin [2006] describe a formal access control model for documents that employs a similar concept of control and ownership over data. Instead of assigning labels they propose to associate policy language objects directly with the relevant data to form a so called *sticky policy*. In their work they focus on document-related actions such as *read*, *print*, *change* and *delegate*. Their model supports multiple owners and sub-policies for document parts and takes document editing into account, where merging and splitting of document content also influences the attached policies.

2.3.4 Usage Control

Traditional research on access control has largely focused on the protection of server-side resources, that typically form a single security domain executing a common security policy. These access control models work well when the protected resource is a fixed physical instance or a controlled service, where access can be granted or revoked depending on the underlying access control policy. If the protected resource is a moveable and cloneable asset, e.g. a data item, it becomes necessary to also control the dissemination of this resource and the subsequent usage. This requirement is immediately obvious in the area of distribution of digital content, where *Digital Rights Management* (DRM) techniques are also providing client-side protection for the necessary enforcement of usage control.

The growing importance of distributed, dynamic computing environments emphasises the necessity for client-side control over locally stored data across different security domains. Park and Sandhu [2004] proposed the UCON_{ABC} usage control model that incorporates the distinct approaches of traditional Access Control, Trust Management and Digital Rights Management (DRM) to define a unifying access control model (see Figure 2.1).

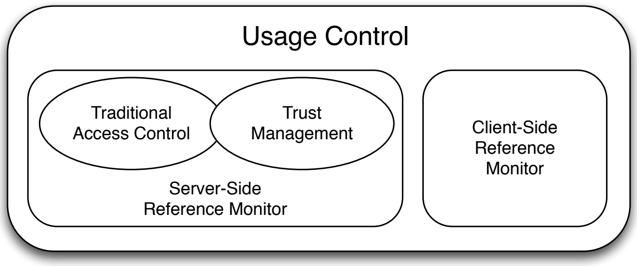

Usage Control
Traditional Access Control
Trust Management
Server-Side Reference Monitor
Client-Side Reference Monitor

Figure 2.1: UCON scope - The UCON model for usage control recognises the necessity of a client-side reference monitor for privacy protection. [Park and Sandhu, 2004]

The UCON_{ABC} model targets the protection of information secrecy, the protection of property rights as well as privacy protection. It incorporates recent approaches of trust management and modern access control that base authorisation decisions not only on the identity of the subject, but incorporate additional attributes (such as trustworthiness) in the decision process. Access decisions can be made dependent on additional external data, policies and conditions.

Digital rights protection and data privacy protection share a very similar goal: a data provider wants to control the dissemination and use of protected data. DRM rules and privacy policies can be expressed with similar policy language constructs as has been shown by Apitzsch, Liske, Scheffler et al. [2008].

However, the enforcement properties of data owner-defined privacy protection systems and DRM systems differ. DRM protected content is made available by a content provider to an end-user, while privacy protected content may be provided by the end-user to a service provider. This asymmetry in bargaining power allows the content owner in DRM systems to enforce the distribution of necessary cryptographic keying material to all client-installations, while the data owner in a privacy enforcement system has no such power and requires cooperation from the data user. DRM systems usually enforce a simple access control policy that may not even be based on the true identity of the user and relies on external attributes such as the possession of a valid license key or proof of payment. Client-side reference monitors for DRM systems are usually closed-source solutions that enforce predefined actions on atomic resources (e.g.: view movie, play MP3-song, etc.) and have no need to communicate an extensive policy from the policy administration system to the client-side reference monitor.

Critique

UCON$_{ABC}$ explicitly recognises the need for client-side access control through the use of a reference monitor. However, it does not provide any discussion or guideline on architectural or implementational issues. The important area of policy administration, management and distribution is mentioned by the authors, but they assume a suitable policy already exists and leave the other issues to further examination.

The privacy enforcement scheme that is developed in this thesis conforms to the basic architecture of the UCON$_{ABC}$ model as described by Park and Sandhu. We extend this work especially in the area of policy management, where we develop a solution for policy creation issues that result from a data-owner based approach to policy writing and we go far beyond in our discussion of practical and implementational issues.

2.4 Authorising Data Access

The discussion in the previous section has shown that the protection requirements of private data can be modelled using existing access control models and that the *authority* over the protection policy of the system is an important differentiation factor between these access control models.

An *authorisation policy* governs the administration of data access policies. The choice of access control model usually implies a certain authorisation policy and implementation strategy. The following sections will elaborate on the subproblems of authorising data access, representation of authorisations, decision making and decision enforcement.

2.4.1 Authorisation Policy

Authorisation policies, also sometimes called *Administrative policies*, determine who is authorised to set or modify permissions in the access control scheme. These policies can be fixed, such as in the case of Mandatory Access Control schemes, or flexible.

Mandatory access schemes usually employ the concept of a *Trusted Subject*. They assume that there exists the role of a Security Administrator or Privacy Officer that has the power and legitimacy to grant or deny data access and override system rules (cf. Karjoth and Schunter [2002]). The trusted subject, usually a system administrator, might also set and modify policy rules and information about objects and subjects in the system, depending on the properties of the access control scheme. The trusted subject is not bound by any administrative policy themselves.

In distributed policy schemes it is preferable to have explicit system authority rules that are part of an *Administrative Policy* and have a subset of rules governing the policy creation process, rather than rely on trusted subjects that have unbounded authority. Explicit authority rules make it possible to evaluate and enforce a privacy creation and management policy. Policy schemes can be differentiated depending on whether they support authorisation rules that guarantee that the policy administrator operates within the specified restrictions.

In the context of privacy enforcement that uses data-owner controlled policies, the data owner must be given the authority over protection policies of personal data. These schemes must ensure that the authority does not change as the data is released to the data user. If this constraint can be enforced, the data owner can be sure that data can not be misappropriated.

2.4.2 Authorisation Decision Function

From a mathematical standpoint an authorisation policy can be regarded as the set of all possible access decisions over the sets of subjects S and objects O supported by this policy. A definition of such an authorisation policy is given by Woo and Lam:

> An authorisation policy is the 4-tuple $(\mathbf{P}^+,\mathbf{P}^-,\mathbf{N}^+,\mathbf{N}^-)$ where each component is a subset of $\{(r,s,o)\,|\,r \in R, s \in S, o \in O\}$ over the set of subjects S, objects O and access rights R. P^+ and N^+ record the rights that are explicitly granted or denied. Whereas P^- and N^- record the rights that should not be explicitly granted or denied and are needed to define the semantics of policy composition.
>
> [Woo and Lam, 1993]

A policy $Auth = (P^+,P^-,N^+,N^-)$ is sound if there exists no request (r,s,o) such that $(r,s,o) \in P^+ \cap P^-$ or $(r,s,o) \in N^+ \cap N^-$.

The following *authorisation decision function* is defined for an authorisation request (r,s,o) over a sound policy $Auth = (P^+,P^-,N^+,N^-)$:

$$\begin{aligned} Auth\ grants\ (r,s,o) \quad & \text{iff} \quad (r,s,o) \in P^+ \\ Auth\ denies\ (r,s,o) \quad & \text{iff} \quad (r,s,o) \in N^+ \\ Auth\ fails\ (r,s,o) \quad & \text{iff} \quad (r,s,o) \notin P^+ \cup N^+ \end{aligned} \tag{2.2}$$

A decision request is granted if a positive authorisation can be found and denied if a negative authorisation is derived. The decision function fails if the authorisation request does not match the authorisation base. It will be shown later how an default policy can be used to change the decision result in this case.

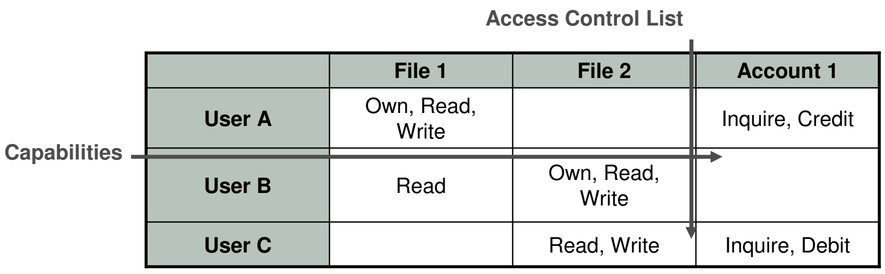
The explicit inclusion of rights that should never be granted or denied (P^-,N^-) allows to check the consistency of the authorisation base. Conflicts in policies must be resolved in this model prior to the authorisation decision.

2.4.3 Representation of Authorisations

Over the history of access control systems many different approaches for authorisation specification have been developed. A suitable authorisation specification must be powerful enough to capture the intentions of the policy writer – the protection goal of the security policy. The representation must be easy to implement and should aid the task of policy management through a clear conceptional model that should be easy to understand. It should be easy and fast to derive an authorisation decision from the system.

Access Control Matrix

One of the best understood policy representation is the *Access Control Matrix* (cf. [Lampson, 1971], Graham and Denning [1972], Sandhu and Samarati [1994]), where rows represent subjects, columns represent objects and the matrix entries are used to store the access rights (see Figure 2.2).


Access Control List
Capabilities

	File 1	File 2	Account 1
User A	Own, Read, Write		Inquire, Credit
User B	Read	Own, Read, Write	
User C		Read, Write	Inquire, Debit

Figure 2.2: Exemplary Access Control Matrix - The Access Control Matrix has long served as the reference model for the formulation and execution of access control statements and decisions.
Capabilities represent the Access Control Matrix by row, whereas Access Control Lists represent the matrix read by column.

An Access Control Matrix in its basic form lists only positive permissions and defines an implicit deny rule for access requests that do not correspond to a permission entry in the matrix. On a typical system with many objects and users an access matrix would only be sparsely populated and would allocate to much memory. Actual implementations therefore store access rights as *Access Control Lists* or as *Capabilities*:

Access Control Lists (ACL) can be seen as a representation of the Access Control Matrix read by column. An ACL belongs to an object and lists the set of permissions for this given object. These permissions define the subjects and operations that are allowed by the subjects. ACLs have the drawback that it now becomes difficult to determine the set of authorisations that are held by a particular subject, because all object in the system would need to be examined to get this information.

Capabilities store the permissions of the Access Control Matrix by row. It thus becomes easy to capture all the permissions for one specific subject, but now it is difficult to find all the subjects that might have access rights associated with a certain object.

Access Control Lists are the most popular implementation of the Access Control Matrix because they are easy to implement and understand, but have limitations when it comes to expressiveness and maintainability of policies:

Expressiveness ACLs have difficulties to capture hierarchies and external dependencies. Certain security models, such as the Chinese Wall security policy

[Brewer and Nash, 1989], introduce dependencies that can not be directly modelled at the level of expressiveness found in ACLs (e.g.: a financial analyst that has access to files from Company A must not have simultaneous access to files from Company B).
Similarly, it is difficult to express permission-sets that contain the *all-permission* with only a small set of exceptions (e.g. granting read access to the whole department, with the exception of one person). Support for wild-cards and negative permissions have subsequently been added to enhance the expressiveness of Access Control Lists but have thus added complexity to this simple model.

Maintainability Every single object in the system must be attached with access rights at creation time and must be updated if the trust level for a subject changes. This can be a resource intensive task for systems containing numerous objects and can lead to errors if external dependencies are not correctly resolved.

Enhancements of expressiveness for ACLs have lead to the fact that the interpretation of the ACL now depends on rule-ordering and evaluation strategy, where more specific permissions need to be defined before general permissions and ACLs have to be evaluated in a defined order (usually top-down). Different ACLs can no longer be easily combined, because the simple concatenation that was possible for ACLs that express only positive authorisations becomes problematic when ACLs express exceptions and negative permissions.

Authorisation Languages

Authorisation languages provide syntactical elements for greater expressiveness and maintainability than simple *Access Control Lists* (ACLs) or Capabilities. They allow the expression of an access control policy in a declarative language and thus provide an abstraction from the actual authorisation decision.

The Access Control Matrix directly stores authorisations as an entry of an ACL or Capability. An authorisation decision can be made simply by finding and retrieving the action entry for an object and subject match. Authorisation languages usually do not store the individual authorisations, but rather create a policy rule-base that forms the basis for authorisations. This rule-base must be evaluated by a decision mechanism at the time an access request is made, in order to derive at the appropriate access decision. Since this evaluation needs to be executed every time an access is requested, the decision mechanism needs to derive at this decision quickly or else the responsiveness of the system suffers.

Attribute-based authorisations Authorisation languages allow flexible authorisation decisions that can also be based on other attributes than the identity of the requesting subject [Woo and Lam, 1993]. For instance an authorisation could depend on the identity of a subject, its current location and the local time at the current location. Abrams, LaPadula, Eggers et al. [1990] calls this the *Access Control Context* and remarks that this information is independent of the set of particular subjects or objects specified in the rule base.

Structural properties Authorisations can be highly structured and thus reflect real world systems that themselves have a high degree of structure [Woo and Lam, 1993]. Authorisation languages allow the specification of dependencies in access decisions through logical constructs that would not be possible to express in an ACL, where every entry constitutes an independent access decision.

Default policy To simplify the creation and maintenance of access specifications, access policies regularly implement a default policy. The default policy will be used if no authorisation decision can be made from the consulted rule base. Default policies can be defined in two different ways as either a *permissive* or a *restrictive* policy (also called *open* or *closed* policy). A *restrictive* or *closed* policy denies all actions that are not explicitly authorised. The *permissive* or *open* policy allows all actions that are not explicitly denied.

Default policies ease the task for the policy administrator. However, the exclusive reliance on an implicit default policy can lead to the situation, where absent authorisations can not be distinguished from authorisations in line with the default policy [Tschantz and Krishnamurthi, 2006]. If for whatever reason the default policy of a rule base changes, these rules would not be present and could lead to erroneous authorisations.

Critique

Policy rule-sets must be evaluated at the time of access in order to determine the access decision. This evaluation process can be computational intensive for large policy bases. This additional computational overhead of evaluating the authorisation rule-base is outweighed by the ability to base an authorisation decision on a richer set of attributes than a simple Access Control Matrix. Policy-rules can contain conditions that must be fulfilled for the granting of an access request and obligations that are requirements that will be passed to the enforcing entity. Access control languages allow the expression of dependencies between different policy decisions. The ability of these languages to logically group objects and subjects, as well as the ability to evaluate environmental conditions allows the creation of concise policies.

The support for explicit positive and negative authorisations improves expressiveness in policy languages. Potential evaluation conflicts (as described in Section 2.7) can be resolved through the implementation of conflict resolution strategies in the decision mechanism.

2.5 Distributed Access Control

When we move away from a centralised system architecture to access control in distributed systems new requirements arise. From a security perspective we can distinguish between two different scenarios: a) small and medium scale distributed systems that enforce a single security policy for all members of the system, and b) large scale distributed systems that are composed from multiple independent security domains that typically enforce their own security policy but may have trust relationships with each other.

The joint evaluation of policies in distributed systems requires the development of a common policy semantics and the deployment of a framework for authorisation requests and responses. This common framework must be flexible enough to support different usage scenarios, but rigid enough to enforce system interworking. Reliance on such a policy framework would enable distributed systems to be bound by a centralised common policy, although distributed policy decisions can be made and enforced at the individual host level.

Distributed evaluation and enforcement requires externally defined policies. The externalisation of policies has the following benefits:

- Policy representations can be chosen to represent the problem at hand and can be changed without any modifications at the implementation .
- Policies can be evaluate externally and are independent from the actual implementation.
- Run-time evaluation allows support for dynamic policy changes and the evaluation of additional attributes for policy decisions.

2.5.1 Generalised Framework for Access Control

Abrams, LaPadula, Eggers et al. [1990] proposed to separate the policy decision component from the rest of the security and access mechanisms of the system. The access control policy is constructed from *Access Control Rules* (ACR) that are used to derive access decisions for access requests made by subjects to objects of the system. This functional decomposition makes it possible to separate the access control functions from the actual enforcement functions and evaluate additional context information. This work has lead directly to the distributed access control framework described in the next section.

2.5.2 ISO/OSI - Access Control Framework

In 1996, the International Organization for Standardization (ISO) specified a distributed access control framework [ISO/IEC 10181-3, 1996] as part of the Open Systems Interconnection (OSI) security framework for open systems [ISO/IEC 10181-1, 1996]. The standard identified specific access control functions that can be part of a distributed access control service.

The basic entities of the access control model are the Initiator, the Access Control Enforcement Function (AEF), Access Control Decision Function (ADF) and the Target. The AEF enforces compliance with access control decisions made by the ADF for resource access from the initiator to the target. The architecture separates the access control from other systems functions. It provides a generic decision function as a common service that can be used for the provisioning of enforcement functions throughout the distributed system.

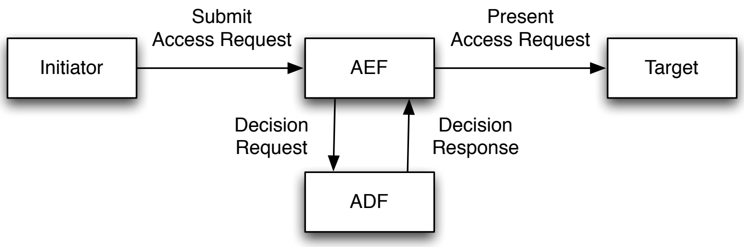


Figure 2.3: Access control functions as defined by ISO/IEC 10181-3 [1996] - The framework models the different functional elements of access control as separate entities that can serve generic access requests issued within a distributed system.

Figure 2.3 shows the abstract access control elements as they are defined by the standard. No specific access control policy or policy representation is assumed, although several possible alternatives are discussed in the ISO access control framework document.

2.5.3 Framework for Policy-based Admission Control

The *Internet Engineering Taskforce* (IETF) has defined a framework for providing policy controlled admission control to networks offering different Quality-of-Service categories in RFC 2753 [Yavatkar, Pendarakis, and Guerin, 2000]. The framework has the goal to provide an abstract architectural framework that does support many styles of policies and support a distributed implementation where the actual policy decisions are made by specific entities in the network. The framework does not mandate the use of a specific policy mechanism to achieve this goal.

Yavatkar, Pendarakis, and Guerin [2000] specify two main architectural elements for policy control: a *Policy Decision Point* (PDP) and a *Policy Enforcement Point* (PEP) (cf. Figure 2.4). The PDP uses mechanisms and protocols for user authentication, accounting, policy information storage and may support additional functions which are not defined in the standard.

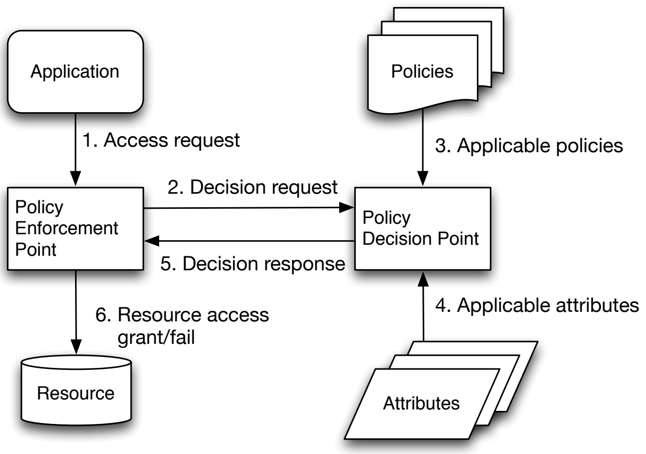


Figure 2.4: Distributed Access Control Framework as defined by Yavatkar, Pendarakis, and Guerin [2000] - The distributed access control framework for policy controlled network access, as defined by the IETF, serves now as a reference model for other distributed frameworks, such as XACML-2.0 [2005].

The standard provides a very generic framework and makes no restrictions on the deployment of the framework elements. The PDP can be a central server – accessible by a large number of PEP's, or it could be collocated with the PEP on the same machine.

2.6 Privacy Policy Languages

In section 2.2.5 we already talked about the importance of policy languages for the expression of privacy policies. This section now discusses the advantages and disadvantages of different policy languages in greater detail.

There exists a variety of policy specification languages that differ in their expressiveness, maturity and applicability to certain problem domains. For the purpose of this thesis we are going to analyse three dedicated privacy policy languages, the Platform for Privacy Preferences (P3P), the Enterprise Privacy Authorization Language (EPAL) and the GEOPRIV Common Policy and compare them to a general purpose authorisation language: the eXtensible Access Control Markup Language (XACML). We are focusing on these languages because they are standardised, have clear support for the expression of privacy preferences and some have been actively deployed (P3P and XACML). Their use for the formulation access control and privacy policies has been extensively discussed in the literature, e.g. by Vimercati, Samarati, and Jajodia [2005]; Langendörfer, Maaser, Piotrowski et al. [2008]; Anderson [2005]; Gupta and Bhide [2005]; Ashley and Karjoth [2003]; Karjoth, Schade, and Herreweghen [2008] and others.

There exist other policy languages, like the *Flexible Authorisation Framework* [Jajodia, Samarati, Sapino et al., 2001] and the *eXtensible rights Markup Language* [XrML, 2001], that have a lesser degree of popularity and are not examined in further detail in this thesis.

2.6.1 Platform for Privacy Preferences (P3P)

The Platform for Privacy Preferences project of the World Wide Web Consortium (W3C) has defined a standard for the expression of privacy policies for websites – together with a protocol that enables web browsers and other tools to evaluate privacy policies automatically.

The first version of P3P [P3P v1.0, 2002] has been ratified as a W3C Recommendation in 2002. In 2006, the P3P working group issued a W3C Group Note [P3P v1.1, 2006] with the specification for a revised standard that picks up feedback to the standardisation committee. The specification includes updated definitions for some key elements of the language. P3P 1.1 introduces a new binding mechanism that allows privacy policies for content that is not referenced through an URI, such as requests to Web Services.

Work on the P3P standard started in the mid-1990s and followed the aim to let users define the conditions under which they accept the release of their personal information. The initial idea was to rate websites according to their information practices and negotiate agreements about website privacy practices. The standardisation process was far from smooth and many important features, such as the ability to negotiate privacy levels, got dropped from the final standard. An account of the development and changing requirements for the first version of the standard can be found in Cranor [2002].

Protection Goal

The owner of a website (data user) describes the terms of conduct for data that is collected while the user (data owner) is browsing the site, filling in forms and submitting data. The protection goal consists of the following main areas:

- Specification of terms for data collection and data retention
- Acceptable use of collected data for current and additional purposes
- Definition of terms for the transfer of data to other departments, partners and third parties

The data owner (service user) is given the ability to evaluate the privacy policy of a website prior to any data release. He or she should be able to decide whether to accept this policy or to refrain from interacting with this website if the stated privacy practice does not match with personal privacy preferences. P3P follows the principle of informed choice, where the data owners retain control over their personal data and can decide under which conditions they are willing to release it. Service providers can use P3P policy statements to make the privacy practices of their websites explicit in a standardised way.

Mechanism

A P3P policy is encoded as an XML document that can be found at a well known location at the website[5]. This mechanism allows a P3P enabled web browser to check for the availability of a suitable policy prior to the loading of the page for every site the user visits. P3P defines a base data schema for classification of release cases and data categories that allows the creation of standardised policies. This data schema can be extended by the policy creator to incorporate domain specific attributes in their policies if needed.

The P3P consortium simultaneously developed *A P3P Preference Exchange Language* (APPEL) [P3P APPEL, 2002] that allows the creation of rule-sets for the expression of privacy preferences by the data owner. A P3P aware web browser can then be used to automatically check the compliance of a site policy with the privacy preferences of the data owner.

Language Elements

The `<POLICIES>` element is the top-level element of the P3P language. More than one policy can be specified for the support of multiple human languages or sub-policies for individual parts of the website. Figure 2.5 gives an overview of the P3P language model (showing only required policy elements). P3P-enabled sites can make the following general assertions about their privacy policy:

Policy The `<POLICY>` element allows the specification of the location of a human-readable privacy policy via a *Disclosure URL* `discuri` and an optional opt-out mechanisms `opturi` for data collection.

Entity The `<ENTITY>` element contains contact information for the legal entity that is responsible for the website privacy policy and data collection.

Access If the website collects identifiable data the `<ACCESS>` policy element states the type of access that the data owner has to the identifying data. It does not specify an access method to the data.

Disputes Contact information for possible dispute resolution is provided via the `<DISPUTES>` section. Remedy actions for erroneous privacy claims can be stated via the `<REMEDIES>` element.

P3P policies also make the following *data-specific assertions* that specify the scope of data collection and usage. These assertions are combined within a `<STATEMENT>`:

[5]The P3P specification mandates the following location for the policy reference file: `/w3c/p3p.xml`

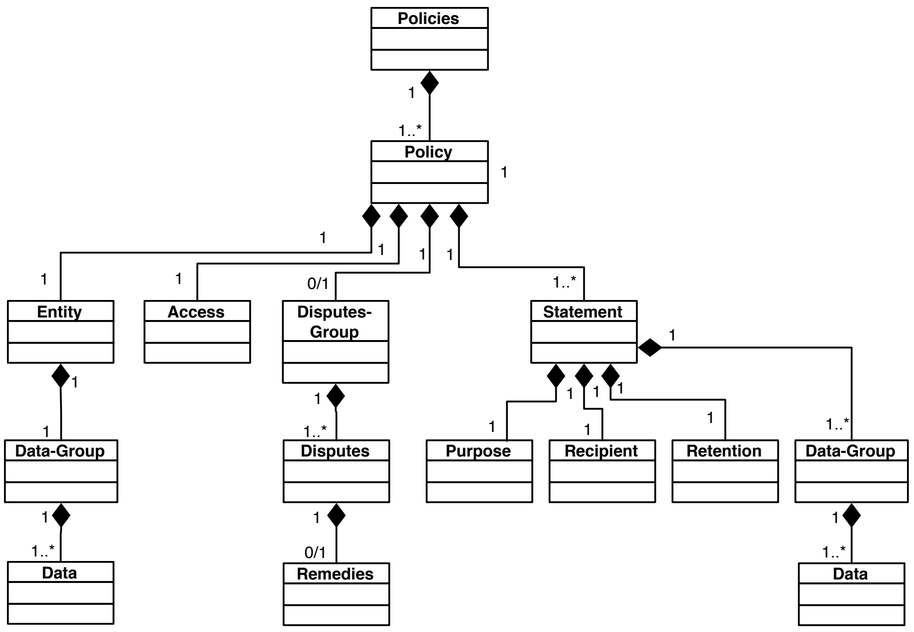


Figure 2.5: P3P language model - The figure shows a simplified language model for the P3P language. Optional language elements have been omitted. The P3P language is conceptionally closer to an ontology than a true privacy authorisation language. Enforcement of privacy requirements require the formulation of privacy preferences using the P3P Preference Exchange Language [P3P APPEL, 2002]

Purpose The reasons for data collection is specified via the `<PURPOSE>` element. P3P defines a vocabulary of 12 pre-defined purposes. Examples include:

- Research and development `<develop/>`
- Marketing `<telemarketing/>`
- Completion and support of the current activity by the service provider `<current/>`

Recipient The `<RECIPIENT>` element specifies who has access to the collected data. This can be the entity referenced in the privacy policy together with agents acting for this entity, other specified partners as well as public forums and directories.

Retention Terms for data usage and data retention are published via the `<RETENTION>` element. The data retention policy is defined by one of five predefined values ranging from `<no-retention/>` to `<indefinitely/>`. It does not specify a specific time when collected data will be purged.

Data The type of collected data is specified via one or more `<DATA>` elements inside a `<DATA-GROUP>` element. P3P includes a *base data schema* that identifies data items frequently entered in web forms such as name, phone-number and others. P3P also specifies 17 data categories that provide hints to the data owner about intended use of the collected data. This list includes contact and purchase information, user health data and preferences as well as navigation and clickstream data generated by the browser visiting the website.

The example in Listing 2.1 defines a privacy policy for the website at `www.example.com`. It specifies that `www.example.com/BrowsingPolicy.html` a human readable version of this policy can be found. Navigational data will be collected, stored and used for administrative and development purposes only by the Example Corporation. Any disputes about this policy should be resolved via the 'PrivacySeal' organisation and should lead to an adaptation of the current practice.

Listing 2.1: P3P Policy Example

```
<POLICIES xmlns="http://www.w3.org/2002/01/P3Pv1">
        <POLICY name="Privacy Policy"
                 discuri="http://www.example.com/BrowsingPolicy.html" xml:lang="en">
        <ENTITY>
                <DATA-GROUP>
                        <DATA ref="#business.name">Example Corp.</DATA>
                        <DATA ref="#business.contact-info.online.email">p3p@example.com
                            </DATA>
                        <DATA ref="#business.contact-info.postal.city">New York</DATA>
                    ...
                </DATA-GROUP>
        </ENTITY>
        <DISPUTES resolution-type="independent" service="http://www.PrivacySeal.org">
                <REMEDIES><correct/></REMEDIES>
        </DISPUTES>
        <STATEMENT>
                <PURPOSE><admin/><develop/></PURPOSE>
                <RECIPIENT><ours/></RECIPIENT>
                <RETENTION><stated-purpose/></RETENTION>
                <DATA-GROUP>
                        <DATA ref="#dynamic.clickstream"/>
                </DATA-GROUP>
        </STATEMENT>
</POLICY>
</POLICIES>
```

Critique

P3P policies provide a privacy promise that specifies a protection goal targeted at data collected from web-forms and user behaviour.

The creation of a P3P policy is entirely voluntary for the data user – the data owner can not force the creation of a (specific) privacy policy. The ability to negotiate the privacy level for released data has been dropped from the P3P standard. In the absence of the ability to negotiate, the data owner can only approve or disapprove the site policy as a binary decision. The data owner has to either accept the policy or refrain from any further transaction with the data user. The stated privacy policy may also change at a later time, while the data owner has agreed to a previous version of the policy.

The granularity level for protection specifications is the web-site/page level. P3P does not take into account that there might be different sensitivity levels for individual data items or categories and that these might have a higher protection need. Specification of protection properties for individual data items is not supported.

P3P policies can not be directly enforced, because they do not specify concrete actions and conditions for data access. To make a P3P directly enforceable, the policy would have to be interpreted and translated into a suitable access policy. No such translation mechanism currently exists. Ashley and Karjoth [2003] have analysed P3P with respect to the automatic generation of authorisations from stated policies. They found that the use of pre-defined data categories and types, as well as the vague and limiting categories for purpose binding and actions make P3P unsuitable as an authorisation language. Certain necessary elements, such as support for conditions, are missing from the language.

The original P3P data schema can be extended to incorporate domain specific attributes, however this may lead to policy evaluation problems, as not all P3P agents might be able to interpret these extensions.

2.6.2 Enterprise Privacy Authorisation Language (EPAL)

IBM has developed the Enterprise Privacy Authorization Language (EPAL) as a language for the exchange and propagation of privacy policies within and between organisations. The language specification [EPAL 1.2, 2003] has been submitted as a Member Submission to the W3C in November 2003 to be considered as a privacy policy language standard. W3C has declared EPAL as being out of scope of the chartered P3P 1.1 working group, but relevant for further P3P working groups [W3C Comment, 2003]. This status has not changed to date.

Protection Goal

EPAL has been developed to describe enterprise-internal privacy practices and enforce these policies throughout the company workflow. EPAL can potentially be used to link internal enforcement procedures of the organisation with an external privacy policy. Karjoth, Schunter, and Herreweghen [2003] proposed a mechanism for the automatic and dynamic translation of authorisations specified in the Enterprise Privacy Authorisation Language into a corresponding P3P Policy, so that the externally displayed privacy policy could always in line with the internal enforcement practices.

Mechanism

EPAL is a declarative policy language that can express positive and negative authorisation. Rule ordering is important for policy evaluation because a *First match* evaluation strategy is used to resolve potential conflicts in the rule base. Rules describing exceptions must be listed before general rules in the policy base, because evaluation is stopped when a matching entry is found.

EPAL supports the distributed policy decision/enforcement architecture defined in Section 2.5. A decision component using EPAL policies can generate implementation independent access decisions. EPAL rules support the specification of conditions and the specification of obligations for every rule in the policy.

EPAL rules use subject, object and purpose categories (`user-categories`, `data-categories` and `purpose-categories`) for the expression of authorisations that provide an abstraction from actual data-models and system users, which can lead to a significant reduction of the number of rules in the policy.

The abstract attributes used in rules and conditions that are required for policy evaluation are defined in a vocabulary. EPAL does not define a global terminology, such as the one used by P3P.

Vocabularies could, however, be used to define sector specific terminologies. The exchange of policies between different partners requires matching vocabularies.

Language Elements

EPAL policies are defined under the `<epal-policy>` top-level element. Figure 2.6 provides an overview of the elements in the language model.

EPAL-Policy Each policy has a required attribute `default-ruling` that implements a default policy (cf. Section 2.4.3) in case no authorisation decision can be derived from the policy. Default-rulings are ‘allow’, ‘deny’ or ‘not-applicable’.

EPAL-Vocabulary EPAL uses vocabularies to define sector specific language elements. Enterprises will typically define their own vocabulary. Merging of policies from different enterprises is only possibly if all partners can agree upon a common subset of the vocabulary. The `<epal-vocabulary>` element defines all attributes that will be referenced in the policy.

Rule The main building block for a privacy policy is the `<rule>` element. The `ruling` attribute determines if the rule describes a positive or negative authorisation. Allowed values are ’allow’ and ’deny’.

User, data and purpose categories can be defined as hierarchical categories. Access decisions are always granted for the complete sub-tree of the hierarchy specified in the rule. Rules must be ordered in the policy so that specific rules precede general rules (e.g. permission of access to category ’health/medication’ needs to be placed before the more general prohibition on the category ’health’).

An EPAL rule uses language elements defined in the `<epal-vocabulary>`. The used elements must be known by the interpreting system through their definition in the vocabulary prior to evaluation. Rules contain the following elements:

Data Category - One or more `<data-category>` element defines the matching data category.

User Category - One or more `<user-category>` element defines the matching user category.

Purpose - The request will also be matched against one or more `<purpose>` element defining privacy relevant purposes.

Action - The `<action>` element defines the privacy relevant actions of the policy.

Condition - The `<condition>` element defines the pre-condition for the applicability of the rule.

Obligation - Rules may include `<obligation>` elements. The obligations are not processed by EPAL, but are passed on to the environment and specify additional activities that need to be performed, when an action is granted.

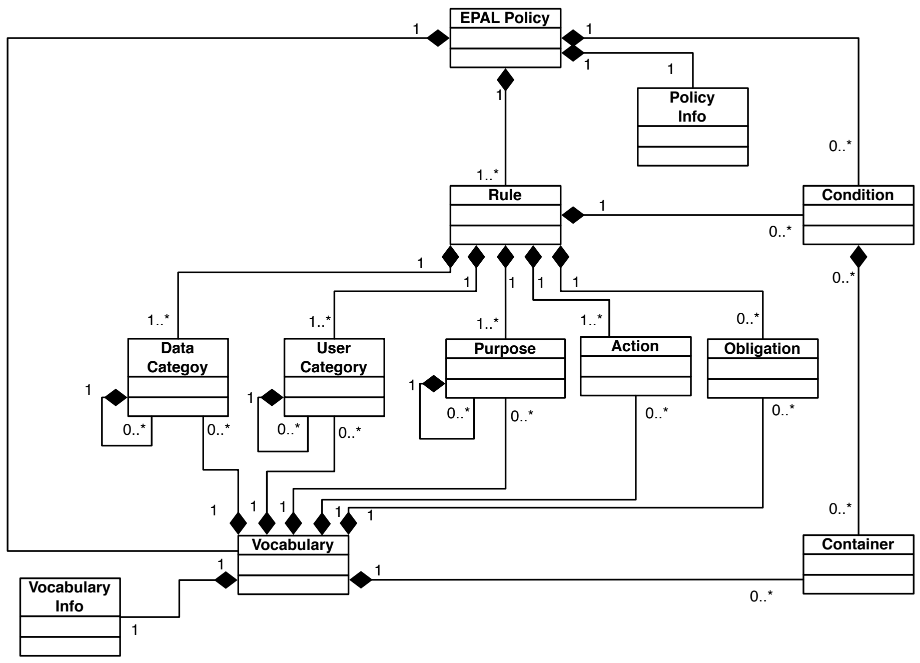


Figure 2.6: EPAL language model [EPAL 1.2, 2003] - The language elements of the EPAL language and their interrelations are shown in the figure. Not all elements must be present in a particular EPAL policy.

Condition evaluation Conditions determine if a rule applies to the decision request. Only if all the stated conditions are satisfied does the rule apply, otherwise it is ignored. Conditions are evaluated to a boolean value and EAPL defines a number of functions and predicates based on an earlier draft of the XQuery standard [XQuery, 2007].

Critique

EPAL is not designed as a general access control language. Access requests need to be translated by the Policy Enforcement Point (PEP) before a policy decision can be made. Policy requests and decisions are based on abstract data and user categories that must match against the pre-defined categories in the EPAL vocabulary. The PEP has the task to reliable map actual subjects, objects and purposes into the supported categories before an decision request can be issued. This mapping operation is outside the scope and control of EPAL and could potentially lead to different access decisions between implementations. EPAL does not define how privacy-categories and policies are associated with collected data. The PEP would need to be very application and data aware to support EPAL policy evaluation.

EPAL does not allow policy nesting and combining. It should be possible to evaluate several EPAL policies independently. However, EPAL does not define a policy combination mechanism within the standard.

Policy management is very difficult due to the rule-ordering requirement. Additions to or deletion from the policy rule-base can have unforeseen consequences for the policy writer because the *First Match* evaluation potentially hides rules further down in the rule-base. The same would be true for the merging of different policy rule-bases.

2.6.3 eXtensible Access Control Markup Language (XACML)

The eXtensible Access Control Markup Language (XACML) is a declarative access control policy language and a processing model that is standardised by OASIS [XACML-2.0, 2005]. The current version 2.0 was ratified in 2005 and there is ongoing work on the specification of XACML_3.0. XACML provides a core specification which is enhanced by profiles that define best practices on how to use XACML for specific well-defined scenarios such as role-based access control and privacy protection. The implementation of XACML profiles is optional.

Protection Goal

XACML has been defined as a general purpose access control language that should be able to express arbitrary access control policies. It defines a rich policy language model for the expression of access policies together with a standardised policy evaluation approach.

XACML has been developed to support policy evaluation in distributed environments where multiple policy sources exist and an access decision from joint policy evaluation must be reached. XACML supports attribute-based authorisations, where additional information taken from the environment can be taken as input into the access control decision. Decisions can be based on and directly applied to the content of an XML-resource.

Mechanism

XACML is a declarative language that supports positive and negative authorisations. The decision mechanism evaluates the whole policy rule base for an access decision. Policy evaluation is not dependent on a particular rule ordering, although ordered evaluation options exist for the resolution of policy conflicts. Potential conflicts in the rule base can be resolved through the application of dedicated conflict resolution strategies provided by specific *combining algorithms*.

Distributed Policy Support XACML supports a distributed decision/enforcement architecture, which can be seen in Figure 2.7. The communication between Policy Enforcement Point (PEP) and Policy Decision Point (PDP) is based on a dedicated Request/Response Language, although combined PEP/PDP elements are also possible. Policies and policy decisions are implementation independent and need to be enforced by the PEP.

The XACML reference model separates authorisation policies from the resources. However, it makes no assumptions about the origins of the policies, other than that they are provided by an abstract entity called Policy Administration Point (PAP). This opens the possibility to include XML-based resources within the XACML policy objects and reference the resources directly from the policy via XPath [XPath, 1999].

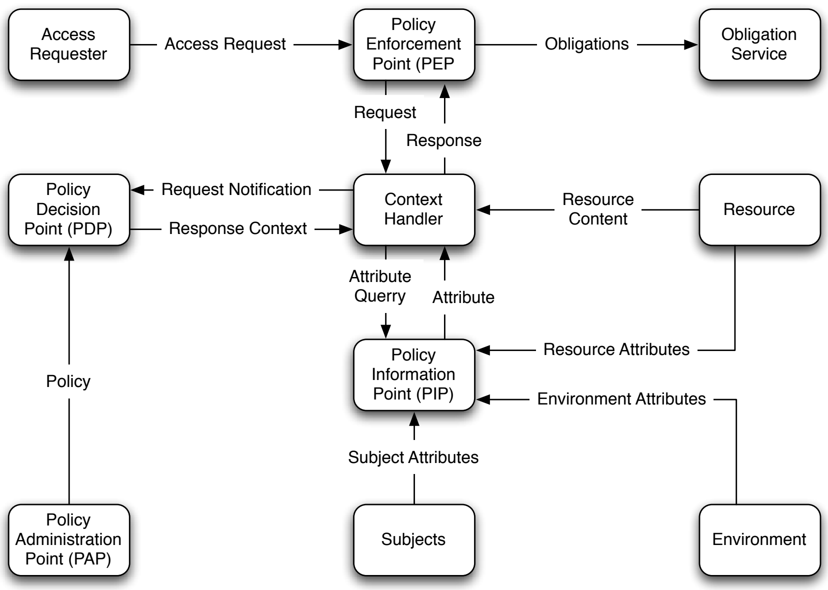


Figure 2.7: XACML Architecture Framework - The XACML Architecture framework defines the interworking of the different, possibly distributed architectural elements that are needed for the formulation and successful execution of XACML authorisation requests and responses. This architectural model can have different instantiations depending on the actual requirements. Crucial elements are the Policy Enforcement Point (PEP) and the Policy Decision Point (PDP).

Language Elements

Figure 2.8 provides an overview of the language elements. The top level element of an XACML policy is the `<Policy>` element. Different individual policies can be jointly evaluated by defining their combined evaluation inside a `<PolicySet>`. Figure 2.9 illustrates the formation of policy sets and policies from sub-elements.

PolicySet The `<PolicySet>` element contains a set of `<Policy>` or other `<PolicySet>` elements and a policy combining algorithm to determine the outcome from the joint evaluation of different elements.

Policy The `<Policy>` element represents a single access control policy. It contains a set of rule elements and a rule combining algorithm to determine the joint evaluation result of the policy. The `<Policy>` forms the basis of an authorisation decision.

Rule The `<Rule>` element contains an authorisation expression that can be evaluated in isolation and provides the basic unit of policy management. Rules can grant positive or negative authorisations. The attribute `Effect` is used to define a positive (‘permit’) or negative (‘deny’) authorisation expression. Rules may contain a `Condition` element that defines the applicability of the rule.

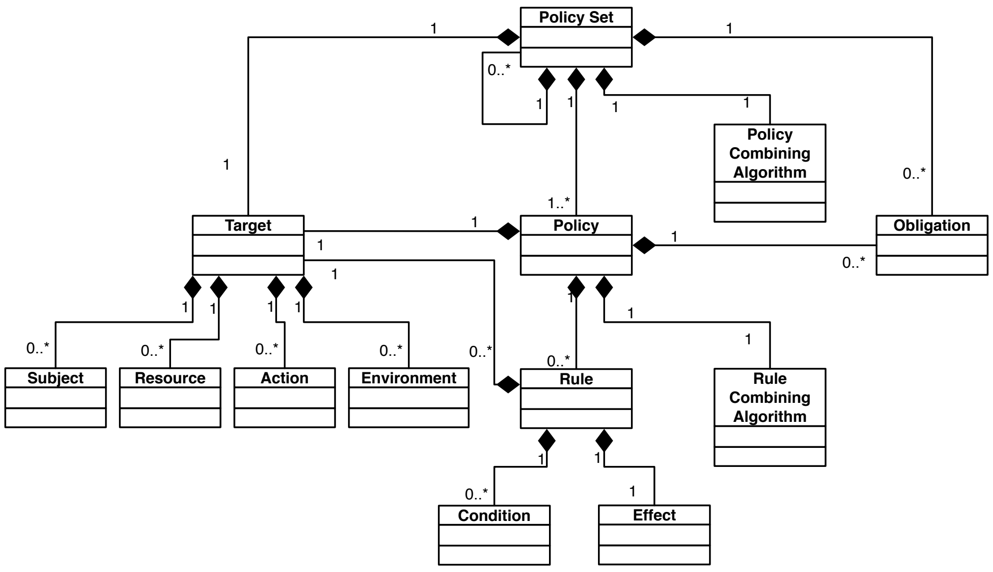


Figure 2.8: XACML language model [XACML-2.0, 2005] - The XACML language model consists of the language elements and their interrelation shown in the figure. Not all elements must be present in a particular XACML policy.

Applicability of Policies and Rules XACML implements a special feature to identify which policy rules have to be evaluated for a given request. The language uses the `<Target>` element to specify the set of applicable `<Subject>`, `<Resource>`, `<Action>` and `<Environment>` attributes. Targets are supported for the `<Rule>`, `<Policy>` and`<PolicySet>` elements and decision requests will only be evaluated against policies and rules with a matching `<Target>`. Targets in `<Policy>` and`<PolicySet>` elements can be used to narrow the evaluation scope of the policy rule-base.

Matching attributes are determined through the use of a *matching function* that compares the attribute-values of the XACML decision request to the attribute-values stored in the policy. If no specification has been made in the `<Target>`, the element will match against any value in the request. This part of the policy will therefore have global evaluation scope.

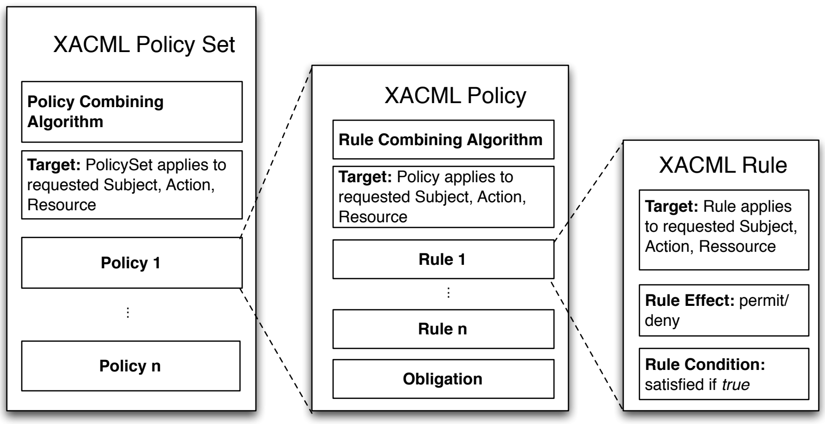
Target The following child-elements are contained in a `<Target>` element:

Subject matches the identifying attributes of the requesting actor against the values stored in the policy.

Resource defines the attributes of the requested data, service or component.

Action matches on attributes defining an operation on a resource.

Environment is an optional element used for the specification of additional attributes in the request context that are not directly associated with the subject, resource or action of the request.



XACML Policy Set
Policy Combining Algorithm
Target: PolicySet applies to requested Subject, Action, Resource
Policy 1
Policy n
XACML Policy
Rule Combining Algorithm
Target: Policy applies to requested Subject, Action, Resource
Rule 1
Rule n
Obligation
XACML Rule
Target: Rule applies to requested Subject, Action, Ressource
Rule Effect: permit/ deny
Rule Condition: satisfied if true

(a) Composition of language elements

```
<Policy...CombAlg>
 <Rule ... Effect>
   <Target>
     ...
   </Target>
   <Condition>
     ...
   </Condition>
 </Rule>
</Policy>
```

(b) Code Example

Figure 2.9: Formation of XACML Policies - XACML Polices are formed from a combination of `<PolicySet>` , `<Policy>` and `<Rule>` elements. The joint evaluation of these entities is guided by the appropriate Combining Algorithms. The `<Target>` element allows the dynamic inclusion/exclusion of rule-sets for a given request by the decision mechanism. Only rule-sets that are relevant and apply to a query need to be evaluated.

Condition Evaluation Conditions determine if a rule applies to a decision request. A rule only applies to the request, if the stated condition is satisfied, otherwise the rule is ignored. Conditions are evaluated to a boolean value and XACML defines a large number of functions and predicates based on an earlier draft of the XQuery standard [XQuery, 2007] that can be used for the creation of sophisticated expressions.

Combining Algorithms XACML allows explicit positive and negative evaluation of rules, as well as the combination of policies from different sources within a `PolicySet`. Combining algorithms, as an essential part of the language specification, are needed to derive an authorisation decision from potentially conflicting individual rules and policies. Their application is controlled via the `RuleCombiningAlgId` and `PolicyCombiningAlgId` attributes.

Standard combining algorithms are:

Deny-overrides implements a pessimistic default that prioritises access denial. As soon as one rule evaluates to a deny, the policy evaluation results in a *Deny*, even if other rules are present that permit the action.

Permit-overrides implements a optimistic default that prioritises access permission. As soon as at least one rule evaluates to a permit, the whole policy evaluation results in a *Permit*.

First-applicable evaluates the policy sequentially in the listed order. Policy evaluation is based on the first rule that matches the target and where the condition of the rule evaluates to true.

Only-one-applicable this combining algorithm ensures that there exists exactly one applicable policy. In this case the evaluation result is determined by the evaluation of this policy. If the request matches more than one policy the evaluation result is *Indeterminate*.

Table 2.2 shows which standard combining algorithms are applicable to different XACML policy elements:

Table 2.2: Applicability of XACML Combining Algorithms

	applicable on		
Combining Algorithm	**Rule**	**Policy**	**PolicySet**
Deny-overrides (ordered / unordered)	x	x	
Permit-overrides (ordered / unordered)	x	x	
First-applicable	x	x	x
Only-one-applicable		x	x

Policy Evaluation

The Policy Decision Point (PDP) receives a decision request, evaluates the policy and generates a decision response. During the policy evaluation, the PDP analyses the policy-base to find applicable policy and rule elements. Depending on the evaluation of these elements it might generate one of the following results:

Permit - the access request is allowed.

Deny - the access request is denied.

Indeterminate - the PDP can not evaluate the decision request and signals an evaluation error. An evaluation error might be caused by missing attributes, syntax errors in rules, or others.

NotApplicable - the PDP can not find a single target policy (rule) that matches the decision request. It is now up to the PEP to implement a suitable default policy to either allow or forbid the requested action. Depending on the general preferences the PEP could implement an *Open Policy* or a *Closed Policy* model (cf. Section 2.4.3).

```
<Policy RuleCombiningAlgId="deny-overrides">
  <Rule Effect="Permit">
    <Target>
     <Subject><AnySubject/></Subject>
     <Resources>
       <ResourceMatch MatchId="xpath-node-equal"> /HealthRecord/fileData </
           ResourceMatch>
     </Resources>
     <Action>view</Action>
    </Target>
  </Rule>
  <Rule Effect="Permit">
    <Target>
     <Subject>Dr. CD</Subject>
     <Resources>
```

```
        <ResourceMatch MatchId="xpath-node-equal"> /HealthRecord/medHistory/event[
            Diagnosis="Gonorrhoea"]
        </ResourceMatch>
      </Resources>
      <Action>view</Action>
      <Condition FunctionId="date-less-than-or-equal">
        <Apply FunctionId="date-one-and-only"> </Apply>
        <AttributeValue>2012-03-22</AttributeValue>
      </Condition>
    </Target>
  </Rule>
</Policy>
```

Listing 2.2: Example of an XACML Policy protected EHR. [Apitzsch, Liske, Scheffler et al., 2008]

Critique

XACML is a general purpose access control language and thus not specifically targeted at the expression of privacy policies. Anderson [2005] has shown that XACML can express all the relevant concepts that are required from a dedicated privacy language like EPAL, while at the same time being much more flexible and expressive. An exemplary XACML policy can be seen in Listing 2.2. Here we used standard language constructs to specify a privacy policy for an Electronic Health Record (EHR), that allows a certain practitioner to view the patients examination data related to a specific diagnosis until a final date.

XACML also defines a privacy extension [XACML Privacy Profile, 2005] that implements the additional attribute `resource:purpose` to express the purpose for which the data resource was collected. The corresponding attribute `action:purpose` indicates the purpose of the decision request that must be matched against the `resource:purpose` attribute.

In general, XACML uses abstract attribute values in the decision request and the corresponding policy and is agnostic to the mechanisms that are used by the PEP or the environment to determine these attributes. However, in the case of XML-resources, XACML provides support for XPath [XPath, 1999] that allows attribute extraction directly from XML-documents.

The matching functions that are used in XACML are based on XQuery [XQuery, 2007] and thus condition evaluation currently does not support higher order functions, since XQuery does not support such functions. It is therefore not possible to pass a function as an argument to another function or to specify recursive evaluation strategies.

XACML is defined as a statically-typed language and provides no function overloading or type casting. This means that the matching of expressions other than the ones supported by the pre-defined functions requires the implementation of a custom function. If, for example, we wanted to match `x500name` attributes against string attributes, we would have to write a custom function. While the standard supports language extensions, this practice may lead to incompatible implementations.

XACML provides support for distributed policy evaluation, however, the evaluation model of XACML seems to be targeted mainly at intra-organisational policy evaluation, since the following limitations apply:

- There is weak support for dynamic policy finding. Policy locations must be known at policy creation time to be correctly referenced. There exists no discovery mechanism for dynamic resources and attributes.

- There exists no support for policy prioritisation based on advanced metrics. If, for example, we wanted to prioritise policies depending on their origin from trustworthy sites, we would have to develop our own policy combining algorithm.
- The standard does not mandate authentication and integrity protection between the architectural building blocks. These would have to be provided by the individual implementations and could again lead to incompatibilities.

2.6.4 GEOPRIV Common Policy

The Internet Engineering Taskforce (IETF) has developed a privacy framework for the protection of location and presence information of resources and entities. This work was motivated by the insight that nowadays many applications access and use geographic location data and that the representation and transmission of such data raises privacy and security issues.

The privacy framework was developed within the GEOPRIV working group and consist of a core specification that defines the key elements of the language, its representation and evaluation algorithm and several application specific extensions. The core specification RFC4745 [Schulzrinne, Tschofenig, Morris et al., 2007] has been submitted as a *Proposed Standard* to the IETF standards process.

Protection Goal

The GEOPRIV privacy framework has been developed to unify efforts for the protection of sensitive location and presence data within a number of different networking contexts, such as instant messaging or location-based services. The work was initiated to integrate different approaches that had been taken for the expression of authorisation policies for location and presence data and to create a suite of protocols that allow such applications to represent and transmit location objects [Geopriv-Charter, n.d.].

The approach taken by GEOPRIV differs from other policy frameworks insofar, as it does not define a generic privacy policy which is used to derive access decisions for resources, instead it focuses on the currently available location data on a presence or location server and adds a description on what constitutes legitimate access by the data user. It is usually the owner or originator of the data that defines the policy on how this data should be exposed.

Schulzrinne, Tschofenig, Morris et al. [2007] state that goals for the Common Policy have been efficient rule representation, permission-only rules and additive permissions, extendability of the framework and individual rule-bases. External references in the rule-base, regular expressions and repeated time intervals are currently not supported because the authors felt that they introduce complexity that might lead to inconsistencies and false assurances in the policies.

Mechanism

The GEOPRIV Common Policy is a declarative policy language that can only express positive authorisations. This allows the easy combination of distributed rule sets and eases policy management. Rule ordering is not important for policy evaluation. The whole rule base will be evaluated to make a decision, because permissions are additive. The Common Policy supports distributed policy evaluation and enforcement. However, it only describes an abstract architectural model that might be implemented in the application itself or through a distributed framework such as defined in Section 2.5.

Common Policy rules differ from other policy description rules insofar, as they do not include a direct reference to the actual resource. It is assumed that the policy is targeted at the complete set of location and presence data that would currently be available on the server, such as the last position update or location information. Subsequent protection of this data, after it has been exposed, is not part of the GEOPRIV privacy protection model.

Rules have the ability limit the visibility of certain parts of this data set, as well as to modify location data before it is exposed (e.g. in order to reduce data granularity). The exact transformation mechanisms are defined by application specific extensions to the Common Policy. The Common Policy itself defines very few attributes, namely the identity and domain of the data user and a valid time range. The policy framework must be extended by the definition of conditions, actions and transformations for specific application domains.

Language Elements

The Common Policy uses an XML representation for the description of the elements of the language, but makes it clear that the policy is targeted at being represented in a database form, where each rule represents a table row that can be easily evaluated.

We will continue to use the XML representation for the description of the language to ease the comparison. Privacy policies following the Common Policy are defined under the `<ruleset>` top-level element. Figure 2.10 provides an overview of the language elements.

Ruleset A policy is defined as the set of rules within a `ruleset`. Since the Common Policy explicitly only defines positive authorisations it is not possible to define a different default behaviour. Permissions from individual rules are combined to a permission-set. A common procedure for the combining of permissions from multiple matching rules is defined within the standard. It computes the union of all permissions and finds the highest permission value for numerical permission. It is not foreseen to deploy additional algorithms.

Rule The main unit of authorisation is a `rule`. Each rule carries an `id`-Attribute that is used to uniquely identify a rule. Rules define one precondition and two post-conditions. The *action* post-condition can be compared to the obligations found in other privacy and authorisation languages, while the *transformation* allows the specification on how the current location data needs to be manipulated by the server before it is exposed.

Rules contain the following elements:

Conditions A condition determines if a rule applies to the decision request. Only if all the stated conditions are satisfied does the rule apply, otherwise it is ignored. Conditions are structured using the following elements:

Identity The `<identity>` element defines the data user that has access to the location data. It is possible to define a list of known data users by using the `<one>` element. The `<many>` element bases the authorisation decision on the domain part of the authenticated entity. An empty `<many>` element authorises any authenticated data user. Exceptions may be specified through the <except> element.

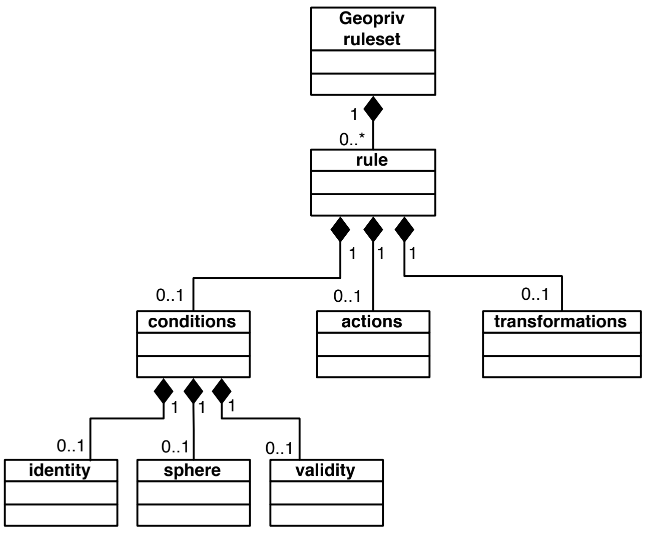


Figure 2.10: GEOPRIV language model [Schulzrinne, Tschofenig, Morris et al., 2007] - The language elements of the GEOPRIV Common Policy and their interrelation are shown in the figure. The Common Policy only provides a base framework for authorisation specifications that must be extended through application specific specifications.

Sphere The `<sphere>` element defines the current location state of the data owner. It is used to convey information about the current status, such as 'work', 'home', 'meeting', etc. and matches only if the data owner is currently in the described state. Different subsets of rules may apply for different spheres and allow the data owner to control data release.

Validity The `<validity>` element defines the time period within a rule is valid. It uses two elements `<from>` and `<until>` to set a start and end time within this rule condition evaluates to true.

Transformations The `<transformations>` element describes operations that must be executed by the location server after access has been granted in order to modify the result that is returned to the data user.

Actions The `<actions>` element defines all other operations that the location server has to perform after access has been granted and which are not `<transformations>`. They closely correspond to the concept of obligations in other privacy frameworks. `<transformations>` and `<actions>` are application specific extensions to the Common Policy.

Critique

The GEOPRIV Common Policy is not designed as a general access control language. The focus on expression of policies for location and presence data simplifies the generation and evaluation of targeted policies. However, it also limits the usefulness of the policy language if other types of access and usage policies need to be described or systems need to be designed to include such policies in future implementations. There are currently no known implementations of the standard and substantial work would have to be invested to define application specific attributes and transformation operations.

2.6.5 Summary

Most existing implementations of policy-based privacy protection schemes have so far followed the approach that the effective policy is set by the party that is collecting the data - the data user, with GEOPRIV the only exception to this rule.

P3P is a privacy language that only informs the data owner about the privacy promises that the data user is willing to make. In P3P there exists no direct link between the promises about the treatment of collected data to the actual data protection mechanisms.

EPAL was constructed to remediate this situation and provide enforceable policies. EPAL is an expressive privacy language that is able to capture the semantics of privacy policies and generate appropriate access decisions from a policy-base created by an enterprise privacy manager. However, the approach to base policies exclusively on semi-fixed data- and user-categories makes an implementation in an open and distributed environment difficult.

Anderson [2005] compares EPAL and XACML and comes to the conclusion that although EPAL is directly targeted at the expression of privacy policies, all relevant privacy policies could also be expressed by using XACML. XACML also has the great advantage to allow policy decisions based on actual resource content (for XML-data) and flexible user attributes. EPAL offers only a subset of the functionality of XACML and the GEOPRIV Common policy is to narrowly scoped. We can therefore conclude that XACML is a very powerful policy language that is well suited to the expression of generic privacy policies.

2.7 Conflicts in Authorisation Specifications

Authorisation specifications are the basis of authorisation decisions. In order to form a conclusive decision, it must be possible to derive a non-ambiguous evaluation from the given specification - the specification must be *decidable*.

One way to reach the *decidability* criterion would be to make sure that the authorisation specification itself is free of conflicting authorisations - the policy should be *consistent*. Alternatively, we could use a *conflict resolution mechanism* to resolve authorisation conflicts as they occur during the evaluation of the policy base.

Conflicts in specifications arise from a number of reasons such as errors in the specification, unreachable goals, differences in policy sources and scope, or through changes to the policy during lifetime management. Distributed systems can contain multiple subsets of the authorisation specifications that introduce inconsistencies, while each one alone is conflict free. For example, an enterprise policy could contain different conflicting departmental policies that in itself are conflict free. Another source of conflicts comes from the evaluation of dynamic attributes, whose value are only known at evaluation time.

The underlying source of conflicts is typically the simultaneous support for positive and negative authorisations in the policy. While the support for negative authorisations can greatly simplify the task of policy writing, it can also introduce inconsistencies in the policies where an action upon a resource might be simultaneously allowed and denied for a given access request.

We already hinted at the two approaches that are generally taken to resolve conflicts in the authorisation specification: a) avoid conflicts through careful policy creation and selected policy language features and b) try to solve conflicts as they arise.

2.7.1 Conflict-free Authorisations

If it is possible to guarantee that the policy rule base itself is conflict free, no conflicting authorisations can be derived from it. The following approaches might be used to generate conflict free policies:

Monotonic authorisations An authorisation policy using monotonic authorisation has a policy base that exclusively grants P^+ or denies N^+ access rights. Since it will not be possible to grant or deny an access right at the same time, it is not possible to derive conflicting authorisations from such a policy base (cf. Formula 2.2 in Section 2.4.2).
Granting only monotonic authorisations in the policy eliminates the underlying source of conflicting rules. However, it also makes a policy language much less expressive. The easy specification of exceptions to general rules (e.g. all members of the department have access, except Mr. Jones) is no longer possible. Permissions either have to be spelled out completely for each subject or subjects need to classified into groups with identical permissions. This then generates additional administrative overhead because subject might need to be reclassified as authorisations change.

Explicit priorities Another approach for the generation of conflict free policies is the assignment of explicit and unique priorities to authorisations by the policy author in order to define a precedence relation. This way any policy can be made consistent, because simple rule ordering can be used to derive an authoritative decision. However, it is hard for the policy administrator to assign meaningful priorities and inconsistencies can arise in distributed systems that have more than one policy source. Inconsistencies may also occur over the lifetime of the policy when rules will be inserted or deleted [Lupu and Sloman, 1999].

2.7.2 Conflict-resolution Mechanisms

Policies for distributed systems have to be managed by different entities and there might exist considerable overlap between the sets of subjects, actions and resources that are used within different policy-bases [Lupu and Sloman, 1999]. It is also often valuable to have the ability to express positive and negative permissions in policy rule bases. So, it is not always possible or desirable to define policies that are completely conflict free.

Another strategy is therefore to allow conflicts in policies to exists and to solve occurring conflicts when they arise during actual access requests. A meaningful access-decision is derived through the application of some decidable algorithm that determines which authorisations apply to a particular access decision and which can be ignored. These conflict resolution schemes can be roughly classified in schemes that deploy a fixed resolution strategy and schemes, where conflicts are resolved using changeable algorithms.

Fixed conflict resolution strategy

Several different conflict resolution strategies have been found to be useful. The most widely implemented and used are the *ordered execution* and the *negative authorisations take precedence* mechanisms that will be explained next. Several other mechanisms can be found in the literature, e.g. in Vimercati, Samarati, and Jajodia [2005].

Ordered execution Policy rules can be evaluated in a deterministic, sequential fashion and policy evaluation is stopped once an authorisation decision is reached. This strategy ensures that no conflicting decisions can be derived from the policy base. However, some negative effects exist: Policy evaluation now depends on rule ordering. It becomes the responsibility of the policy author to ensure that more specific rules precede general rules in the policy base, otherwise these rules could become unreachable. Policies from different sources can not be merged automatically, because different results would be derived depending on the evaluation order of sub-policies [Barth, Mitchell, and Rosenstein, 2004].

Negative authorisations take precedence Negative authorisations are mainly used to express exceptions to a general rule. For example all member of the development team are allowed to check-in software builds, except for newly hired software developers. This authorisation can be described through a general positive rule that grants permission to check-in software builds and a more specific rule that forbids check-in for a subset of the developers. Precedence is assigned to the negative authorisations that deny access. One drawback of this approach lies in the fact that this is an implicit, inflexible solution that does not work if some policies need to define positive exceptions.

Dynamic conflict-resolution through explicit algorithm definition

The most flexible handling of authorisation conflicts can be achieved through the use of explicitly defined conflict resolution algorithms that can be applied to the whole policy base or parts of it. Jajodia, Samarati, Sapino et al. [2001] have developed a framework that allows the dynamic definition of a conflict resolution and decision strategy depending on the requirements of the particular authorisation scheme. The policy decision component analyses conflicting authorisations and derives a decision in accordance with the conflict-resolution algorithm that has been set for the policy or the policy part.

A policy can explicitly specify the dynamic conflict-resolution algorithm that will be used to evaluate different parts of the policy rule base. Examples for such algorithms include: *Permit-takes precedence*, where permissive rules take precedence over rules that deny access and *Denial-takes precedence*, where rules that deny access take precedence over permissions. It is also possible to define more elaborate algorithms such as the *Most-specific takes precedence*, where more specific rules take precedence over general rules.

The XACML policy language [XACML-2.0, 2005] uses so called *combining algorithms* to allow the policy administrator to set the current conflict resolution strategy. It is also possible to define and implement new algorithms for use with a particular policy base.

2.8 Enforcement of Authorisations

Policy languages provide the basis for the determination of an access control decision. They express the constraints that are in effect if subjects want to use resources on the protected system. However, an authorisation decision in itself only describes the course of actions that our enforcement system should take and requires the presence of an enforcement mechanism that makes this decision effective through the implementation of an enforcement function.

The literature distinguishes between two types of enforcement mechanisms, depending on where the enforcement function is located in respect to the application that requires access to the protected resource: inline and external policy enforcement.

2.8.1 Inline Policy Enforcement

The inline enforcement mechanism places the enforcement functions directly into the application code to enforce policy decisions. Policy checks are implemented as a policy checking function that, if invoked, returns a policy decision. Alternatively the policy check can also be implemented as a wrapper function that encapsulates the protected function.

The hard-coding of policy enforcement functions ensures very efficient policy checking with comparatively little overhead. Authorisations can be tailored to the access control needs of the individual application. However, the reuse of inline mechanisms between different applications can be difficult. Implementing inline policy checks requires great care from the application designer, because a missing or incorrectly implemented policy check might compromise application security or bypass the privacy protection for application data. If a workflow requires the interworking of different applications, all these applications need to implement consistent policy checks. If programs change and are rewritten the policy enforcement functions must be adapted to reflect any new or changed access restrictions.

Inline mechanisms usually require access to the source code, which makes the reuse of existing code-libraries and object-level code difficult. There are some research projects under way that study the feasibility of adding policy checking code after the actual implementation stage. Lehmann and Thiemann [2006] describe the possibility to add Bytecode-checks for Java programs at run-time.

2.8.2 External Policy Enforcement

Section 2.5 has outlined different access control architectures for distributed systems. These architectures commonly separate the policy decision functions from the enforcement functions in separate architectural modules. Under the Framework for Policy-based Access Control [Yavatkar, Pendarakis, and Guerin, 2000] a *Policy Decision Point* (PDP) implements an access decision function that evaluates the current policy. The architectural counterpart for the enforcement is the

Policy Enforcement Point (PEP) that makes the policy decisions effective. The PEP controls and effectively restricts all data accesses from principals in the system in accordance with the access policy.

Such an enforcement function is usually implemented as a *Reference Monitor* (see [Anderson, 1972]). A reference monitor enforces the access authorisations between the principals and the other elements of a system. According to Anderson, a reference monitor implementation must have the following properties:

tamper-proof it should not be possible to alter it, otherwise the integrity can not be guaranteed (possible alterations must be reliably detected and reported)

non-bypassable it must mediate all accesses to the system and its resources and must always be invoked

verifiable it must be of limited size to be susceptible of rigorous verification methods to ensure its correctness

Definition 8: Reference Monitor

A trusted system component that validates each and every request to system resources against those authorised for the subject.

Access control mechanisms based on the Reference Monitor principle can be implemented at different levels in the computer system architecture [Anderson, 2001]. Figure 2.11 distinguishes four different levels where access control systems might be implemented. Each system level has distinct advantages and disadvantages for the implementation of access control mechanisms.

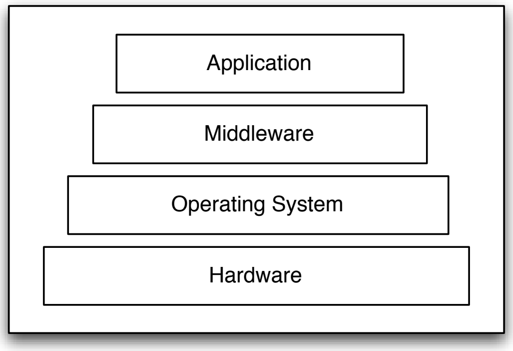


Figure 2.11: Access Control Mechanisms at different system levels - it makes sense to implement protection mechanisms at different system levels, depending on their protection requirements. Access control mechanisms at higher layers are conceptionally closer to the protection requirements of the user, while low-level hardware mechanisms can effectively control memory and device access.

Hardware Level Access Control

Access control mechanisms that are implemented at the hardware level can provide very effective protection and enforce access decisions independent of any software-support from the operating system or application software. The protection mechanisms are hard to circumvent and can guarantee policy enforcement. However, since hardware mechanisms operate at a very low conceptual level they have difficulties to enforce data usage restrictions for access decisions that are dependent on multiple high-level attributes, such as the support for roles, the evaluation of external attributes and others. In order to support such concepts the hardware mechanisms need to rely on the operating system to provide a mapping between the entities and attributes known by the hardware protection mechanism and those known by the operating system. The Trusted Computing Group [Trusted Computing Group, 2007] has developed a specification that provides a dependable interface between security functions provided by a *Trusted Platform Module* and higher-level functions provided by the operating system.

Hardware mechanisms are the ideal choice for the protection of very sensitive information that require a certified security level. However, the strong protection properties of hardware mechanisms can become a burden if dynamic policies and access control models need to be supported. The implemented mechanisms are strongly tied to the actual hardware and it might be difficult or impossible to update if new access control models need to be supported.

Operating System Level Access Control

Mechanisms that operate at the operating system level implement standard access control mechanism, such as Access Control Lists. These mechanisms are usually very efficient and can provide strong protection properties. Their main drawback lies in the fact that they are mainly targeted at the requirements of the operating system. They provide protection from unauthorised system access and process separation for different users of the system. The granularity of access restrictions is usually file or device based. Access restrictions are formulated and enforced for whole documents, so that it is not possible to enforce fine-grained access restrictions that offer different protection properties for certain document-parts. Access control mechanisms for operating systems must be very performant since they are invoked for every resource access. This has a limiting effect on the complexity of the implemented mechanisms.

Most general purpose operating systems follow the Discretionary Access Control model and do not enforce data flow protection (cf. Section 2.3.2). Adapting or changing operating system protection mechanisms is difficult because they are usually under the control of the OS vendor and must support other legacy applications that might require the presence and expected behaviour of certain mechanisms.

Middleware Level Access Control

Middleware components are usually placed between the operating system and the applications. The most widely implemented access control mechanisms at the middleware level are access control features in database systems.

The generation of access decisions at the middleware level can implement advanced access control mechanisms for the generation of access decisions and can evaluate complex access policies.

The mechanisms are comparatively easy to adapt and can provide a better granularity for access restrictions down to the level of individual data records for most database-systems.

Application Level Access Control

Application level access control mechanisms provide the greatest flexibility for the creation, evaluation and enforcement of data access restrictions, because at this level exists the greatest knowledge about data access patterns and usage purposes. Access control mechanisms can be precisely tailored to the specific problem domain.

However, the implementation of application level access control mechanisms also requires direct access to either the application code base or a dedicated API, which is not always possible. Uniform implementation of protection mechanisms would be required for different classes of applications. Otherwise it is not possible to offer effective data protection in work-flow scenarios. When access control components are directly referenced from the applications, the changing of functionality may require the patching and secure distribution of many dependent applications.

2.8.3 Summary

The concept of a reference monitor is a useful abstraction for the implementation of the enforcement function of an access control mechanism, because it provides a clean separation of duties. The policy enforcement through a separate enforcement module also makes the implementation more robust, since it allows to develop and test the enforcement function independently from the application logic. The reference monitor can be implemented at different architectural levels in the system. Deciding which is the most suitable level is not easy because a number of tradeoffs have to be made.

Enforcing access control decisions at higher layers provide useful abstractions and is conceptionally closer to the protection requirements of the user, while low-level hardware mechanisms provide protection for function calls and can effectively control memory and device access. Lower level mechanisms have difficulties to incorporate high level abstractions such as roles and conditions and usually only enforce a relatively simple (binary) access policy. Low-level as well as high-level protection mechanisms require operation system support for the identification of principals and other system attributes, such as the current system time and geographic location. The protection from the enforcement function becomes ineffective, if these functions of the operating system can be compromised.

When we compare the general protection strength of the system layer implementations, it can be noted that attacks that are started from a lower system layer are usually able to bypass the higher layer access control: an attacker that has full access to the operating system of a machine can install debugging software to analyse memory dumps from applications that are protected via middleware or application-level access controls.

Table 2.3 gives an overview of the different properties of access control mechanisms. In Chapter 5: *Java PrivMon - Privacy Protection for Personal Health Records* we will propose an approach for a client-side reference monitor implementation where policy evaluation and enforcement are based on the unique features of a language-based security model that supports application sandboxing and merges middleware with application-level access control mechanisms to provide a platform that can be used by different applications.

This approach is later extended to also cover cases where data processing is done in a server-based scenario where we use Aspect-oriented Programming (AOP) and the Java Reflection API to intercept data accesses in existing application frameworks and provide a way to enforce data user defined privacy policies for business applications [Scheffler, Schindler, and Schnor, 2012].

Table 2.3: Comparison of Access Control Mechanisms

	Hardware	**Operating System**	**Middleware**	**Application**
Granularity	Device access	Files + Devices	Records	Sub-document + Devices
Policy flexibility	Very Low	Low	High	Very High
Scope & Protection Level	Highest	High	Medium	Low
Circumvention	Very difficult	Difficult	Difficult	Easier
Portability	Low	Low/Medium	High	Language dependent

2.9 Conclusion

Our analysis of available privacy enhancing technologies, access control methods and privacy policy languages comes to the conclusion that currently no privacy protection framework exists that can offer protection for data release scenarios where the data owner defines his or her privacy preferences and wants to base privacy protection on the Owner-Retained Access Control (ORAC) model.

Electronic data is easily copyable and electronic copies are indistinguishable from the original data. These copies can be easily distributed without the knowledge of the data owner. The Discretionary Access Control (DAC) mechanism, implemented by most current operating systems, does not implement data flow restrictions and anybody who can access data is also able to copy it. Intentional or unintentional misuse of data by authorised subjects thus becomes very easy. The sustained protection of data access and data use over the lifetime of sensitive private data is the main challenge for privacy protection.

An externalised data access and usage control, based on the Reference Monitor model, is therefore needed to enforce the privacy requirements for electronic data. The reference monitor ensures that data access can not bypass the authorisation system and data flow restrictions are enforced. The UCON_{ABC} model is one representation of an access control model that recognises the need for data flow control through the incorporation of trusted, server-side and client-side protection measures. Data access by users is mediated via a client-side reference monitor to provide data usage control, making it suitable for the enforcement of privacy policies.

The access control mechanisms (both, client- and server-side) must be able to automatically enforce authorisations. It therefore becomes necessary to explicitly formulate the required authorisations in a policy. Policy languages provide powerful and flexible authorisation specification mechanisms that can also be used for the description of privacy policies. Based on our evaluation of policy languages in Section 2.6, we have chosen to use the eXtensible Access Control Markup Language (XACML) within our own privacy enforcement framework. All privacy policies of our use cases can be expressed as XACML policies (cf. Sections 5.4.2 and 6.2.4).

An important part that is missing from the literature is the discussion of usability and safety issues of privacy policy creation and management. Many privacy protection schemes require the existence of a dedicated policy administrator or simply assume the pre-existence of a suitable set of authorisations. This approach may be suitable for privacy enforcement schemes that deploy data user-defined privacy policies, but it creates problems for our approach of data owner-defined policies, where ordinary people have to define their privacy requirements.

In the following chapters we will rectify this omission and develop and discuss our approach to policy administration for data owner-defined privacy policies. This approach will then be used for policy administration and maintenance within our own privacy enforcement framework.

3 Owner-Retained Access Control Policies

In this thesis we assume the position that privacy is the right of individuals to determine for themselves when, how and to what extent information about them is communicated to others [Ashley and Karjoth, 2003]. The Federal Constitutional Court of Germany confirmed this view in its ruling against the 1983 census and established the principle of *informational self-determination* for its citizens.

The German Federal Constitutional Court ruled[1] that:

> ”The free development of the individual in the context of modern data processing requires the protection of the individual against unlimited collection, storage, use and disclosure of his/her personal data. This protection is based upon the basic rights specified by Article 2 (1) and Article 1 (1) in the Basic Law for the Federal Republic of Germany. This basic right safeguards the authority of the individual to generally decide about the disclosure and use of his or her personal data.
>
> [...]
>
> The right to informational self-determination is not without bounds. [...] The individual has to accept limitations in his or her right to informational self-determination in case of overriding public interest.” [BVerfG 65,1, 1984]

In the following sections we analyse the different approaches of authority over privacy policy creation in more detail, where we distinguish between policies that are controlled by the data user and policies controlled by the data owner. We analyse challenges and solutions for the problem of policy creation and maintenance in an owner controlled scheme and evaluate trust relationships for privacy policies.

3.1 Data Release under a Privacy Policy defined by the Data User

Data user-defined policies, where the user and custodian of the personal private data defines the protection properties are well established. The data user has a clear understanding of the type

[1]Original text of the ruling found in [BVerfG 65,1, 1984], Part C, Section II:
“Freie Entfaltung der Persönlichkeit setzt unter den modernen Bedingungen der Datenverarbeitung den Schutz des Einzelnen gegen unbegrenzte Erhebung, Speicherung, Verwendung und Weitergabe seiner persönlichen Daten voraus. Dieser Schutz ist daher von dem Grundrecht des Art. 2 Abs. 1 in Verbindung mit Art. 1 Abs. 1 GG umfaßt. Das Grundrecht gewährleistet insoweit die Befugnis des Einzelnen, grundsätzlich selbst über die Preisgabe und Verwendung seiner persönlichen Daten zu bestimmen.
[...]
Dieses Recht auf "informationelle Selbstbestimmung" ist nicht schrankenlos gewährleistet. [...] Grundsätzlich muß daher der Einzelne Einschränkungen seines Rechts auf informationelle Selbstbestimmung im überwiegenden Allgemeininteresse hinnehmen.”

and amount, as well as the potential uses of the data that need to be collected. Based on this information the data user defines a privacy policy and announces this policy to the data owner in form of a privacy promise [Barth and Mitchell, 2005]. It is the responsibility of the data user to enforce this promise and refrain from using the data for other than the intended and agreed purposes.

Data user-defined policies are widely used. Examples for this type of privacy policies include informal privacy policies, such as described in Section 1, as well as policies that use a more formal representation such as the privacy policy language defined by the Platform for Privacy Preferences [P3P v1.1, 2006].

Benefits of data user-defined policies

Data-use policies that are defined by the user of the data have the following advantages over policies defined by the data-owner:

Efficiency - The data use policy has to be declared and referenced only once. The operation and effectiveness of the policy is relatively easy to audit, all possible restrictions are known at policy creation time. There is no need to store and reference different policies between data owner and data user. The data user has no need to negotiate policies and provide protection properties that are stronger than an average protection profile.

Conformity - Policies can be very well adapted to the organisational procedures and there is no risk that certain workflows can not be executed because of policy restrictions.

However, data owner-defined policies have deficits and we show in the following that the current approach does not always deliver adequate privacy protection.

Challenges for data user-defined policies

It is possible to identify several areas of concern that are not adequately covered by policies generated under the control of the data users. The following list provides an overview of potentially problematic areas:

Unfounded trust - A data user defined policy scheme, such as a P3P policy, states only a privacy promise that has no obligatory link to privacy enforcement. Data owners can not be certain that an adequate protection enforcement exists.

The data owner has to trust that the data user competently enforces the privacy policy and has deployed effective data access protection. It has been shown that data user-defined privacy protection schemes are not effective in their protection against a trusted, but careless data user [Anton, He, and Baumer, 2004].

Weak enforcement - Data release policies specify a so called purpose-binding. Data should be collected and used for a specified purpose only [Ashley and Karjoth, 2003]. As purposes specify very domain specific concepts, the data owner needs to make a judgement about the validity of the stated purpose. The policy enforcement scheme needs to map data access requests against these defined purposes. However, most access control mechanisms do not easily support the interpretation and enforcement of purposes.

Policy interpretation - It becomes the responsibility of the data owner to understand, compare and verify different privacy policies created by different data users that might utilise different, incompatible schemes. If, for example, a patient is sharing medical data with various healthcare providers, practitioners and insurance companies, each of these entities might use their own tools and mechanisms to manage their privacy policy [Cranor and McDonald, 2008].

Jensen and Potts [2004] have shown that privacy policies differ greatly from site to site and often address issues that are not relevant to the data owner. They find that form, location and legal context of published privacy policies make them unusable as decision-making aids for a person that is concerned about privacy.

Diverging goals - The data user might collect, use and share more data than necessary and store data longer than strictly necessary for the completion of the task.

Limited choice - The privacy policy is set by the user of the data, which could have other goals for data protection than the data owner. The data user could be forced to accept a privacy policy that is sub-optimal for the protection of their own protection requirements. The data owner has no power to force the application of a privacy policy.

Weak negotiation position - With data user generated privacy policies the data owner can not negotiate or force a specific requirement in the privacy policy of the data user. If the data user offers no or only an unacceptable privacy policy, the data owner can only choose between refraining from the data release or releasing data without adequate protection.

Unstable policies - The privacy policy is set by the data user and can be changed anytime to offer a different level of privacy protection.

Changing authority - The authority over data protection rules and the collected data is handed over to the data user. Control by the data owner is only exercised at the time of data release.

This list of problematic issues makes clear that even with trusted service provider offering data user-defined privacy policies, there still exist areas for improvement. Especially the weak enforcement binding opens the possibility for intentional and unintentional data breaches.

In the data release taxonomy presented in Section 1.1, we have made the distinction between data releases under privacy policies set by the data owner and policies that are set by the data user. The following section analyses the advantages and disadvantages of policies that are under the control of the data owner.

3.2 Data Release under a Privacy Policy defined by the Data Owner

Data owner-defined policies derive their name from the circumstance that it is the data owner that defines the data release policy. The data owner might have a clear understanding of his or her privacy preferences and requirements. If private data is collected by the data user, the data owner should be able to formulate these requirements in a privacy policy and distributed the policy together with the data to the data user.

Benefits of data owner-defined policies

The following advantages can be identified for data owner-defined policies in comparison to policies that are defined by the data user:

Higher specificity - Data owner-defined policies have the ability to specify very detailed policies for individual data items. It is no longer necessary that a privacy policy needs to apply for all data items collected over all data owners. Policies can be specifically adapted to reflect the personal requirements of the data owner.

Expressiveness - The data owner actively formulates the data release policy through the expression of his or her specific privacy requirements, rather than silently consenting to the uniform policy of the data user.

Extended referencing - The data user needs to store and retain an individual policy together with the data that has been received from the data owner. This policy can be passed to other data users when the data is moved, allowing for distributed policy referencing.

Direct trust - With data owner-defined policies it becomes possible to express direct trust relationships between the entity that owns the personal identifying data and the user of this data.

Challenges for data owner-defined policies

The application of owner-defined privacy policies is not without hurdles. Data owner-defined policies requires a higher level of cooperation between the involved parties. Besides the technical difficulties of expressing and communicating the privacy policy, it must be made certain that the actual privacy policy is enforceable:

Efficiency - Many data owner specific privacy policies need to be stored, referenced and evaluated over all collected data items, whereas a data user-defined policy scheme only requires the referencing of a single privacy policy.

Enforceability - Data owner-defined policies are usually not well adapted to the organisational procedures of the data user. The defined policies are typically not tied to the procedures of the data users. It might even be possible for the data owner to specify a policy that is not enforceable by the data user.

Policy creation effort - The data owner needs to define suitable privacy policies that capture his or her privacy requirements. This creates an additional burden for the data owner and data owner must be educated about the protection properties of privacy policies. Policies need to be revised as privacy requirements change over time.

Default protection - The policy scheme must provide at least minimal protection, even when data owner choose to opt-out of protection or mistakenly create a faulty protection profile.

This analysis shows that data owner-defined policies offer interesting properties that might enable privacy protection that is tailored to the privacy preferences of the data owner. We base our preference of data owner-defined privacy policies on the advantages these policies possess over policies that are defined by the data user. However, their application in the area of privacy enforcement has some consequences and challenges that must be understood and solved in order to reach the best possible privacy protection. Usability aspects of policy creation and policy management tasks have to be analysed and taken into account for the development of our particular policy protection scheme. The next section highlights these aspects and presents our particular solutions for some of the potential problems.

3.3 Using the ORAC Access Control Model for the Definition of Data Owner-defined Privacy Policies

McCollum, Messing, and Notargiacomo described in 1990 an alternative access control scheme where the access policy for a resource is determined by its owner or creator and not by a system rule or the security administrator (cf. Section 2.3.3). The creator of an object sets the access policy which can not be revoked or changed later by other users that have access to the object.

When we base a privacy protection scheme on the implementation of the Owner-Retained Access Control (ORAC) model, we give the data owner the ability to create privacy policies that reflect their personal protection needs, as was already discussed in Section 3.2. However, the implementation of the ORAC model burdens the data owner with the responsibility to create an adequate protection policy and manage this policy over time. This is an extra effort that is not required if data is released under a data user-defined policy. The following list explains some of the challenges for data owner generated policies and compares them to data user defined policies:

Convenience - Privacy policies defined by the data user are convenient. The data owner simply accepts or denies an existing policy. He or she does not have to create their own policy from scratch every time some data item is released to the data user.

Global reach - If the data user has defined a privacy policy, all data items that are released by the data owner automatically fall under the protection of this policy. This is not the case with ORAC policies. The data owner has to choose and compose an adequate policy each and every time, otherwise released data might not be protected.

Safety - ORAC policy administrators are ordinary people that have no particular training in privacy policy creation and maintenance. This could lead to situations, where defined policies do not correspond to the intention of the data owner or might actually be harmful, such when privacy protection is lost through the accidental deletion of rules.

With greater power there also comes greater responsibility and several challenges need to be addressed in order to implement a data owner-defined policy scheme:

- The data owner needs to be guided in his authority to specify permissions to provide safeguards against the specification of disadvantageous policies.

- It should not be possible to opt-out of basic protection.

- It should not be possible to specify policies that are not enforceable by the data user.
- Authorised data use should be influenced as little as possible to achieve a high acceptance rate for the system, while at the same time reliably prohibiting unauthorised data access.

Traditional access control models focus on the evaluation and enforcement of authorisations and to a large extent assume the pre-existence of an access control policy. The access policy itself is created and exists outside of the access control architecture that enforces it.

The XACML specification recognises the importance of the policy administration process through the definition of a special entity in the XACML architecture, the Policy Administration Point (PAP) [XACML-2.0, 2005]. It is the purpose of the PAP to provide the policy object to the Policy Decision Point (PDP). The PAP has complete authority over policy creation and the standard does not further specify the administrative actions and its implementation – there are no restrictions imposed on the policy creation process itself. Figure 3.1 shows the location of the architectural elements within the XACML architecture.

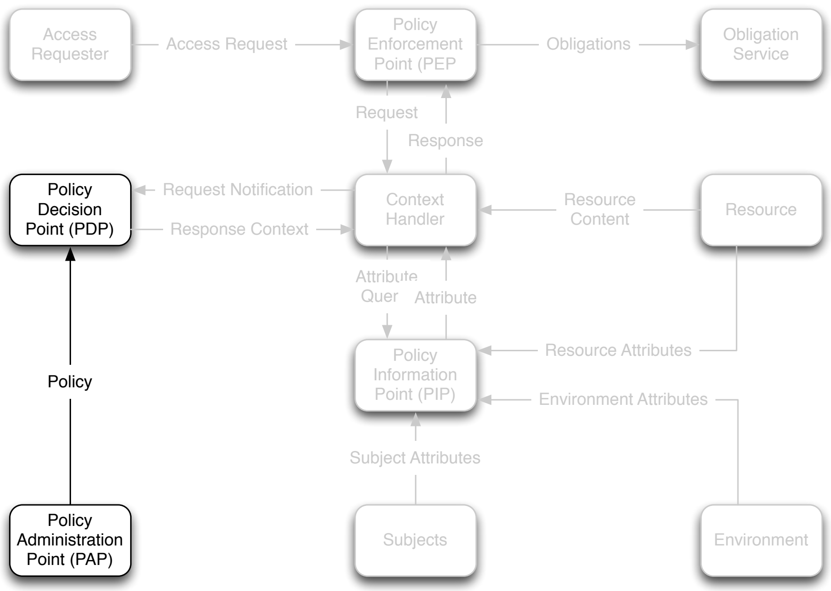


Figure 3.1: Policy Administration in the XACML Architecture - The XACML Architecture (see Section 2.6.3) defines a Policy Administration Point (PAP) that provides the policy object to the Policy Decision Point (PDP). The PAP has full authority over the policy and is not restricted in its administrative power by the architecture.

Other current authorisation architectures assume the presence of a policy administrator who is responsible for the design and assessment of privacy policies. The standards for EPAL [EPAL 1.2, 2003] and P3P [P3P v1.1, 2006] explicitly specify the role of a *Privacy Administrator*.

The privacy policy administrator creates and maintains the access rules that are later evaluated by the PDP and enforced by the PEP. The policy administrator has unrestricted administrative powers over the policy administration process, which results in the ability to create, modify and delete arbitrary policy specification rules.

As we have already discussed, one important prerequisite for the usage of owner-defined policies, is the necessity for the data owner to create an adequate protection policy and manage this policy over time. While a professional policy administrator is expected to have the necessary skills to create, review and adapt formal access policies, the same can not be said about the ordinary computer users that want to protect his or her personal data.

Usability aspects for policy creation and policy management tasks have to be taken into account when a protection scheme for data owner-defined policies is developed. We think that we can fulfil this requirement by restricting the administrative authority of the policy administrator (the data owner) to the formulation of valid, favourable and enforceable policies, so that the task of policy creation can be handled safely by non-experts. The developed mechanism should be flexible, so that it can be adapted as needed to different requirements and many application domains. The implemented restrictions should be directly visible, so that they can be questioned and evaluated by the data owner as well as external reviewers.

3.4 Trust Model

Privacy policies can be seen as the expression of trust relationships between data owner and data user. The implementation of a privacy protection mechanism serves as a trust building tool between data owner and data user and lets data owners exercise control over who has access to their personal private data. The policy allows the data owner to explicitly formulate his or her trust relationship with the data user. A privacy protection mechanism based on the ORAC model ensures that an authenticated, non-malicious user will be granted data access - in accordance to the owner-defined policy. Not authorized data usage will be prohibited by the system.

The expression of granular trust relationships in the form of privacy policies serves the following purposes:

a) It lets the data owner articulate and document personal privacy concerns.

b) The explicit definition of the privacy policy allows the data owner to reason about the trust relationship with the data user.

The support of Owner-Retained Access Control policies is also beneficial for the data user. It allows the data user to focus on the guaranteed enforcement of individual data owner-defined policies, without having to develop a single privacy policy that applies to all data owners and is based on a compromise between the individual privacy requirements of the data owners and the data processing needs of the data user.

3.4.1 Sources of Trust

An important prerequisite for the correct attribution of trust is the successful and reliable enforcement of restrictions on data usage by the reference monitor. The data owner needs to trust the reference monitor to securely enforce the stated privacy policy – it acts as a *trusted system*:

Definition 9: Trusted System

A system that operates as expected, according to design and policy, doing what is required – despite environmental disruption, human user and operator errors, and attacks by hostile parties – and not doing other things. [Shirey, 2007]

Beyond the trust in the correct operation of the privacy enforcement, the data owner also has a certain level of personal trust in the data user itself. While the trust in the policy enforcement system is usually a binary decision between trust and no trust, which could be influenced by external attestation and audits, the personal level of trust in the data user is a more complex matter:

Definition 10: Trust

Generally, an entity is said to 'trust' a second entity when the first entity makes the assumption that the second entity will behave exactly as the first entity expects. This trust may apply only for some specific function.
... [Shirey, 2007]

The level of personal trust a data owner shows towards the data user will be expressed in the privacy policy. The properties of this particular type of trust will be explored in the following sections.

3.4.2 Properties of Trust Relationships

Trust relationships can be categorised depending on their level of transitivity. Many technical systems use a transitive definition of trust, because it lets the system designers reason about the security properties of a trusted system. In a transitive trust relationship such as the chain of certificates issued by different Certificate Authorities (CA) of a Public Key Infrastructure (PKI), the following assumption holds ($\vDash$ is used to notice a trust relationship between subjects):

$$(A \vDash B) \wedge (B \vDash C) \Rightarrow (A \vDash C) \tag{3.1}$$

Many data user-defined privacy polices assume such a transitive relationship to some degree. They allow the passing of personal private data to associated organisations for purposes such as data processing and marketing. This can be seen in the following example:

> "We employ other companies and individuals to perform functions on our behalf. Examples include fulfilling orders, delivering packages, sending postal mail and e-mail, removing repetitive information from customer lists, analysing data, providing marketing assistance, providing search results and links (including paid listings and links), processing credit card payments, and providing customer service. They have access to personal information needed to perform their functions, but may not use it for other purposes." [Amazon, 2008]

When we examine the privacy statement from Amazon, we find that it contains no identifying information about these potential other data users. The data owner therefore has no direct knowledge about data handling practices of the data users other than the information that these entities are trusted by Amazon.
Marsh [1994] has shown that trust relationships between independent agents are typically not transitive. Each trust relationship reflects an independent decision made by an agent and this decision is made separately for each pair of subjects. This can be explained by a simple example: If a person trust a certain individual, this trust does not automatically extend to other individuals trusted by this person. We must therefore assume that personal trust is not transitive and it is safe to conclude that the following equation expresses the trust-relationship between individuals:

$$(A \vDash B) \wedge (B \vDash C) \not\Rightarrow (A \vDash C) \tag{3.2}$$

It also follows that trust relationships are not symmetric. If person A trusts person B, this does not mean that B similarly trusts A:

$$(A \vDash B) \neq (B \vDash A) \tag{3.3}$$

3.4.3 Dynamics of Trust

Trust relationships between subjects are not static. As privacy requirements change over time, it becomes necessary to re-value the current privacy policy. A change in trust could result in an update to the privacy policy or the complete withdrawal of the data usage decision. Depending on the origin of the privacy policy we find strong differences in the handling of change in trust:

Data user-defined policies assume a stable trust relationship between data user and data owner. The data owner has effectively handed the authority over the access policy to the data user, it becomes very difficult to change or suspend this trust relationship at a later time. If no opt-out mechanisms are supported by the data user, the data owner has no mechanism to revoke his trust.

Data owner-defined policies are better suited to handle dynamic changes in the trust relationship. A data owner can start with a very restrictive privacy policy, which can be loosened at a later time when the data user has been proven trustworthy. Similarly it is possible to tighten access policies if trust wanes. Fine-grained permissions allow the construction of access policies that correspond to the personal trust level of the data owner.

The data owner needs to communicate these changes to the data user and this generates two problems: The data user may find ways to use the old privacy policy and ignore the new one, or it may not even possible for the data owner to contact the data user about these changes.

For the reliable support of changing trust values it is therefore necessary, that the privacy scheme supports mechanisms to limit the lifetime of a policy during which the data user has access to the data. The policy lifetime has to be renewed in regular intervals and this allows the data user to review and adapted the trust-relationship, if necessary.

3.4.4 Managing Trust Domains

Real world trust in subjects, organisations and services is typically limited to certain areas of expertise and experience. This means that trust is bound to a specific domain and does not extend automatically and equally over different domains.

If, for example, a medical practitioner is trusted by the patient to correctly diagnose and treat a specific illness, the patient may not trust the same practitioner to be a computer security expert that can install and maintain a secure and trusted computing environment for the safe storage of the medical data.

From the perspective of privacy protection this means that the patient trusts the practitioner to see and read the private medical report. But, he or she does *not* trust the practitioner that this report will be stored safely on the computer system, so that it can not also be accessed by other parties and data is protected from inadvertent exposure.

Partial Trust and Trust Domains

Trust domains can be build around the domain of competence of a data user. Pretty Good Privacy (PGP), for example, reflects such a split trust-domain in its key signing model [Callas, Donnerhacke, Finney et al., 2007]. A user can trust the key to correctly attest the identity of a person, but does not trust other keys that are signed by this key, because the user has no trust in the competence of the key-holder to securely sign the keys of other users.

Managing separate trust domains has already been proposed, for example, by the Electronic Health Record (EHR) communication standard EN-13606-4 [2007]. The standard defines several Access Domains for Electronic Health Record data, as can be seen from Table 3.1.

Table 3.1: Access Domains for Electronic Health Record Data as defined by EN-13606-4

Access Domains for Health Record Data
1. Private entries shared with General Practitioner
2. Entries restricted to sexual health team
3. Entries accessible to administrative staff
4. Entries accessible to clinical support staff
5. Entries accessible to direct care teams
6. Private entries shared with several named parties
7. Entries restricted to prison health services

Such standards normally only capture a very narrow and specific domain knowledge. It might be possible to base such trust domains on ontologies to make them independent from a particular field of expertise. However, it is not clear if people, which are no domain experts in this particular field, will find it easy to make the right decisions for privacy protection based on these ontologies.

We think that a better way for the management of different trust domains consists in the separation of data into domain specific containers and the binding of authorisations to a specific data user, since it is very unlikely that an ordinary data owner can understand and maintain sensible privacy policies based on domain-specific, formalised specifications. The separation of data in domain specific containers enables the data owner to effectively build and manage separate trust domains that are within their own field of expertise.

3.5 Efficient Policy Specification and Maintenance through the Use of a Precedence Relation for Sub-Policies

Giving unrestricted administrative power to the data owner might create problems when delete operations on the policy base lead to data protection loss or cause permissions for the data owner to be revoked. If, for example, a data owner could revoke his or her permissions to access the policy, the working order of the system could be seriously affected. Similarly, a data owner could decide to delete all existing policy rules, creating a state, where no privacy protection is provided.

One approach to make policy creation and maintenance more user friendly could be the restriction of the administrative power of the policy creator. These restrictions could be implemented through rules that are implicitly defined in policy management tools or through the use of meta policies. These options have certain limitation that are discussed in Section 3.7.

We choose to develop a new approach for the controlled restriction of administrative power in data owner-generated policies, which is based on the conflict resolution approach of modern policy languages. These languages can tolerate inconsistencies in the rule-base and still derive a valid access decision. Our main idea consists in the structuring of the policy base in different functional modules. Each module will contain its own set of policy rules that can be independently managed. There exists a strong precedence relationship between the functional modules, which will be jointly evaluated in order to reach an access decision. We propose the following modules:

1. **Default policy rules -** generate a minimum level of data protection even if the data owner is not able or willing to generate a specific data protection policy. A default policy always applies, even if no data owner-defined policy is specified.
2. **Owner policy rules -** are actively managed rules that express privacy protection statements made by the data owner.
3. **Safety and compliance rules -** protect the data owner against the creation of unsafe policies that are potential harmful (e.g. creation of access rules that prevent data access for the data-owner itself). They can guarantee a minimal set of rights for other actors and/or the compliance with legal requirements.

Different priority values will be given to the individual modules. We define a *Privacy Policy Precedence Relation* $\prec$ that prefers access decisions from policies carrying higher priority values. In our use case safety policy rules will have a higher precedence value than default priority rules and owner policy rules.

$$\textit{Default policy} \prec \textit{Owner policy} \prec \textit{Safety policy} \tag{3.4}$$

Each sub-policy may contain identical authorisations, however, only the authorisation decision from the sub-policy with the highest priority value applies. This setup allows us to define a very broad default policy that provides general data protection. This default policy can be overwritten by access decisions from policy parts with a higher priority.

We assume different administrative authorities for the individual sub-policies:

- The *default policy* is designed by a professional policy administrator and attached to any data item as soon as it is created. The policy itself cannot be adapted by the data owner. The default policy will be typically a 'closed' policy. Every action that is not explicitly allowed will be denied.

- The *owner policy* rules can be modified arbitrarily by the data owner. It adjusts the default policy to reflect any privacy decision by the data owner. This policy part can be used to define the privacy preferences of the data owner and make exceptions to the default policy. If the owner policy is deleted, the default policy will again provide basic privacy protection. It is therefore not possible to accidentally lose complete privacy protection through the deletion of the owner policy.
- The *safety policy* again is not under the administrative authority of the data owner. The rules of the safety policy describe specialised authorisations, where it must be guaranteed that these restrictions or permissions persist. The rules in this policy part are meant to safeguard the integrity of a privacy policy. An example would be a safety rule that prevents the data owner from revoking his or her right to access the data.

3.5.1 Implementing Precedence Relations for Sub-Policies

XACML offers a suitable mechanism for the creation and maintenance of different policy parts. Sub-policies can be expressed via `<Policy>` elements. The joint evaluation of different `<Policy>` elements is guided by an adequate policy combining algorithm under a common `<PolicySet>` element. See Listing 3.1, where separate policies for the data owner and the data user are maintained in separate sub-policies:

```xml
<PolicySet  PolicySetId="FirstExample"  PolicyCombiningAlgId="deny-overrides">
   <Target/>
   <Policy RuleCombiningAlgId="deny-overrides">
     <Target>
       <Subjects>Data Owner</Subjects>
       <Resources/>
       <Actions/>
     </Target>
     <Rule RuleId="urn:oasis:names:tc:xacml:2.0:example:rule_id:1"  Effect="Permit">
     </Rule>
   </Policy>

   <Policy RuleCombiningAlgId="deny-overrides">
     <Target>
       <Subjects>Data User</Subjects>
       <Resources/>
       <Actions/>
     </Target>
     <Rule RuleId="urn:oasis:names:tc:xacml:2.0:example:rule_id:2"  Effect="Deny">
     </Rule>
   </Policy>
</PolicySet>
```

Listing 3.1: Separation of XACML Policy Parts through the use of a `<PolicySet>`

The precedence relation could be implemented through a simple ordering operation where policies with a higher priority are evaluated first and policy evaluation is stopped, once an access decision has been derived. XACML already defines the `FirstApplicable` combining algorithm that implements such an evaluation strategy. However, implementing our precedence relation through the `FirstApplicable` combining algorithm has some undesired side-effects. In a declarative policy language, such as XACML, rule ordering usually has no effect on the applicability of a rule. All rules are jointly evaluated and a decision is reached through the use of an explicit combining algorithm that decides potentially conflicting authorisations. The same is true for the evaluation of sub-policies: individual sub-policies have no particular ordering, a policy decision is derived from the joint evaluation of all applicable sub-policies.

If the precedence relation is implemented using the `FirstApplicable` combining algorithm, this property is lost. We would have to make sure that our sub-policies are specified in exactly the right order or wrong authorisations could be derived from their joint evaluation. We think that re-using the existing policy combining method is to fragile and may lead to future problems, especially when we plan to support distributed policy management and automatic policy merging.

We therefore decided to extend the current language specification to allow the specification of explicit priority values for sub-policies and support the prioritised evaluation of sub-policies in XACML through the definition of a new `<PolicyCombiningAlgorithm>` that derives the applicable access decision from the `<Policy>` that has the highest priority value within a `<PolicySet>`. The XACML implementation from Sun [SunXACML, 2006] allows the extension of existing datatypes, functions and combining algorithms through a simple system of factories. We implemented our *Privacy Policy Precedence Relation* (P3R) by defining a new policy combining algorithm class named `PriorityPolicyAlg`, which extends `PolicyCombiningAlgorithm` and implements the decision algorithm shown in Figure 3.3. The full implementation of this algorithm is given in Appendix *Priority Policy Algorithm*.

This implementation must be announced to the XACML framework before it can be used. Our PDP implementation uses an API defined by the `CombiningAlgFactory` to add the algorithm to the set of standard combining algorithms, as can be seen from Listing 3.2:

```
CombiningAlgFactory combFactory = CombiningAlgFactory.getInstance();
        try {
                combFactory.addAlgorithm(new PriorityPolicyAlg());
        } catch (URISyntaxException e1) {
                System.out.println("URISyntaxException has occured in MyPdP.class");
                e1.printStackTrace();
        }
```

Listing 3.2: Registration of the new P3R policy combining algorithm with the SunXACML `CombiningAlgFactory` in `MyPDP.java`

The next Subsections present short examples that highlight particular policy management issues and explain how they might be resolved and implemented as XACML policies using our approach.

3.5.2 Assigning Explicit Priorities to Sub-Policies

Explicit priority values can now be assigned to specific sub-policies using the `<PolicyCombinerParameters>` element of XACML. We define a new parameter `Priority` that carries an integer value which specifies the priority value of the sub-policy.

Listing 3.3 shows the assignment of an explicit priority value to the named `<Policy>` element `"DefaultPolicy"`.

```
<PolicyCombinerParameters PolicyIdRef="DefaultPolicy">
   <CombinerParameters>
      <CombinerParameter ParameterName="Priority">
         <AttributeValue DataType= "http://www.w3.org/2001/XMLSchema#integer">
            1
```

```
        </AttributeValue>
      </CombinerParameter>
    </CombinerParameters>
</PolicyCombinerParameters>
```

Listing 3.3: Specification of explicit priority value using XACML `<PolicyCombinerParameters>`

Instead of relying on sub-policy ordering, we can now explicitly define the priority of a `<Policy>` element, so that the combining algorithm knows if a sub-policy is considered more specific than the other. This has the benefit that the different policy parts are clearly named and the evaluation order can not be changed accidentally through insertions and deletions in the policy base. Figure 3.3 shows our custom combining algorithm for multiple `<Policy>` elements in a `<PolicySet>` and it can be easily seen, that the combining algorithm does not assume or require a specific element ordering in the policy base.

Table 3.2 and Figure 3.2 list the different priority values and their interpretations as they are currently recognised by our implementation. Explicit policy prioritisation allows us to enforce different semantics for the individual parts of the policy. The Figure also shows the intended authorisation scope for the policy parts. The data owner will have control over one particular level of the precedence hierarchy, the `OwnerPolicy`, while the `DefaultPolicy` and the `SafetyPolicy` are generated by an external policy designer and are pre-installed on the system.

Table 3.2: Defining priority values for policy combining

Priority	Value	Interpretation
low	1	Default policy
medium	2	Owner policy
high	3	Safety policy

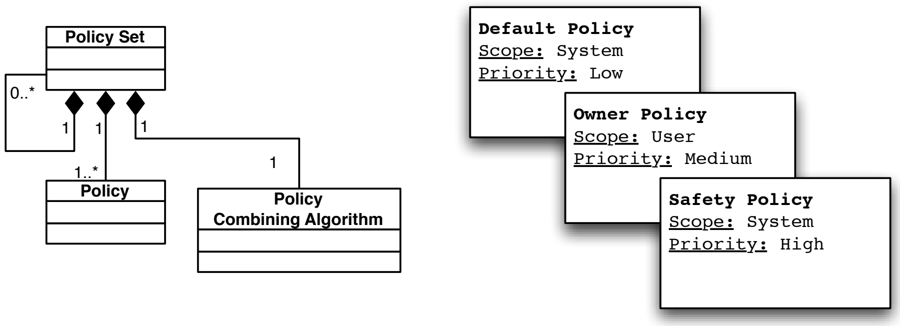


Figure 3.2: Priority Policy Combining - The diagram shows the different policy parts carrying different priority values and scopes. Explicit priority values are assigned to different `<Policy>` elements and jointly evaluated within the XACML `<PolicySet>`.

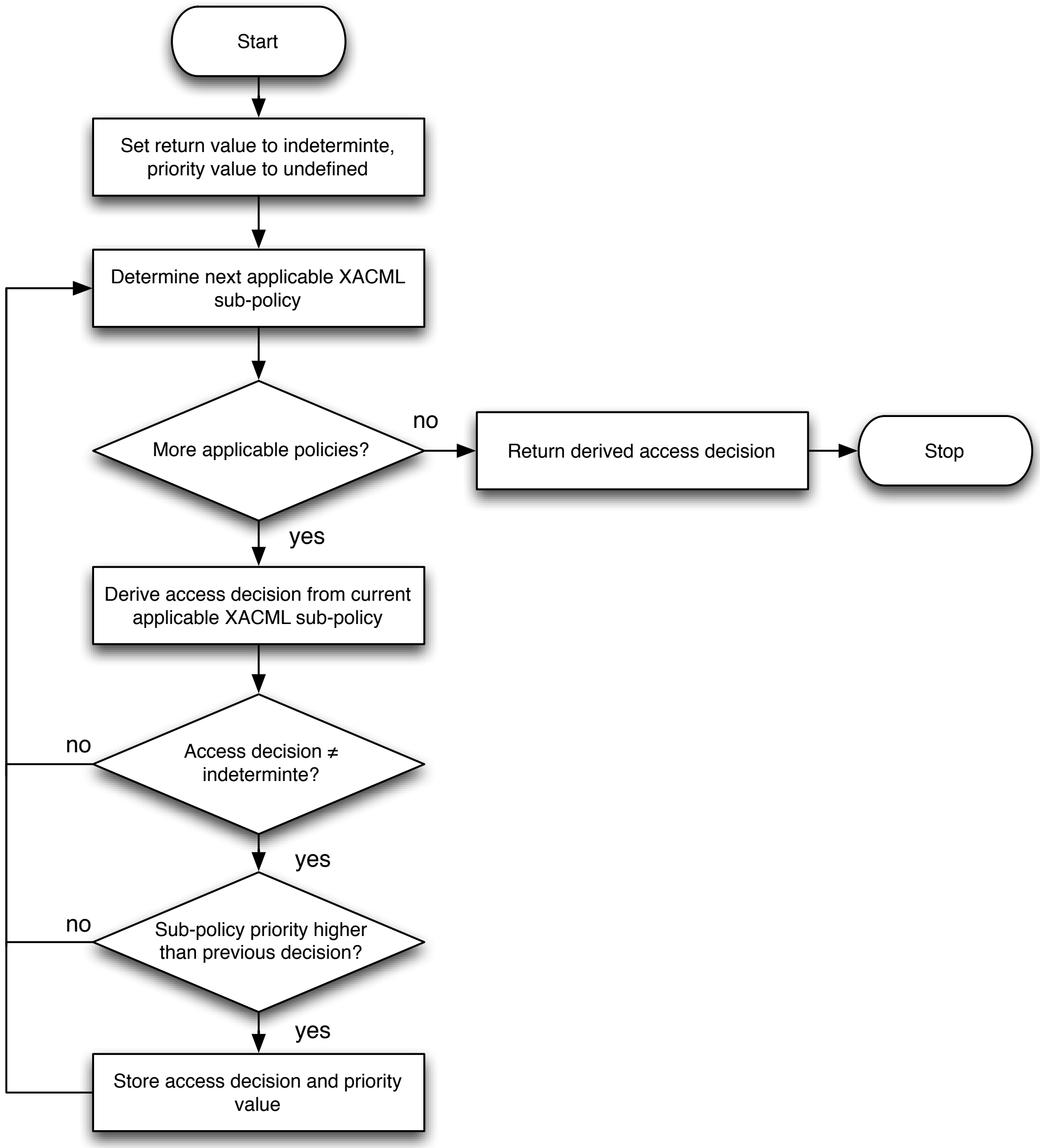

Start
Set return value to indeterminte, priority value to undefined
Determine next applicable XACML sub-policy
More applicable policies?
no
Return derived access decision
Stop
yes
Derive access decision from current applicable XACML sub-policy
no
Access decision ≠ indeterminte?
yes
no
Sub-policy priority higher than previous decision?
yes
Store access decision and priority value

Figure 3.3: Priority Policy Combining Algorithm - The diagram visualises the policy decision process for an XACML `<PolicySet>` that consists of `<Policy>` elements with differing priority values. If more than one policy applies the decision from the policy with the highest priority value is returned.

We now have all the elements needed for the implementation of an ORAC Policy Administration Point that fulfils our usability requirements for a ordinary computer user:

1. Different `<Policy>` elements of a `<PolicySet>` can belong to different administrative authorities and a unique priority value is attached to each `<Policy>` element.
2. The policy base is split into three distinct sub-policies with rising priority level: `Default-Policy`, `OwnerPolicy` and `SafetyPolicy`.
3. The data owner can only manage the user `OwnerPolicy`. `DefaultPolicy` and `SafetyPol-icy` are predefined and protect the policy administration process.
4. Rules in the different policy parts can be missing or contradicting. Rules in different policy parts are not overwritten by rules in other parts that apply to the same data entities. Policy access is determined if at least one part contains an applicable rule. Contradicting decisions are resolved through the priority-combining algorithm according to the priority value of the `<Policy>` element.

Discussion

Lupu and Sloman [1999] have argued that assigning meaningful explicit priorities to policy rules is difficult for the policy creator and might result in inconsistent policies. They therefore discourage the use of explicit prioritisation for policy rules.

Our approach prioritises `<Policy>` elements in an XACML `<PolicySet>` rather than individual authorisation rules. There exists only a very small number of possible prioritisation values, so that it should be easy to understand and manage them. It should also be noted that the priority values are used to define a meta policy that guards the policy administration process itself. It is only very seldom necessary to manipulate policy parts other than the `OwnerPolicy`, which is under the direct control of the data owner. For the data owner are the different priorities not even visible, so explicit policy values have no influence on his or her task of privacy policy management. We are confident that the introduction of the *Privacy Policy Precedence Relation* for individual sub-policies is a valuable contribution to the management of data owner-defined privacy policies. We will discuss relevant applications to specific use cases in Section 5.4.2 and Section 6.2.4.

3.6 Managing Default Protection through Template-based Privacy Policies

The initial process of policy creation for the release of a private data item is carried out by the Policy Administration Point (PAP). The PAP installs a suitable `DefaultPolicy` representation into the `<PolicySet>`. The `DefaultPolicy` in our privacy protection scheme is a policy-part that defines an initial protection settings for the data, even if no `OwnerPolicy` has been configured.

The data owner can not change the default protection rules. It is, however, possible to override the default rules through rules in the `OwnerPolicy`, because this sub-policy has a higher priority. If the data owner decides to delete rules in the `OwnerPolicy`, the `DefaultPolicy` will again be visible for the decision mechanism and data will still be protected.

The prioritised handling of sub-policies makes it easy to support the handling of different protection profiles through so called *Policy Templates*. The `DefaultPolicy` representation could be based on one of several predefined templates that implement different protection properties. The data owner could then choose the `DefaultPolicy` from a set of meaningful, pre-defined template policies according to his or her privacy protection needs.

To show the opportunities of this arrangement, we present two distinct default policies in Listing 3.4 and 3.5 that could be chosen by people according to their personal privacy preferences. A very privacy conscious person might want to closely monitor data usage and explicitly authorise every data access, while a more liberal-minded person only wants to restrict data access from certain subjects and generally allows others to access his or her data. The data owner now can decide which policy behaviour better describes his or her privacy needs, making the management of the `OwnerPolicy` easier:

Restrictive Default Policy A restrictive policy template would implement the *closed policy* model. Data access is prohibited unless explicitly granted via a data owner-defined rule with a higher priority than the default rule.

```xml
<Policy PolicyId="DefaultPolicy"  RuleCombiningAlgId="permit-overrides">
   <Description>Default policy: restrictive</Description>
   <Target />
   <Rule RuleId="defaultAccessPermission" Effect="Deny">
      <Target />
   </Rule>
</Policy>
```

Listing 3.4: Example of a restrictive `DefaultPolicy`

Permissive Default Policy A permissive policy template implements the *open policy* model. Data access is allowed unless a data owner-defined rule is defined that prohibits data access.

```xml
<Policy PolicyId="DefaultPolicy"  RuleCombiningAlgId="deny-overrides">
   <Description>Default policy: permissive</Description>
   <Target />
   <Rule RuleId="defaultAccessPermission" Effect="Permit">
      <Target />
   </Rule>
</Policy>
```

Listing 3.5: Example of a permissive `DefaultPolicy`

3.7 Managing Authorisation Policies and Safety Rules in the Policy Rule-base

Privacy policies usually reflect the personal trust of the data owner in some other person or organisation. This trust might change as personal relationships evolve and this leads to the necessity to alter and adapt exiting privacy policies. The resulting policies may therefore change significantly over time as an authorisation rule base is created, extended and modified.

Policy modification may also introduce conflicts, when newly generated permissions contradict existing ones. We have discussed in Section 2.7 the importance of *consistent* authorisation specifications that allow the successful derivation of a policy decision. It is therefore important to develop a strategy to prevent or manage these conflicts as they arise.

3.7.1 Tool-based Approach

Using a graphical tool as Policy Administration Point can make the generation of policies easier for the human policy administrator. It can supply visual clues regarding expected input values and provide immediate feedback on the syntactical correctness of the specified rules. There have been some experiences with tool based generation of P3P policies, such as the IBM Privacy Editor[2]. Consistency checks could be directly incorporated into the policy management tool. If a data owner wanted to create or modify an authorisation rule that might contradict existing rules or might otherwise be considered harmful, the tool could prevent the user from manipulating the policy.

We researched the possibility to check the policy management activities against an externalised administration policy. Dedicated administration rules could be incorporated into the policy as part of a management policy in the rule base.

Definition 11: Policy Administration Rules

Special rules in the policy base that have the purpose to restrict and guide the policy administration process. *Policy Administration Rules* do not specify access decisions, but rather restrict the creative power of the policy creator, to specify only such access rules that are permitted under the Administrative Policy.

When a Policy Administration Point (PAP) tries to incorporate a new access rule in the rule-base, we could now use the Policy Decision Point (PDP) to check this newly created access rule against the set of policy administration rules and prohibit the inclusion of conflicting rules in the policy base. Only such rules could be added to the policy base that do not conflict with the administrative policy. Navarro, Firozabadi, Rissanen et al. [2003] have used a similar technique for the management of delegation rights that have to be treated independently from the actual access rights.

This approach is expected to work, as long as the set of policy administration rules remains static. However, there exists a high probability that also the administrative policy might change over time. Changes in the administrative policy can lead to the circumstance that the policy-base contains already existing rules that are no longer compatible with the current administrative policy. The question then arises, how to detect and treat those conflicting rules?

The existing authorisation rule base would have to be re-evaluated in order to derive any possible conflicts. If existing rules were found that are in conflict with the new administrative policy, two alternatives exist: The conflict resolution mechanism could simply erase conflicting rules from the policy base. In this case the data owner would find a changed policy rule-set which might no longer represent his or her privacy requirements. If for some reason our administrative policy becomes corrupted and inaccurate, this would be reflected in the rule-set of the policy base, which will now also be defective. A second approach would be to go interactively through the rule set and resolve potential conflicts together with the data owner. This may take time for large policy bases and the data owner might not be able to resolve all arising issues.

Both approaches have the drawback that a changing administrative policy will permanently alter the existing rule-base. Since we can not guarantee that our administrative policy will stay static,

[2]IBM Privacy Editor: http://www.alphaworks.ibm.com/tech/p3peditor

we choose a different approach for the treatment of conflicts in the authorisation base, as will be described next.

3.7.2 Conflict-resolution Approach

Conflicts in authorisation rule-bases can also be handled through a conflict-resolution strategy that is able to derive a policy decision from contradicting rules and policies (cf. Section 2.7.2). Conflict-resolution algorithms specify how policy decisions will be derived from potentially conflicting rules or policies without the need to alter the existing rule base. XACML uses so called *combining algorithms* to solve potential conflicts.

XACML allows us to use a flexible combination of different algorithms for conflict resolution at different levels in the privacy policy. We can use rule-combining algorithms within a sub-policy and separate policy-combining algorithm between sub-policy elements of the privacy policy.

Our implementation of a policy creation tool makes use of the *Privacy Policy Precedence Relation* to limit the authoritative power of the data owner and enforces only the following basic restrictions:

a) Only the data owner can alter the privacy policy.
b) The data owner can choose a suitable `DefaultPolicy`.
c) The data owner can only modify the `OwnerPolicy` part of the rule-base.
d) A predefined `SafetyPolicy` protects substantial access rights of the data owner, as well as other important rights and restrictions.

Our *priority-based combining algorithm* will be used to resolve conflicts between policy parts by giving precedence to policy decisions from parts with a higher priority. We can now include a `SafetyPolicy` that has a higher priority than all the other sub-policies and can be used to specify policy administration rules within our rule-base. This `SafetyPolicy` overrides decisions that are derived from the editable `OwnerPolicy` and thus can resolve problematic authorisations that have been made by the data owner. However, if for some reason our `SafetyPolicy` changes, the data-owner generated rules will still exist in the rule-base and can be used to derive an updated access decision.

The `SafetyPolicy` allows the construction of policies, where the data owner is not able to revoke certain basic permissions, such as his or her own access to the data. Listing 3.6 shows a policy example that guarantees data access for the data owner under all circumstances. Even if the data owner enters a restricting rule in the `OwnerPolicy`, he or she would still be able to access all their private data.

```
<PolicySet PolicySetId="urn:uni-potsdam:healthrecord:example:policysetid:1"
    PolicyCombiningAlgId="urn:policy-combining-alg:priority">
  ...
   <Policy PolicyId="SafetyPolicy" RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0
       :rule-combining-algorithm:deny-overrides">
     <Rule RuleId="dataOwnerRule1" Effect="Permit">
      <Target>
        <Subjects>
          <Subject>DataOwner</Subject>
        </Subjects>
        <Resources>
          <AnyResource />
        </Resources>
        <Actions>
          <AnyAction />
        </Actions>
      </Target>
    </Rule>
  </Policy>
  ...
    <PolicyCombinerParameters PolicyIdRef="SafetyPolicy">
         <CombinerParameters>
                <CombinerParameter ParameterName="Priority">
                        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#
                            integer">3</AttributeValue>
                </CombinerParameter>
        </CombinerParameters>
 </PolicyCombinerParameters>
</PolicySet>
```

Listing 3.6: Example of a `SafetyPolicy` that enforces an administrative policy for the data owner. It guarantees data access for the data owner itself, even if restricting rule have been entered in the `OwnerPolicy` policy part.

Figure 3.4 shows the different approaches to conflict resolution in authorisation rule-bases. Our implementation uses a combination of tool-based and conflict-resolution approach to derive a consistent policy decision. We keep the data owner-defined rules in the rule-base because it conserves the intentions of the data owner, even if they might not be currently enforced. We use a policy creation tool to enforce the syntactic correctness of a rule, without applying an administrative policy to the rule creation process.

This approach eases the construction of the policy creation tool, because it does not have to resolve potential conflicts in the policy. Conflicting rules will not be removed from the authorisation base and could later be used to understand and re-evaluate the privacy intensions of the data owner without impeding the ability to derive a valid policy decision.

3.8 Conclusion

The use of Owner-Retained Access Control (ORAC) policies for the expression of privacy policies offers a number of benefits for the data owner but also holds some challenges that must be solved in order to become a usable and safe alternative to data user-defined policies.

Our trust model for data owner-defined policies assumes a direct trust relationship between the data owner and the data user. The data owner must be able to specify and revoke the necessary authorisations for data access and use the privacy policy to reflect his or her trust relationship with the data user. The trust model for our privacy enforcement framework makes the following assumptions about the relationship between data owner and data user :

- The data user is a non-malicious user that has a mutual interest to comply with the owner-defined privacy policy.
- The trust relationships is not transitive. This means that data forwarding by the data user to not directly unauthorised principals is not supported.

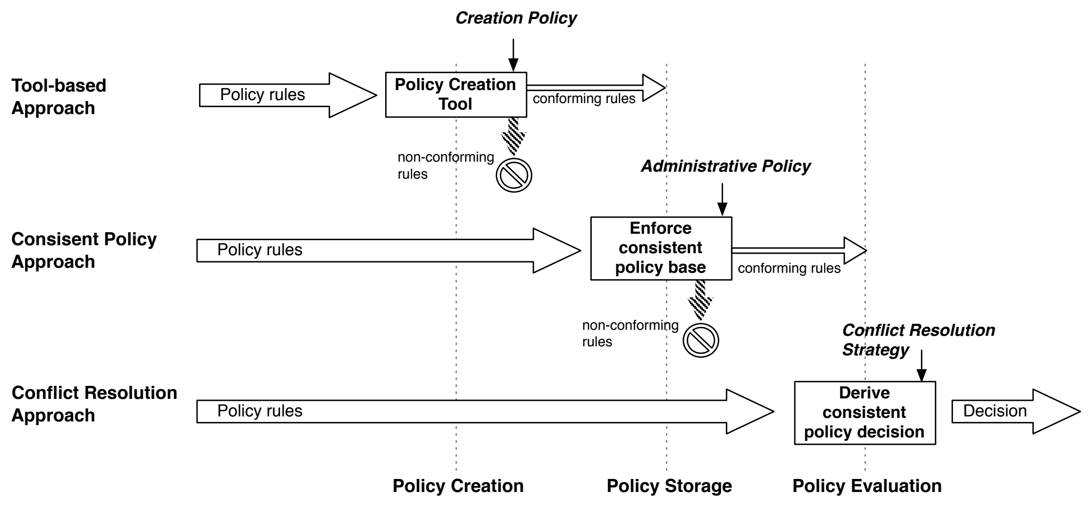


Figure 3.4: Maintaining consistency in the policy rule-base - The figure shows the different approaches that might be used to derive consistent decisions from the policy base. The first two approaches try to eliminate existing rules from the rule-base and are applied when policies are created or changed. The last approach keeps conflicting rules in the rule-base and creates a consistent decision at policy evaluation time.

We defined a *Privacy Policy Precedence Relation* (P3R) and extended the specification of the eXtensible Access Control Markup Language (XACML) with a priority-based combining algorithm. The algorithm uses explicit priority values for individual sub-policies to derive a policy decision. Policy decisions with a higher priority override decisions from policies with lower priority.

We then defined three sub-policies `DefaultPolicy`, `OwnerPolicy` and `SafetyPolicy` that correspond to the priority levels *low*, *medium* and *high*. Short examples were presented that highlight particular policy management issues and explain how they might be resolved using our approach:

- Authorisation rules in the `DefaultPolicy` will be used to form a default policy and provide privacy protection when the data owner has not specified an explicit `OwnerPolicy`.

- The data owner can express his or her protection preferences in the `OwnerPolicy` at the medium priority level. The policy decisions from this sub-policy have a higher priority than the decisions from the default policy and take precedence without changing the default policy. So, even if the data owner later decides to delete all rules from the `OwnerPolicy`, the joint evaluation of the privacy policy would again protect the data through the default policy.

- A third element of our privacy policy is the `SafetyPolicy`, which can be used to describe specialised authorisations, where it must be guaranteed that these restrictions or permissions persist during the lifetime of the policy. It allows the creation of non-deniable access rights that can not be revoked through authorisation rules generated by the data owner.

4 Privacy Enforcement Framework

A very important prerequisite for the implementation of our Owner-Retained Access Control privacy protection scheme is the reliable enforcement of the data owner-defined privacy policy. Section 1.3 of the introductory chapter argued that an effective privacy protection scheme must provide consistent policy evaluation and guaranteed policy enforcement, as personal private data will be used by different data users and the privacy policy must be evaluated and enforced consistently across these parties.

The following points need to be addressed by the privacy enforcement framework when we want to implement our protection scheme:

Communication of the policy - we need to communicate the individual privacy policy of the data owner to the data user in a reliable way. This is a prerequisite for the referencing, evaluation and execution of the policy by the enforcement system.

Protection of policy and data - personal private data must be protected from illegitimate access. It is therefore necessary to encrypt data when it is transferred via the network and during storage on the system of the data user. We similarly have to protect the privacy policy itself from alterations by the data user.

Client-side enforcement component - it was already discussed in Section 2.3 and 2.5, that the enforcement of data-use policies in a distributed environment requires a protection framework that implements client-side protection mechanisms, where each instance of the protection system supports the common access control scheme.

The following sections will model a privacy protection framework for the necessary privacy enforcement that makes private data available under a data owner-defined policy and uses a common mechanism for the enforcement of this policy.

Important building blocks for a distributed privacy enforcement scheme are a distributed client-side reference monitor implementation and the combination of the data use policy with the privacy data to form a so-called sticky policy object. The protection scheme must effectively prohibit any unauthorised data access. Authorised data use, on the other hand, should be influenced as little as possible to achieve a high acceptance rate for the system. Data access is granted on data items that are relevant to the data user and access to all other data items is denied. Enforcement of this simple principle allows us to limit the consequences of unintentional misuse of data by authorised data users.

4.1 Reference Monitor

It is our goal to reliably enforce data protection policies for private data that has been distributed to different computer systems.

A commonly used mechanism for the enforcement of access control decisions is the reference monitor concept (see Definition 8, Anderson [1972]). A trusted system component (the reference monitor) controls every request to system resources and can not be bypassed. The reference monitor guarantees that access to private data conforms to the data access restrictions specified to the privacy policy. It intercepts every user activity and allows the access request if the privacy policy grants access and denies the request if the policy prohibits data access.

Figure 4.1 shows the reference monitor as the central component of the access control framework. Auxiliary components provide user authentication, policy and data storage. Data access is only possible via the reference monitor functions.

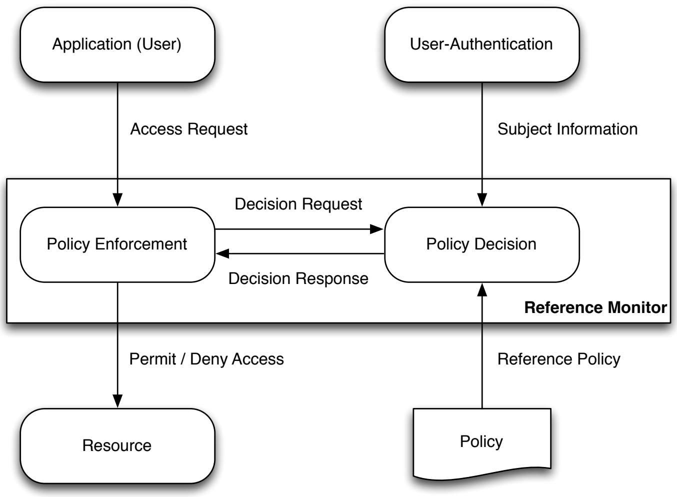


Figure 4.1: Access Control Architecture - The Reference Monitor is structured in a Policy Decision and a Policy Enforcement component. The access request of the data user is intercepted by the Policy Enforcement and triggers a *Decision Request* to the Policy Decision. The decision component will issue an appropriate *Decision Response* based on the evaluation of the available policy. The enforcement component then grants or denies resource access to the user in accordance with this policy decision.

Current distributed access control architectures, such as the UCON_{ABC} usage control model [Park and Sandhu, 2004], recognise that it is not enough to provide access restrictions only at the system or server where the data is initially stored. A client-side enforcement component is necessary for comprehensive data protection, usage and information flow control.

Most commonly used operating systems implement access control schemes that follow the Discretionary Access Control model (cf. Section 2.3.2) and therefore require the presence of an additional trusted policy enforcement element for privacy protection at the site of the data user. Our protection system must implement client-side controls that enforce the owner-defined privacy policy as data propagates from the system of the data owner to the data user.

Structure of the Reference Monitor

A reference monitor can be represented conceptually by two dedicated modules that can be separately modelled and implemented:

Policy Decision Component - is responsible for the generation of an access control decision based on the currently defined policy, the actual access request and the state of the environment, such as the current time, authentication information, access location and possibly other attributes.

Policy Enforcement Component - imposes the access control decision, thus allowing or disallowing access to the protected resource. The enforcement component needs to be system specific insofar as it interfaces directly with the actual execution environment.

This separation of components mirrors best practice requirements from Saltzer and Schroeder [1975] and Woo and Lam [1993] that request the separating of policies from the mechanisms that implement the policy decisions. The functional separation of decision and enforcement component is beneficial in many ways:

(a) It enforces the externalisation of policies. The policies have to be defined explicitly and thus can be evaluated, updated and changed independently of the implementation mechanisms.

(b) Implemented enforcement mechanisms can be system specific and still enforce a common policy across different systems.

(c) The modularisation of reference monitor implementations allows the construction of flexible solutions where the decision component can be implemented on a different machine and can be consulted by many different enforcement implementations.

This functional separation is now an integral objective for the design of modern access control schemes (cf. Section 2.5). It allows the construction of distributed policy enforcement systems but may also be implemented within a monolithic system.

Client-side Reference Monitor

The implementation of a client-side reference monitor requires the controlled distribution of the privacy policy and the associated private data. Distributed networking architectures can be broadly categorised into *peer-to-peer* and *client-server* based architectures. Participating nodes in peer-to-peer architectures simultaneously act as provider and consumer of resources, whereas client-server architectures provide a clean separation of duties: *servers* offer access to their resources, whereas *clients* request access to those resources.

Requests are initiated differently in the two architectures: a client-server system requires the client to initiate the communication session - the server passively waits for a connection attempt by a client. In peer-to-peer systems communication sessions can be initiated by every communication partner.

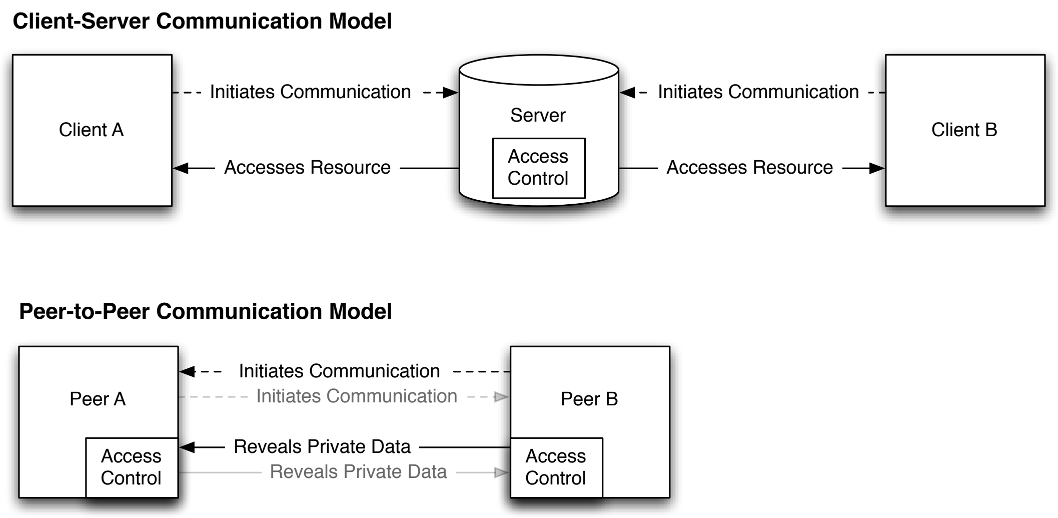


Figure 4.2: Access Control for different communication models - Stationary resources can be protected through a *central access control model* as it is implemented by existing client-server infrastructures. Protection of data that is inherently mobile and might also be distributed between peers requires a distributed client-side reference monitor at every data access location.

These networking architectures typically deploy corresponding access control structures. Client-server systems implement centralised access control mechanism at the server location. A connecting client needs to authenticate itself and can than request access to certain resources on the server. The corresponding access control component will be installed at the server and forms an access decision based on a policy available to the server. There usually is no client-side reference monitor installed that would further limit the use and redistribution of the accessed data.

This *central access control model* works well for stationary resources that can not be moved or distributed freely. For resources that are inherently mobile, such as data, the central access control model is not well suited. When data is accessed from a central repository it is usually copied to the accessing client. There exist now two copies of the data that must be equally protected and thus require a separate *client-side* reference monitor that is able to reference and enforce the access policy. Similarly, when private data is copied between data users, a distributed access control architecture is needed that can enforce the data privacy policy at every peer that is allowed to access the data.

Figure 4.2 compares the two communication and access control models. It shows that each peer needs to contain an access control module - a client-side reference monitor. This access control module needs to be trustworthy and must enforce any access control decision specified in the privacy policy. However, the deployment of a client-side reference monitor is only the necessary condition for distributed privacy enforcement. The reference monitor must also be able to reference the current privacy policy in order to derive an access decision. The next section explains how this requirement can be solved.

4.2 Policy Storage

The decision component of the reference monitor must be able to access the current privacy policy to evaluate it, and inform the enforcement component about the access decision. The storage location and distribution model for the privacy policy is an important design criterion that influences the implementation of our protection framework. A client-side reference monitor evaluates the policy whenever personal private data needs to be accessed and it is therefore necessary to easily and reliably reference, retrieve and evaluate the corresponding privacy policy.

The privacy enforcement framework should provide protection under the assumption, that private data could be copied between different data users. This means that it must be possible to share the privacy policy (or a reference to it) together with the data and ensure that the reference monitor can access the policy. Two principal alternatives exist for the storage location of access policies:

Separated policy and data storage - private data and the corresponding policies can be stored in different repositories and each client-side reference monitor instance consults the policy store before an access decision is made. A policy repository offers a central point of reference for the policy decision component and allows the easy maintenance of a coherent policy for all data objects.

This storage model has the advantage that the policy repository could be implemented as a service that can be accessed by the data owner and the data user simultaneously, which would ease the administration of privacy policies – it would be simple for the data owner to change or revoke permissions. It must be ensured that the repository is reachable by the reference monitor to derive an access decision and the reference to the policy is maintained , when private data is copied between data users.

Combined policy and data storage - data privacy policies can also be stored together with the data that this policy is protecting. A policy distribution method, which is well suited to handle distributed policy and data storage, is the *sticky policy* concept (cf. Karjoth, Schunter, and Waidner [2003]; Mont, Pearson, and Bramhall [2003]). The data access policy is stored and distributed together with the data that it is protecting and therefore can always be referenced.

The sticky policy paradigm provides a high level of flexibility for the dissemination of access control policies. The direct attachment of the policy object to the data object allows the creation of multiple protected objects with very focused policies. It becomes possible to share similar data objects with different recipients and implement recipient-specific policies. So could, for example, the policy grant temporal or unlimited access to the data object depending on the trustworthiness of the recipient (cf. Section 3.4.4).

This model has the advantage that any policy can be directly referenced from the data and can also move with the data when the data is copied or transferred between data users. Having potentially multiple copies of data and policies in unknown locations makes policy administration more difficult. The existence of multiple copies of policies and data can lead to diverging policies, when changes are made to a subset of the distributed data and policy items. It might become impossible to reliably revoke access to already distributed data. It is therefore advisable to limit the timeframe in which the data can be accessed by the data user.

Figure 4.3 shows the interworking of the different policy enforcement components under the *sticky policy* model. Access to data and policy is again mediated through the reference monitor. The protected resource and its access policy are created, stored and referenced together. The sticky object can be disseminated to different data users without having to fear that policy and data become separated. If a sticky object is deleted, both data and access policy are removed from the system.

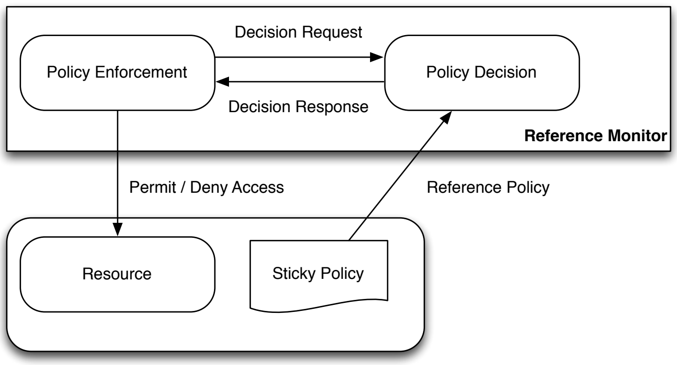


Figure 4.3: Access Control Architecture using *Sticky Policies* - The generic architecture is adapted so that the data and the corresponding privacy policy are now contained inside a single object. This architecture allows the controlled distribution of data under the assumption that data and policies can only be accessed via a trusted reference monitor and will never be separated during data usage.

When we analyse the binding between data and policies in the different storage models, we find that the separated policy and data storage enforces only a weak connection between policies and data. We find this storage strategy typically in data user-defined policies, where the common policy of a data user organisation is stored, maintained and referenced throughout the organisation. If data is shared across access domains, e.g. as part of a workflow that involves different organisational entities, it becomes difficult to maintain the link between the data and the original privacy policy. If the reference between the policy and the data breaks, the reference monitor can no longer enforce the data access restrictions. This type of event can easily lead to a data breach as described by Anton, He, and Baumer [2004].

The combined data and policy storage of the *sticky policy* enforces a strong binding between data and policy. If data and policy are stored together, e.g. as an XML-file, all operations at the file-level influence policy and data simultaneously. Policies stay attached to individual data items as they are distributed to the intended data-users and can be much more focused than global policies for all data objects. Policies stay connected to the data object as it is copied between data users. Policy decisions can be made locally and do not depend on the reachability of the policy store. We therefore decided to use in our implementation of the privacy enforcement framework a combined policy and data store.

4.3 Data and Policy Protection

The security of our privacy enforcement scheme depends on the fact that only the trusted reference monitor has direct access to the protected resource as well as to the included privacy policy. The reference monitor reads and interprets the privacy policy and derives an access control decision that allows or disallows the data user access to the data. If an attacker could otherwise extract private data or alter the included policy, the privacy protection offered by our enforcement framework would be lost. The attacker could then copy, alter and delete private data or change the privacy policy to be less strict and allow data usage in ways that have not been authorised by the data owner.

A scenario, where data access becomes possible without control of the reference monitor constitutes a privacy violation and potential data breach. The consequences of privacy violations are much more difficult to handle and contain than other computer security events. Recovery from data breaches almost ever requires the identification and cooperation of all the involved parties. Simple recovery measures such as the changing of passwords and sanitation of compromised systems can not restore the safe state of compromised private data, because copies have been made of the data that might reside on systems outside the access of the data owner.

Protected Data Object

In order to guarantee that data access is only possible through the reference monitor it is necessary to store the data and policy always in an encrypted form inside a *protected data object*. The choice of encryption mechanism influences the security properties of our enforcement framework. Ideally we could use a Public Key Infrastructure (PKI) that gives the data owner access to the public key of the data user. In this case it is possible to use a asymmetric encryption mechanism such as RSA [Rivest, Shamir, and Adleman, 1978] and encrypt the data under the public key of the data user, so that only the reference monitor of the legitimate data user has access to the protected data object. Another alternative might be the use of Identity-Based Encryption (IBE) that does not require the sharing of the public key of the data user and instead derives a suitable encryption key directly from the identity of the data user [Boneh and Franklin, 2003].

If we can not use asymmetric encryption because no PKI is available, or the data user has no suitable public/private key pair, each trusted reference monitor could share a common encryption key and use the Advanced Encryption Standard (AES) [FIPS PUB 197, 2001] to access the protected data object. This method is less secure than the asymmetric encryption mechanism, because now all reference monitors share a common secret that might be extracted from the reference monitor by a capable attacker. Countermeasures such as the use of a Trusted Platform Module (TPM) as proposed by Sevinç, Strasser, and Basin [2007] are not yet widely deployed and usable.

We implemented two different solutions in our use cases. The protected data object for the Personal Health Record is encrypted by a symmetric key that is shared between all instances of the reference monitor, while the theme park scenario uses a public-private key pair based on RSA to secure the distribution of location data. Sections 5.3.1 and 6.2.4 provide implementational details for each approach. Figure 4.4 shows the data and policy access by different instances of the reference monitor using a shared symmetric key.

For the rest of the chapter we assume that our encryption keys can be managed securely and have not been compromised. If this assumption holds, the protected data object can now be distributed

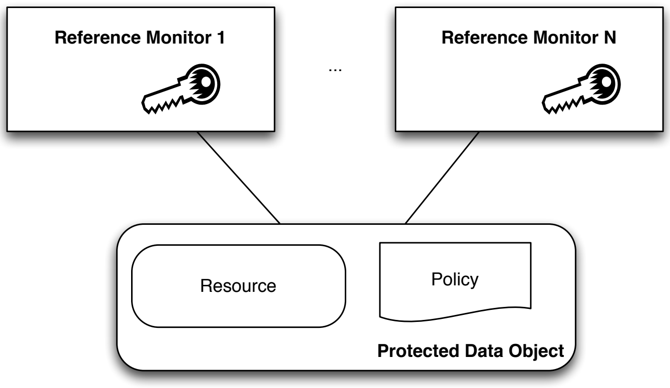


Figure 4.4: Data and Policy Protection using Shared Key Access - The *Privacy Enforcement Framework* uses data and policy encryption to build a *protected data object*. A shared encryption key could be used to restrict access to the protected object to trusted reference monitors.

and copied freely between peers without privacy concerns.

Policy management

The reference monitor bases all access decisions on the privacy policy that is explicitly documented in the protected data object. The encryption and secure distribution of the protected data object is not sufficient for the secure enforcement of privacy policies. The framework must also support the secure creation and management of policies.

This means that the policy itself must be suitably protected and any access to the policy rule base (e.g. for the purpose of policy creation and maintenance) must be made through the reference monitor. Policy management must be guided by an administrative policy that only allows the data user to make changes to the policy rule base. Otherwise an attacker could gain data access indirectly through the manipulation of the access control policy. The attacker could then access protected data and compromise data integrity through a number of different attacks on the policy rule base:

Insertion Attacks - the attacker inserts an additional rule into the policy base that grants data access, where previously no access was granted.

Deletion Attacks - the attacker deletes a restrictive rule in the policy base, thus allowing access, where previously access had been forbidden.

Cut & Paste Attack - the attacker substitutes a restrictive policy with a copy of a permissive policy that has been stored previously.

Denial-of-Service Attack - the attacker alters rules in the policy base, so that legitimate use through authorised users is no longer possible.

4.4 User Authentication

Human users are represented in computer systems as principals - entities to which authorisations are granted [Saltzer and Schroeder, 1975]. A single human user can be represented through different principals depending on the current role of the user, e.g. a human can be logged into a computer system simultaneously as an ordinary user and as the system administrator.

It is the responsibility of the authentication system to create a reliable and secure association between the identity of the human user and the principals known by the system. A subject is "a process in a computer system that represents a principal and that executes with the privileges that have been granted to that principal" [Shirey, 2007].

The reference monitor mediates data access based on the identity of the principal that is issuing an access request to the resource. The decision component of the reference monitor bases access decisions, amongst other factors, on the subject that issues the access request. For the secure and trustworthy operation of the privacy protection scheme it is important that only fully authorised subjects are able to access and manipulate resources. The policy enforcement therefore requires the secure and reliable authentication of the human user that started this process. Otherwise it might be possible for an attacker to gain illegitimate access to a resource through the adoption of a spoofed identity.

The privacy enforcement framework presented in this thesis does not mandate the use of a specific authentication mechanism. Several secure authentication mechanisms, such as public-key, smartcard or password authentication might be implemented as long as they can securely bind the user identity to the principal issuing the access request.

It is important to note that secure authentication is equally important for the identification of the data user as well as the data owner. If a data user can spoof the identity of the data owner, it can gain additional rights that would allow him or her to alter the privacy policy.

4.5 Application Program

The last major component of the privacy enforcement framework that needs to be discussed is the application component. Privacy protection differs from traditional access control insofar, as it introduces the special requirement that it is necessary to still enforce usage control restrictions after the data has been released to the data user.

We already described that actions of the data user are carried out by the computer system through processes that are associated with a principal known to the system. These processes typically belong to an application program that executes certain application-specific tasks under the responsibility and control of a human user. The application program itself can not be a predefined part of the privacy enforcement framework, since its implementation is usually domain specific and not controlled by the framework. Therefore, the application is regarded as an untrusted component that needs to be restricted in its interaction with the private data.

Once data access has been granted to the application, it must be guaranteed that private data is only used according to the usage restrictions implied by the policy. The application must be restricted in its ability to process, store, copy or forward data arbitrarily. It is the responsibility of the reference monitor to restrict data-usage by controlling interactions of the application with external processes and systems (e.g. making changes in the filesystem, writing to devices and network sockets, etc.).

A popular approach that limits the execution capabilities of untrusted applications on a host-platform is the sandboxing of applications. The Java programming language first made this protection model popular, because it protected hosts from the potentially malicious actions of Applets that were downloaded from a web-server and executed on the local machine [Gong, Mueller, Prafullchandra et al., 1997]. Systrace [Provos, 2009] and AppArmor [AppArmor, 2007] are more recent sandboxing approaches that restrict the capabilities of arbitrary application programs according to a protection profile.

Chapter 5 *"Java PrivMon - Privacy Protection for Personal Health Records"* will introduce a framework that uses a modified Java Security Manager to provide data privacy protection for Java applications. It requires only minimal adaptation to existing application components and can enforce the required data usage protection. It will be shown how privacy policies can be dynamically mapped to Java policies in order to enforce the owner defined privacy policies. The application component is started under the control of the Java Security Manager and thus can be trusted to execute only functions allowed by the privacy policy.

Chapter 6 *"Privacy Protection for Server-based Information Systems"* extends this work to also offer privacy protection for server-based business applications that follow a tiered architecture where different functions such as business logic, data access and data representation are separated. We extended our reference monitor design to use Aspect-oriented Programming (AOP) and reflections to intercept data accesses in existing applications and provide a way to enforce data user defined privacy policies for business applications.

4.6 Related Projects

The following section gives a short overview of alternative approaches that either directly targeted at data privacy issues or exhibit interesting properties that have been tried in the implementation of data privacy schemes. This is not a complete list of all available solutions and approaches, but highlights interesting concepts that have had an influence on our work.

4.6.1 Key Concepts

Karjoth, Schunter, and Waidner [2003] describe a *Sticky Policy* scheme that allows the direct association of a privacy policy with a data record, so that privacy requirements can be honoured consistently by different data users and different processing systems. We make extensive use of this concept for data and privacy storage in our privacy enforcement framework.

Lehmann and Thiemann [2006] have developed a field access analyser that is able to analyse existing Java programs in order to determine the points in the program code where object methods are accessed. Static policy checking code is inserted to enforce access controls in accordance with the access-control policy for the program. In our work we choose to clearly separate the policy enforcement from program execution. No access to the application source code is necessary for the policy enforcement and policies can be expressed, evaluated and enforced independently from the application.

4.6.2 Architectural Frameworks

Sevinç and Basin [2006] describe a formal access control model for documents based on the sticky policy paradigm. In their work they focus on document related actions such as read, print, change and delegate. Their model supports multiple owners and sub-policies for document parts and takes document editing into account, where merging and splitting of document content also influences the attached policies. Our work fits within their problem definition, but we focus more on the issues of policy generation by the data owner and on implementation issues that need to be resolved for practical deployment.

Mont, Pearson, and Bramhall [2003] propose a privacy architecture that uses sticky policies and obfuscated data that can only be accessed if the requester can attest compliance with the requested privacy policy for this data. Data access is mediated via a Trusted Third Party that can reliably enforce time-restricted access. Our work aims to provide similar protection but does not depend on functions provided by an Third Party. We use the functions of a trusted reference monitor to enforce policies and provide data-flow protection.

4.6.3 Server-based Solutions

Author-X [Bertino, Braun, Castano et al., 2001] is a Java-based data protection solution that allows the definition and enforcement of access restrictions on (parts of) XML-documents. It is a server-based solution, where the document access is mediated by the access component, based on the collocated authorisation store. Access can be restricted to certain parts of the complete document. Author-X does not provide client-side data protection. Once data access has been granted, the data user has complete control over the data.

Damiani, De Capitani di Vimercati, Paraboschi et al. [2002] developed an access control system for XML documents that is able to describe fine grained access restrictions for the structure and content of XML documents. Their system generates a dedicated user view according to the permissions granted in a server-side authorisation file: the XML Access Sheet (XAS). The system does not implement any control over data that has been released by the server to the client. Consequently any data that can be read by a data user could be locally stored, copied and processed by the client. The generated view restricts data processing for single action classes only, e.g. the 'read' action. No support is given for orthogonal action sets, e.g. restricting a document to be read, but not to be printed.

4.6.4 Client and Server-based Solutions

Gupta and Bhide [2005] describe an XACML-based authorisation scheme for Java that is based on the Java Authentication and Authorisation Service. The work describes a generic implementation that extends the Java policy class with the ability to interpret XACML policies. While this work allows the Java Security Framework to enforce permissions for different users of an application, it might not be possible to enforce ORAC policies where different permissions need to be enforced depending on the data object that is currently accessed.

The authors Hohl and Zugenmaier [2006] describe an implementation for data owner-based privacy policies using DRM technology. The data owner sends private data to a service provider and restricts the use of the data to fulfil a particular service. Once the data owner stops using the service, the data is deleted from the server. The system provides only very limited action support.

The data owner can grant the service to *view* and *anonymise* the data. The implementation is based on the Microsoft Rights Management Service for Windows Server and therefore limited by the enforcement functions provided by this particular platform. Data user and data owner must agree to use the same server for the issuing of data usage permissions and server must be online, in order to issue a license to the data user.

4.6.5 Expiring Data

A reoccurring demand for privacy protection is the ability to support expiration dates for distributed private data after which this data is no longer available.

Geambasu, Kohno, Levy et al. [2009] have developed a practical approach for the forced expiration of data called *Vanish*. It protects private data by making it unavailable to any potential data user after a certain time period and requires no actions from the data owner after the data has been released. Vanish encrypts the private data locally using a secret sharing scheme [Shamir, 1979]. The key shares are distributed in the form of index entries across different large-scale distributed hash tables (DHT) for P2P networks and the local copy of the data encryption key is deleted. A special data access key is used to reference these individual indexes. This approach ensures that after a certain amount of time less and less elements of the encryption key are available, as the hash tables evolve and old hash entries are deleted. The access time for data decryption can be influenced by modifying the threshold value for the used key sharing scheme. The developers of Vanish have implemented a Firefox plugin for Gmail and other web sites and a Vanishing File application which can be used to protect local content on computers. Vanish is safe against attackers that may obtain a copy of the original, encrypted data.

The Vanish system operates under a similar assumption than our system: the data user is non-malicious and has a mutual interest to safeguard the privacy of the shared data. It trusts the legitimate receiver of the data not to make local copies of the unencrypted data and does not protect against attacks, where a legitimate data user choose to forward or use the unencrypted private data against the intentions of the data owner. It provides no usage restrictions or policy support on the shared data other than the enforcement of an expiration date for the encrypted data.

4.7 Summary

The proposed privacy enforcement framework combines existing elements of distributed access control systems and builds a new framework for the enforcement of owner-defined privacy policies. The framework is targeted at the controlled and secure distribution of private data – something that has been requested by privacy advocates [Stytz, 2005; Sevinç and Basin, 2006], but has not yet become widely available for common usage.

We rely on a distributed *reference monitor* and a *protected data object* for the distribution and access to private personal data under the assumption that the reference monitor implementation can reliably enforce the defined privacy policy in arbitrary data access locations. Our protected data object contains the private data together with the corresponding access policy. This simplifies the referencing of the policy and enables fine-granular permissions for individual data items in the protected data object. No central infrastructure or third-party is required for the determination of the data access decision. This makes the framework useable in situations where no access to the

Internet is available.

The private data and corresponding *Sticky Policy* are protected through a suitable encryption mechanism and form a *Protected Data Object* that can be stored and distributed freely between potential data users without privacy concerns. The reference monitor must be installed at every data access location and consists of a *Policy Decision* component that interprets the privacy policy and a local *Policy Enforcement* component, which is responsible for the secure data access by the data user. Data and policy access happens exclusively via the reference monitor and we assume that every reference monitor is a fully trusted implementation. A suitable encryption and key-sharing mechanism is implemented by all reference monitor instances.

5 Java PrivMon - Privacy Protection for Personal Health Records

The realisation of our *Privacy Enforcement Framework* requires the presence of trusted system components at every data access location. A *Client-Side Reference Monitor* acts as such component in our framework and it has the responsibility to reliably evaluate and enforce the data privacy policies for the data owner.

The implementation and maintenance of a trusted, distributed reference monitor infrastructure is the main challenge in our proposed privacy protection scheme, because every participating data user needs to trust, install and maintain the necessary components. We therefore decided to base our first implementation of the Privacy Enforcement Framework on a widely deployed, popular and proven platform – the Java programming language and virtual machine concept. We assumed that the Java Security Framework can be used directly for the enforcement of our privacy policies. After some experimentation, we found, that our assumption was only partially true. The Java Security Framework can be used for privacy enforcement, but we had to define and implement our own extensions, such as a specific class loader that allowed us to bind different Java Protection Domains to instances of a class.

This chapter describes our extensions to the standard Java and XACML mechanisms that are used for the expression and enforcement of authorisations for distributed access to private personal data. We assume that a reliable authentication mechanism is used for the identification of principals in the system. Authentication issues will not be discussed further in this chapter, because our use case has no special requirements that can not be handled by standard mechanisms and therefore authentication issues are not in the scope of our research activities.

5.1 Java Security Architecture

The Java programming language provides a *Security Framework* that is aimed to protect the local system from threats that may arise from the execution of untrusted Java code [Gong, Ellison, and Dageforde, 2003; Oaks, 2001]. This security framework has its origin in the protection requirements for untrusted program code that is downloaded from Internet sources and executed on the local system, such as Java Applets. Untrusted programs will be run under the supervision of the *Java Security Manager* within a sandbox environment and are restricted in their access to system resources. For each Java virtual machine there exists exactly one instance of the `SecurityManager` class.

With the introduction of the Java 2 Security Architecture the early sandbox model became much more refined and flexible [Gong, Mueller, Prafullchandra et al., 1997]. The first version of the language specification restricted the control of the Java Security Manager to Java applets that had been downloaded from a web-server. The Java 2 security architecture removed this restriction and it is now possible to control the execution of all Java programs. Whereas before the support for fine

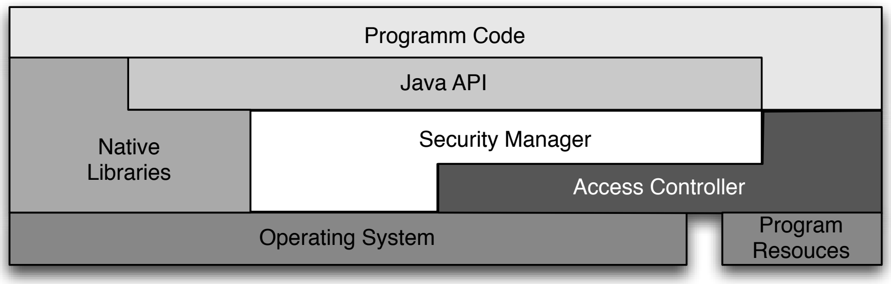


Figure 5.1: Java Security Architecture - The Security Manager and the Access Controller are the main building blocks of the Java Security Architecture. The Access Controller has been introduced with the Java 2 Security Architecture. [Oaks, 2001]

grained authorisations required the sub-classing and customising the `SecurityManager` class, it is now possible to use a hierarchy of typed and parameterised access permissions for this purpose. The `SecurityManager` now only provides the interface that is used by the core Java API. The implementation of the necessary mechanisms has been moved into the `AccessController` and its related classes, as can be seen in Figure 5.1, where the different elements of the Java Security Architecture are shown.

5.1.1 Invoking the Java Security Manager

The Java 2 security framework allows the definition of application-specific security policies for local programs. However, local programs are *not* started under the control of the Java security manager by default. This means they are fully trusted and operate with the full set of rights to access any resource on the system. To restrict the permissions of a local program, it must be explicitly started under the control of the security manager. This can either be done through the `-D java.security.manager` command line option or programmatically with the code given in the following listing:

```java
SecurityManager sm = System.getSecurityManager();
if (sm == null) {
   sm = new SecurityManager();
   System.setSecurityManager(sm);
}
```

Listing 5.1: Programmatically invoking the Java Security Manager

5.1.2 Java Policies and Permissions

The default Java policy implementation uses text-based configuration files (`java.policy`) to determine the actual set of permissions for a program. The policy file(s) grant only positive permissions to avoid the problem of conflicting specifications. Grants specify what permissions are given to an application from a specified source and executed by different principals. Java supports fine grained permissions for method calls that try to access the filesystem, network sockets, specific system resources and the Java virtual machine itself.

Listing 5.2 shows the basic format of a grant entry in the java.policy file. The codeBase identifies the location of the code source, the signedBy value points to a certificate stored in the keystore and the principal value specifies the subject that is executing the code. The signedBy, codeBase, and principal values are optional. If they are omitted from a grant, it means that this grant applies to any code signer and principal, and it does not matter where the code originated from.

```
grant signedBy "signer_names", codeBase "URL",
      principal principal_class_name "principal_name",
      principal principal_class_name "principal_name",
      ... {

    permission permission_class_name "target_name", "action",
        signedBy "signer_names";
    permission permission_class_name "target_name", "action",
        signedBy "signer_names";
    ...
};
```

Listing 5.2: Basic format of a grant entry in the java.policy file

5.1.3 Java Protection Domain

The default implementation of the Java Security Framework uses different classes to represent the individual elements of an access control policy. The Permission class contains the name and a list of possible actions of the permission. Individual permissions can be grouped together by the PermissionCollection class. This makes the handling of permissions easier, because usually a whole set of permissions applies to an application.

Permissions are granted depending on the origin of the code, a potential code signer and the user of the application. The Principal class represents an authenticated user or service that is associated with the current execution context of the code. The CodeSource class encapsulates the origin of the code in the form of an URL and the set of certificates that have been used to sign the code.

When a class is loaded its protection domain is set and can not be changed later. The Java ProtectionDomain class encapsulates the following [Gong, Ellison, and Dageforde, 2003, p. 75]:

- A CodeSource describing the code origin and signing certificates.
- A Principal array that may be set during execution to indicate who is executing the code
- A ClassLoader reference, possibly null, to the class loader defining the class.
- A PermissionCollection containing permissions granted to the code statically when the class was loaded. The dynamic permissions for the protection domain are determined by consulting the policy.

The ProtectionDomain is determined and irrevocably bound to the class at class-loading time by the Java class loader. Protection domains can be used to isolate different applications within the Java runtime environment. “This could be done by using distinct class loaders to load classes belonging to different domains in such a way that any permitted interaction either must be through system code or explicitly allowed by the domains concerned." [Gong, Ellison, and Dageforde, 2003, p. 79]

5.2 Java PrivMon Architecture

The Java runtime environment already provides reference monitor functionality for the safe execution of untrusted code and it was our aim to re-use this proven mechanisms for the enforcement of privacy policies. We had to solve the problem that all the existing Java security mechanisms are class-based. This means that policies and permissions are bound to the codebase of the program and not to the principals that executes the code or the resources that are accessed by this program. If we wanted to use the existing Java Security Framework for the enforcement of privacy policies, we had to extend it, to support different security policies for different code instances. Instance-based protection was needed, because the same application might be used to access distinct data-sets with different privacy policies.

Figure 5.2 shows the main elements of the Java PrivMon that closely match the architectural framework presented in Chapter 4 *Privacy Enforcement Framework*:

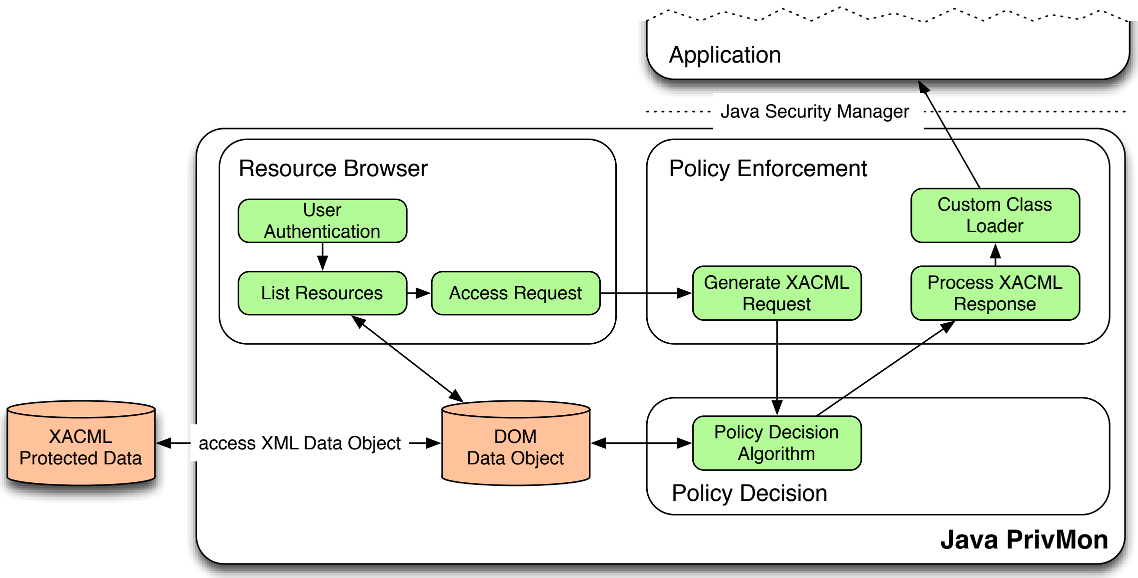


Figure 5.2: Java PrivMon Architecture - The *Java PrivMon* reference monitor consists of three distinct elements: Resource Browser, Policy Decision and Policy Enforcement. Access to private data is initiated from the Resource Browser. The Policy Enforcement starts an application component under the control of the Java Security Manager with the set of permissions that have been derived from the privacy policy by the Policy Decision component.

The private data of the data owner is encoded into a suitable XML record format and stored together with the corresponding policy as a single, encrypted *Protected Data Object*. Privacy policies are specified using the eXtensible Access Control Markup Language [XACML-2.0, 2005].

The *Policy Decision* component (see Section 5.4) evaluates the XACML privacy policy that has been defined by the data owner and generates an access decision.

The *Policy Enforcement* (see Section 5.5) is conceptionally different from the simple reference monitor concept. Instead of directly intercepting data access from the application, the policy enforcement of the Java PrivMon relies on the Java Security Manager to intercept data access and enforce the privacy policy. It was our idea to provide a mapping between XACML policies and Java permissions. The Policy Enforcement component builds a controlled execution environment that is entirely based on the existing Java Security Framework. It builds a dedicated `ProtectionDomain` and starts the application under the control of the Java `SecurityManager`.

The *Resource Browser* (see Section 5.6) is an additional component that has not been discussed previously. It lets the data user interact with the private data in order to select interesting data items from the *Protected Data Object*.

All the components of the Java PrivMon are fully trusted, this means they have full access to the private data in the *Protected Data Object*. The following sections explain the architectural components of the Java PrivMon in greater detail.

5.3 Personal Health Records

The discussion of the features of a privacy protection scheme benefits from a practical example that clarifies the concepts and determines crucial features that need to be available. One application domain where privacy issues are obvious and have been discussed, long before widespread electronic processing of patient data became available, is the medical domain.

Practitioner records contain sensitive personal data about patients and special provisions have to be made for their protection. Typical health care situations require interactions between different actors, which must enter a very private trust-relationship with the patient in order to make a diagnosis, administer treatment, provide rehabilitation and aftercare. A study conducted by Evered and Bögeholz [2004] showed the complexity of modelling an access control system for even a small health information system.

In the past, medical records have been kept at practitioners offices and it was seldom possible to access previous examination results made by other practitioners. With the ability to store and access health records electronically, the idea developed to create Personal Health Records (PHRs), where different health related information about a single person can be kept together. The Personal Health Working Group of the Markle Foundation defined a Personal Health Record as following:

> The Personal Health Record (PHR) is an Internet-based set of tools that allows people to access and coordinate their lifelong health information and make appropriate parts of it available to those who need it. PHRs offer an integrated and comprehensive view of health information, including information people generate themselves such as symptoms and medication use, information from doctors such as diagnoses and test results, and information from their pharmacies and insurance companies. Individuals access their PHRs via the Internet, using state-of-the-art security and privacy controls,

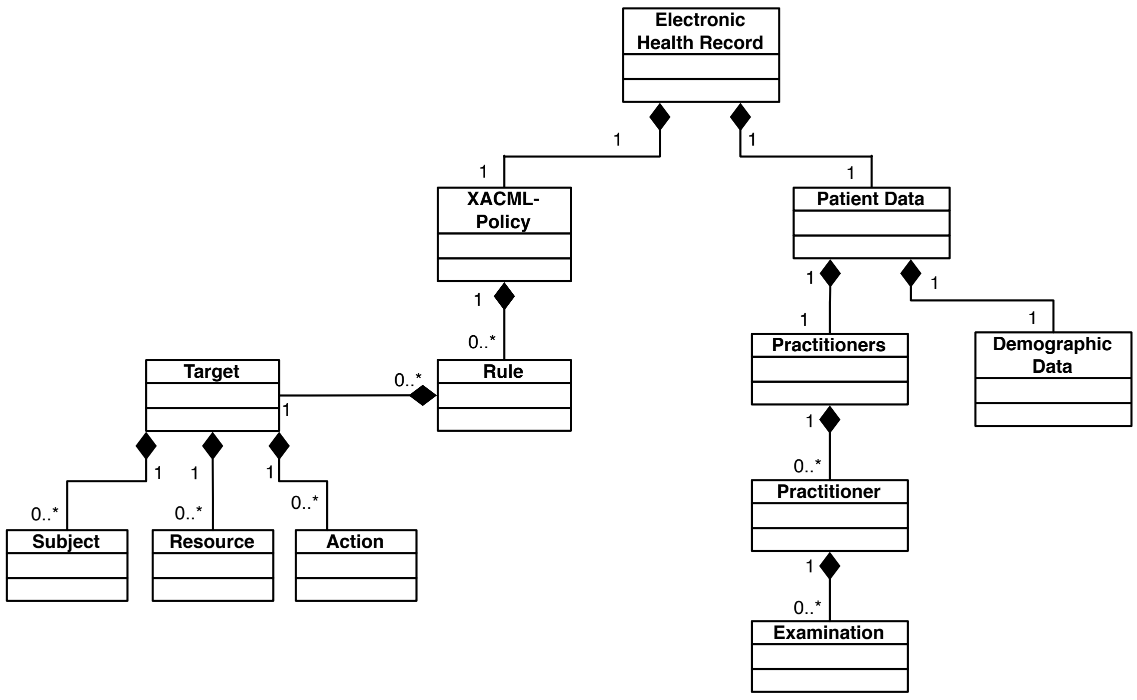


Figure 5.3: Data model of a Personal Health Record - The PHR is stored as a semi-structured XML-document. The document consists of a XACML policy part that encodes the access policy for the data part of the document. Medical data is stored in the data part of the document, which is structured as a document tree, where each practitioner owns a separate branch.

> at any time and from any location. Family members, doctors or school nurses can see portions of a PHR when necessary and emergency room staff can retrieve vital information from it in a crisis. People can use their PHR as a communications hub: to send email to doctors, transfer information to specialists, receive test results and access online self-help tools. [Markle Foundation, 2003]

According to the definition, a PHR has the following distinct characteristics that differentiate it from other electronic health records that are kept by doctors or health care organisations:

- Each person controls his or her own PHR. Individuals decide which parts of their PHR can be accessed, by whom and for how long.
- PHRs contain information from the patients entire lifetime.
- PHRs contain information from all health care providers.
- PHRs are private and secure.
- PHRs are *transparent*. Individuals can see who entered each piece of data, where it was transferred from and who has viewed it.
- PHRs permit exchange of information with other health information systems and health

professionals.

Several implementations of PHR systems exist and can be used by patients. Google [Google Health, 2008] and Microsoft [Microsoft HealthVault, 2010] are the largest players in this field, however in Germany also health insurance companies provide *Personal Health Portals* for the storage of PHRs [Barmer, 2010].

The German government has mandated the use of patient smart cards for access to the general health care system [GKV 2003, 2003]. The health cards will have the ability to store personal health record data of the patient, so that it can be accessed and exchanged by different practitioners participating in the treatment process and act as a repository for future diagnosis.

Despite promised privacy controls, most PHRs have only rudimentary privacy settings. Google Health allows to share a health profile with arbitrary other users. Profile sharing is limited to 30 days and can be revoked earlier; a data user can make no changes to the shared profile. However, no other restrictions exist: The data user has full access to any data that is currently stored in the profile. They can print or export the profile to a PDF document and can download files and images that have been stored in the profile by the patient without restriction. Profile sharing is complete, which means that no fine grained policies are supported that would limit the visibility to certain items in the profile.

In the case of Google Health, the data owner has to trust the data user that the data in the shared profile is not copied, printed, stored and reused for purposes never intended by the patient. The potential life-long history of the patient data and the ability to link external events with the health record would make the patient profile a favourite target for data correlation attacks. Different direct or indirect attacks can be envisioned and can be executed with or without the cooperation of the data owner. For example, one could imagine that employers force their employees to allow access to their Google Health profile as part of their job application and store the available data for later reference.

5.3.1 Data Model of the Personal Health Record

Several standards exist for the structured data representation in Electronic Health Records (EHRs) (cf. CEN/TS-15211 [2006]; ISO/HL7-21731 [2006]). Since the focus of this work is not the exact representation of medical data, but rather the creation, management and enforcement of access decisions, the PHR is represented as a simple XML document structure, which is flexible enough to incorporate standard-based data representations if necessary. Health record data are stored in a structured way and data access policies can be applied to these structures.

Hierarchical grouping is a widely used concept in the field of access control. It minimises the effort for policy management - rules can be defined and enforced at the group level, thus minimising the number of rules in the policy.

All treatment records generated by the same practitioner are stored as virtual examinations in the document tree (cf. Figure 5.3). All treatment records generated by the practitioner (a $1 : m$ relationship between practitioner and examination is assumed) are stored under this particular node and form a single zone of trust similar to the *existing patient – practitioner relationship*. The practitioner, as data author, has specific access rights for his or her sub-tree in the personal health record of the patient. These rights can be specified as a generic rule affecting all groups of a certain type and are applied consistently for every instantiation of this type.

5.3.2 Protection of the Personal Health Record

The patient data and the included XACML policy form a *Protected Data Object* (cf. Section 4.3). Our prototypical implementation uses the XML Encryption standard [XML Encryption, 2002] to protect data and policies from access and modification when they are stored on the system or during transit.

XML Encryption is a very flexible encryption standard that can be used to encrypt individual XML elements, XML elements including their child elements or whole XML documents. Our system uses a symmetric encryption key that is shared between all reference monitor instances in the framework and encrypts privacy sensitive data in the PHR using the Advanced Encryption Standard (AES) [FIPS PUB 197, 2001] in Cipher Block Chaining (CBC) mode.

The current implementation does not offer a very high level of security and it may be possible to extract the shared key through memory or byte-code analysis. Since we have not implemented any re-keying mechanism the security of our Protected Data Object can be broken. The implementation can be made much more secure through the inclusion of a Trusted Platform Module [Trusted Computing Group, 2007] for secure key storage and encryption and decryption support.

5.4 Policy Decision Component

Owner-defined policies in our privacy enforcement architecture are described using XACML and must be evaluated at the time of data access in order to determine the access decision. It is the responsibility of the *Policy Decision* component to evaluate the current policy and generate an access decision through a deterministic policy evaluation algorithm.

We used an existing Java-based XACML implementation that had been provided by Sun under a liberal open-source license [SunXACML, 2006]. This implementation already supports the main elements of XACML-2.0 [2005] and can be modified and extended freely. We implemented our priority-based policy combining algorithm for the support of precedence relationships within policies (cf. Section 3.5) and further extended the capabilities of the XACML language to support dynamic referencing, as will be described in Section 5.4.3.

5.4.1 Extended Authorisations

It is a distinctive feature of the use case that data ownership and authorship for PHRs are shared between patient and practitioner. The principals in the medical use case have shared responsibilities and rights that are derived from the legal context of medical practice. While the patient is the owner of the data, he or she is not the author of medical data. Only authorised practitioners are allowed to create medical entries in the repository. Otherwise it would be possible for the owner of the PHR to forge examination results, which might lead to misdiagnosis or other unintended consequences.

We model this distinction through the additional role of *data author* which is assigned to the diagnosing practitioner and gives special rights to this role:

Definition 12: Data Author

The data author is the subject that contributes medical data to the Personal Health Record of the data owner. Information about the data author is recorded so that it can be clearly identified. Data authors should always be able to access the data that they have created themselves.

The data user is usually a practitioner, who wants to access a prior diagnosis or treatment report that has been created by a different author. Other data users, such as physiotherapists, pharmacists, etc., could be supported but are not explicitly mentioned in the current design.

5.4.2 Use Case Policy Example

The following list of rights and restrictions explains some essential properties of our implementation of a PHR policy:

Patient rights and restrictions The patient is the owner of the Personal Health Record and administrates the sharing policy for the document. Instead of the sharing policy by Google Health, which reveals the complete document, our policy model supports fine grained control, where the patient can allow and restrict access to different parts of the document.

- Patients can grant access rights for practitioners to read examination entries of other practitioners
- Patients can grant the right to export entries from the health record into the medical information system of the practitioner
- Patients have no right to create/modify entries in the *Patient Data* branch of the document.

Practitioner rights and restrictions In our use case, practitioners can add medical data from examinations and treatment processes to the electronic repository. For this purpose the repository is sub-structured into separate compartments that will be guarded by an appropriate access policy. The implementation of a suitable policy-set guarantees the same level of privacy between the different visits to practitioners that the patient can currently expect.

- Practitioner can create new examination entries in his/her practitioner sub-tree
- Practitioner can read examination entries from his/her personal practitioner sub-tree

Use of prioritised Sub-Policies for Policy Management

We use our *Privacy Policy Precedence Relation (P3R)* from Section 3.5 to define different prioritised sub-policies that are evaluated by the Policy Decision Point:

- A *default policy* is attached to the PHR as soon as it is created and can not be altered by the data owner. The default policy will be a ‘closed’ policy that denies every data access that is not explicitly allowed.

- The *owner policy* rules can be modified by the patient to reflect the privacy preferences of the patient. For example, a patient grants a practitioner access to the examination results from a previous period of illness.
  If the owner policy is deleted, the PHR will again be protected by the default policy.

- The *safety policy* is not under the administrative authority of the data owner and describes authorisations that need to be guaranteed, such as the *Data Author Rule* shown in Listing 5.3. Other examples are the rules that ensure that the patient as data owner can always access the stored data, but is not able to modify medical data.

5.4.3 Dynamic Referencing and XPath Evaluation

Based on the requirements of our use case, new resource-nodes and permissions can be added and deleted from the XML document-tree at any time. We therefore need to base authorisations on XML attributes within this dynamic document structure and have to find, access and process XML attribute values as part of the XACML policy evaluation.

An example for this requirement comes directly from our use case, where we gave a *data author* the right to access data that had been provided by him or her. The PHR contains the identities of all the practitioners that have contributed to the patient health history. In order to allow access to the sub-tree of the health record that contains the data that has been provided by this *data author*, we need a rule that compares the current data user with the recorded practitioner-id of the currently selected resource sub-tree.

The XACML standard provides XPath expression-based functions for the selection of XML attributes and uses the `<AttributeSelector>` element to identify a particular attribute value based on its location in the request context. The `RequestContextPath` attribute of the `<Attribute-Selector>` element takes a static XPath expression and evaluates it to a bag of values, given by the `DataType` attribute. However, this function has the drawback that the attribute value for the `RequestContextPath` attribute handles only fixed XPath expressions that must be fully known at policy creation time.

In our example this does not work, since we need to evaluate the request relative to the currently selected resource. We therefore had to define and implement a new XPath-based function that enables dynamic referencing and comparing of XML nodes relative to the currently selected resource.

Our function `<function:xpath-node-element-x500-compare>` takes two arguments:

- The first argument is of data type: `urn:oasis:names:tc:xacml:1.0:data-type:x500Name`
- The second argument is of data type: `http://www.w3.org/2001/XMLSchema#string`
  and this argument is interpreted as an XPath expression and evaluates to a
  `urn:oasis:names:tc:xacml:1.0:data-type:x500Name`

This function allows us to dynamically create XPath expressions for the second argument, using the standard XACML string manipulation functions. In our case we concatenate a partial location path with the currently selected resource in order to access an element relative to the currently selected one. Both arguments of the function are treated as `X500Name`-values. The function compares the arguments and returns the result of the comparison as a `boolean`.

Listing 5.3 shows the application of this function in a specific policy rule that grants data authors access to their own sub-tree in the PHR. If there is a match between the `subject-id` of the access request and the `practitioner-id` of the selected resource, the function evaluates to **`true`** and data access is granted.

```
<Rule RuleId="dataAuthorRule0" Effect="Permit">
    <Target>
        <Subjects>
            <AnySubject />
        </Subjects>
        <Resources>
            <AnyResource />
        </Resources>
        <Actions>
            <AnyAction />
        </Actions>
    </Target>

    <Condition FunctionId="function:xpath-node-element-x500-compare">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:x500Name-one-and-only"
            >
            <SubjectAttributeDesignator DataType="urn:oasis:names:tc:xacml:1.0:data-
                type:x500Name" AttributeId="urn:oasis:names:tc:xacml:1.0
                :subject:subject-id" />
        </Apply>
        <Apply FunctionId="urn:oasis:names:tc:xacml:2.0:function:string-concatenate">
            <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-
                only">
                <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0
                    :resource:resource-id" DataType="http://www.w3.org/2001/XMLSchema#
                    string" />
           </Apply>
           <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
               >/../../../../@id</AttributeValue>
        </Apply>
    </Condition>
</Rule>
```

Listing 5.3: Default Permit Rule for Data Authors

5.5 Policy Enforcement Component

The Policy Enforcement component is a very important item in the privacy enforcement framework, because its enforcement properties strongly influence the ability to write meaningful policies and have consequences for the security and trust that such a framework can provide. The enforcement mechanism must support the separation of policies and protection mechanisms [Saltzer and Schroeder, 1975; Woo and Lam, 1993], and must be flexible in its deployment on different computing platforms. Based on the requirements of our use case, it is necessary to support access control restrictions that work at the sub-document level and enforce different access restrictions for different document parts.

The Java programming language has been build from the ground up with a dedicated security and trust model that supports the fine grained specification of access permissions for different system resources. It also has an efficient and tested reference monitor in the form of the Java Security Manager.

We therefore developed the idea to implement a mapping between XACML privacy policies and Java permissions and use the existing Java Security Manager to enforce privacy protection at the level of Java permissions. Our own Policy Enforcement component translates the XACML privacy policy into a corresponding set of Java permissions, which is the basis of a dedicated `PermissionCollection`. It then binds this `PermissionCollection` to the application class that is processing the private data and starts the application under the control of the Java `SecurityManager`.

The enforcement of privacy restrictions at the level of the Java virtual machine has many advantages: We require only very little adaptation at the application level. We can work with untrusted application code-bases, because the Java Security Manager will enforce the necessary access protection. No application code checks are necessary, a method-call succeeds if access is granted and fails with an exception from the Java security framework if it has not the necessary permissions.

Several problems had to be solved in order to build a practical privacy enforcement from this ideas. Java permissions are typically class-based (all application instances share the same policy setting), but we wanted to restrict the application depending on the user and the data that this application is accessing. A second problem is that XACML policies are much more expressive than Java permissions. A suitable translation had to be found and it needed to be shown that Java permissions have sufficiently strong protection properties for the expression of privacy policies.

5.5.1 Enforcing XACML-Policies through the Java Security Manager

The Java security manager can not directly enforce our XACML privacy policy. We need to map the XACML policy actions into a corresponding set of Java permissions, so that our privacy policy can be enforced through the Java Security Framework. It therefore becomes the responsibility of our policy enforcement component to generate an appropriate `PermissionCollection` from the XACML privacy policy, bind this `PermissionCollection` to the application code and safely launch the application program under the protection of the Java security manager.

The permissions of a Java application are normally derived from the standard `java.policy` file, which is not a very flexible way to handle dynamically changing permissions. Our enforcement framework needs to change this standard behaviour and derive the set of permissions for the `PermissionCollection` from the XACML policy of the currently accessed data object. Two alternatives exist to base permissions for Java classes on XACML policies rather than the `java.policy` file:

1. We could write a new implementation of the `Policy` class that derives permissions directly from XACML policies, rather than policy files and uses the standard Java class loader, or
2. we could implement our own class loader that is able to assign a custom `PermissionCollection` when we load the class. The `PermissionCollection` will be derived from an XACML policy by our own code and does not use the standard `Policy` class.

Writing a new `Policy` class would allow us to derive a `PermissionCollection` directly from XACML. This `PermissionCollection` would then become part of a `ProtectionDomain` and could be bound to the application code via the Java class loader (see Section 5.1.3). The drawback of this solution lies in the fact that this `PermissionCollection` applies equally to all instances of the application code, because a normal class loader caches the `ProtectionDomain` and reuses it for all instances of the class.

We therefore choose the second approach and implemented our own custom class loader.[1] Each `ProtectionDomain` also includes a reference to the class loader that was used to load the class. If we can use a different class loader instance for every application instance that we load, we have the ability to differentiate protection domains and assign different policies for application instances. Section 5.5.3 describes the exact changes to the Java class loading process that is implemented in our solution. When we use the Java security manager for policy enforcement and want to bypass the Java `Policy` class, we also need to adapt our XACML policy evaluation strategy. A policy decision request is normally issued at the time when an application function accesses the protected resource. This is also the case within the Java Security Framework. The Java security manager consults the `PermissionCollection` of the current `ProtectionDomain` to make an access decision. The `ProtectionDomain`, however, must have been determined and set already at class loading time.

We therefore have to translate our XACML policy into a Java `PermissionCollection` *before* a new application instance is loaded. This transformation requires the issuing of an XACML decision request for the currently requested resource, user and every potential action. The policy decision included in each XACML response is then used to populate the Java `PermissionCollection`. If an action is granted an adequate permission is added to the `PermissionCollection` and left out otherwise. The peculiarities of this translation process will discussed in the next section.

The timing of the decision requests is problematic, because when we issue the XACML requests we do not yet know which methods will be invoked later in the application program. All we know is the current user and the resource that needs to be accessed. We solved this problem in the following way. Our Policy Decision component determines an XACML policy decision for the complete set of actions that are supported within our policy. Several XACML requests are executed in rapid succession, which evaluate every potential action for the currently active data user. This action set is currently static and includes the *read*, *copy*, *save*, *print*, *append* and *delete* action, but could also be determined dynamically from the consulted XACML policy object.

5.5.2 Translating an XACML-Policy into Java Permissions

The translation of the XACML policy in Java permissions generates a restricted execution environment that enforces access control decision through the Java security manager without support from the application itself. The application can implement a full set of functions for data handling without awareness or consideration for the specific privacy policies that will be enforced by the security manager.

This policy translation process, is not without problems. Java policies and XACML policy differ in their level of expressiveness. While an XACML policy can express abstract concepts of subjects, resources and actions, a Java policy needs to be a concrete instantiation of that policy that can be enforced within the current runtime environment. We therefore have to map our XACML decisions into corresponding Java permissions that can be enforced by the Java security manager. This mapping is not arbitrary and we need to make sure that the concepts that are expressed in our XACML policy match the enforcement properties of the Java Security Framework. Furthermore, there might be certain actions that can not be expressed as simple single-step permissions or might

[1] A first implementation was developed as part of the diploma thesis of Stefan Geiß [Geiß, 2007]. The system was later extended to support explicit priorities for sub-policies and the corresponding conflict resolution approach.

require additional support from the execution environment. An example is the *append* action for non-sequential XML documents that can not be enforced at the level of the Java `FilePermission`.

Such actions can only be partially mapped to a corresponding Java permission and need additional enforcement support from the policy enforcement framework or the application itself. Which requires that we need to trust the application to implement and enforce access checks in the methods that handle private data. Table 5.1 shows the mapping that we use to translate XACML policy responses into Java permissions.

Table 5.1: Mapping of XACML policy actions against Java Permissions

		XACML policy actions					
		read	***copy***	***save***	***print***	***append***	***delete***
Java Permission	AWT:accessClipboard		x				
	Runtime:queuePrintJob				x		
	FilePermission:read	x					
	FilePermission:write			x		x	
	FilePermission:delete						x

Semantics of *action* attributes and data-flow protection

One extended goal for our privacy enforcement framework is the support for data-flow protection. If our application issues a function call that is not supported under the current access policy, the Java security manager intercepts and blocks the execution of this call. However, this behaviour alone does not yet enforce data-flow protection.

As we have seen from the example of the Discretionary Access Control (DAC) model in Section 2.3.2, if we grant the permission to *read* a particular file, this also gave the user the ability to redistribute the file-content, because we could not further restrict the access to the filesystem, the network or the printing system. The execution environment of the Java sandbox allows the definition of such restrictions depending on the sensitivity of the accessed data. If we want to enforce data-flow protection, we can define of a very narrow meaning for the granted actions and can enforce them as described in the following:

Read The application has read-access to the protected resource. The data can be transiently visualised on the local system, such as displaying it on-screen for the purpose of reading or viewing by a human user. The data can not be used for any other purpose, such as writing it to a file under the control of the data user.

Write/Save/Append The application can write to the protected resource. User input or other data accessible to the application can be written to the protected resource. It needs to be further defined if write-access also implies the ability to delete and/or modify existing data or if new data can only be appended to the existing data.

Print The application has read-access to the protected resource. The data can be processed and send to the system printer.

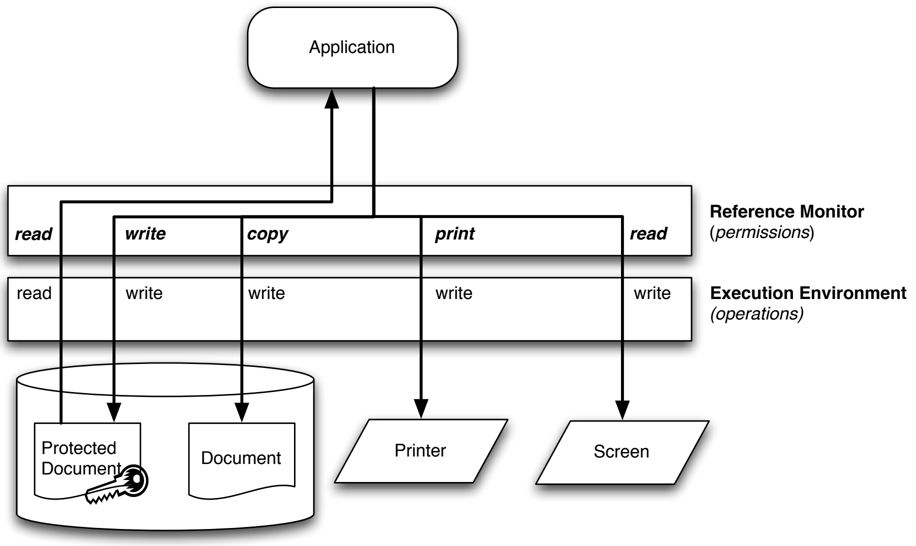


Figure 5.4: Restricted application execution - The reference monitor implements and enforces a narrow action semantics to support data flow protection for specific access primitives.

Copy The application can read-access the data. The data can be further processed and copied to resources outside the protected resource. Data flow is no longer protected, although it might also be possible to copy policy and data owner information and create a new Protected Data Object (see the work of Sevinç and Basin [2006] for an extended discussion of such functionality).

Figure 5.4 shows how this actions could be implemented in the protection environment of the Java Security Framework. If the policy defines data-flow protection (authorising *read-only* data access), the reference monitor restricts the interaction of the application with external processes, files, devices such as printers, remote filesystems and network sockets and the system clipboard.

The definition of a narrow meaning for granted actions has the benefit that this model supports the intuition of the data owner. Because in most real-world situations restricting data access to *read-only* is associated with the property of data confinement and restricted distribution. Under the narrow meaning, the *copy* action is the only action that breaks data-flow restrictions. But even here can the data owner trust its intuitive understanding of the policy and can grant this permission only if the data user is fully trusted to handle copying and forwarding of private data.

5.5.3 Assigning Instance-Level Permissions

The current Java Security Architecture is targeted towards class-based policing. Trust is attributed to the origin of the code and not to the running instance based on this code. This behaviour is a consequence from the distinctive Java threat model which attributes trust on the creator of the code and severely restricts permissions for code that stems from unknown sources such as Applets that have been downloaded from the Internet. The application user, on the other hand, is fully trusted to run and operate any program on his or her computer and modify the Java policy settings accordingly.

For the realisation of the Owner-Retained Access Control (ORAC) we need to make changes in this trust model. We now want to limit access to resources depending on the privacy policy of this resource and need to limit the actions of the application user. This means that the reference monitor has to enforce different policies depending on who is accessing a resource (the data user) and the privacy settings of this particular resource. The application (code source) no longer determines the applied protection preferences. We may even invoke the same application code while enforcing different privacy policies and protection preferences.

An instance-based policing is necessary to distinguish between multiple invocations of an application that accesses different data items with varying privacy policies. Data users need to be restricted in their actions, based on the privacy policy of the data that they are trying to access. When a data user accesses different data objects during one session, it becomes necessary to enforce more than one access policy simultaneously and keep application instances separate.

We already mentioned that we decided to develop our own class loader for the privacy enforcement of XACML policies. The assignment of a new `ProtectionDomain` to a class is only possible at class loading time. It is important to know that Java provides no mechanism to change or revoke a `ProtectionDomain` after the class has been loaded, so it should not be possible for the application to break out of the Java sandbox. We started our implementation with the extension of the `ClassLoader` class. The `ProtectionDomain` class has at least two different constructors, whereby one of them ignores any permissions defined by the `java.policy` file and creates a static set of permissions. Our class loader overrides the `getPermissions()` method, which allows us to create the `PermissionCollection` directly from the decisions of our XACML policy.

The default implementation of the `loadClass()` method in `ClassLoader` loads a class in the following order (cf. Gong, Ellison, and Dageforde [2003]):

1. Call `findLoadedClass()` to check if the class is already loaded. If this is the case, return that object. Otherwise,
2. call the `loadClass()` method of the parent class loader, in order to delegate the task of class loading to the parent (this is done to ensure that system classes are only loaded by system class loaders).
3. If none of the parent class loaders in the delegation hierarchy is able to load the class, the `findClass()` method of this class loader is called in order to find and load the class.

Dynamic policy enforcement requires the construction of a new `ProtectionDomain` for every data object that will be accessed. Under the default behaviour of the `loadClass` method, the existing class would be found and re-used and we would end up with the same `ProtectionDomain` for each application instance.

This behaviour needs to be changed when we want to support instance-based privacy policies. In order to decide how such a feature could be implemented without minimal impact, we analysed how Java implements the separation of namespaces. Namespace separation for applets loaded from different websites is enforced in Java through the use of different class loaders. Two classes are considered distinct if they are loaded by different class loaders and therefore can have a different set of permissions. We decided to use a similar approach and modified our own class loader, so that it creates a new class loader instance each time we need to load the application class with a different policy.

Our privacy class loader uses a modified `loadClass()` method that no longer calls `findLoadedClass()` to check if the parent class loader already knows this class. Instead `findClass()` is called directly to load the application class with a new `ProtectionDomain`. Figure 5.5 shows the resulting class loader hierarchy, both for classes loaded from different code sources and classes loaded with different privacy policies for their data.

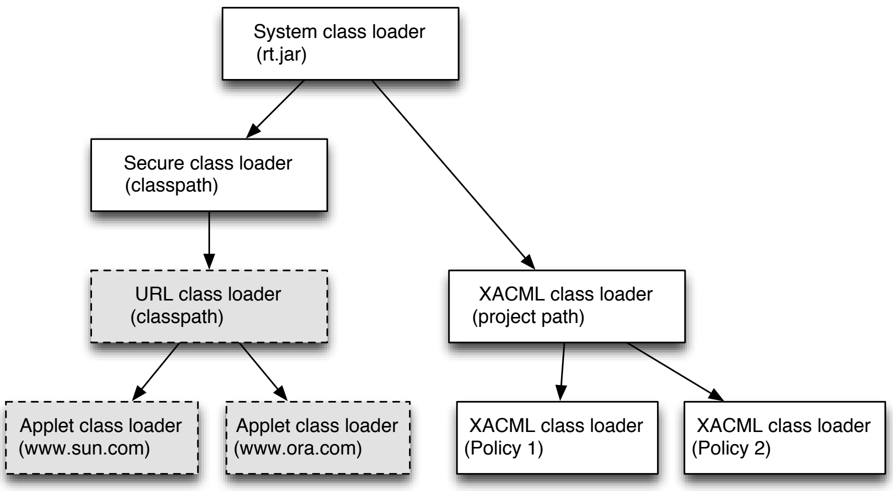

System class loader (rt.jar)
Secure class loader (classpath)
URL class loader (classpath)
XACML class loader (project path)
Applet class loader (www.sun.com)
Applet class loader (www.ora.com)
XACML class loader (Policy 1)
XACML class loader (Policy 2)

Figure 5.5: Java class loader hierarchy - Java normally loads classes from different code sources with dedicated class loader instances (left side) [Oaks, 2001]. With our modified class loader (right side) each instance of the application is loaded with a dedicated protection domain that enforces resource access checks through the Java Security Manager.

5.6 Resource Browser

The current Java implementation of our enforcement framework has a small bootstrap problem. Whereas normal applications can be started and documents can be loaded from the application itself, our privacy protection framework requires that the document is selected first, the corresponding privacy is referenced and resolved and only then can the application be started under the control of the Java security manager. If the data user wants to access a new or additional document, a new `ProtectionDomain` has to be constructed and the application class has to be reloaded.

This requirement has consequences for the application workflow. We need an additional component in our framework that accesses and loads the private data from the protected data object and starts the application with the necessary policy. Policy and data resources use a suitable XML-representation and are stored together in a *Protected Data Object* (cf. Section 4.3). We assume that a Protected Data Object contains multiple data elements that might have different permission settings and will be referenced via XPath [XPath, 1999]. The *Resource Browser* component in our

PrivMon privacy enforcement framework allows the data user to securely interact with the private data of the data owner (see Figure 5.2).

The data user needs to authenticates itself, so that the policy decision component can evaluate the decision request correctly. The data user utilizes the Resource Browser to identify the requested resource node within the XML document-tree. Once this selection is made, the Resource Browser invokes the Policy Enforcement component that requests a policy decision from the Policy Decision component and subsequently launches the application.

5.7 Implementation of a Health Record Viewer

So far we have focused our discussion on the implementation of the PrivMon privacy monitor. In order to test our assumptions about the protection properties of the privacy enforcement framework it became necessary to also provide an application component.

We implemented a so called *Health Record Viewer* (HRV) that is able to display graphical medical data, such as X-ray images, which have been stored as XML-encoded resources in the Protected Data Object. Depending on the privacy policy, the graphical data can be viewed, saved as a JPEG-file, copied to the system clipboard, printed on paper or a combination of the above. The append and overwrite functions were added as placeholder for further development, whereby we wanted to support the insertion and change of data in the Personal Health Record itself. Figure 5.6 visualises the implemented functions within our Health Record Viewer (HRV) application.

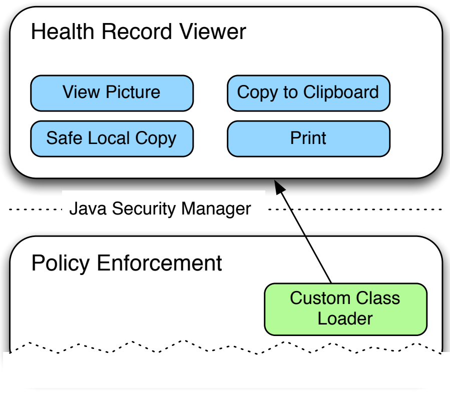
The Resource Browser component of the PrivMon lets the data user authenticate and select interesting events in the Personal Health Record. If a suitable resource has been selected, the data user invokes an application component, such as the Health Record Viewer. The HRV application will be started under the control of the Java security manager and is given access to the selected data element.

An appropriate permission setting will be derived from the XACML policy stored in the Personal Health Record. If the privacy policies allows it, the HRV will visualise the medical images contained in the personal health record, store local copies of the data or print the images. The application itself can implement the full set of functionality without having to care about privacy settings and no special trust is required into the safe implementation of these functions. Execution of the functions will be restricted at runtime according to the specified data-use policy by the Java security manager.

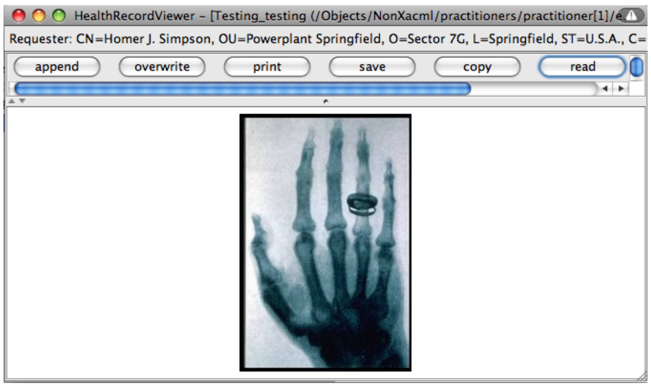
Multiple instances of the HRV can be started simultaneously for different data objects and may allow the visual comparison of different diagnoses or illnesses. Each instance of the HRV will carry its individual set of permissions based on the data that is being accessed and is not influenced by other running instances of the HRV.

5.7.1 Results

The XACML policy is evaluated at the moment before the application is loaded via the PrivMon privacy monitor. The PrivMon iterates through the set of actions contained in the policy-base for a given subject/resource pair to gather all the related permissions and generates an appropriate set of Java permissions. The actual policy enforcement is offloaded to the Java Security Framework and XACML requests have to be evaluated at the time of resource access of the HRV. The performance



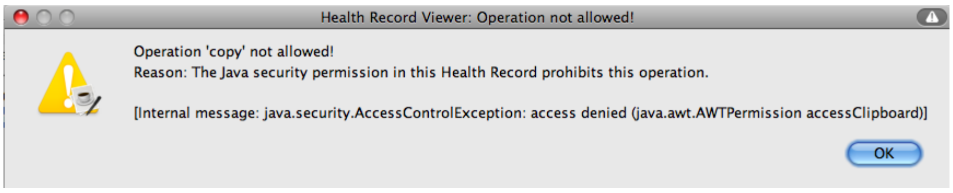
(a) Application example implements different functions


HealthRecordViewer – [Testing_testing (/Objects/NonXacml/practitioners/practitioner[1]/

Requester: CN=Homer J. Simpson, OU=Powerplant Springfield, O=Sector 7G, L=Springfield, ST=U.S.A., C=

(b) Application screen shot

Figure 5.6: Privacy-aware Applications: Personal Health Record Viewer - Application components are started under the control of the Java Security Framework and support the privacy-aware execution of application functions.


Health Record Viewer: Operation not allowed!

Operation 'copy' not allowed!
Reason: The Java security permission in this Health Record prohibits this operation.

[Internal message: java.security.AccessControlException: access denied (java.awt.AWTPermission accessClipboard)]

Figure 5.7: Java AccessControlException - An exception is thrown if the application component tries to access a method that requires permissions that have not been granted to the class.

of the application component therefore does not depend on the parsing and evaluation of XACML policies.

Privacy policy enforcement works as expected. The application component is effectively controlled through the Java Security Framework without any cooperation from the application itself. If a necessary Java permission is not available while the user tries to exercises a particular function, the Java security manager throws an AccessControlException, which can be intercepted by the application to show a meaningful error message to the human user (cf. Figure 5.7). Different instances of the application can be started simultaneously with different sets of permissions and allow the effective protection from unintended data-flow between different applications and the application and the operating system.

A working prototype of the privacy protection system for Personal Health Records was presented at *CeBIT 2008* in Hannover, Germany, at the *3rd Telematik-Konferenz 2009* in Potsdam, Germany and *conhIT 2009* in Berlin, Germany.

5.7.2 Restrictions

The direct enforcement of privacy policies is currently limited by the support of native permissions in the Java Security Framework – which is primarily focused to support the original Java threat model.

We could extend the Java permission model through application specific extensions, however in this case the application needs to be trusted to correctly implement the necessary access checks. Implementation of new permissions would therefore require that we extend our trust also to the application, whereas currently only the PrivMon privacy monitor needs to be trusted.

Policies that can not be directly enforced through the native Java permission mechanism include the support for controlled write and change operations within the data structure of the Personal Health Record. These functions currently need specific support from the PrivMon. Another example are time-dependent policies, which enforce access restrictions based on the current date or time. Time-dependent policies can currently only be enforced at application start-up. Their correct enforcement requires reliable access to a trusted time-base, because it would otherwise be possible to trick the reference monitor into incorrectly permitting data access.

5.8 Performance Measurements

One open question for the practical generation of the set of Java permissions from the XACML policy description is the expected performance of this approach. It must be possible to derive the Java `PermissionCollection` quickly, so that a human user can use the system in an interactive manner, without experiencing delays at application launch. Once a Java permission set is generated, the Java Security Manager automatically enforces these permissions. Since the policy enforcement from the security manger is standard Java behaviour for any Applet, we expect that the enforcement of these permissions will be highly optimised and does not slow down the execution of our application component.

When the data user selects a certain data-set from the Resource Browser, the Policy Enforcement component will issue a series of XACML-requests, gathering the access decisions for all the supported actions from the XACML-policy description and forms a Java `PermissionCollection` from the response.

We needed to measure the additional delay this operation incurs on the loading of the application via our custom class loader. This required the adaptation of our implementation. We implemented an additional method `generateRequests()` in our Reference Monitor class, which repeatedly calls the original `generateJavaPermissions()` method and measures the execution time of the XACML-request using `System.nanoTime()`. The measurement platform consisted of a single-core Pentium-M 1.73 GHz, using 1 GByte RAM and the Java 2 Runtime Environment - Standard Edition (build 1.5.0_14-b03) running on OS Windows XP SP3.

We formed the hypothesis that the execution time for XACML requests depends on two factors: a) the number of rules contained in the XACML policy and b) the number of resources that these policies need to be matched against, since the policy decision process always considers the complete XACML without stopping the evaluation after a first match has been found.

For the test we generated two sets of files, one containing a single data-set and two exemplary rules, the other containing 25 data-sets together with 20 XACML rules. The policy rules were mainly *permit* rules so no conflicts needed to be solved by the XACML `rule-combining-algorithm`. In policies that contained multiple resources, each rule contained multiple references to the resources in order to reference every resource in the data-set.

Each policy was tested repeatedly 100 times, against a set of 6 different action attributes, generating 600 XACML-requests. The repetitions were done in order to offset singular effects and gain a level of confidence in the reliable reproduction of the measurement results. The test results (see Figure 5.8 and 5.8) show that a single XACML-request can be evaluated in approximately 120-250 microseconds[2]. Further, the execution time shows almost no influence by the amount of rules that had to be processed. Similar results were obtained for policies that contain multiple resources. Figures 5.9 shows the results for the XACML policy evaluation with multiple resources in the policy. We see a little bit more variance in the execution time of the policy decision for read-requests with multiple resources.

We currently use the standard SunXACML implementation [SunXACML, 2006] and do not employ any optimisation, such as support for XACML-requests that include more than one action attribute, or caching of decision results. The measurements have shown that the current execution time is fast enough, even when several requests have to be made in sequence to generate the Java `PermissionCollection`.

In very few cases (<1%) we saw some outliers in the execution time, notably in the processing of the larger resource files, where the evaluation of an XACML-request took up to 3 ms. These outliers were randomly distributed and do not seem to be caused directly by the XACML-evaluation code.

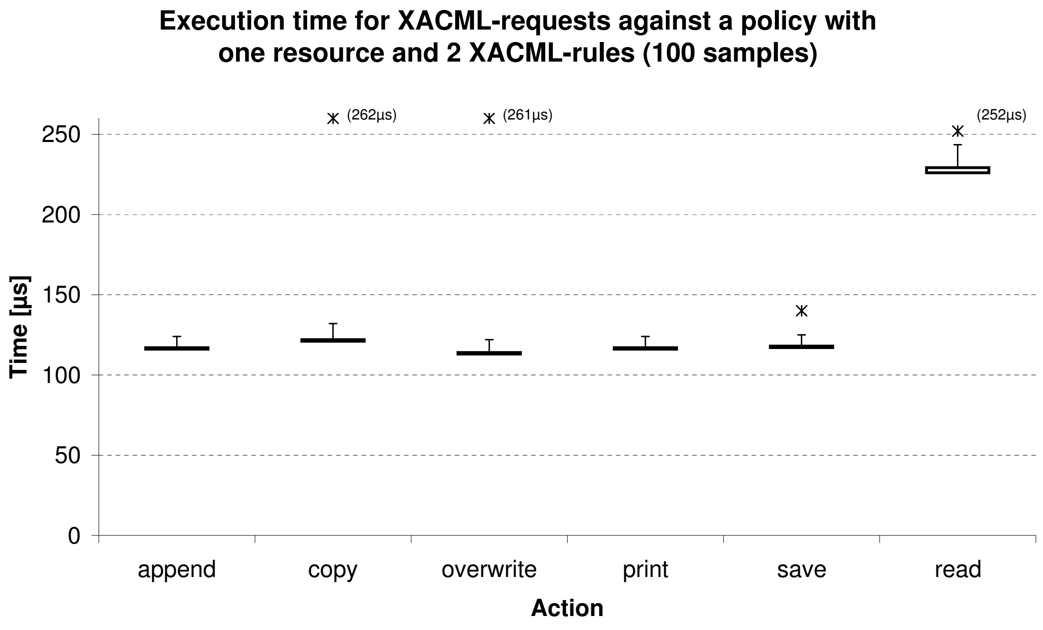


Figure 5.8: Execution times for XACAML requests - single resource / two XACML rules

[2]The error bars represent the distribution of 95% of the observation results. The boxes contain 75% of our observations.

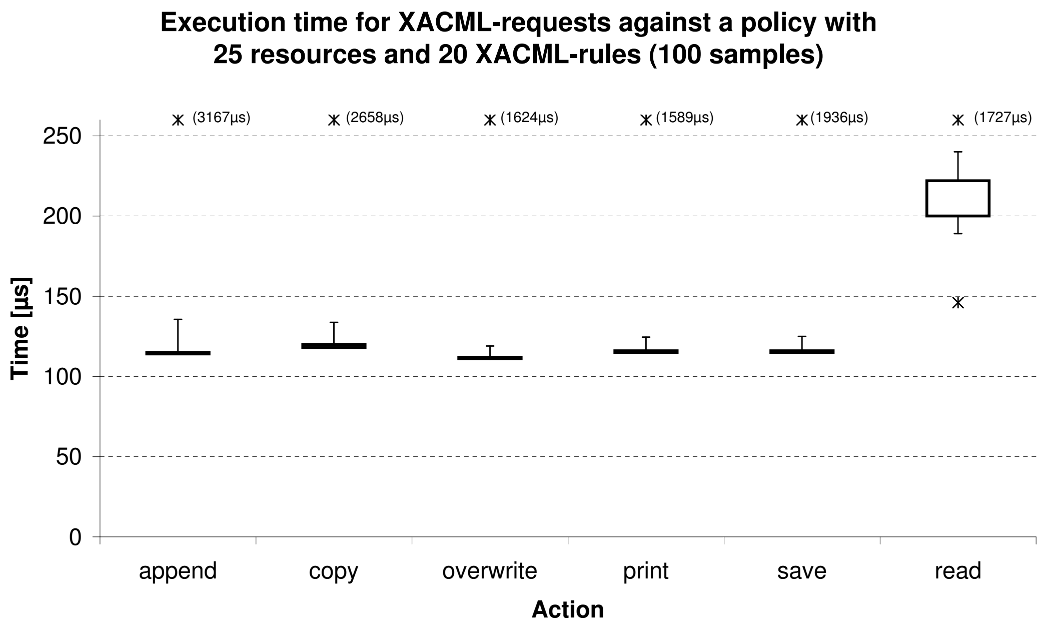


Figure 5.9: Execution times for XACAML requests - multiple resources / multiple XACML rules

5.9 Summary

The development of the PrivMon privacy monitor is the main proof for our concept of a client-side privacy enforcement architecture of owner-defined policies.

We choose to base our implementation on the Java Security Framework and use the permission concept of the Java security manager for the enforcement of privacy policies. This allows us to treat the application as untrusted code that could be contributed by external parties. The application can implement a full set of functions, even functions that are likely to be disallowed by the policy. The application does not need to implement and perform any permission checks against the policy itself. It has full control over its internal data-structures and can make arbitrary function calls.

Our client-side privacy enforcement framework forces the application to treat data from the Personal Health Record (PHR) as ephemeral data. The data can be accessed freely within the sandboxed environment, but data-flow outside the sandbox is restricted and the data object is deleted from memory, once the application finishes using it. The application is part of a containment architecture, where data access is only possible through a reference monitor that enforces the privacy policy for each and every access request.

The privacy settings are documented as declarative policies and are bound together with the medical data in the XML structure of the Personal Health Record as *sticky policies*. The policies support fine-grained control over individual data-elements in the XML-document structure.

XACML policy decisions are translated into *Java Permission Collections* that control resource access and use. These policies apply to the running instance of the application class. Our custom class loader generates a new class loader instance for every application instance and creates a unique Java `ProtectionDomain` for each application instance. Client applications will be started by the adapted class loader and run under the full control of the Java security manager. If the application tries to invoke a method for which it has not been granted the necessary permissions by the current privacy policy, the Java Security Framework intercepts the call and throws a security exception.

The enforcement framework uses a very narrow definition for the *read* action attribute in the policy rules that deviates from the usual interpretation of the attribute in the Discretionary Access Control model (cf. Section 2.3.2). Private data with a *read-only* permission can only be visualised on the computer screen of the data user. Data-flow protection prohibits the storing, processing or forwarding of this data outside the protected environment. We think that this narrow interpretation is better suited to capture the intuition of the data owner and prohibits unintended data breaches through carelessness and neglect from the side of the data user.

6 Privacy Protection for Server-based Information Systems

The previous chapters presented our solution for a client-side privacy enforcement scheme and a corresponding reference monitor implementation based on the Java Security Framework. The approach uses sticky policies written in the eXtensible Access Control Markup Language (XACML) to formulate access rules for protected resources under the Owner-Retained Access Control model. The reference monitor converts the XACML policy of the accessed resource into Java permissions. These permissions will be attached to the application class when they are loaded with our custom class loader. This solution has the advantage that it supports the easy implementation of a client-side reference monitor. Application developers do not need to evaluate privacy policies and enforce security checks, because they are already done by the Java Security Manager.

A growing number of applications are now based on the client-server model, where a lightweight client, such as a web-browser, uses services and resources stored on a network connected server. Client-side data privacy protection can not provide sufficient privacy protection for such scenarios, because private data is processed by and forwarding between different service instances. We need to answer the question, if our solution can be applied to the problem of enforcing privacy policies in server-based usage scenarios or if it is necessary to extend our work and develop enforcement solutions that offer enforcement for owner-defined privacy policies in such scenarios.

We base our evaluation on the implementation of two privacy-aware localisation services. The first is the localisation component of the *KopAL* ambient assisted living system, the other solution uses aspect-oriented programming (AOP) techniques within a hypothetical theme park localisation service for the privacy enforcement in tiered business applications.

6.1 Privacy for Location-based Services

Location data belong to the fastest growing data types within business and private applications [Manyika, Chui, Brown et al., 2011]. The continuously growing smartphone market, as well as the emerging 'Internet of Things' enable service providers to very accurately track and map persons and their devices through a variety of different localisation mechanisms. The unbroken interest in social networks entices users to make their private data available to various services and it is not uncommon that devices and applications indiscriminately collect location data [Allan and Warden, 2011].

In most cases, a user has no ability to disallow data access or to restrict the data collection to a specific purpose. Additionally, location data might also be forwarded to third parties for processing and storage. A user cannot be sure that collected data is only used for the fulfilment of a particular service and that no data is used for further purposes like market analyses or generation of movement profiles.

The privacy problems that emerge when location-based services have access to the geographic coordinates of a person have been studied by different groups of researchers. Location services that are privacy aware, typically try to incorporate anonymisation, reduction of data granularity or one-time-use of data in order to limit the possibilities of profile building and other attacks on data privacy. However, many current localisation services are still developed without explicit support for privacy functions. The referenced projects in this section stand exemplarily for the different data protection approaches taken by the community.

The GEOPRIV working group of the IETF developed a *Common Policy* for the expression of privacy preferences of location data by a presence and location server [Schulzrinne, Tschofenig, Morris et al., 2007]. We already presented this work in section 2.6.4 and will extensively compare the Common Policy with XACML in the next section.

Maaser and Langendörfer [2009] use the Java Security Framework for the safe execution of private location services by service providers. They define a Privacy Guaranteeing Execution Container (PGEC) that allows the service user to have private location data processed by the service provider while enforcing a privacy contract. The privacy contract is negotiated between service provider and user and uses extensions to P3P [P3P v1.1, 2006]. The Java permissions that are needed for the enforcement of the privacy contract are predefined for a number of data usage scenarios. A PGEC protects location information from being extracted by the service running inside the container as well as malicious programs running outside of the container. As soon as the service finished processing the private data, the container implementation deletes it. The private data is never stored permanently in the Execution Container and only used once. Privacy Guaranteeing Execution Container are targeted at the provisioning of location services to the service user itself. It is explicitly not intended to pass location data to other users or protect these data exchanges. This approach is suitable if a service only needs temporary access to location data, e.g. to compute a path or offer services based on the current location. It does not offer protection for location data that needs to be available for later processing.

Kung Chen and Da-Wei Wang describe a privacy protection approach for Struts-based[1] enterprise web applications using *Aspect-oriented Programming* (AOP) [Chen and Wang, 2007]. Their solution implements a modular enforcement mechanism that enables fine-grained access control for private data. Accesses to resources are monitored using advices. The provided concept enables a user to restrict data accesses to a certain purpose. This is achieved by binding advices to all important methods of the monitored application. These advices are aware of their methods purpose and add it to a table as soon as the corresponding method gets called. A further advice which needs the purpose to compute the access decision can use the aforementioned table to obtain it. The authors state that their concept can be used with different policy languages like EPAL. However, the provided example implementation contains hard coded decision logic instead of a sophisticated policy language.

In Kruppa [2011] the authors presented a user localisation component for the Smart Senior Project. The project tries to help elderly people to stay as long as possible in their familiar environment. The main purpose of the localisation component is the detection of disoriented or even unconscious people at home or outdoor. Each participant gets a mobile phone running Windows mobile 6.5. The systems uses the Global Positioning System (GPS) to determine the outdoor location of a person. GSM is used in cases where GPS is not available. The position of a user is

[1] `http://struts.apache.org/`

collected every 30 seconds and a set of collected locations is sent every 30 minutes via UMTS or GSM. Because GPS is not working indoor, Smart Senior uses a combination of WiFi and Bluetooth for the indoor localisation. The current solution is not privacy aware. The localisation data is gathered, sent and processed without privacy protection for the data owner.

6.1.1 Comparison between XACML and GEOPRIV Common Policy

Our solution for a location privacy service should again be based on the *sticky policy model* that requires the attachment of a privacy policy to the location data. There exist different policy description languages that can be used for the expression of authorisation rules and we had to choose a suitable language for our server-based implementation. We decided to reevaluate our initial choice for XACML and compare XACML [XACML-2.0, 2005] with the Common Policy of the GEOPRIV working group of the IETF [Schulzrinne, Tschofenig, Morris et al., 2007], to determine potential benefits and drawbacks of each language. The Common Policy has been developed specifically for the expression of privacy preferences for location data, while XACML is a generic policy description language. Most of the comparison criterions are based on the work by Anderson [2005] that evaluated EPAL and XACML and have been augmented where we felt that this is important for our work.

Rule syntax

Both languages use rules as the basic building block on which authorisation decisions are made. XACML allows the specification of positive and negative authorisations that grant or deny access to a resource. XACML uses the `Effect` attribute to define the outcome of an applying rule. Possible outcomes are `Permit` and `Deny` to allow or forbid access to a resource. In contrast, a rule in GEOPRIVs Common Policy can only grant permission. It is not possible to define a rule which denies access to a resource.

XACML policies can be filtered by the `<Target>` elements before evaluating their conditions. The `<Target>` element of an XACML rule is used to define to which set of subjects, actions, resources or environments a rule applies. Further conditions can be defined in the `<Condition>` element of a rule. In both languages the rule order does not influence the permission decision.

XACML rules follow the standard convention found in many access control schemes, where a clear structuring of authorisations in *subjects*, *objects* and *access rights* is provided (cf. Section 2.4.2). The GEOPRIV Common Policy uses a implicit definition of *objects* and *access rights*. A Common Policy is directly targeted at the referenced location data of a particular location service and does not allow the specification of an external resource object. Access restrictions for *subjects* are specified through the rule conditions, which are placed in the `<conditions>` element of that rule. The assumed *access rights* are access and use of the data itself and can not be explicitly specified.

Obligations

XACML as well as the Common Policy rules may define obligations. Obligations are actions that must be performed if resource access is granted. For example, an obligation may state that is has to be logged *when* and *to whom* access was granted.

The Common Policy distinguishes between obligations which perform a resource transformation before granting access to it `<transformations>` and other obligations `<actions>`. Transformations can be applied to obfuscate location data before giving access to it. XACML does not make this distinction between *actions* and *transformations*.

Resolving rule conflicts

Both privacy policy languages use rule combining algorithms to resolve rule conflicts. A rule conflict occurs when two rules with different outcomes are applicable.

It can happen, even though GEOPRIV Common Policy rules contain only positive permissions, that a set of applying rules has different actions or transformations. If multiple rules apply to a request, then the proposed combining mechanism generates a combined permission. The combining algorithm defined by the Common Policy separately analyses each group of actions and transformations. The resulting combined permission for this group depends on the data type of the permission. For boolean permission the combined permission is TRUE if at least one matching rule evaluates to TRUE and FALSE otherwise. Integer, real-value and date-time values are evaluated to the maximum value across the matching rule set. The resulting permissions from the individual action and transformation groups are then combined to the permission-set for this specific request.

XACML allows explicit positive and negative evaluation of rules (permit/deny), as well as the combination of policies from different sources for distributed policy evaluation. Combining algorithms are needed to derive an authorisation decision from potentially conflicting individual rules and policies and may not always be able to make a decision. A rule or policy combining algorithm may have an "indeterminate" outcome that can be resolved through a *default policy* rule (cf. Section 2.4.3). XACML allows the definition of custom combining algorithms. The following combining algorithms are defined by the language standard (cf. Section 2.6.3):

- Deny-overrides
- Permit-overrides
- First-applicable
- Only-one-applicable

Attributes

XACML as well as the Common Policy may use resource, environment or data user attributes to evaluate a policy. In contrast to XACML, GEOPRIV does not define how the attributes are retrieved and assigned.

Furthermore, XACML supports XPath references [XPath, 1999] and allows a policy to directly enclose the requested resource, which can be used to directly reference the elements of this XML resource when evaluating the policy.

Data types and functions

XACML is a strongly typed policy language that implements a set of data types from the subset of XML schema types and some XACML specific types. It is possible to combine the simple types to form complex data types.

Each attribute in the language has a well defined data type. Conversion functions have to be used when working with different types, for example in comparison functions. The GEOPRIV Common Policy does not define data types and provides no mechanism for the specification of data types without extending the policy.

The XACML standard specifies a number of strongly typed functions. They can be used to perform operations on attributes, e.g. a function could be used to restrict access to a resource to certain time spans by performing time-based operations. It is possible to use a combination of simple functions to achieve complex operations, however their function types have to match, because XACML is statically typed and has no support for type-casting. Functions are currently not supported in the Common Policy.

Purpose

The optional XACML privacy profile [XACML Privacy Profile, 2005] defines two attributes that can be used to define restrictions on the access by a *purpose* attribute.

The attribute "`urn:oasis:names:tc:xacml:2.0:`**resource**`:purpose`" can be used to describe the purpose for which this resource can be used or has been collected. This purpose definition is then matched against the attribute "`urn:oasis:names:tc:xacml:2.0:`**action**`:purpose`" in the XACML access request to find out if the resource can be used for this particular purpose. The GEOPRIV Common Policy does not provide a purpose attribute.

Request and Response Language

Besides defining the structure and semantics of security policies, XACML also defines a message format for the exchange of decision requests and responses in a distributed authorisation architecture. A Policy Enforcement Point (PEP) can use this format to send a decision requests to a Policy Decision Point (PDP) for evaluation and get a well defined answer back. The GEOPRIV working group does not define a message format for decision requests and responses. RFC 5491 only specifies a "Location Object" that can be used to represent location information and may be contained in a response from the location server [Winterbottom, Thomson, and Tschofenig, 2009].

Implementation

There exist many implementations to ease the creation, evaluation and validation of XACML policies under a wide variety of licences, some of them Open Source[2]. Implementations of policy decision and support tools for the Common Policy are not yet available. The IETF expects that the GEOPRIV Common Policy will be applied in their future standards, especially in their VoIP protocols [Morris and Peterson, 2007]. An implementation that is comparable to the XACML implementation from SUN [SunXACML, 2006] does not yet exist.

Suitability for Privacy Policies

After comparing the GEOPRIV Common Policy and XACML we found that XACML is a richer policy description language that offers greater flexibility for the generation of privacy policies.

[2]A list of available XACML implementations is provided on the OASIS XACML project website: `https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml`

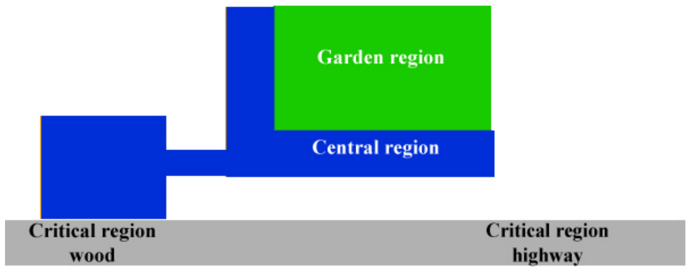


Figure 6.1: Map of the Stahnsdorf area - shows the surroundings of the nursing home and the different areas that are accessible by patients. The two critical regions 'Highway' and 'Wood' present risks for certain patients. [Fudickar, Schnor, Felber et al., 2011]

GEOPRIV focuses on creating a simple and extensible policy language. This results in a language where many features stay undefined. The lack of functions and data types requires the implementation of non-standard extensions and the implicit definition of the *object* and *action* elements in the authorisation rules limits the usefulness of the Common Policy outside its narrowly defined application domain.

The advantage of GEOPRIVs differentiation between transformations and actions is not needed in our use-cases, because we have no need to reduce the granularity or precision of our location information prior to the data release.

A Common Policy condition may specify data access depending on data users, spheres (domain) or valid access times. We would need to extend the Common Policy in order to evaluate a policy depending on other attributes. XACML provides a richer set of attributes and functions that can immediately be used for the policy creation and evaluation and can be extended as easily as the Common Policy. So, we came to the conclusion, that the Common Policy does not offer any particular advantage over the use of XACML other than the less complex policy description. Since XACML policies will be automatically generated and parsed, and implementations for the creation, parsing and evaluating policies are already available, the decision has been made to use XACML for the description of the privacy policies in server-based scenarios.

6.1.2 KopAL Mobile Orientation System

A research team at Potsdam University developed the KopAL mobile orientation system [Fudickar and Schnor, 2009; Fudickar, Schnor, Felber et al., 2011] that tries to support elderly patients and patients suffering from mild forms of dementia. KopAL assists people in everyday activities and offers support for problems, like remembering appointments and keeping track within their surroundings. The KopAL team is operating at a nursing home in Stahnsdorf (Germany) that resides next to a forest and a highway, as shown in Figure 6.1.

Each patient is assisted by a mobile device, the *KopAL Assistant*, which acts as an appointment reminder and features an emergency button that can be activated by the patient to inform caretakers about critical situations. During the trial of the system it became clear that the system needs to

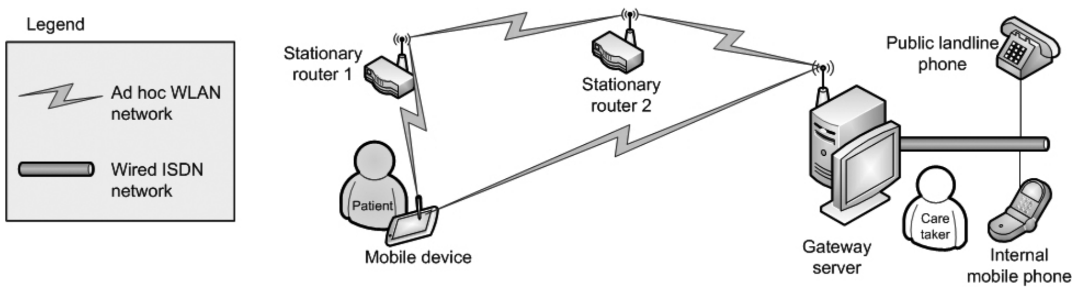


Figure 6.2: Network components of KopAL - the graphic visualises the different components and their interworking in the KopAL network architecture. Mobile devices use the OLSR routing protocol to form an Ad-hoc network and send position updates to the central server [Fudickar, Schnor, Felber et al., 2011].

provide a localisation function for cases where patients are in distress and can not identify their current location, or where patients suffering from dementia simply walk away from the nursing home. If such a dangerous situation develops, the KopAL Assistant should be able to inform the caretakers and provide helpful localisation data.

Network Architecture

The KopAL system uses a distributed architecture of connected nodes in a local network environment (see Figure 6.2) and consists of the following components:

Mobile devices (KopAL Assistants) are given to the patients and communicate with other devices via the Wi-Fi network.

A central server hosts the software components for the management system and the location database. The central server receives position updates from the KopAL Assistants.

WLAN routers provide the required network coverage within important regions of the building and surrounding area. In addition they act as localisation beacons. Since the position of the routers is fixed we use their visibility for location determination of the mobile device.

The network utilises the Optimized Link State Routing (OLSR) protocol [Clausen and Jacquet, 2003], which has been defined by the Mobile Ad-hoc Network (MANET) working group of the IETF, for the routing between nodes that have not direct visibility. If a KopAL Assistant device is outside the coverage area of the stationary Wi-Fi routers, it may still communicate in a hop-by-hop manner via other mobile devices.

The KopAL Privacy-Aware Localisation Service

Extending the KopAL system with the ability to localise the KopAL Assistance device has serious implications for the privacy of the patient. It becomes technically possible to constantly track the position of device and the patient who carries it. It is therefore important to implement privacy

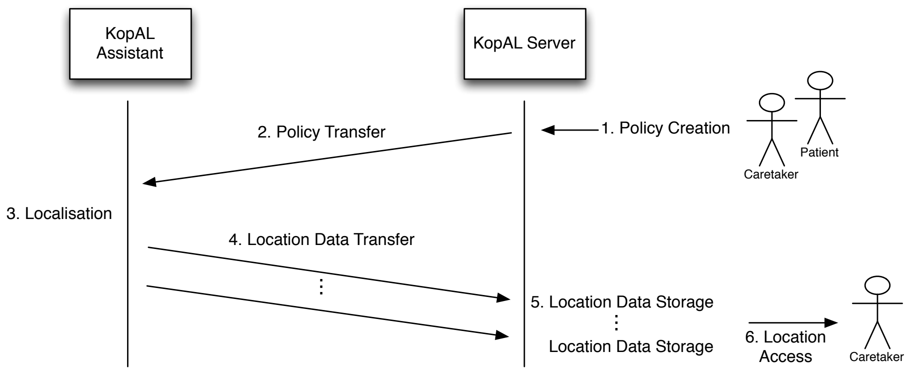


Figure 6.3: Sequence diagram of the KopAL privacy function - the diagram visualises the privacy creation process and the resulting flow of localisation data within KopAL. Privacy settings are configured via the KopAL Server on the Assistant device. Location data flows from the KopAL Assistant to the Server.

protection techniques that make sure that the localisation function respects the privacy of the patient. The patient or a responsible custodian should be able to decide the acceptable use policy for the private location data.

The use of the localisation service in KopAL starts with the policy creation process (cf. Step 1 in Figure 6.3). The responsible caretaker discusses the different privacy options with the patient or a responsible custodian before handing out the KopAL Assistant. The caretaker explains what data is collected for what purpose and the patient decides between the different privacy options.

This creates a privacy policy file, which is moved to the KopAL Assistant (Step 2). The Server does not store any copy of the policy. If the patient later wants to change the privacy policy, he or she can repeat this process to update the privacy settings of the device.

The KopAL Assistant evaluates the local policy file and sends a position update to the server only when the patient has allowed the processing of location data. The KopAL Assistant determines the current position of the patient and sends a location update to the KopAL location server (Step 3 & 4). No data is send when the patient has chosen to disallow localisation.

The system follows the *sticky policy paradigm* in the composition of the messages from the client to the server. With every location update we also send the currently valid privacy policy to the KopAL server. This has the advantage that the policy is originating from the device of the data owner. If the privacy policy changes, these changes would be communicated in a timely manner to the KopAL location server and no additional mechanisms are needed to manage privacy settings on the server.

KopAL provides a light-weight PKI that issues self-signed certificates to all devices. An RSA-Key pair is generated for each device of the system, including the server and KopAL Assistant. We use the public key of the KopAL localisation server to cryptographically protect our position updates. The mobile device encrypts the location and the attached policy with the public key of the KopAL Server.

Individual position updates are stored directly in the file-system of the location server (Step 5).

The stored location data object contains all the necessary authorisation information in the form of the attached policy rules and will not be accidentally processed without consulting the policy. A caretaker has to authenticate him or herself before requesting access to location data. The policy of the location data is checked for the necessary access permission (Step 6) before the certain location of a patient is shown to a caretaker via the KopAL web interface.

6.1.3 Findings and Discussion

KopAL is a server-based system, where distributed mobile clients send private localisation data to a central server, where it is processed. When we tried to adapt our enforcement solution from the Java PrivMon to a server-based scenario, we found that we could not use our custom classloader to automatically enforce different privacy policy settings within a server-based scenario.

We already mentioned in Section 5.5.3 that the Java Security Framework does not support dynamic updates of protection domains without reloading a class. Our approach so far has been to reload the class of the location service with a different Java `ProtectionDomain` to take advantage of the automatic privacy policy enforcement via the Java Security Manager. Reloading the application class is feasible for interactive use by a human operator, where application components have to be restarted relatively infrequently as different data sets are accessed. In server-based usage scenarios the server process is usually a long-running instance that needs to quickly handle data from different user devices with different privacy settings and reloading the application class has serious performance implications and might not even be possible.

Since the localisation component had to be integrated into the already running KopAL system we had to find a different enforcement solution to build our privacy-aware localisation server component. We decided to implement the necessary enforcement functions directly in the server code and build a separate solution that could be used to study other policy enforcement options for server-based systems.

6.2 Enforcing Location Privacy Policies through an AOP-based Reference Monitor

The experiences from the KopAL system showed us that privacy policies which follow the *Owner-Retained Access Control* (ORAC) principle can be implemented and enforced in server-based information systems. However, the requirements for their enforcement functions differ from client-side protection systems.

Modern server frameworks are typically build on service-oriented architectures and provide well-defined business functionality through the interworking of distinct service objects. Direct access to source code and retrofitting of access control checks to these service objects is not possible in most business scenarios. It is also frequently the case that a single service object is providing functionality for a number of requesting services and the object itself has no information other than the identity of the calling service to make an access decision.

In order to study the enforcement of data owner-defined privacy policies in such systems, we developed the hypothetical use-case scenario of a *theme-park location service*. This use-case allowed us to highlight problem areas and research solutions for the server-based privacy enforcement[3]. Our example implements the popular Model-View-Controller paradigm [Reenskaug, 1978] and uses a layered service architecture to encapsulate data access and storage.

The following section introduces our enforcement mechanism which is based on *Aspect-oriented Programming* (AOP) methods. We combine the flexibility of the AOP method interception with a generic policy evaluation component to build an AOP-based reference monitor that controls the interactions and data-flows between different services in the architecture. The policy evaluation component uses *data owner-defined* privacy policies that are directly attached to the relevant data objects as *sticky policies*. We will show how this approach allows the enforcement of individual privacy policies by a common service object.

6.2.1 Exemplary Use-Case: Theme Park

Before we explain our enforcement solution, we want to quickly introduce the properties of our use-case. We designed a hypothetical theme park location service that highlights privacy protection issues and allowed us to test the applicability of our reference monitor concept in tiered applications.

People typically visit theme parks as groups of families and friends. The park area tends to be geographically dispersed, crowded and it is easy to loose contact to members of the group. If group members could see each others locations, they could visit different attractions individually and use the system to meet each other again. Such a system would also be beneficial for the theme park operator. The operator could advertise new or under-utilised attractions and get live updates of crowd movement.

Localisation

Localisation is achieved by handing out electronic guides to visitors entering the park or installing a dedicated app on visitor smart-phones. While the group is visiting the park, each mobile device is sending its location to a central server. The server receives, stores and processes these location updates and distributes the processed location information to clients of the same group. The location server is able to provide additional location-based information, like the distance to the next restaurant or a nearby event.

Privacy Settings

To make the service privacy friendly, the visitor should have the ability to adjust privacy settings. He or she can decide for what particular purpose location data might be used. For example, the visitor can decide whether the theme park operator is allowed to analyse the location data to improve the park quality, etc. The individual privacy preferences are constructed as XACML policies. Each location update that is send to the central server also carries the current policy information.

[3]This section is based on joint work with Sven Schindler [Scheffler, Schindler, and Schnor, 2012]

6.2.2 Service Architecture

The theme park server is designed as a multi-layered JavaEE application to find out whether our concept of a privacy reference monitor is compatible with the design of modern business applications. Figure 6.4 shows a simplified server application architecture. It illustrates the different components that will be used to provide the theme park location service.

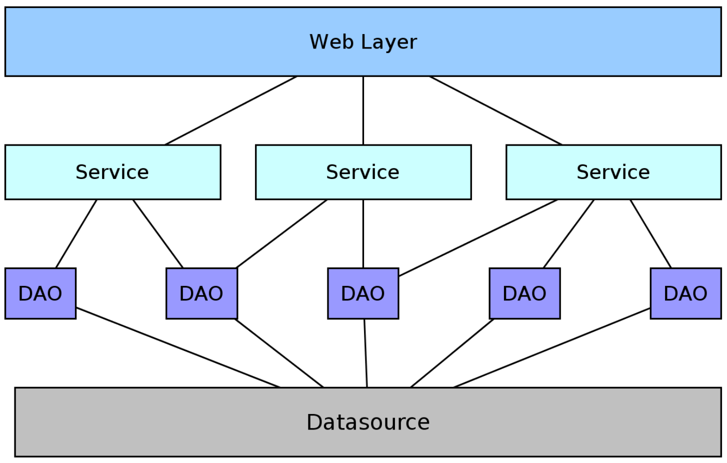


Figure 6.4: Layered application server architecture - modern business applications employ a layered service architecture, where data access is encapsulated through so called Data Access Objects (DAOs).

Theme park employees should be able to compose new visitor groups and analyse visited locations using a web interface on the web layer. Each activity, like creating a new visitor entry in the system, triggers a service in the service layer. A service contains the business logic of the application. It does not operate directly on the datasource but uses one or more *data access objects* (DAOs), so that the implementation details of the datasource are hidden from the service layer.

A visitor's mobile device is continuously sending its current position together with the sticky-policy to the central server. The server distributes the received location data to the visitor's group members and stores a copy in the data source. The responsible service for the distribution of location data, the *LocationBroadcastingService*, is periodically accessing the stored locations in order to find new position updates. The theme park operator may analyse the stored locations by using the web interface to trigger the *LocationAnalysisService*.

A visitor is able to express his or her privacy preferences and can choose to prohibit access to the location data for certain purposes. Our enforcement architecture needs to restrict data access by services depending on the evaluation result of the individual privacy policy that is attached to the data. We could not use the existing Java PrivMon reference monitor implementation within the theme park server application, because every visitor regularly produces new position updates that need to have their individual privacy policies enforced. The Java PrivMon implementation would require the reloading of the DAO classes when location data for different customers have to be processed, because the Java Security Framework is not able to dynamically update the

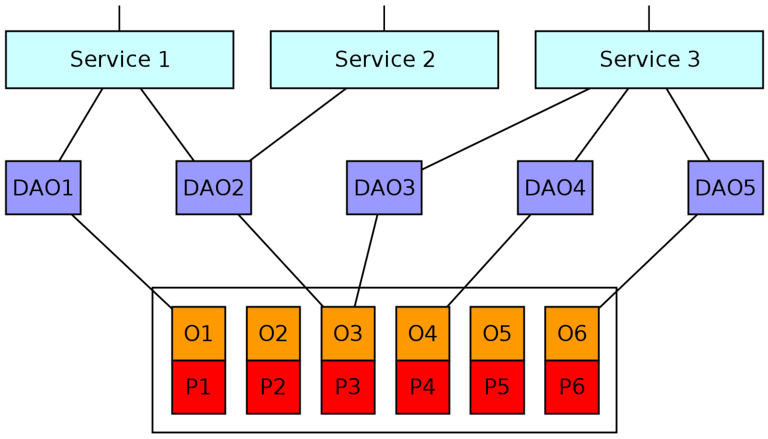


Figure 6.5: Access permissions in the DAO layer - depend on the identity of the calling service and not the identity of the DAO. We use Aspect-oriented Programming (AOP) and Java Reflections to decide if data access by the service is allowed or not.

`ProtectionDomain` of a class.

A further issue lies in the determination of the subject of an access request. Figure 6.5 visualises this problem. The depicted architecture contains two services *Service*1 and *Service*2 trying to access object *O*3 that has an attached policy *P*3. Assume that *P*3 contains a rule that allows *Service*1 to access *O*3 but does not allow *Service*2 to access the resource. For example, a theme park visitor may decide to deny the operator to analyse his location, but still wants to receive service notifications. Ordinary service oriented architectures can not enforce this distinction at the level of the data access objects. DAOs provide no data access control to services and both services are using the same DAO to access the location data. When we are evaluating a privacy policy, we also have to take the calling service into account.

We decided to tackle this problem by changing our existing reference monitor implementation and replace the Java Security Framework with a custom security layer based on aspect-oriented programming (AOP) and the use of the Java Reflection API. The next section provides a short introduction to aspect-oriented programming and explains how it can be used to develop a security and privacy layer.

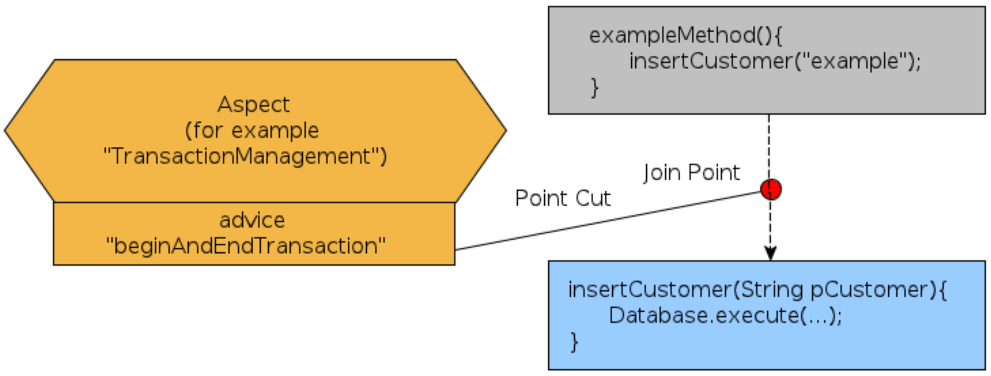
6.2.3 Aspect-oriented Programming

Aspect-oriented programming (AOP) is a programming paradigm that can be used to improve source code modularity. Even though the object-oriented programming paradigm provides plenty of features like inheritance and encapsulation to increase the modularity of a program, there are still certain cases where it is not possible to avoid code repetition.

Consider, for example, an object of a business application that adds or removes customers from a database. The class would have two methods *deleteCustomer* and *insertCustomer* which are used to delete or add a customer. Each operation should be executed within a database transaction to avoid an inconsistent database state. Hence it is necessary to add at least one line of code to open and close a transaction before each of the mentioned operations.

AOP resolves that issue by encapsulating repeating source code segments in so-called aspects. An aspect is a modularisation of a concern [Spring Framework, 2011], which can be bound to different points, so called join points, in the source code. Figure 6.6 shows an overview over the most important terms and definitions of AOP using the afore mentioned transaction handling example.

Basically, an aspect is a set of functional units called advices. The aspect *TransactionManagement* contains one advice *beginAndEndTransaction* to define a transaction. The advice needs to be executed before the *insertCustomer* method gets called as well as when the method is finished in order to properly open and close a database transaction. Therefore the advice is bound to a point in the source code between the two methods. In this example, the advice can be considered as a wrapper around the *insertCustomer* method.



```
exampleMethod(){
    insertCustomer("example");
}
```

```
insertCustomer(String pCustomer){
    Database.execute(...);
}
```

Figure 6.6: AOP-Components overview - the diagram shows how an aspect that starts and ends a database transaction is weaved into the program flow of the transaction management example.

A set of join points is also called a point cut. The binding of an aspect is called weaving and can be done at compile- or loadtime depending on the framework and programming language. There exist different AOP frameworks for different languages that software developers can use to create and bind their aspects. Many AOP frameworks are using so-called interceptors to model advices [Spring Framework, 2011]. An interceptor is an object that wraps around a method call.

The ability of AOP to intercept methods of existing applications led us to the idea of using it to intercept and monitor accesses of classes to protected resources. The following sections show how AOP can be used in Java and presents a privacy architecture that uses AOP to monitor and control policy protected resources.

6.2.4 A Reference Monitor based on AOP

We are using the capabilities of the AOP concept to monitor and control access to protected resources. This is done by implementing a privacy protection layer that enforces the data owner-defined policies for privacy protected location data.

Concept

AOP provides a very flexible way to interact with existing methods in our code-base. Additional code can be executed before and after a method invocation. We can access and manipulate the parameters of the target method if needed and it is also possible to stop the target method from executing at all. So, instead of using the Java Security Framework for the enforcement of privacy policies, we based our server-side policy enforcement component on AOP. A custom privacy framework implementation, which is encapsulated in aspects, is responsible for the enforcement of access control decisions.

The access decisions are provided by our existing XACML decision component that is able to evaluate the sticky XACML policies. Figure 6.7 depicts the AOP-based reference monitor concept. It shows an object trying to access a resource that has an attached sticky-policy. The resource may, for example, be a file in the file system or a data item in a database.

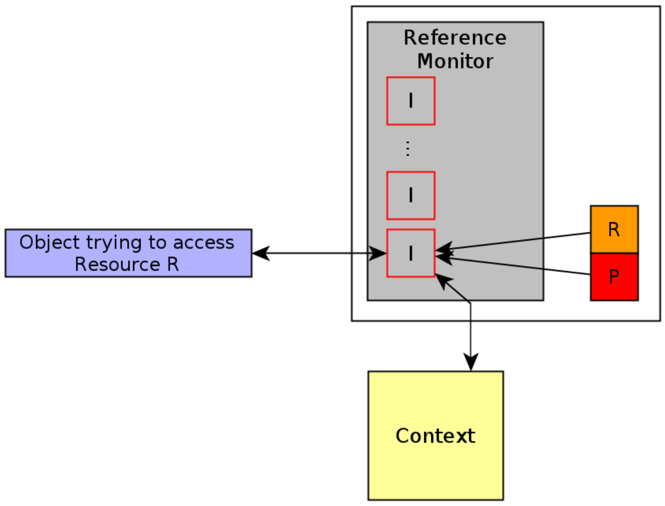


Figure 6.7: An AOP-based reference monitor - task specific advices *I* are used to monitor access to resource *R* and allow or deny access depending on the evaluation of policy *P*. Available context information can also be considered in the policy evaluation process.

The reference monitor uses a set of task-specific advices to monitor the access to the resource *R* and interrupt it, if necessary. Before granting access to a resource *R*, an advice *I* evaluates the sticky policy *P*. If the evaluation is not successful, the advice prohibits the access to the resource by cancelling the method invocation that is trying to access the resource. The monitored system is able to provide additional information for the policy evaluation to the reference monitor by using a context object.

Furthermore, it is possible to include the calling services into the policy evaluation. We are using the Java Reflection API to evaluate the call stack, so that we can determine all accessing objects. The determined Java objects are added as subject when evaluating the XACML policy.

```
<?xml version="1.0" encoding="UTF-8"?>
<aop xmlns="urn:jboss:aop-beans:1.0">
   <interceptor class="org.mobilePositioningSystem.aop.security.LocationDAOInterceptor"
       />

   <bind pointcut="execution(* org.mobilePositioningSystem.daos.location.impl.
       LocationDAOImpl->loadLocationFile(..))">
       <interceptor-ref name="org.mobilePositioningSystem.aop.security.
           LocationDAOInterceptor"/>
   </bind>

</aop>
```

Listing 6.1: Definition of an interceptor and point cut in JBoss AOP for our theme park example implementation

Implementation

AOP frameworks are available for various programming languages. This section explains one possible way of weaving aspects into the source code by using the JBoss AOP framework for Java [JBoss Inc., 2012]. JBoss AOP can be used to define and bind aspects during compiletime, loadtime and runtime. Our example implementation uses the AOP framework to load a security layer at loadtime.

JBoss AOP provides the `org.jboss.aop.advice.Interceptor` interface which enables programmers to develop their own advices. The interface contains a signature for an *invoke* method which is called when a defined join point is reached. An *Invocation* object is passed to the method that can be used to invoke the target method. Therefore, an object implementing the interface is able to execute code before or after it uses the *Invocation* object to invoke the actual target method and acts like a wrapper around the method.

There are several ways to define the point cuts for the implemented interceptors. One way is to create an external XML file which contains the associations between interceptors and point cuts. Consider the example file in Listing 6.1. The file contains one point cut that includes all `loadLocationFile`-calls of our *locationDAO* and one interceptor that is bound to that point cut. When running Java with the JBoss AOP framework and the example definition above, each call to a file constructor will be intercepted. In this way, the interceptor can monitor and interrupt all file accesses by the application. Because the weaving is done at loadtime, there is no need to recompile the underlying application. Hence, with the example interceptor above, it is even possible to monitor file accesses of foreign and proprietary Java applications.

Java provides several ways that enable AOP frameworks to interact with the JVM and to weave advice code into the classes at loadtime. JBoss AOP is using the `java.lang.instrument` package for loadtime weaving. The package is an instrumentation API for the Java programming language and was introduced in Java 5. It allows the development of so-called Java agents that are able to register byte-code transformers to manipulate the behaviour of a class. An agent gets passed to the JVM on startup.

Our theme park example implementation stores all locations for one user in a single location file. The location data is stored together with the sticky-policy. As soon as a location DAO tries to access a location, our reference monitor implementation intercepts the access attempt and

```
<Rule RuleId="lostDeviceRule" Effect="Permit">
 <Target>
  <Subjects>
   <Subject>
    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:x500Name-match">
     <AttributeValue
      DataType="urn:oasis:names:tc:xacml:1.0:data-type:x500Name">
       CN=object_org.mobilePositioningSystem.services.location.analysisImpl.
            LocationAnalysisServiceImpl
     </AttributeValue>
     <SubjectAttributeDesignator
      AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
      DataType="urn:oasis:names:tc:xacml:1.0:data-type:x500Name"/>
    </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>
</Rule>
```

Listing 6.2: This XACML policy excerpt of our 'theme park location system' permits our location analysis service to analyse the corresponding location.

evaluates the corresponding policy. A policy, like the XACML policy in Listing 6.2, may define the services that are allowed to access the location. The reference monitor implementation detects the service which uses the DAO to access the location by using the Java Reflection API and includes it into the evaluation progress.

Use of prioritised Sub-Policies for Policy Management

Within the use case of the theme park we can again use our *Privacy Policy Precedence Relation (P3R)* from Section 3.5 to define different policy parts that aid the policy management task of the data owner and offer protection against accidental mistakes in the policy management process.

We propose to use the following sub-policies:

- A *default policy* is attached to any data item as soon as it is created and can not be altered by the data owner. The default policy will be a 'closed' policy that denies every data access that is not explicitly allowed.
  For example, a default policy for a very privacy conscious person defines that no subject except the visitor itself is allowed to access the location data.

The theme park visitor should be able to choose a suitable default policy for his or her privacy needs from a number of templates. A less privacy conscious person might also allow location data access to the theme park operator for the purpose of marketing or evaluation of service quality.

- The *owner policy* rules can be modified arbitrarily to reflect any privacy decision by the data owner. This policy part will usually define exceptions to the default policy.
  For example, a visitor should be able to allow other persons to access the personal location, when for example, a larger troop of visiting people want to define or re-define ad-hoc groups. Another example is the override of the default policy, where a visitor may decide to change his or her mind in respect to the initial protection choice of the default policy.
  If the owner policy is deleted, the default policy will again provide basic privacy protection. It is therefore not possible to accidentally lose complete privacy protection through the deletion of the owner policy.

- The *safety policy* is not under the administrative authority of the data owner and describes specialised authorisations, where it must be guaranteed that these restrictions or permissions persist. For example, a theme park employee in our use case must always have the ability to locate a device that has been reported lost. An access rule defined by a visitor should not be able to remove this permission.
  The safety policy can also be used to define persistent visitor grouping, e.g. between parents and their children, that must not be deleted because this would distress or hurt the involved people.

Protection of the Location Update

The location update and the included policy form a *Protected Data Object* (cf. Section 4.3) that has to be cryptographically protected during data transit and storage. Our system provides an individual RSA public key pair for each device in the system [Rivest, Shamir, and Adleman, 1978]. Client and server will establish a unique symmetric session key K_{Sess} that stays valid for the duration of the theme park visit. Client and Server will each have a pre-established key pair K_{Cpriv}, K_{Cpub} and K_{Spriv}, K_{Spub} which is used for encryption and message signing.

The session key will be generated and transferred to the device just before the theme park guide is handed out to the park visitor. The server will also send a sequence number SEQ to the client that is used to detect and fend off replay attacks on the key exchange procedure, where an attacker has gained knowledge of the current session key and replays the initial key exchange on further theme park visitors that use the same device:

$$S \longrightarrow C : \{Sign_{K_{Spriv}}\{K_{Sess}, SEQ\}\}_{K_{Cpub}}$$

This session key is used to symmetrically encrypt the location updates as they are send by the client using the AES encryption algorithm [FIPS PUB 197, 2001]:

$$C_1 \longrightarrow S : \{Location_{C1}\}_{K_{Sess\_C1}}$$

The server relays this location data to other clients in the same group, using their individual session keys.

$S \longrightarrow C_2 : \{Location_{C1}\}_{K_{Sess\_C2}}$
$S \longrightarrow C_3 : \{Location_{C1}\}_{K_{Sess\_C3}}$

Location data that is stored on the server by the Location DAO will use a secret encryption key that is only known to this particular service.

6.2.5 Performance Tests

The AOP privacy layer performs a policy evaluation for each access to a protected resource. Since this requires the acquisition, parsing and evaluation of an XML-encoded XACML policy, we needed to quantify the performance impact of the privacy protection layer on a live system. A number of performance measurements were conducted using the afore mentioned *theme park location system*. The system collects location data from theme park visitors in simple text files. The location system contains a statistics function that analyses the location file of a visitor and determines how much time the visitor spent at each attraction.

Each location entry in the file also contains the XACML privacy policy, which must be evaluated by the AOP security layer. In order to measure the performance of our privacy protection mechanism, we compared the time this analysis took with and without an active security layer. Our test system had 4096 MB of memory and an Intel Core 2 Quad Q9550 processor with 4x2.83GHz and was running Ubuntu 8.04 on a SATA hard disk with 5400 rpm.

A visitors mobile device sends its computed location three times a minute to our theme park location server. Hence, a four hour visit results in 720 locations which need to be stored and analysed. We measured the time needed to analyse up to 50 location files, each containing 720 locations. As shown in Figure 6.8, the location analysis of 50 location files takes about 380 seconds to complete if our privacy layer is enabled. An analysis without the access control needs about half the time.

6.3 Summary and Discussion

With this part of our work we have shown that it is possible to implement automatic privacy enforcement mechanisms based on sticky policies in modern business applications, which are build on tiered architectures, using an AOP-framework and XACML-polices.

One of the major advantages of the AOP access control layer is its transparency to the existing application layer. Applications that are developed in a language which supports loadtime weaving, such as Java, do not even have to be recompiled, the AOP privacy enforcement layer can simply be added on application start-up. The XACML processing in the reference monitor allows us to evaluate dynamic policies without requiring changes to the application or the enforcement layer. Since we use the Java Reflection API to base our access decision on the identity of the requesting service and can therefore keep the number of advices to a minimum: there is no need to define different advices for different purposes of data access. Our exemplary implementation is able to work with a single interceptor for three defined access purposes, which are:

- location update to group members
- generation of statistics for the theme park management

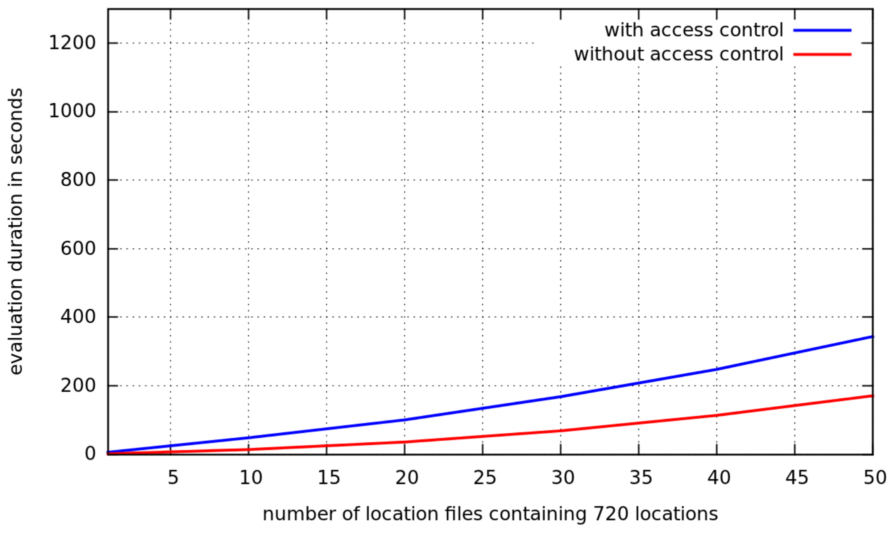


Figure 6.8: Measurement results - Evaluation duration for up to 50 location files each containing 720 locations.

- localisation of misplaced devices

An important criteria for the applicability of any proposed mechanism is its performance under real-world conditions. We have conducted several tests based on our use case scenario. We found that especially the XACML parsing takes a lot of time and can be improved by using appropriate caching mechanisms, since both, resources and policies, are encoded in XML. Overall, our test results have shown that the performance impact of our privacy layer is acceptable and can be controlled, so that this is no hurdle for potential deployment.

Our privacy framework is not restricted to a certain policy or programming language. Every object oriented language that supports AOP and reflections can be used to implement this security framework. Similarly, other policy languages than XACML could be used for the expression of privacy policies as long as they are expressive enough and can be evaluated consistently.

The use of the sticky policy concept, which keeps policy and private data together, is a key element of our privacy framework. We think that it could also be used in the future to ease distributed policy enforcement, where the computationally intensive location analysis is outsourced to external service provider infrastructures and compute-clusters without risking the users privacy.

7 Conclusion and Future Work

Martin R. Stytz demanded in an article for the *IEEE Security and Privacy Magazine* 2005 that “Individual ownership of PPI [Personally Private Information] should be permanent, and owners should receive the same rights over their personal data as they would over any other property”. He made suggestions on possible changes that would be required to achieve better privacy protection for personal data:

> To let owners control access to their personal information, we must change the fundamental way in which information transfer occurs. Currently, organizations can transfer an individual’s PPI, such as bank-account balances, bill-payment history, and even income, to another company without the individual’s knowledge, much less his or her permission. [Stytz, 2005]

It this thesis we have shown that giving control to data owners and enabling privacy-aware data exchanges are indeed possible. We analysed a possible solution for the specification of priorities in data owner-defined privacy policies and constructed a *Privacy Enforcement Framework* that takes the policy administration and client-side data protection into consideration. We conceived, build and analysed several prototypes to confirm our initial hypothesis that access to personal private data can be controlled even when data is shared with a data user.

The presented work also shows the broad scope of our undertaking. A complete, deployable solution extends well beyond the realm of computer science and must answer questions from the economic, legal and psychological domains to become a viable privacy protection tool in the hand of ordinary computer users. Several of these issues go far beyond the scope of this thesis and need to be taken up by further research projects.

The next sections will summarise the findings of this thesis. During the work on this thesis several topics arose that constitute interesting questions for further research but were clearly out of scope for this body of work. Several of these topics will be presented here as further research directions.

7.1 Research Contribution

This thesis makes the following contributions to the field of research into privacy enhancing technologies:

Motivation and Requirements: We identified weaknesses with the currently deployed privacy protection approaches when personal private data needs to be exchanged between data owner and data user. Most existing privacy protection approaches use a privacy policy that is set by the data user and offers no enforcement guaranties, because it is only loosely

connected to the collected data items and the deployed enforcement mechanisms.
We motivate our solution through the description of three use cases that highlight privacy issues in data exchanges that offer access to Electronic Health Records (EHRs) as well as privacy issues in location-based services, where location data is used to support social interactions and assisted living.

Privacy Protection Concepts and Principles: We identified key principles and concepts for a privacy enforcement infrastructure under the control of the data owner through the analysis of existing *Privacy Enhancing Technologies*, *Access Control Frameworks* and *Privacy Policy Languages*. We also contributed a comparison between the GEOPRIV Common Policy and XACML with respect to their applicability for location-based services.
There exists a substantial body of related work for this thesis, however, many previous projects focused on organisational privacy practices and are not directly applicable to data owner-defined policy description and enforcement mechanisms.

Owner-Retained Access Control (ORAC): We provided an in-depth comparison between privacy policies that were defined by data users and policies that have been defined by the data owner. Several areas were found, where ORAC policies offer better support for privacy preferences of the data owner.
However, giving administrative power to the data owner burdens the individual with the non-trivial task of policy management. The user-friendly maintenance of ORAC policies is an important prerequisite for the reliable operation of privacy protection within our framework and needs to be solved for a practical implementation.
We created a novel approach to policy management, which is based on prioritisation of specific policy parts. We propose a *Privacy Policy Precedence Relation* (P3R) that specifies a ordering relationship between sub-policies through the assignment of explicit priority values to these policy parts. We implemented the Privacy Policy Precedence Relation as a conflict resolution mechanism for the XACML language. The use of sub-policies makes it possible to define *default protection profiles* and *safety rules* that augment *owner-defined policies* and aid and protect the data owner in its policy management duties. We further use the *sticky policy* model to support secure policy distribution and storage for Owner-Retained Access Control policies of individual data items.

Java PrivMon: It was our plan to use standard access control enforcement mechanisms, such as the Java Security Framework, to implement privacy policy enforcement. The Java Security Framework with its standard policy and sandboxing mechanisms looked like an ideal enforcement platform, because it is widely deployed and offers robust access control.
During our work we found out that the current Java policy enforcement offers little support for the enforcement of dynamic access decisions that change as different data items with different privacy policies will be accessed. The Java Security Framework operates under the assumption that the computer user has complete control over any program behaviour and it is the program-code that is assigned different levels of trust.
We had to develop our own class-loading behaviour to support different access control policies for instances of the same code-base. Our implementation allows the start of untrusted Java programs under the control of the Java Security Framework. The relevant access permissions of the application are derived at runtime from the policy of the data object that is

being accessed. We tested our framework with a prototypical privacy protection system for personal health records.

Privacy Protection for Server-based Information Systems With the growing numbers of mobile devices, more and more usage scenarios nowadays use server-based or server-assisted data processing and it is no longer the human data user that must be restricted in its ability to access private data. We therefore extended our client-side reference monitor to a server-based solution that uses Aspect-oriented Programming (AOP) for the enforcement of data owner-defined privacy policies. We still use the sticky policy approach for the communication and storage of the privacy policy element.

The XACML policy language was used for the definition of data-use policies by the data owner. The developed privacy enforcement framework provides support for fine grained privacy policies for the expression of privacy preferences.

Private personal data is translated into a suitable XML record format and stored together with the corresponding XACML policy as a single XML data object. Data access policies are defined and bound to the data at creation time and revised later as access decisions need to be granted or revoked. Policy management is aided through the separation of generic default-policies from user-editable specific policies. The private data is referenced from the XACML policy via XPath.

7.2 Discussion

Our privacy enforcement framework operates under the assumption that private data is released to a non-malicious data-user that has a strong interest to cooperate with the data owner. This would be the case for the typical practitioner that treats different patient over the course of the day and needs to protect the privacy of the electronic data that is generated.

Although this assumption sounds as if it might limit the usefulness of our framework, this is not the case. When we compare our approach of *data owner-defined policies* to the more common *data user-defined policies*, we notice that both approaches require trust in the data user to deploy the necessary policy enforcement. When we look at private interactions between persons, we find again that a non-malicious communication partner is assumed. People reveal very personal information to close friends and trust them to respect the privateness of this information. If a person does have no trust in the other person, it usually tries to reveal less or nothing about him- or herself.

Our privacy enforcement framework follows the same pattern. We want to enable the communication of personal private data to the data user, however, the data owner must be able to control to what extend private data is visible and limit the actions of the data user. Other projects use the same assumption about the data user. Vanish [Geambasu, Kohno, Levy et al., 2009], for example, assumes that a data user never stores an unencrypted copy of the sensitive private data and implements no mechanism that prohibits such an attempt.

Our privacy enforcement framework automatically enforces the privacy preferences of the data owner and allows data access that is conforming to the policy. It is our intention to make conforming data access as easy as possible in order to raise the level of acceptance of the framework. Nonconforming access, however, such as unintended data release due to carelessness or unintended copying of data is no longer possible.

However, no protection mechanism can guarantee perfect enforcement. If we must assume a malicious data user, there are several attack vectors that could lead to a successful compromise of our protection scheme. We therefore recommend that in data release cases, where the data owner has no trust in the data user he or she should refrain from the data release. Data that has not been released can not be compromised.

We can, however, identify certain areas that need to be considered. These include:

Attacks on the Java Virtual Machine: The implementation of the reference monitor based on a Java Virtual Machine has different potential attack vectors. Encryption keys can be extracted through a memory dump from the underlying operating system or the virtual machine implementation itself could be manipulated so that necessary access control checks are bypassed. Maaser and Langendörfer [2009] propose to check the hash values of the sandbox environment and the operating system before private data is processed on the system. However, a full solution would require a secure bootstrap process assisted by a Trusted Platform Module [Trusted Computing Group, 2007] in order to have a reliable reference that can not be altered by the attacker.

Social engineering attacks: In most cases of data processing final or intermediate results will be made known to a human data user. The main way of presenting data is done through the rendering of text and images on a computer display. Normal operating systems usually allow the copying of displayed text or images between applications. The Java Security Framework can control access to the system clipboard and it might also be possible to suppress the ability to take screen-shots. Rendering text output in an image format such as JPEG or PNG could further complicate the subsequent electronic processing. However, a determined attacker could still photograph the computer screen or take hand-written notes of the displayed data.

Timing Attacks: Different time interpretations between the data owner and data users systems (different timezones, daylight saving time, etc.) could be used to gain access to data that is protected by rules that enforce time-based access restrictions. Synchronisation issues between different devices might result in situations, where it is difficult to determine the exact meaning of a rule with time related permissions. Possible attacks might arise if an attacker can manipulate time and date information on the reference monitor system. Therefore, the enforcement systems needs to have access to a trusted time-base.

Privacy Considerations: In our PrivMon prototype implementation we allow the browsing of meta-data by every practitioner, even if the entry itself is not accessible, certain information about an event can be gathered and conclusions can be made.
This is a general problem for many privacy critical systems. Access to meta-data already allows the data user to draw certain conclusions. If, for example, a certain oncologist has made a number of entries in the PHR, it is safe to assume that the patient has been diagnosed with cancer.
A potential solution for our system would be to generate a restricted resource browser-view that only includes entries that can are attributed to a certain practitioner. Such a solution has better privacy properties than our current implementation, because no conclusion can be made about other events. We could even differentiate this view according to personal trust

values. So could, for example, the family doctor be allowed to see all entries in the health record and advice the patient on comprehensive medical issues.

Conclusion

At the end of this thesis we try to answer a question that arises from the motivation of our work: “Could the application of our privacy enforcement framework and the consequent usage of Owner-Retained Access Control policies have avoided the data breaches at the University of Texas and the Jet Blue airline passenger data?”

Our privacy enforcement framework can eliminate privacy breaches that are the result of careless data handling practices or that originate from cases where the data usage policy is ignored or simply unknown by the data user. If we assume that the mentioned cases fall into this category of data breaches and were not maliciously conducted, we can answer this question with "Yes, these data breaches could have been avoided if our privacy protection approach would have been used".

We have given strong evidence in this thesis that storing private data together with the applicable policy in a protected object improves privacy-aware data handling. The protected object can still be copied, however, the policy and protection settings are copied as well. Access to the encrypted data is only possible through a server- or client-based privacy monitor that automatically enforces data flow protection by restricting the data user to the execution of authorised actions.

Our system is not designed to offer protection against a capable attacker that has full access to the machines where the private data is used. It may, however, make the life of such an attacker more difficult, because the simple copying of personal private data no longer automatically leads to a data breach.

7.3 Future Work

This thesis makes important contributions to the field of privacy protection techniques and extends currently available mechanisms. Our privacy enforcement framework incorporates elements from very many areas of privacy and security research, such as access control, safe execution of code and others. We think that it is only natural that this body of work leads to new questions and opens up possibilities for further research. In the following, we therefore want to describe several research areas that are directly relevant to or derived from our work.

7.3.1 Delegation of Authority

Delegation can be described as the assignment of authority from one entity to another. The delegatee acquires the ability to carry out functions and make decisions in accordance with the granted authority including the right to further delegate this authority.

Delegation of authority is a powerful concept for the management of permissions. However, in the context of privacy protection it can potentially violate the trust relationship between data owner and data user. Assume that subject *Owner* wanted to prohibit subject $User_b$ from accessing a certain document. If *Owner* grants $User_a$ the authority to further delegate rights, $User_a$ might then be able to grant $User_b$ access to the protected document. A potential solution would be for the *Owner* to explicitly specify the set of subjects U_{deny} that should not have access to the document.

In the case of access restrictions on private documents the set of permitted subjects U_{permit} is usually much smaller than the set of denied subjects:

$$\mid U_{permit} \mid \ll \mid U_{deny} \mid$$

This makes the enumeration of subjects unpractical and if we wanted to implement a *closed policy* it is not even possible to enumerate the set of denied subjects. In our privacy protection framework we therefore directly authorise the trusted subjects and do not implement delegation support.

However, Navarro, Firozabadi, Rissanen et al. [2003] have shown that delegation support is possible within XACML and it should be an interesting task to further examine the problem of enabling restricted delegation within a privacy enforcement system.

7.3.2 Revocation of Access Rights

Personal trust levels change over time and we foresee the need to adjust existing policies to reflect the changing personal trust between data owner and data user. It is therefore important to support the ability to revoke currently existing permissions.

Revoking access to electronic data is inherently difficult, since data can be easily copied, processed, forwarded and stored. Unknown data copies might exist in forms of computer backups, which can be used even if access to the original data source is no longer possible.

The main protection property of our privacy enforcement scheme demands that data should be stored as a protected object that includes the corresponding privacy policy. Nevertheless, it will be possible to make copies of the protected data object itself. Revoking access that has already been granted would require to trace all possible copies of the protected data object and change the attached policy.

A possible solution could be the maintenance of revocation lists for protected data objects on a central repository. However, we would lose some of the benefits of distributed data access, because a centralised revocation infrastructure would be required that would need to be maintained. If we do not mandate revocation checks, an attacker could simply prevent access to the revocation service in order to retain its original authorisations.

For our use case we assume that policy changes that lead to rights revocation are rather rare. Once a trust relationship is formed between a patient and a practitioner it remains stable over a long period of time and authorisations need not be revoked. In cases, where this trust relationship has not been fully formed and policy changes might be likely, we propose a different approach for the granting of authorisations: Data users that are not fully trusted should be only granted temporary authorisations. This can be implemented directly at the policy level. The condition element of XACML data access rules can facilitate time-based functions for the expression of temporary restrictions.

7.3.3 Knowledge Representation in Privacy Policies

During the course of this thesis we have treated policies as instruments for privacy protection that are used by the data owner to communicate privacy preferences and that allow the reference monitor to derive access decisions for sensitive data.

During our research we found that data owner-defined privacy policies have an additional purpose. They also document the current privacy preferences and expectations of the data owner. This documentation aspect is important, because privacy preferences might change over time, and different circumstances might require the adaptation of an existing privacy policy. The data owner needs a frame of reference upon which he or she can re-evaluate their initial choice and former believes.

Privacy policies in organisational contexts are commonly expressed and documented in the form of high-level, informal textual expression, similar to the privacy policy found in Section 1 of this thesis:

> The University of Texas at Austin (U. T. Austin) is committed to ensuring the privacy and accuracy of your confidential information.
>
> ...
>
> U. T. Austin also complies with the Family Educational Rights and Privacy Act (FERPA), which prohibits the release of education records without student permission....
>
> [University of Texas, 2008]

High-level privacy policies are mainly written down as human readable text (based on a suitable encoding such as HTML) and can be directly posted on a website or referenced through the `discuri` element of a P3P policy. They are the basis for the construction of *low-level*, enforceable policy representations that consists of data structures that facilitate fast policy evaluation. They also often have the mentioned documentation purpose, in that they capture and explain the original intentions of the privacy policy, potential restrictions and make references to applicable regulations and legal frameworks.

This clear differentiation between high-level and low-level policies is typical for *data user-defined policies*, where a dedicated privacy administrator maintains these policy representations. In the case of *data owner-defined policies* we normally find no documented high-level policy representation that could be used to reference the intended protection properties of the enforceable low-level policy. The data-owner typically has some intuitive personal preferences that will be expressed through the rules of the current privacy policy, but no formal documentation. We think that it is therefore important to find a policy representation that supports policy enforcement and that is expressive enough to also enable the data-owner to reason over the enforcement properties of the privacy policy.

High-level policies are not directly enforceable by the underlying policy enforcement mechanisms due to their informal specification and rather broad meaning. They must be refined into low-level expressions that correspond to the granularity of objects, subjects and actions known by the authorisation scheme and implemented in the enforcement system. Figure 7.1 shows this process, which closely involves the Policy Administrator, who must create the necessary low-level policy and has complete control over this process.

Low-level policy representations, such as Access Control Lists (ACL), are optimised for rapid policy evaluation and direct enforcement. They are derived from the corresponding high-level

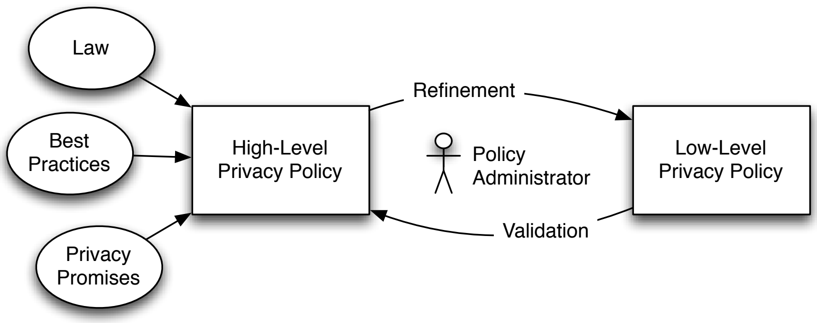


Figure 7.1: Policy Refinement and Validation Process - Privacy policies are usually documented as high-level, human readable policies. The Policy Administrator is responsible for a suitable transformation into an enforceable low-level policy that can later be validated against the high-level representation.

policy but the individual rules typically contain no link to the high-level policy part that was responsible for their creation. The specific knowledge about the high-level policy goals is eliminated from the resulting low-level policy during the policy refinement process – rules in the low-level policy don't carry information from which part of the high-level policy they were derived. It is usually not necessary to preserve this information in the low-level policy representation, because it is not needed for the enforcement and still available in the high-level policy.

However, for the reliable lifetime management of policies the connection between high-level and low-level policies is important. It must be made certain that a low-level policy truly corresponds to its high-level counterpart and changes in the policy representation do not violate any higher level goal. Similarly, it is necessary to adapt the low-level policy if some accepted protection goals change. The high-level policy is documented separately from the low-level policy and must be able to be referenced so that their correspondence can be ascertained. Organisational procedures specify the role of a policy administrator that is responsible for the policy refinement process and the maintenance of both policies.

Knowledge Representation in Owner-Generated Privacy Policies

The policy management process for owner-generated policies differs from the preceding description insofar, as there typically exists no separately documented high-level policy. Therefore, the process of generating and maintaining an enforceable policy representation can not be aided by it.

We assume that the typical data owner has only an intuitive knowledge about potential privacy threats and a naïve concept of trust that builds the foundation of his or her individual privacy requirements. The low-level policy representation itself must be expressive enough to capture and document high-level policy goals for the data owner and the data owner needs tool-support to formally specify his or her privacy requirements in the policy. Figure 7.2 visualises the policy maintenance process for ORAC policies where only one policy representation exists.

A policy validation tool should be developed that lets the data owner test assumptions about the protection properties of the policy and check for weaknesses and errors in the policy. We

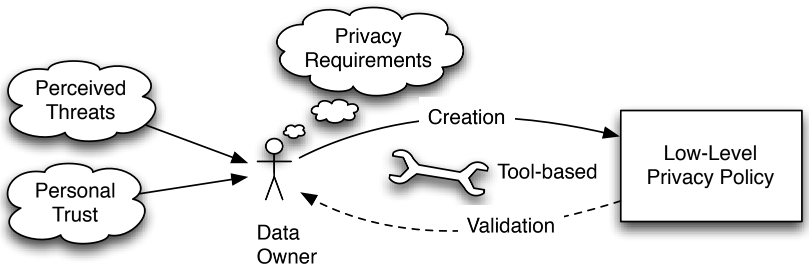


Figure 7.2: ORAC Policy Creation and Validation Process - It is the responsibility of the Data Owner to express his or her privacy requirements as a low-level, enforceable policy. No separate documentation exists if policies need to be verified or adapted at a later time.

envision that it would be possible to interrogate the formal privacy policy by answering questions about what actions are allowed or denied for a particular data user. Potential weaknesses and inconsistencies in the policy could be listed, e.g. when user rules are masked or overwritten by the safety rules of the policy. If an existing policy needs to be adapted, the validation tool could be used to find out if the policy corresponds to the intentions of the data owner. The tool should be able to generate a human readable summary of the privacy policy.

7.4 Final Words

Privacy-aware sharing of data is possible. Our research has shown that a privacy framework can be build that respects and automatically imposes privacy requirements that have been set by the data owner. Uptake of our scheme depends on a number of factors that are no longer under our control, such as the implementation of our ideas in open data sharing infrastructures and the testing of these solutions with a diversified number of users to make privacy-aware data sharing robust and user friendly.

We very much hope that it soon becomes possible to have a level of control over the distribution and release of electronic personal data that is similar to the level of privacy that we enjoy and value in personal interactions with people around us.

Appendix

OECD Privacy Guidelines

The OECD Privacy Guidelines [OECD, 1980] define the following eight principles for the collection of personal identifiable data:

1. **Collection Limitation Principle** There should be limits to the collection of personal data and any such data should be obtained by lawful and fair means and, where appropriate, with the knowledge or consent of the data subject.
2. **Data Quality Principle** Personal data should be relevant to the purposes for which they are to be used, and, to the extent necessary for those purposes, should be accurate, complete and kept up-to-date.
3. **Purpose Specification Principle** The purposes for which personal data are collected should be specified not later than at the time of data collection and the subsequent use limited to the fulfilment of those purposes or such others as are not incompatible with those purposes and as are specified on each occasion of change of purpose.
4. **Use Limitation Principle** Personal data should not be disclosed, made available or otherwise used for purposes other than those specified in accordance with Paragraph 3 except:
   (a) with the consent of the data subject; or
   (b) by the authority of law.
5. **Security Safeguards Principle** Personal data should be protected by reasonable security safeguards against such risks as loss or unauthorised access, destruction, use, modification or disclosure of data.
6. **Openness Principle** There should be a general policy of openness about developments, practices and policies with respect to personal data. Means should be readily available of establishing the existence and nature of personal data, and the main purposes of their use, as well as the identity and usual residence of the data controller.
7. **Individual Participation Principle** An individual should have the right:
   (a) to obtain from a data controller, or otherwise, confirmation of whether or not the data controller has data relating to him;
   (b) to have communicated to him, data relating to him
      - within a reasonable time;
      - at a charge, if any, that is not excessive;
      - in a reasonable manner; and
      - in a form that is readily intelligible to him;
   (c) to be given reasons if a request made under subparagraphs (a) and (b) is denied, and to be able to challenge such denial; and
   (d) to challenge data relating to him and, if the challenge is successful to have the data erased, rectified, completed or amended.

8. Accountability Principle A data controller should be accountable for complying with measures which give effect to the principles stated above.

Priority Policy Algorithm

The following code shows the implementation of the *Privacy Policy Precedence Relation* (P3R) as a policy combining algorithm within the SunXACML framework [SunXACML, 2006]:

```java
/**
  * @(#)PriorityPolicyAlg.java
 */
package de.uni_potsdam.reference_monitor.pdp;

import java.net.URI;
import java.net.URISyntaxException;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Set;

import com.sun.xacml.AbstractPolicy;
import com.sun.xacml.EvaluationCtx;
import com.sun.xacml.Indenter;
import com.sun.xacml.MatchResult;
import com.sun.xacml.combine.CombinerParameter;
import com.sun.xacml.combine.PolicyCombinerElement;
import com.sun.xacml.combine.PolicyCombiningAlgorithm;
import com.sun.xacml.ctx.Result;
import com.sun.xacml.ctx.Status;
import com.sun.xacml.attr.IntegerAttribute;

import de.uni_potsdam.reference_monitor.helper.DEBUG;


/**
 *
 * This is the standard Priority Policy combining algorithm. It looks
 * through the set of policies, derives an access decision from each of the applicable
     policies,
 * and returns the evaluation result for the policy with the highest priority.
 *
 * @author Thomas Scheffler
 * @version 1.1
 */


public class PriorityPolicyAlg extends PolicyCombiningAlgorithm {

    /**
     * The standard URN used to identify this algorithm
     */
    public static final String algId =
        "urn:policy-combining-alg:" +
        "priority";

    /**
     * Standard constructor.
     */
    public PriorityPolicyAlg() throws URISyntaxException {
        super(new URI("urn:policy-combining-alg:priority"));
    }

    /**
     * Applies the combining algorithm to the set of policies based on the
     * evaluation context.
     *
     * @param context the context from the request
     * @param parameters a (possibly empty) non-null <code>List</code> of
     *                  <code>CombinerParameter<code>s
     * @param policyElements the rules to combine
     *
     * @return the result of running the combining algorithm
     */
    public Result combine(EvaluationCtx context, List parameters, List policyElements)
         {

        if (DEBUG.LEVEL2){
                System.out.println("\n in Methode "
                                + "PriorityPolicyAlg.combine");
        }

        boolean atLeastOnePermit = false;
        Set permitObligations = new HashSet();
        AbstractPolicy selectedPolicy = null;
        Iterator it = policyElements.iterator();
        long policyPriorityParameter=-1;
        Result policyResult = null;


        while (it.hasNext()) {
            long currentPolicyPriority = -1;
            PolicyCombinerElement policyCombElement = (PolicyCombinerElement)(it.next()
                );
            // get the policy
            AbstractPolicy policy = policyCombElement.getPolicy();
            // get the associated Combining Parameters for this policy
            List parList = policyCombElement.getParameters();
            // check if Combining Parameters are present, get value of parameter named
                "Priority"
            for(int i=0, size=parList.size(); i < size; i++){
                CombinerParameter combPar = (CombinerParameter)parList.get(i);
                if (combPar.getName().equals("Priority")){
                    IntegerAttribute intAttr = (IntegerAttribute)combPar.getValue();
                    currentPolicyPriority = (long) intAttr.getValue();
                    if (currentPolicyPriority < 1){
                        List code = new ArrayList();
                        code.add(Status.STATUS_PROCESSING_ERROR);
                        String message = "Policy prioity out of bound. Must be > 0";
                        return new Result(Result.DECISION_INDETERMINATE,
                                      new Status(code, message),
                                      context.getResourceId().encode());
                    }
                    if (currentPolicyPriority == policyPriorityParameter){
                        //INFO: we assume a strict hierarchy and do not combine
                            multiple policies
                        //      at the same priority level.
                        List code = new ArrayList();
                        code.add(Status.STATUS_PROCESSING_ERROR);
                        String message = "Policies using same prioity level found! Must
                             be different.";
                        return new Result(Result.DECISION_INDETERMINATE,
                                      new Status(code, message),
                                      context.getResourceId().encode());
                    }
                    if (DEBUG.LEVEL2){
                        System.out.println("Current Policy Priority:" +
                            currentPolicyPriority);
                    }
                }
                else{
                    List code = new ArrayList();
                    code.add(Status.STATUS_PROCESSING_ERROR);
                    String message = "No policy prioity level found! Must be set.";
                    return new Result(Result.DECISION_INDETERMINATE,
                                  new Status(code, message),
                                  context.getResourceId().encode());
                }

                if (DEBUG.LEVEL2){
                    System.out.println("End Policy Priority:" + currentPolicyPriority);
                }
            }

            // see if the policy matches the context
            MatchResult match = policy.match(context);
            int matchResult = match.getResult();

            // if there is an error in trying to match any of the targets,
            // we always return INDETERMINATE immediately
            if (matchResult == MatchResult.INDETERMINATE)
                return new Result(Result.DECISION_INDETERMINATE,
                              match.getStatus(),
                              context.getResourceId().encode());

            if (matchResult == MatchResult.MATCH) {
                Result evalResult = policy.evaluate(context);
                int effect = evalResult.getDecision();
                if (DEBUG.LEVEL2){
                    System.out.println("PolicyEvaluation: " + effect);
                }
                // if this policy has a higher priority than the previously visited
                    policy,
                // and decision is permit or deny, remember it for later
                if (effect <2 && (policyPriorityParameter == -1 ||
                currentPolicyPriority > policyPriorityParameter)){
                    policyPriorityParameter = currentPolicyPriority;
                    policyResult = evalResult;
                }
            }
        }

        // if we got through the loop and found one or more matches, then
        // we return the evaluation result of the policy with the highest
        // priority.
        // Otherwise it's NOT_APPLICABLE
        if (policyPriorityParameter > 0)
            return policyResult;
        else
            return new Result(Result.DECISION_NOT_APPLICABLE,
                          context.getResourceId().encode());
    }
}
```

Listing 1: Implementation of the Privacy Policy Precedence Relation as a policy combining algorithm within the SunXACML framework [SunXACML, 2006]

List of Figures

List of Tables

Listings

Abbreviations

AAL Ambient Assisted Living

ACL Access Control List

ACR Access Control Rules

ADF Access Control Decision Function

AEF Access Control Enforcement Function

AES Advanced Encryption Standard

AOP Aspect-oriented Programming

API Application Programming Interface

CA Certificate Authority

EPAL Enterprise Privacy Authorization Language

DAC Discretionary Access Control

DAO Data Access Object

DRM Digital Rights Management

EHR Electronic Health Record

GPS Global Positioning System

HRV Health Record Viewer

HTML Hypertext Markup Language

IBE Identity-Based Encryption

IETF Internet Engineering Taskforce

ISO International Organization for Standardization

MAC Mandatory Access Control

MANET Mobile Ad-hoc Network

OLSR	Optimized Link State Routing
ORAC	Owner-Retained Access Control
OSI	Open Systems Interconnection
P3R	Privacy Policy Precedence Relation
P3P	Platform for Privacy Preferences
PAP	Policy Administration Point
PDP	Policy Decision Point
PET	Privacy Enhancing Technology
PEP	Policy Enforcement Point
PGP	Pretty Good Privacy
PHR	Personal Health Record
PII	Personally Identifiable Information
PPI	Personally Private Information
PKI	Public Key Infrastructure
RSA	Rivest, Shamir, & Adleman (public key encryption technology)
TPM	Trusted Platform Module
W3C	World Wide Web Consortium
XACML	eXtensible Access Control Markup Language
XML	Extensible Markup Language

Bibliography

Abrams, Marshall D., Leonard J. LaPadula, Kenneth W. Eggers, and Ingrid M. Olson. 1990. A generalized framework for access control: An informal description. *Proceedings of the 13th National Computer Security Conference*, 135–143.

Agre, Philip E. and Marc Rotenberg, eds. 1997. *Technology and Privacy: The New Landscape*. MIT Press.

Allan, Alasdair and Pete Warden. 2011. iPhone Tracking "What Your iPhone Knows About You". O'Reilly Where 2.0 Conference.

Amazon. 2008. Privacy notice. `http://www.amazon.com/gp/help/customer/display.html?nodeId=468496`.

Anderson, Anne. 2005. A Comparison of Two Privacy Policy Languages: EPAL and XACML. Tech. Rep. Technical Report 2005-147, Sun Microsystems Laboratories. `http://research.sun.com/techrep/2005/abstract-147.html`.

Anderson, J. P. 1972. Computer security technology planning study. Technical Report ESD-TR-73-51. Tech. Rep., Electronic System Division/AFSC.

Anderson, Ross J. 2001. *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley &Sons, Inc.

Anton, Annie I., Qingfeng He, and David L. Baumer. 2004. Inside JetBlues's Privacy Policy Violation. *IEEE Security and Privacy* 2, no. 6: 12–18.

Apitzsch, Felix, Stefan Liske, Thomas Scheffler, and Bettina Schnor. 2008. Specifying Security Policies for Electronic Health Records. *Proceedings of the International Conference on Health Informatics (HEALTHINF 2008)*, vol. 2, 82 – 90. Funchal/Madeira, Portugal.

AppArmor. 2007. AppArmor Application Security for Linux. `http://www.novell.com/linux/security/apparmor/`.

Ashley, Paul and Günter Karjoth. 2003. Shortcomings of P3P for Privacy Authorization - Lessons learned when Using P3P-based Privacy Manager 1.1. `http://www.w3.org/2003/p3p-ws/pp/ibm1.html`.

Barmer. 2010. Barmer - Gesundheitsakte. `https://www.lifesensor.com/de/barmer`.

Barth, Adam and John C. Mitchell. 2005. Enterprise privacy promises and enforcement. *Proceedings of the 2005 workshop on Issues in the theory of security*, 58–66. Long Beach, California: ACM Press.

Barth, Adam, John C. Mitchell, and Justin Rosenstein. 2004. Conflict and combination in privacy policy languages. *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, 45–46. Washington DC, USA: ACM Press.

Bell, D. Elliot and Leonard J. LaPadula. 1973. Secure computer systems: Mathematical foundations. Tech. Rep., MITRE Technical Report 2547, Volume I.

Benassi, Paola. 1999. TRUSTe: an online privacy seal program. *Communications of the ACM* 42: 56–59.

Bertino, Elisa, M. Braun, Silvana Castano, Elena Ferrari, and Marco Mesiti. 2001. Author-X: A Java-Based System for XML Data Protection. *Proceedings of the IFIP TC11/ WG11.3 Fourteenth Annual Working Conference on Database Security: Data and Application Security, Development and Directions*, 15–26. Kluwer, B.V.

Boneh, Dan and Matthew Franklin. 2003. Identity-based encryption from the weil pairing. *SIAM J. Comput.* 32, no. 3: 586–615.

Brewer, D. F. C. and M. J. Nash. 1989. The Chinese Wall security policy. *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, 206–214. doi:http://dx.doi.org/10.1109/SECPRI.1989.36295. http://dx.doi.org/10.1109/SECPRI.1989.36295.

BVerfG 65,1. 1984. Bundesverfassungsgericht: BVerfGE 65, 1 "Volkszählung". *Neue Juristische Wochenschrift*, 419. Verlag C. H. Beck.

Callas, J., L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. 2007. OpenPGP Message Format. RFC 4880, Internet Engineering Task Force. http://tools.ietf.org/html/rfc4880.

CEN/TS-15211. 2006. Health informatics - Mapping of hierarchical message descriptions to XML. European Committee for Standardisation, http://www.cen.eu.

Chaum, David. 1985. Security without identification: transaction systems to make big brother obsolete. *Communications of the ACM* 28, no. 10: 1030–1044. http://doi.acm.org/10.1145/4372.4373.

Chaum, David L. 1981. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM* 24, no. 2: 84–90. http://doi.acm.org/10.1145/358549.358563.

Chen, Kung and Da-Wei Wang. 2007. An Aspect-Oriented Approach to Privacy-Aware Access Control. *Machine Learning and Cybernetics, 2007 International Conference on*, vol. 5, 3016 –3021. doi:10.1109/ICMLC.2007.4370665.

Clausen, T. and P. Jacquet. 2003. Optimized Link State Routing Protocol (OLSR). RFC 3626, Internet Engineering Task Force. http://tools.ietf.org/html/rfc3626.

Cranor, Lorrie and Aleecia McDonald. 2008. The Cost of Reading Privacy Policies. *Proceedings of 36th Research Conference on Communication, Information & Internet Policy (TPRC)*. http://tprcweb.com/files/CostOfReadingPrivacyPolicies.pdf.

Cranor, Lorrie Faith. 2002. *Web Privacy with P3P.* O'Reilly.

Damiani, E., S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. 2002. A Fine-Grained Access Control System for XML Documents. *ACM Transactions on Information and System Security* 5, no. 2: 169–202.

DataLossDB. 2012. DataLossDB. Open Security Foundation. `http://datalossdb.org/`.

Dierks, T. and E. Rescorla. 2008. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, Internet Engineering Task Force. `http://tools.ietf.org/html/rfc5246`.

Directive 95/46/EC. 1995. On the protection of individuals with regard to the processing of personal data and on the free movement of such data. `http://ec.europa.eu/justice_home/doc_centre/privacy/law/index_en.htm`.

DoD 5200.28-STD. 1985. Trusted computer system evaluation criteria. Department of Defense Standard 5200.28-STD.

DTOS. 1997. DTOS General System Security and Assurability Assessment Report. Tech. Rep. MD A904-93-C-4209 CDRL A011, Secure Computing Corporation. `http://www.securecomputing.com/randt/HTML/dtos.html`.

EN-13606-4. 2007. Health informatics - Electronic health record communication - Part 4: Security. European Committee for Standardisation. `http://www.cen.eu`.

EPAL 1.2. 2003. Enterprise Privacy Authorization Language (EPAL 1.2). W3C Member Submission. `http://www.w3.org/Submission/2003/SUBM-EPAL-20031110/`.

Evered, Mark and Serge Bögeholz. 2004. A case study in access control requirements for a Health Information System. *Proceedings of the second workshop on Australasian information security, Data Mining and Web Intelligence, and Software Internationalisation - Volume 32*, 53–61. Dunedin, New Zealand: Australian Computer Society, Inc.

Fair Information Principles. 1973. A Review of the Fair Information Principles: The Foundation of Privacy Public Policy. `http://www.privacyrights.org/ar/fairinfo.htm`.

FIPS PUB 197. 2001. Specification for the Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197. `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.

Fischer-Hübner, Simone. 2001. *IT-Security and Privacy: Design and Use of Privacy-Enhancing Security Mechanisms*, vol. 1958 of *Lecture Notes in Computer Science*. Springer-Verlag New York, Inc.

Fudickar, S. and B. Schnor. 2009. KopAL - A Mobile Orientation System For Dementia Patients. *Communications in Computer and Information Science, Int. Conf. Intelligent Interactive Assistance and Mobile Multimedia Computing*, vol. 53, 109–118. Berlin Heidelberg, Germany: Springer.

Fudickar, Sebastian, Bettina Schnor, Juliane Felber, Franz J. Neyer, Mathias Lenz, and Manfred Stede. 2011. KopAL - An Orientation System For Patients With Dementia. *Behaviour Monitoring and Interpretation - BMI*, 83–104. IOS Press.

Geambasu, Roxana, Tadayoshi Kohno, Amit Levy, and Henry M. Levy. 2009. Vanish: Increasing data privacy with self-destructing data. *Proceedings of the 18th USENIX Security Symposium.*

Geiß, Stefan. 2007. Realisierung eines Referenzmonitors für die kontrollierte Nutzung verteilter elektronischer Daten. Diplomarbeit, Universität Potsdam.

Geopriv-Charter. n.d. Charter of the Geographic Location/Privacy Working Group. `http://datatracker.ietf.org/wg/geopriv/charter/`.

GKV 2003. 2003. Gesetz zur Modernisierung der gesetzlichen Krankenversicherung, SGB V, §291a. *Bundesgesetzblatt*, vol. 55. Bundesgesundheitsministerium.

Goldberg, Ian. 2003. Privacy-enhancing technologies for the internet, II: five years later. *PET'02: Proceedings of the 2nd international conference on Privacy enhancing technologies*, 1–12. Berlin, Heidelberg: Springer-Verlag.

Goldberg, Ian, David Wagner, and Eric A. Brewer. 1997. Privacy-enhancing technologies for the Internet. *Proceedings of the 42nd IEEE International Computer Conference*, 103. IEEE Computer Society.

Gong, Li, Marianne Mueller, Hemma Prafullchandra, and Roland Schemers. 1997. Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit. *USENIX Symposium on Internet Technologies and Systems*. Monterey, California.

Gong, Li, Gary Ellison, and Mary Dageforde. 2003. *Inside Java 2 Platform Security - Second Edition*. Boston: Addison-Wesley.

Google Health. 2008. Google - personal health record. `https://www.google.com/health`.

Graham, G. Scott and Peter J. Denning. 1972. Protection - Principles and Practice. *Spring Joint Computer Conference*, vol. 40, 417–429. AFIPS Press.

Gupta, Rajeev and Manish Bhide. 2005. A Generic XACML Based Declarative Authorization Scheme for Java. *Lecture Notes in Computer Science: Computer Security - ESORICS 2005*, vol. 3679. Springer Berlin / Heidelberg.

Hohl, A. and A. Zugenmaier. 2006. Safeguarding Personal Data using Rights Management in Pervasive Computing for Distributed Applications. Technical Report, University of Freiburg. `http://www.telematik.uni-freiburg.de/opendownloads/hozu06.pdf`.

ISO 7498-2. 1991. Security Architecture for Open Systems Interconnection for CCITT Applications. Obtainable from http://www.itu.int/itudoc/ itu-t/rec/x/x500up/x800.html.

ISO/HL7-21731. 2006. Health informatics - HL7 version Reference information model Release 1).

ISO/IEC 10181-1. 1996. Information technology - Open Systems Interconnection - Security frameworks for open systems: Overview.

ISO/IEC 10181-3. 1996. Information technology - Open Systems Interconnection - Security frameworks for open systems: Access control framework.

Jajodia, Sushil, Pierangela Samarati, Maria Luisa Sapino, and V. S. Subrahmanian. 2001. Flexible support for multiple access control policies. *ACM Trans. Database Syst.* 26, no. 2: 214–260.

JBoss Inc. 2012. JBoss AOP - Aspect-Oriented Framework for Java. `http://docs.jboss.org/aop/1.1/aspect-framework/reference/en/html/index.html`.

Jensen, Carlos and Colin Potts. 2004. Privacy policies as decision-making tools: an evaluation of online privacy notices. *Proceedings of the SIGCHI conference on Human factors in computing systems*, 471–478. Vienna, Austria: ACM Press.

Karjoth, Günter and Matthias Schunter. 2002. A privacy policy model for enterprises. *Proceedings of the 15th IEEE Computer Security Foundations Workshop (CSFW'02)*, 271. IEEE Computer Society.

Karjoth, Günter, Matthias Schunter, and Els Van Herreweghen. 2003. Translating Privacy Practices into Privacy Promises-How to Promise What You Can Keep. *POLICY '03: Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, 135–146. Washington, DC, USA: IEEE Computer Society.

Karjoth, Günter, Matthias Schunter, and Michael Waidner. 2003. Platform For Enterprise Privacy Practices: Privacy-enabled Management Of Customer Data. *2nd Workshop on Privacy Enhancing Technologies (PET2002)*, vol. 2482 of *Lecture Notes in Computer Science*, 69–84. Springer Verlag.

Karjoth, Günter, Andreas Schade, and Els Van Herreweghen. 2008. Implementing ACL-Based Policies in XACML. *Proceedings of the 2008 Annual Computer Security Applications Conference*, ACSAC '08, 183–192. Washington, DC, USA: IEEE Computer Society. `http://dx.doi.org/10.1109/ACSAC.2008.31`.

Kent, S. and K. Seo. 2005. Security Architecture for the Internet Protocol. RFC 4301, Internet Engineering Task Force. `http://tools.ietf.org/html/rfc4301`.

Kruppa, Michael. 2011. Emergency indoor and outdoor user localization. *Demographischer Wandel - Assistenzsysteme aus der Forschung in den Markt (Proceedings 4. Deutscher AAL-Kongress)*. Berlin: VDE Verlag.

Lampson, Butler W. 1971. Protection. *Proceedings of the 5th Princeton Symposium on Information Sciences and Systems*, 437–443. Princeton University. Reprinted in ACM Operating Systems Review, 8, 1, January 1974, pp. 18-24, `http://doi.acm.org/10.1145/775265.775268`.

Landwehr, Carl E. 1981. Formal models for computer security. *ACM Comput. Surv.* 13, no. 3: 247–278. `http://doi.acm.org/10.1145/356850.356852`.

Langendörfer, Peter, Michael Maaser, Krzysztof Piotrowski, and Steffen Peter. 2008. *Privacy Enhancing Techniques: A Survey and Classification.* Handbook of Research on Wireless Security. Hershey, PA: Information Science Reference - Imprint of: IGI Publishing.

Lehmann, Kathrin and Peter Thiemann. 2006. Field Access Analysis for Enforcing Access Control Policies. *Proceedings of the International Conference on Emerging Trends in Information and Communication Security (ETRICS 2006)*, vol. 3995 of *Lecture Notes in Computer Science*, 337–351. Berlin, Heidelberg: Springer-Verlag.

Lupu, Emil C. and Morris Sloman. 1999. Conflicts in policy-based distributed systems management. *IEEE Trans. Softw. Eng.* 25, no. 6: 852–869. doi:http://dx.doi.org/10.1109/32.824414.

Maaser, Michael and Peter Langendörfer. 2009. Privacy from Promises to Protection: Privacy Guaranteeing Execution Container. *Mobile Networks and Applications* 14, no. 1: 65–81. doi: http://dx.doi.org/10.1007/s11036-008-0116-7.

Manyika, James, Michael Chui, Brad Brown, Jacques Bughin, Richard Dobbs, Charles Roxburgh, and Angela Hung Byers. 2011. Big data: The next frontier for innovation, competition, and productivity. Tech. Rep., McKinsey Global Institute. `http://www.mckinsey.com/Insights/MGI/Research/Technology_and_Innovation/Big_data_The_next_frontier_for_innovation`.

Markle Foundation. 2003. Connecting for health. The Personal Health Working Group Final Report. `http://www.connectingforhealth.org/resources/final_phwg_report1.pdf`.

Marsh, Stephen Paul. 1994. Formalising trust as a computational concept. Ph.D. thesis, University of Stirling.

McCollum, Catherine Jensen, Judith R. Messing, and LouAnna Notargiacomo. 1990. Beyond the pale of MAC and DAC-defining new forms of access control. *IEEE Computer Society Symposium on Research in Security and Privacy*, 190–200. `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=63850`.

Microsoft HealthVault. 2010. Microsoft HealthVault - Personal Health Record. `http://www.healthvault.com/Personal/index.aspx`.

Mont, Marco Casassa, Siani Pearson, and Pete Bramhall. 2003. Towards accountable management of identity and privacy: Sticky policies and enforceable tracing services. *Proceedings of the 14th International Workshop on Database and Expert Systems Applications*, 377. IEEE Computer Society.

Morris, John and Jon Peterson. 2007. Who's watching you now? *IEEE Security and Privacy* 5: 76–79. doi:http://doi.ieeecomputersociety.org/10.1109/MSP.2007.24.

Navarro, Guillermo, Babak S. Firozabadi, Erik Rissanen, and Joan Borrell. 2003. Constrained Delegation in XML-based Access Control and Digital Rights Management Standards. *CNIS03, Special Session on Architectures and Languages for Digital Rights Management and Access Control*, ed. M. H. Hamza, 271–276. Acta Press.

Oaks, Scott. 2001. *Java Security*. Sebastopol: O'Reilly, 2nd edn.

OECD. 1980. OECD Guidelines on the Protection of Privacy and Transborder Flows of Personal Data. http://www.oecd.org/document/18/0,2340,en_2649_34255_1815186_1_1_1_1,00.html.

P3P APPEL. 2002. A P3P Preference Exchange Language 1.0 (APPEL1.0). W3C Draft. http://www.w3.org/TR/2002/WD-P3P-preferences-20020415.

P3P v1.0. 2002. The Platform for Privacy Preferences 1.0 (P3P 1.0) Specification. W3C Recommendation. http://www.w3.org/TR/2002/REC-P3P-20020416/.

P3P v1.1. 2006. The Platform for Privacy Preferences 1.1 (P3P 1.1) Specification. W3C Group Note. http://www.w3.org/TR/2006/NOTE-P3P11-20061113/.

Park, Jaehong and Ravi Sandhu. 2004. The $\mathrm{UCON}_{\mathrm{ABC}}$ usage control model. *ACM Transactions on Information and System Security* 7, no. 1: 128–174. doi:http://doi.acm.org/10.1145/984334.984339.

PRIME. 2006-2008. PRIME - Privacy and Identity Management for Europe. https://www.prime-project.eu/.

Privacy Rights Clearinghouse. 2012. A chronology of data breaches. http://www.privacyrights.org/data-breach.

Provos, Niels. 2009. Systrace - Interactive Policy Generation for System Calls. http://www.citi.umich.edu/u/provos/systrace/.

Ramsdell, B. and S. Turner. 2010. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2. RFC 5751, Internet Engineering Task Force. http://tools.ietf.org/html/rfc5751.

Reenskaug, T. M. H. 1978. MVC XEROX PARC 1978-79. http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html.

Rivest, Ronald L., Adi Shamir, and Leonard Adleman. 1978. A method for obtaining Digital Signatures and Public Key Cryptosystems. *Communications of the ACM* 21, no. 2: 120–126.

Rössler, Beate. 2001. *Der Wert des Privaten*. Suhrkamp.

Saltzer, Jerome D. and Michael D. Schroeder. 1975. The Protection of Information in Computer Systems. *Proceedings of the IEEE* 63, no. 9: 1278–1308.

Samarati, Pierangela and Sabrina De Capitani di Vimercati. 2001. Access control: Policies, models, and mechanisms. *FOSAD '00: Foundations of Security Analysis and Design*, vol. 2171 of *Lecture Notes In Computer Science*, 137–196. London, UK: Springer-Verlag.

Sandhu, Ravi S. and Pierrangela Samarati. 1994. Access control: Principles and practice. *IEEE Communications Magazine* 32, no. 9: 40–48. http://citeseer.ist.psu.edu/sandhu94access.html.

Scheffler, Thomas, Stefan Geiß, and Bettina Schnor. 2008. An Implementation of a Privacy Enforcement Scheme based on the Java Security Framework using XACML Policies. *Proceedings of the IFIP TC 11, 23rd International Information Security Conference*, vol. 278/200, 157–171. Springer Boston. doi:10.1007/978-0-387-09699-5_11.

Scheffler, Thomas, Sven Schindler, Marcus Lewerenz, and Bettina Schnor. 2011. A Privacy-Aware Localization Service for Healthcare Environments. *Privacy and Security in Pervasive Environments (PSPAE'11) Workshop at PETRA 2011*. Crete, Greece.

Scheffler, Thomas, Sven Schindler, and Bettina Schnor. 2012. Enforcing Location Privacy Policies through an AOP-based Reference-Monitor. *Proceedings of the World Congress on Internet Security (WorldCIS-2012)*. Guelph, Canada.

Schneider, Cornelia and Elisabeth Häusler. 2011. Mobilitätssichernde Assistenzsysteme - Ergebnisse einer Akzeptanzstudie, Mobility safeguarding assistance systems - Results of an acceptance test. *Demographischer Wandel - Assistenzsysteme aus der Forschung in den Markt (Proceedings 4. Deutscher AAL-Kongress)*. Berlin: VDE Verlag.

Schulzrinne, H., H. Tschofenig, J. Morris, J. Cuellar, J. Polk, and J. Rosenberg. 2007. Common Policy: A Document Format for Expressing Privacy Preferences. RFC 4745, Internet Engineering Task Force. `http://tools.ietf.org/html/rfc4745`.

Sevinç, Paul E. and David Basin. 2006. Controlling Access to Documents: A Formal Access Control Model. Technical Report No. 517, Department of Computer Science, ETH Zürich, 8092 Zürich, Switzerland.

Sevinç, Paul E., Mario Strasser, and David Basin. 2007. Securing the distribution and storage of secrets with trusted platform modules. *WISTP 2007*, vol. 4462 of *LNCS*, eds. Damien Sauveron, Konstantinos Markantonakis, Angelos Bilas, and Jean-Jacques Quisquater, 53–66. Springer. `http://www.springer.com/dal/home/computer/mathematics?SGWID=1-151-22-173738321-0`.

Shamir, Adi. 1979. How to share a secret. *Communications of the ACM* 22, no. 11: 612–613. `http://doi.acm.org/10.1145/359168.359176`.

Shirey, R. 2007. Internet Security Glossary, Version 2. RFC 4949, Internet Engineering Task Force. `http://tools.ietf.org/html/rfc4949`.

Spring Framework. 2011. Spring - The Standard for Enterprise Java Development. Tech. Rep., VMware Inc. `http://www.springsource.com/developer/spring`.

Stytz, Martin R. 2005. Protecting Personal Privacy: Hauling Down the Jolly Roger. *IEEE Security and Privacy* 3, no. 4: 72–74.

SunXACML. 2006. Sun's XACML implementation. `http://sunxacml.sourceforge.net/`.

Tavani, Herman T. 1999. Privacy online. *SIGCAS Comput. Soc.* 29: 11–19.

Trusted Computing Group. 2007. TCG Architecture Specification v1.4. `http://www.trustedcomputinggroup.org/resources/tcg_architecture_overview_version_14/`.

Tschantz, Michael Carl and Shriram Krishnamurthi. 2006. Towards reasonability properties for access-control policy languages. *SACMAT '06: Proceedings of the eleventh ACM symposium on Access control models and technologies*, 160–169. New York, NY, USA: ACM Press. doi: http://doi.acm.org/10.1145/1133058.1133081.

UDHR. 1948. Universal declaration of human rights. `http://www.ohchr.org/EN/UDHR/Pages/Language.aspx?LangID=eng`.

University of Texas. 2008. The University of Texas at Austin Web Privacy Policy. `http://www.utexas.edu/policies/privacy/`.

Vimercati, Sabrina De Capitani di, Pierangela Samarati, and Sushil Jajodia, eds. 2005. *Policies, Models, and Languages for Access Control.* Proc. 4th International Workshop on Databases in Networked Information Systems (DNIS 2005). Japan: Springer Lecture Notes in Computer Science Vol. 3433.

W3C Comment. 2003. W3C Team Comment on EPAL 1.2. `http://www.w3.org/Submission/2003/07/Comment`.

Warren, Samuel D. and Louis D. Brandeis. December 15, 1890. The right to privacy. *Harvard Law Review* IV, no. 5: 193–220. `http://www.lawrence.edu/fast/boardmaw/Privacy_brand_warr2.html`.

Winterbottom, J., M. Thomson, and H. Tschofenig. 2009. GEOPRIV Presence Information Data Format Location Object (PIDF-LO). RFC 5491, Internet Engineering Task Force. `http://tools.ietf.org/html/rfc5491`.

Woo, Thomas Y. C. and Simon S. Lam. 1993. Authorizations in distributed systems: A new approach. *Journal of Computer Security* 2, no. 2-3: 107–136.

XACML-2.0. 2005. eXtensible Access Control Markup Language (XACML). OASIS-Standard. `http://www.oasis-open.org/committees/xacml`.

XACML Privacy Profile. 2005. Privacy policy profile of XACML v2.0. `http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-privacy_profile-spec-os.pdf`.

XML Encryption. 2002. XML Encryption Syntax and Processing. W3C Recommendation. `http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/`.

XPath. 1999. XML Path Language (XPath). W3C Recommendation. `http://www.w3.org/TR/1999/REC-xpath-19991116`.

XQuery. 2007. XQuery 1.0 and XPath 2.0 Functions and Operators. W3C Recommendation. `http://www.w3.org/TR/xpath-functions/`.

XrML. 2001. eXtensible rights Markup Language (XrML) 2.0 Specification. `http://www.xrml.org`.

Yavatkar, R., D. Pendarakis, and R. Guerin. 2000. A Framework for Policy-based Admission Control. RFC 2753, Internet Engineering Task Force. `http://tools.ietf.org/html/rfc2753`.

Index

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbstständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt und keine anderen als die von mir angegebenen Quellen und Hilfsmittel benutzt habe.

Berlin, 02.09.2013

Thomas Scheffler