Draft 2nd Design for Policy Languages and Protocols

Heartbeat: H 5.3.2

Editor:	Dave Raggett (W3C)
Reviewers:	Sandra Steinbrecher (TUD) Jan Schallaböck (ULD) Martin Pekarek (TILT)
Identifier:	H5.3.2
Type:	Heartbeat
Class:	Internal
Date:	07 July 2009

Abstract

This document is a heartbeat for the PrimeLife project and defines the second design of policy languages and protocols for use in enabling a Data Subject and Data Controller to attempt to reach agreement on a) the credentials needed to access a resource, and b) the obligations upon the Data Controller for handling the associated personal data relating to the Data Subject, whether this is collected directly from the Data Subject or indirectly from other sources.

Conventions of this Document

All sections in this specification are normative, unless otherwise indicated. The informative parts of this specification are identified by "Informative" labels within sections.

The key words "must", "must NOT", "REQUIRED", "shall", "shall NOT", "should", "should NOT", "RECOMMENDED", "may", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n^{o} 216483.

Members of the PrimeLife Consortium

1.	IBM Research GmbH	IBM	Switzerland
2.	Unabhängiges Landeszentrum für Datenschutz	ULD	Germany
3.	Technische Universität Dresden	TUD	Germany
4.	Karlstads Universitet	KAU	Sweden
5.	Università degli Studi di Milano	UNIMI	Italy
6.	Johann Wolfgang Goethe - Universität Frankfurt am Main	GUF	Germany
7.	Stichting Katholieke Universiteit Brabant	TILT	Netherlands
8.	GEIE ERCIM	W3C	France
9.	Katholieke Universiteit Leuven	K.U.Leuven	Belgium
10.	Università degli Studi di Bergamo	UNIBG	Italy
11.	Giesecke & Devrient GmbH	GD	Germany
12.	Center for Usability Research & Engineering	CURE	Austria
13.	Europäisches Microsoft Innovations Center GmbH	EMIC	Germany
14.	SAP AG	SAP	Germany
15.	Brown University	UBR	USA

List of Contributors

Contributions from several PrimeLife partners are contained in this document. Individual participants in the Activity 5 are (at the time of writing):

Claudio Ardagna (UNIMI), Carine Bournez (W3C), Laurent Bussard (EMIC), Michele Bezzi (SAP), Jan Camenisch (IBM), Sabrina de Capitani di Dimercati (UNIMI), Aleksandra Kuczerawy (KUL), Sebastian Meissner (ULD), Gregory Neven (IBM), Stefano Paraboschi (UNIBG), Eros Pedrini (UNIMI), Ulrich Pinsdorf (EMIC), Franz-Stefan Preiss (IBM), Slim Trabelsi (SAP), Christina Tziviskou (UNIBG), Dave Raggett (W3C), Thomas Roessler (W3C), Pierangela Samarati (UNIMI), Jan Schallaboeck (ULD), Stuart Short (SAP), Dieter Sommer (IBM), Mario Verdicchio (UNIBG), Rigo Wenning (W3C).

This heartbeat on policy languages and protocols is indebted to the work done on the draft requirements for next generation policies as part of PrimeLife heartbeat H5.1.1.

This deliverable was rendered from HTML pages using PrinceXML from YesLogic Pty Ltd. YesLogic has donated a license of PrinceXML to W3C.

Table of Contents

1 Introduction

This report describes work on a design for a policy language for handling access control and privacy. The design is independent of the underlying transport protocol and bindings would be possible for HTTP and SOAP (Web Services), although such bindings are left to future work. The design of the policy language and associated mechanisms is a work in progress, and a number of areas are still to be completed. These include completion of a formal definition of the language as an XML schema. The set of credentials, triggers and actions is deliberately left open ended, and further work is anticipated on refining a core set for implementation purposes.

The work addresses the scenario where a user is using a Web browser to access a website over HTTP. The website may wish to restrict access to users presenting the appropriate credentials. This report describes extensions to an XML access control language (XACML 3.0), along with the means to provide users with information on what credentials are needed for accessing a given URI. Section 5 provides an introduction to XACML. This is followed by the definition of the policy language model and extensions to XACML for credential-based access control.

The website needs to comply with the requirements of European data protections laws when it comes to the handling of personal data. This report defines a language for the obligations that the website agrees to with the user in regards to the handling of the user's personal data. The obligations take the form of conditions and actions, which need to be executed when the conditions become true, e.g. deleting personal data at the end of the agreed data retention period, or notifying the user when his/her data is accessed for a particular purpose. The obligation language is designed to allow the comparison of user preferences with the website's proposed obligations.

The report finishes with a sketch of the protocol and message flows between Data Subject and Data Controller, and between data controller and downstream Data Controllers (third parties).

An introduction and definitions relating to the protection of personal data can be found on the website of the Data Protection Office of the European Commission. Further information can be found on the European Commission's Data Protection website.

The data protection principles are worth repeating here:

- Fairly and lawfully processed;

- processed for limited and explicit purposes;
- Adequate, relevant and not excessive;
- Accurate;
- Not kept longer than necessary;
- Processed in accordance with the Data Subject's rights;
- Secure;
- Not transferred to third parties without adequate precautions.

Note that some sections have their own references and these have yet to be moved to the references section at the end of the document.

1.1 Terminology

Access Control

The means to control access to resources such as web pages. This may be on the basis of the identity of the entity requesting access, or more generally the presentation of a set of credentials, and possibly some representation of the purpose for accessing the resource, as well as other contextual information, such as the time of day and properties of the resource itself.

Credentials

According to Wikipedia, a credential is an attestation of qualification, competence, or authority issued to an individual by a third party with a relevant de jure or de facto authority or assumed competence to do so. In this document, we define digital credentials to be lists of attribute-value statements certified by an Issuer. Here we abstract from the concrete mechanism (cryptographic or other) by which the authenticity of the attribute values can be verified. We do not impose any restrictions on which attributes can be contained in a credential, but typically these either describe the identity of the credential's owner or the authority assigned to her.

Personal Data

Personal data means any information relating to an identified or identifiable natural person or "Data Subject".

An identifiable person is someone who can be identified, directly or indirectly, in particular by reference to an identification number or to one or more factors specific to his or her physical, physiological, mental, economic, cultural or social identity.

The processing of special categories of data, defined as personal data revealing racial or ethnic origin, political opinions, religious or philosophical beliefs, trade-union membership, and of data concerning health or sex life, is prohibited, subject to certain exceptions (see Article 10 of Regulation (EC) 45/2001).

From the European Directive on the protection of personal data, Regulation (EC) 45/2001, article 2.

Data Controller

The Data Controller means the entity which alone or jointly with others determines the purposes and means of the processing of personal data. The processing of personal data may be carried out by a Data Processor acting on behalf of the Data Controller.

This document describes the means for a User Agent acting on behalf of a user to reach an agreement with a Data Controller over the obligations incurred by the controller for any personal data collected about that user.

Downstream Data Controller

When a Data Controller passes personal data to a third party, that third party incurs obligations in respect to the Data Subject, and is referred to in this document as a "down stream data controller".

Data Subject

The Data Subject is the person whose personal data are collected, held or processed by the Data Controller.

The following strictly speaking refers to EC institutions not generally to EU companies etc.

The controller must give the Data Subject the following information about the data being processed:

1. confirmation as to whether or not data related to him or her are being processed;
2. information about the purposes of the processing operation, the categories of data concerned, and the recipients or categories of recipients to whom the data are disclosed;
3. communication of the data undergoing processing and of any available information as to their source;
4. knowledge of the logic involved in any automated decision process concerning him or her.

The Data Subject has the right to access his data and to require the Controller to rectify without delay any inaccurate or incomplete personal data.

The Data Subject has the right to require the Controller to erase data if the processing is unlawful.

Data Subject’s privacy preferences

The expectation of a Data Subject in terms of how his or her personal data should be handled.

Authorization and Obligations

The Data Subject *authorizes* the Data Controller to process her personal Data Subject to the *obligations* on the Data Controller as agreed with the Data Subject. This document defines a means for Data Controllers to define policies that describe proposed obligations and to pass these to the Data Subject for matching against her preferences. If the Data Subject is satisfied with the match, she will then authorize the Data Controller to proceed. The Data Controller is then required to implement the agreed obligations in respect to the Data Subject's personal data.

In a variant of this approach, the Data Subject could propose obligations to the Data Controller, who would then match them against his policies, and inform the Data Subject if the proposal is acceptable. The end result is the same — a binding agreement on the obligations on the Data Controller for handling the Data Subject's personal data.

Sticky privacy policy

An agreement between Data Subject and Data Controller on the handling of personal data collected from the Data Subject. Sticky policies (as well as privacy preferences and privacy policies) defines how data can be handled. Different aspects are defined:

- Authorizations:
    - *Usage:* what the Data Controller can do with collected data (e.g. use them for a specific purpose).
    - Downstream sharing: under which conditions data can be shared with another Data Controller.
- Obligations: what the Data Controller must do.

Tracking what obligations apply to which items of data is a significant challenge, and further involves the need to track the binding to the Data Subject involved. A complication is that information on the identity of the Data Subject is limited to the credentials provided in the request. The implications have yet to be fully worked through.

Software that needs to access personal data should do so through APIs that enable the Data Controller to identify the entity that is requesting access as well as the purpose involved. This report does not provide such an API, which is left for future work.

User Agent

A software system (such as a web browser) acting on behalf of a user. The user agent acts on user preferences when dealing with a server acting on behalf of a Data Controller.

1.2 High Level Architecture Components

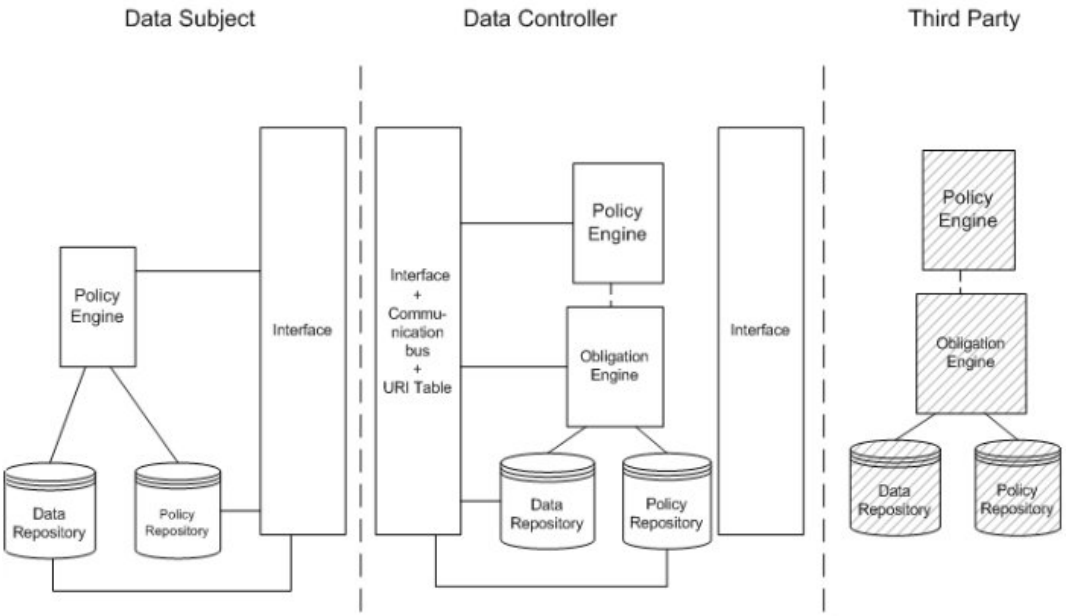


Figure 1: Architecture Components

1.2.1 Data Subject

Policy engine: This component is in charge of parsing and interpreting the privacy preferences of the Data Subject. This policy engine supports the entire PrimeLife Language capabilities (Preferences, Access control, DHP, Obligations etc …). For this reason this module is replicated on the Data Controller side and the third party side.

Data repository: A database containing data owned by the Data Subject. This data could be composed of personal data, credentials, certificated, and other information that should be used during the interaction with the Data Controller application.

Policy repository: A database containing policy files representing the privacy preferences of the Data Subject.

Interface: This interface represents a communication interface with the Data Controller implementing the message exchange protocol (reference to the section Protocols and Message Formats). This interface helps the user to select the appropriate data for the interaction with the Data Controller. The data selected can be retrieved from the data repository module or added directly via the interface (text typing).

1.2.2 Data Controller

Policy engine: this component is the same as the one described in the Data Subject section.

Obligation engine: this component is responsible for handling the obligations that should be satisfied by the Data Controller (reference to the obligation section). This engine sets and executes the triggers related to the actions required by the privacy preferences of the Data Subject.

Data repository: this database contains all the information collected from the Data Subject during an interaction session. This data represents, personal datas, credentials, certificates, and other information provided by the user.

Policy repository: this database contains the privacy policies related to the data provided by the Data Subject.

Interface, Communication bus, URI table: This module has three functional parts:

Interface: This interface represents a communication interface with the data subject implementing the message exchange protocol. This interface plays the role of user interface described in the data subject section, in case of downstream interaction between the data controller and a third party.

Communication bus: in charge of extracting the different information contained in the Data Subject messages then dispatches it to the data repository, the policy repository, the policy engine and the obligation engine.

URI table: this component maintains the links between the data contained in the data repository and the policies contained in the sticky policy repository.

1.2.3 Third Party or Downstream Data Controller

All the components supported by these actors are the same as those described in the Data Controller section. This is due to the fact that the third party plays the role of a Data Controller in case of downstream usage of the data.

1.3 Relationship to Existing Work (State of the Art)

1.3.1 Policy Matching

This section we discuss the different privacy policy matching mechanisms proposed in the literature. We focus on the technical and implementation aspects of the matching without entering in the formalization details.

We can distinguish two implementation architectures: a user-centric and a server-centric architecture. In the user-centric architecture, proposed

for the P3P [P3P], the administrator of a server creates and publishes the privacy policy of the service. Then when the user wants to connect to the server, he will ask first for this policy in order to recover it locally and match it with his privacy preferences. Simple implementations are proposed in Netscape 7 [Netscape 7] and Microsoft Internet Explorer 6 [IE6] allowing the user to specify their privacy preferences for handling cookies. This approach is quite simple, since it is based on a kind check list filled by the user and compared with compact policies published by the servers. The AT&T privacy bird [Privacy Bird] extends the matching scope, limited to cookies, into a more general application field. The user specifies his preferences through a check list related to the usage of different kinds of personal datas. The server expresses its policy with a summarized version of P3P. This summary includes a bulleted summary of each statement in the policy, as well as information from the P3P ACCESS, DISPUTES, and ENTITY elements, including images of any privacy seals referenced. Rather than using the full definitions of each PURPOSE, CATEGORY, RECIPIENT and ACCESS element from the P3P specification. Thibadeau [Thibadeau] proposed a more complex implementation of a matching mechanism called the privacy server protocol. The matching algorithm is iterative; for each rule specified in a customized version of APPEL [P3P Prefs] it compares line by line the rules specified in a P3P file sent by the server. A strong constraint related to the rule writing make this approach limited in terms of expressiveness and matching scope.

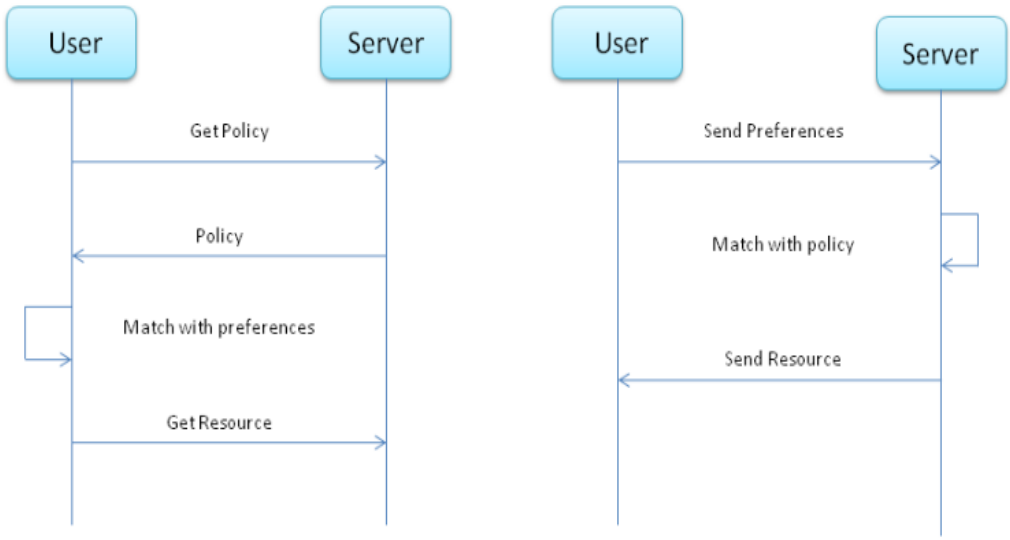


Figure 2: User-centric vs. Server-centric approach

In the server-centric approach, the user agent sends the user's preferences to the server. The server will match locally the preferences with the website's privacy policy. Agrawal et al. [Agrawal] proposed an efficient solution where the server, deploying P3P, installs its privacy policies in a database system. Then database querying is used for matching user’s preferences against privacy policies. They proposed three technical approaches: Convert privacy policies into relational tables and transform an APPEL preference into an SQL query for

matching. Store privacy policies in relational tables, define an XML view over them, and use an XQuery [Boag] derived from an APPEL preference for matching. Or, Store privacy policies in a native XML store and use an XQuery derived from an APPEL preference for matching. This approach seems to be the most powerful in terms of expressivity and dynamicity. Compared to the other approaches relying on abstracted representation of policies (e.g. check lists or simple rules), Agrawal proposed a solution that can be adapted to any XML based privacy language without any expressivity restriction.

Kojima and Itakura [Kojima] proposed a registry based solution relying on an independent privacy matching engine that intermediates between the user agents and the servers. This approach offers a better privacy protection compared to the previous approaches, but it suffers from a lack of expressivity, since it is based on a check list and a pattern based policy structure.

Solutions	User-Centric	Server-Centric	Pros	Cons
Netscape 7 [4]	yes	no	One of the first deployable solutions implementing P3P	Limited to cookies handling. Naïve approach
Internet Explorer 6 [5]	Yes	no	One of the first deployable solutions implementing P3P	Limited to cookies handling. Naïve approach
AT&T privacy bird [6]	yes	no	Implementing P3P. Good exploitation of P3P capabilities Compatible enabled browsers	Failed to conquer a market and never been supported by service providers.
Thibadeau [2]	Yes	No	Advanced P3P matching algorithm	Lack of expressiveness for the matching rules
Agrawal et al [7]	No	Yes	One of the most efficient solutions in terms of expressivity and matching algorithm	Heavy deployment infrastructure. Performance issues.
Kojima et al [9]	No	Yes	High performance for matching	Naïve approach with a serious lack of expressivity

1.3.2 Credential Based Access Control

Technical improvements of Web technologies have fostered the development of online applications that use identity information of users to offer enhanced services. There is a large variety in mechanisms to authenticate and transmit identity information, including X.509 certificates, anonymous credentials, and OpenIDs, and only few of these have provisions in place to protect the privacy of the information that they contain.

In this document we propose an extension to XACML for privacy-enhanced credential-based access control. The credential-based aspect of our language introduces credentials as a common abstraction for the various authentication mechanisms. The privacy enhancements allow to attach two-sided data handling policies to privacy-sensitive resources, specified by means of a concrete set of obligations. Moreover, we change the typical interaction sequence so that the Data Subject is informed about the applicable policies before transmitting his personal information.

1.3.3 Credential-Based Policy Languages

Credential-based access control can be seen as a generalization of a variety of access control models. In (hierarchical) role-based access control (RBAC) [FK92, SCFY96] the decision to grant or deny access to a user is based on the roles that were assigned to her. Clearly, one could encode the roles of a user in a credential, so that RBAC becomes a special case of credential-based access control.

Attribute-based access control (ABAC) [BS02, eXt05, WNR05] comes closer to our concept of credential-based access control, since it grants access based on the attributes of a user. The de facto ABAC standard XACML [eXt05] allows to specify the issuer of an attribute, but does not see them as grouped together in atomic credentials. Moreover, the architecture paradigm is far from privacy-friendly: the user is assumed to provide the policy decision point (PDP) with all her attributes, and lets the PDP decide on basis of his access control policy. The policy that needs to be satisfied is not known to the user, leaving no opportunity for data minimization.

The first proposals that investigate the application of credential-based access control regulating access to a server are done by Winslett et al. [SWW97, WCJS97]. Access control rules are expressed in a logic language, and rules applicable to a service access can be communicated by the server to the clients. A first attempt to provide a uniform framework for attribute-based and credential-based access control is presented by Bonatti and Samarati [BS02]. The language is based on logic expressions and focuses on credential ownership. It does not allow for more advanced requirements such as revealing of attributes or signing statements, however. Besides credentials, this proposal also permits to reason about declarations (i.e., unsigned statements) and profiles of the users that a server can make use of to reach an access decision. The same is true for the language proposed in [ACDS08].

Although the works do not address data and credential typing explicitly, credentials may be organized into a partial order. Several works [WSJ00, IY05, YWS03] describe how trust can be established through the exchange of credentials; our language reuses the same idea by allowing the Data Subject to protect his personal data with a (credential-based) access control policy. The language allows for requiring credentials with certain attribute properties, but does not provide a full syntax. The work of Ni et al. [NLW05] takes the idea of [WSJ00] to cryptographic credentials and defines a grammar for a revised version of the policy language, but does not do so for the conditions imposed on attributes.

Trust-management systems such as Keynote [BFIK98], PolicyMaker [BFL96], REFEREE [CFL 97], DL [LGF00], and others [LMW05, YWS03] use credentials to describe specific delegation of trust among keys and to bind public keys to authorizations. They therefore depart from the traditional separation between authentication and authorizations by granting authorizations directly to keys (bypassing identities).

The languages mentioned above are not targeted to transactions with anonymous credentials and thus lack the ability for expressing conditions for unlinkable yet accountable transactions, which we do achieve through the capability of disclosure to third parties. The first work covering such use cases is due to Backes et al. [BCS05]. The authors of [GD06] extend P3P to allow for describing necessary credential properties for accessing a service, but no precise syntax is specified. The only language featuring a credential typing mechanism and advanced features such as spending restrictions and signing requirements was recently proposed by Ardagna et al. [ACK 09]. This language was focused mainly around anonymous credentials and therefore models concepts that cannot be implemented with other credential technologies. Nevertheless, it served as an important source of inspiration for the work presented here.

In general, the above solutions provide access control languages and solutions that are logic-based, powerful, highly expressive, and permit to specify complex conditions involving credentials and relations between parties in a simple yet effective way. However, in real world scenarios, like the one considered in PrimeLife, fundamental requirements for access control solutions are simplicity and easiness of use, rather than the presence of a complete and highly expressive access control language. Also, although the benefits of all these works (e.g., credential integration), none provides functionalities for protecting privacy of users and regulates the use of their personal information in secondary applications. The work in the PrimeLife WP5.3, instead, is aimed at providing a XACML-based language integrating and supporting credential definition and trust negotiation in the context of a privacy-aware access control system, that protects privacy of users and manages secondary use of data.

The research community has also devoted huge effort in the study and specification of policy languages for regulating access in distributed scenarios. Some works have focused on the definition of privacy-aware languages [eXt05, Web06, AHKS02, W3C02] that support preliminary solutions to the privacy protection issue, as for instance, by providing functionalities for controlling secondary use of data (i.e., how personal information could be managed once collected).

The works closest to the one in PrimeLife and in this heartbeat are represented by PRIME languages [Pri, ACDS08], XACML [eXt05], and P3P [Cra02, W3C02]. PRIME languages represent a good starting point and an inspiration for the work to be done in PrimeLife. XACML represents the most accepted, complete, and flexible solution in terms of access control languages, which could be adapted and integrated with privacy preference-based solutions. P3P represents the most important work done in the context of privacy protection and secondary use management. For the sake of conciseness, we refer the readers interested in more details about these proposals, their pro and cons, to the PrimeLife Deliverable D5.1.1 [Fin09].

1.3.4 Credential-Based Identity Management

We refer to Section 6.2.3 for an overview of the different technologies that are covered under our abstract notion of credentials.

The Simple Public Key Infrastructure (SPKI) [Ell99] has born as a joint effort to overcome the complication and scalability problems of traditional X.509 public key infrastructure [AT02]. SPKI has then been merged with Simple Distributed Security Infrastructure (SDSI) that bounds local names to public keys. The combined SPKI/SDSI allows the naming of principals, creation of named groups of principals, and the delegation of rights or other attributes from one principal to another. SPKI is mainly used in access control and emphasizes decentralization using keys instead of names. Different permissions can be freely defined in SPKI certificates, which can be used as name certificates that provide a mechanism to get public keys according to names or attributes.

Much like a SPKI certificate, an anonymous credential [Cha85, CL01] can be seen as a signed list of attribute-value pairs issued by a trusted issuer. Unlike SPKI certificates, however, they have the advantage that the owner can reveal only a subset of the attributes, or even merely prove that they satisfy some condition, without revealing any more information about the other attributes. Also, they provide additional privacy guarantees like unlinkability, meaning that even with the help of the issuer a server cannot link multiple visits by the same user to each other, or link a visit to the issuing of a credential.

There are two main anonymous credential systems in use today, namely Idemix [CL01, CV02] and UProve [U-P07]. IDEntity MIXer (Idemix) [IDE] is a privacy-enhancing pseudonym-based public key infrastructure. It provides an anonymous credential system that has a number of interesting associated cryptographic tools, such as verifiable encryption [CD00] that allows to prove properties about encrypted values, and limited spending [BCC05] that allows to put restrictions on how often the same credential can be used to access a service, without compromising anonymity.

Beside research on privacy languages, several projects have focused on developing frameworks and architectures that preserve security and privacy of distributed parties communicating among them. These works may be taken as a reference for the development of the infrastructure responsible for evaluating and enforcing the PrimeLife language. International Security, Trust, and Privacy Alliance (ISTPA) [Int] is an open, policy-configurable model including privacy services and capabilities, that can be exploited for designing solutions that cover security, trust, and privacy requirements. The goal of the framework is to provide the base for developing products and services that support current and evolving privacy regulations and business policies.

Reasoning on the Web (REWERSE) [Rea] is a project whose objective is to strengthen Europe in the area of reasoning languages for Web systems and applications. One of its main goals is to enrich the Web

with advanced functionalities for data and service retrieval, composition, and processing. REWERSE's research activities are devoted to several objectives, a part of those might be overlapped with PrimeLife work:rule markup languages aiming at unified markup and tools for reasoning Web languages, policy specification, composition, and conformance aiming at user-friendly high-level specifications for complex Web systems, Web-based decision support for event, temporal, and geographical data aiming at enhancing event, temporal, and location reasoning on the Web.

Enterprise Privacy Architecture (EPA) [KSW02] is an IBM project that wants to improve enterprises e-business trust. The main goal of EPA is to guide organizations in understanding how privacy impacts business processes. To this aim, EPA defines privacy parties, rules, and data for new and existing business processes and provides privacy management controls based on the preferences of the consumer, privacy best practices, and business requirements. TRUSTe [Tru] enables trust, based on privacy protection of personal information on the Internet. It certifies and monitors Web site privacy practices.

Many other projects are focused on providing enhanced identity management system protecting privacy and security of identity information and giving to the users much more power in controlling their private sphere [Lib, Pri, Win]. Liberty Alliance [Lib] project is aimed at providing a networked world based on open standards where consumers, citizens, and governments are able to manage online transactions still protecting the privacy and security of their identity information. In Liberty Alliance, identities are linked by federation and protected by strong authentication. Also Liberty Alliance project tries to address end user privacy and confidentiality concerns, allowing them to store, maintain, and categorize online relationships. Users can manage all of their information using privacy controls built into the system based on Liberty platform.

Windows CardSpace [Win] is an identity management component or identity selector that enables users to provide their digital identity to online services in a simple, secure, and trusted way. Windows CardSpace is based on visual "cards" that have several identity information associated (e.g., name, phone number, e-mail address). These cards are released to the users by identity providers, such as public administration or users themselves, and are used to fulfill the request stated in the Web forms of the service providers. Therefore the users maintain control over the information flow, and are responsible for choosing to release or not information to the online services.

The Higgins Trust Framework [Hig09] intends to address four challenges: the lack of common interfaces to identity/networking systems, the need for interoperability, the need to manage multiple contexts, and the need to respond to regulatory, public or customer requests to implement solutions based on a trusted infrastructure that offers security and privacy. This project is developing an extensible, platform-independent, identity protocol-independent, software framework to support existing

and new applications that give users more convenience, privacy and control over their identity information.

1.3.5 Obligations

The following uses obligation terminology where the "subject" is the subject of the obligation, i.e. the service or Data Controller. Do not confuse with the "Data Subject", which is the user in privacy terminology.

Most of the available policy languages, like XACML [Rissanen ,Moses], EPAL [EPAL], Ponder [Damianou], Rei [Kagal] and PRIME-DHP [Ardagna], provide either only a placeholder or very limited obligation capability. Moreover these languages do not provide any concrete model for obligation specification. XACML and EPAL support system obligations only, i.e. obligation defined and enforced within a single trust domain, as no other subject can be expressed in their proposed language.

Ponder and Rei on the other hand do allow user obligations, however they do not provide a placeholder explicitly for the specification of temporal constraints and they do not support pre-obligations, conditional obligations, and repeating obligations.

PRIME-DHP proposed a new type of policy language which expresses policies as a collection of data handling rules which are defined through a tuple of recipient, action, purpose and conditions. Each rule specifies who can use data, for what purposes and which action can be performed on the data. The language structure limits its expressiveness. PRIME-DHP itself also does not provide any concrete obligation model.

Besides the policy languages, we observed publications on expression, enforcement and formalization of obligations. In the next paragraphs, we collected prior art which is directly related to our approach and point out the key differences to our work.

Mont Casassa et al. [Casassa] proposed the idea of having parametric obligation policies with actions and events having variable parameters. This work was done in conjunction with the PRIME-DHP to support obligations. It is by far the closest work to ours. They propose a formal obligation model and provide the framework to enforce obligations. However, they do not offer the notion of preventive obligations (negative obligations) and multiple subjects. As opposed to their policy expressions, we propose a schema which is not modified when domain specific obligations, including new actions, events and triggers, are added. Unlike [Casassa], we also do not allow multiple actions per rule because of the system integrity problem which arises from the fact that we cannot map fulfillment of a subset of actions in any policy rule as complete fulfillment and we achieve the same behavior through rule cascading without ambiguity.

Irwin et al. [Irwin] proposed a formal model for obligations and define secure states of a system in the presence of obligations. Furthermore,

they focused on evaluating the complexity of checking whether a state is secure. However, the proposed obligation model is very restricted and neither support pre-obligations (provisions) nor repeating and conditional obligations, which are required in different domains and scenarios. They addressed the problem of verification of obligation enforcement while we focus on the expression of a wide range of scenarios, supporting all of the above types of obligations. In other words, the two research efforts are targeting different problems.

Pretschner et al. [Hilty, Pretschner, Schütz] worked in the area of distributed usage control. In [Hilty], they used distributed temporal logics to define a formal model for data protection policies. They differentiated provisional and obligation formulas using temporal operators. Provisions are expressed as formulas which do not contain any future time temporal operators and obligation are formulas having no past time temporal operators. They also addressed the problem of observability of obligations which implies the existence of evidence/ proof that the reference monitor is informed about the fulfillment of obligations. Possible ways of transforming non-observable obligations into observable counterparts have also been discussed. We also consider temporal constraints as an important part of obligation statement. However, we deem observability as an attribute of the reference monitor and not an attribute of the obligation rule. It depends on the scope of the monitor.

The scope could be within the system, within the same trust domain but outside the system, or even sitting outside the trust domain, to observe fulfillment and violations. We currently have not addressed this problem of observability. In [Pretschner] they have proposed an obligation specification language (OSL) for usage control and presented the translation schemes between OSL and rights expressions languages, e.g. XrML, so the OSL expression could be enforced using DRM enforcement mechanisms. We have tried to fill that gap by implementing the enforcement platform for enforcing obligation policies without translation. In [Schütz], the authors have addressed the scenario of policy evolution when the user data crosses multiple trust domains and the sticky policy evolves.

Currently, we are not focusing on evolution of obligation policy, but it could likely be one of the future extensions of our work where we plan to address the interaction of obligation frameworks at multiple services which is complementary to what is discussed in [Schütz].

Katt et al. [Katt] proposed an extended usage control (UCON) model with obligations and gave prototype architecture. They have classified obligations in two dimensions a) system or subject performed and b) controllable or non-controllable where the objects in the obligation would be either controllable or not. Controllable objects are those that are within a target systems domain, while non-controllable objects are outside the systems domain. The enforcement check would not be applied for system-controllable obligations where they assume that since system is a trusted entity so there is no need to check for the fulfillment.

The model does not address the conditional obligations. Rakaiby et al. [Rakaiby] as well as Cholvy et al. [Cholvy] studied the relationship between collective and individual obligations. As opposed to individual obligations which are rather simple as the whole responsibility lies on the subject, collective obligations are targeted toward a group of entities and each member may or may not be responsible to fulfill those obligations. We also consider that the subject of any obligation rule is a complex entity in itself like individual or group, self directed or third party. Our current implementation does not support this but could be extended to include such scenarios.

Ni et al. [Ni] proposed a concrete obligation model which is an extension of PRBAC [Trombetta]. They investigated a different problem of the undesirable interactions between permissions and obligations. The subject is required to perform an obligation but does not have the permissions to do so, or permission conditions are inconsistent with the obligation conditions. They have also proposed two algorithms, one for minimizing invalid permissions and another for comparing the dominance of two obligations. Dominance relation is the relationship between two obligations which implies that fulfillment of one obligation would cover the fulfillment of other which is analogous to set containment.

Gama et al. [Gama] presented an obligation policy platform named Heimdall, which supports the definition and enforcement as a middleware platform residing below the runtime system layer (JVM, .NET CLR) and enforcing obligations independent of application. Opposed to that, we present an obligation framework as an application layer platform in a distributed service-oriented environment which could be used as a standalone business application to cater for user privacy needs. We believe that it is not necessary to have the obligation engine, which is an important infrastructure component to ensure compliant business processes, as part of the middleware. Moreover our service-oriented approach supports interoperability in a heterogeneous system environment.

The work present in this paper incorporates some of the prior art and extends it toward more expressiveness, extensibility, and interoperability. However, we think that some authors addressed different problems, and it would be worthwhile to further combine their results with our approach.

1.4 Contributions of the Proposed Language

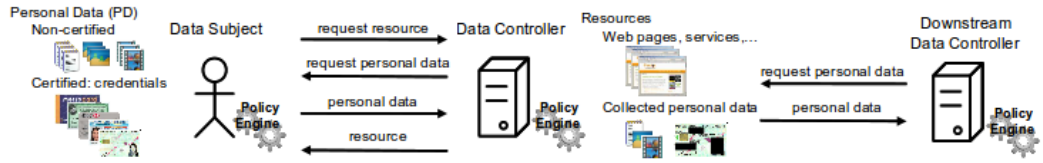


Figure 3: Downstream Usage

The policy language proposed here considers the scenario depicted in Figure 3, where the Data Subject wants to access a resource hosted by the Data Controller, but has to reveal some personal data in order to access the resource. Furthermore, the Data Controller may want to further forward the Data Subject's personal data to a Downstream Data Controller. Our policy language allows the Data Controller to express which personal data he needs from the Data Subject and how he will treat this data, and allows the Data Subject to express to whom she is willing to release her personal data and how she wants her data to be treated.

Our policy language supports the following features that we see as its main contributions over the current state-of-the-art:

- Two-sided data handling policies/preferences with automated matching: Both the Data Controller and the Data Subject can specify in their data handling policies (resp. preferences) how collected personal data will be treated (resp. should be treated). An automated matching procedure detects whether a match can be found between the policies of both sides. Policies and preferences can be specified for explicitly revealed personal data (e.g., name, birth data) as well as data that is implicitly revealed by setting up a connection (e.g., IP address).
- Credential-based access control: The access control conditions can be specified in terms of the credentials that need to be presented. The concept of credentials acts as a useful abstraction for many authenticating technologies, including in particular anonymous credentials.
- Language symmetry: By considering personal data as a special type of resource in its own right, the same language can be used on the Data Subject's side to express to whom and under what conditions she is willing to reveal her data, as on the Data Controller's side to specify which personal data needs to be revealed in order to access a service and how that data will be treated.
- Downstream usage: By exploiting the above symmetry, the Data Subject's personal data can itself become a resource offered by the Data Controller to further Downstream Data Controllers. Our policy language allows the Data Subject to specify to whom and under which such forwarding can take place.

We define extensions to XACML to support all of the above features in the local policies expressed by each of the entities. Moreover, we define extensions to SAML so that it can be used as a wire format to carry resource requests, policies, and credential proof statements from one entity to the other.

2 Introduction to Obligations

We define an obligation as: “A promise made by a Data Controller to a Data Subject in relation to the handling of his/her personal data. The Data Controller is expected to fulfill the promise by executing and/or preventing a specific action after a particular event, e.g. time, and optionally under certain conditions".

Obligations play an important role in daily business. Most companies have a process to collect personally identifiable information (personal data) on customers and ad-hoc mechanisms to keep track of associated authorizations and obligations. State of the art mechanisms to handle collected personal data accordingly to a privacy policy are lacking expressiveness and/or support for cross-domain definition of obligations. Please refer to Section 1.3.5 for a complete evaluation of the state of the art.

We identify four main challenges related to obligations.

1. Service providers must avoid committing to obligations that cannot be enforced. For instance, it is not straightforward to delete data when backup copies do exist. Tools to detect inconsistencies are necessary.
2. Services should offer a way to take user's preferences into account. Preferences may be expressed by ticking check boxes, be a full policy, or even be provided by a trusted third party. Mechanisms to match user's privacy preferences and service's privacy policies are necessary.
3. Services need a way to communicate acceptable obligations to users, to link obligations and personal data, and to enforce obligations.
4. Finally, users need a way to evaluate the trustworthiness of service providers, i.e. know whether the obligation will indeed be enforced. This could be achieved by assuming that misbehavior impacts reputation, by audit and certification mechanisms, and/or by relying on trusted computing.

This report mainly focuses on the first challenge by providing a mechanism to enforce obligations (see Section 3.3.4) and the second aspect by providing mechanisms to match obligations (see Section 3.3.2). The third aspect is also covered in Section 3.3.3. The fourth challenge is addressed by assuming a simple trust model: audit and reputation mechanisms.

2.1 Requirements for Obligations

This section describes general requirements for an obligation language and enforcement framework. This list was mainly compiled by refining requirements related to obligations in D5.1.1 [Fin09], by at scenarios we address in the PrimeLife, and by studying requirements found in related work [ACDS08, Casassa, Irwin, Rissanen, Hilty].

Requirement 1: Independence from policy language. The obligation language must be independent from the “container” policy language, which offers a placeholder for the obligation. Thus the obligation framework should be able to enforce obligations defined by the service, e.g. XACML [Rissanen] or P3P [P3P], and obligations in sticky policies, e.g. PRIME-DHP [Ardagna]. Different serialization of the obligation language may be required.

Requirement 2: Independence from data storage. The obligation handling must be independent from the concrete data store. The obligation travels with the data and should be stored along with the data so that the reference does not get lost. For instance, the obligation may refer to personal data stored in a database or to documents in a file system. Moreover, one obligation may refer to multiple pieces of data.

Requirement 3: Independence from communication protocols. The framework must be independent from the communication protocol. Web Services, REST, or plain HTTP may be used to exchange data and obligations.

Requirement 4: Support for common obligations. The obligation language shall NOT be empty. Usual actions such as, for instance, delete, anonymize, notify user, get approval from user, log should be available with different implementations. The language should support time-based and event-based triggers.

Requirement 5: Support for domain specific obligations. The framework must be extensible and open to define additional domain specific obligations. Mechanisms to define new types of actions and new types of triggers shall be available. Naturally, the semantics of those new elements must be understood by all stakeholders.

Requirement 6: Support for abstraction of actions. The obligation language must offer abstract actions which must be mapped to concrete actions on underlying systems during enforcement. For instance, a “notify user” action may be implemented as sending an e-mail, sending an SMS, sending a voice message, or calling (and authenticating to) a Web Service.

Requirement 7: Support for abstraction of triggers. The obligation language must offer abstract and configurable triggers. For instance, a trigger “access personal data” may react both to a query on a database and to a read operation on a file server.

Requirement 8: Support for distributed deployment. Various deployments of the obligation framework may be envisioned: a corporate-wide obligation framework may cover multiple databases, a desktop obligation framework may deal with local files, or the obligation framework may even be provided as a “cloud service”. In any case, a specific piece of data must be handled by one and only one obligation framework.

Requirement 9: Support for different trust models. When obligation enforcement is not observable by data subjects, they have to trust the service provider, i.e. assume that it will fulfill obligations. The anchor of trust could be based on various technologies, e.g. a trusted stack (certified TPM [TCPA], trusted OS), reputation, or certification by external auditors. The obligation language should be independent from the trust model.

Requirement 10: Transparency of data handling. The obligation enforcement as well as mechanisms to load policies should be comprehensive so that data processors and auditors can easily check whether a specific deployment is compliant with a given specification. This is a prerequisite to enable data-handling transparency toward end users.

Requirement 11: Support for preventive obligations. The obligation language may be able to express preventive statements that forbid the execution of an action. For instance, the obligation to store logs for six months forbids the deletion of log files.

Requirement 12: Matching obligations: It must be possible to check efficiently whether an obligation is enforced by another obligation. For instance, “Notify user at least once a month” is enforced by “Notify the user by e-mail at least once a week”.

3 Specifying Obligations: an Obligation Language

This section describes the language used to specify obligations as triggers and actions. XML schema, i.e. placeholders for triggers and actions, will be provided.

3.1 Different Aspects of Obligations

It is close to impossible to develop an exhaustive list of obligation statements existing in the real world. We could encounter many complex forms of commitments made between individuals, organizations and logical entities. Additionally, the semantics of these commitment and promises may be different in different domains like healthcare, financial services etc. This section identifies and classifies promise and obligation statements. Many aspects have been discussed in existing literature:

Positive versus Negative Statements: A key aspect of obligations is the tone of statement. We could have positive obligations where the commitment is stated in a positive tone. For instance, in “Hospital X commits to delete patient's history in 1 month” and in “Hospital X commits to notify patient whenever his information is shared”, the Data Controller (Hospital X) commits to positive obligation statements. Obligations like “Hospital X commits not to share patient's history to anyone” or “Hospital X commits not to use patient's history for any statistical purposes” are negative statements.

Enforcement of negative statements, i.e. prevention of an action, is quite different than enforcement of positive statements, i.e. execution of an action.

Conditionality: It is very usual to have some form of constraints defined with the obligation statements. If the conditions are not fulfilled, the Data Controller is not held liable for not fulfilling its promises. For instance, “Hospital X commits to notify patient whenever his information is shared if patient is registered at the hospital”.

Iteration: The third aspect is the iterative nature of obligation statements. The simplest forms of positive obligations are fulfilled once. However, we could encounter statements which are required to be fulfilled multiple times iteratively. When the fulfillment of an obligation is required multiple times during its life then we consider them as iterative or repeating obligations. For instance, “Bank X commits to send account

statement to customer C every 3 months" will be triggered every 3 months.

Stateful Obligations: Another important aspect of obligation statement is their stateful nature. In existing literature, we mostly found that obligations are either fulfilled or violated which are two atomic states. However, more complex states could be required. For instance, "Hospital Y commits to share patient data only 3 times" requires a counter to keep track of disclosure.

Time Boundary: The life cycle of obligations may be complex. We generally assume that obligations end when they are fulfilled, when data retention ends, or at specific date.

Observability: Pretschner et al [Hilty, Pretschner] discussed the problem of observability of obligations and suggested methods to translate non-observable obligations into observable ones. However, we perceive that observability is not only dependent on the rule itself but highly related to the authority and scope of the evaluator and monitor. An external evaluator can only monitor the communication channels external to the Data Controller trust domain. Internal evaluator can also monitor the internal communication between obligation enforcement platform and other infrastructure of the Data Controller. Lastly the monitor inside the framework is the one which could log audit trail and monitor the actual processing of the enforcement framework.

Delegation: Delegation of obligations could be another interesting area for extension. The basic idea is to allow constructs in the language which enable the policy issuers to commit promises/obligations on behalf of other Data Controllers. Complementary to the problem of delegation of obligations is the proportion of responsibility in the cases of collective controllers.

3.2 Definition and Schema of an Obligation Language

Based on the different aspects of obligations, in this section we define the structure of an obligation. This is a first version of language structure that is subject to modifications and refinement during development of the enforcement and matching engines. An obligation is often defined as Event-Condition-Action:

On ***Event*** If ***Condition*** Do ***Action***

For facilitating the comparison of obligations, we consider triggers as events filtered by conditions. In other words, we replace the notions of events and conditions by trigger. The triggers are events that are considered by an obligation and can be seen as the set of events that result in actions. Additionally, in order to simplify obligations management, we specify a validity period for each obligation:

Do ***Action*** when ***Trigger*** (from Start to End)

For instance, we can define “Do {Notify User} when {User’s personal data is read} (from .. to ..)”. Obligations are thus defined as a set of triggers, an action, and a validity period. More details can be found in the schema (see Figure 4).

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:ob="http://www.primelife.eu/obligation"
        elementFormDefault="qualified"
        targetNamespace="http://www.primelife.eu/obligation"
        xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!-- List of Obligations -->
  <xs:element name="ObligationsSet" type="ob:ObligationsSet" />
  <xs:complexType name="ObligationsSet">
    <xs:sequence>
      <xs:element name="Obligation" type="ob:Obligation"
         minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
  <!-- Obligation -->
  <xs:complexType name="Obligation">
    <xs:sequence>
      <xs:element name="TriggersSet" type="ob:TriggersSet"
          minOccurs="1" maxOccurs="1" />
      <xs:element name="Action"    type="ob:Action"
          minOccurs="1" maxOccurs="1" />
      <xs:element name="Validity" type="ob:Validity"
          minOccurs="1" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
  <!-- List of Triggers -->
  <xs:complexType name="TriggersSet">
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:any namespace="##any" processContents="lax"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="optional"/>
  </xs:complexType>
  <!-- Action -->
  <xs:complexType name="Action">
    <xs:sequence minOccurs="1" maxOccurs="1">
      <xs:any namespace="##any" processContents="lax"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="optional"/>
  </xs:complexType>
  <!-- Validity -->
  <xs:complexType name="Validity">
    <xs:sequence>
      <xs:element name="start" type="xs:dateTime"
          minOccurs="1" maxOccurs="1" />
      <xs:element name="end" type="xs:dateTime"
          minOccurs="1" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Figure 4: Draft XML Schema for obligations

Figure 5 shows an example of XML serialization of a policy: “WILL delete personal data within 30 days from 01/01/2010”. We assume a schema to specify such triggers and actions as defined in Figure 4.

```
<?xml version="1.0" encoding="utf-8" ?>
<ObligationsSet xmlns="http://www.primelife.eu/obligation"
         xmlns:ext="http://www.primelife.eu/obligation/extension"
         xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <Obligation>
    <TriggersSet>
```

```
      <ext:TriggerAtTime>
        <ext:start> 2010-01-01T00:00:00 </ext:start>
        <ext:maxDelay> P30D </ext:maxDelay>
      </ext:TriggerAtTime>
    </TriggersSet>
    <Action>
      <ext:ActionDeletePersonalData>
        <ext:personalData> "ref to personal data" </ext:personalData>
      </ext:ActionDeletePersonalData>
    </Action>
    <Validity>
      <start> 2000-01-01T00:00:00 </start>
      <end> 2020-01-01T00:00:00 </end>
    </Validity>
  </Obligation>
</ObligationsSet>
```

Figure 5: Draft Sample obligation

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:ext="http://www.primelife.eu/obligation/extension"
   elementFormDefault="qualified"
   targetNamespace="http://www.primelife.eu/obligation/extension"
   xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!-- ActionDeletePersonalData -->
  <xs:element name="ActionDeletePersonalData"
      type="ext:ActionDeletePersonalData" />
  <xs:complexType name="ActionDeletePersonalData">
    <xs:sequence>
      <xs:element name="personal data" type="xs:string"
          minOccurs="1" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
  <!-- TriggerAtTime -->
  <xs:element name="TriggerAtTime" type="ext:TriggerAtTime" />
  <xs:complexType name="TriggerAtTime">
    <xs:sequence>
      <xs:element name="start" type="xs:dateTime"
          minOccurs="1" maxOccurs="1" />
      <xs:element name="maxDelay" type="xs:duration"
          minOccurs="1" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Figure 6: Sample XML schema to specify Triggers and Actions

The language and schema for defining obligation may be slightly different to express obligations in Data Controller's privacy policy, in Data Subject's privacy preferences, and in sticky policies.

- Data Subject's privacy preferences specify "required obligations", i.e. what the Data Subject requires in terms of obligation to provide a given piece of personal data to a given Data Controller.
- Data Controller's privacy policy specifies "proposed obligations", i.e. what the Data Controller is willing (and able) to enforce in terms of obligation for a given collected data.
- Sticky policy specifies "committed obligations", i.e. the obligations Data Subject and Data Controller agreed upon and that must be enforced by the Data Controller.

Further work is necessary to decide whether multiple dialects are required to specify obligations embedded in policies, preferences, and sticky policies.

3.3 Matching Obligations

This section describes how Data Controller's proposed obligations (part of its policy) are matched with Data Subject's required obligations (part of his preferences). Matching is generally done by the Data Subject but, depending on the trust model, may occur at Data Controller-side.

Note that the same mechanism is used by Data Controller in order to decide whether a collected piece of data can be shared with a third party (downstream data controller). In this case, the Data Controller matches the proposed obligation of the downstream Data Controller (part of its policy) with committed obligations (part of sticky policy) attached to a piece of personal data.

3.3.1 Obligation Matching Engine

The obligation matching engine is in charge of deciding whether an obligation is less permissive than another one. In other words, it checks whether enforcing the first will ensure that the second one is not violated. For instance we could define that deleting a piece of personal data is less permissive than anonymizing this same piece of data. Indeed, a data controller not having access to a given personal data is more privacy-friendly than a data controller having access to an anonymized version of this data. Similarly, we could define that notifying the data subject each month is less permissive than notifying the user each year. Notifying the user once a month instead of once a year may be considered as spam. We consider this as a valid enforcement because, for the moment, we do not specify the authorization of enforcing obligations. The obligation engine requires the following inputs:

- Obligation policy: the language to specify obligations in policies (XML schema and/or Domain Specific Language), the required extensions (additional schema and/or DSL extension), an instance of those languages, i.e. the obligation itself.
- Obligation preference: the language to specify obligations in preferences (XML schema and/or DSL), the required extensions (additional schema and/or DSL extension), an instance of those languages, i.e. the obligation itself.
- Matching rules: the language to specify matching rules (XML schema and/or DSL), the required extensions (additional schema and/or DSL extension) corresponding to the policy and preferences to match, instances of those languages, i.e. the set of rules to be used.

The output is a Boolean response, i.e. "True" when the obligation policy is less permissive than the obligation preference or "False" when the

obligation policy is not less permissive than the obligation preference (this does not mean that the obligation preference is less permissive than the obligation policy). Optionally, when the policy does not match the preferences, more details on the mismatch may be provided.

3.3.2 Matching Rules

Comparing service's privacy policy P_S with user's privacy preference P_U is necessary to decide whether users can provide personally identifiable information to service. We express the fact that policy P_S is less (or equally) permissive than preference P_U as $P_S \trianglelefteq P_U$. This intuitively means that P_S provides less (or equal) authorizations than P_U and that P_S defines more (or equal) obligations than P_U. Note that $P_S \not\trianglelefteq P_U$ does not imply $P_U \trianglelefteq P_S$. We define a service data handling policy as the set of authorizations and a set of obligations. We define PS ⊴ PU as following:

> L:Policy ⊴ R:Policy ⇔ ((L.authorizations ⊴ R.authorizations) ∧ (L.obligations ⊴ R.obligations))

NB: This can be read as left-side policy (L) is less (or equally) permissive than right-side policy (R) if and only if the set of authorizations specified in the left-side policy (L.authorizations) is less (or equally) permissive as the set of authorizations specified in the right-side policy (R.authorizations) and the set of obligations specified in the left-side policy (L.obligations) is less (or equally) permissive as the set of obligations specified in the right-side policy (R.obligations). The meaning of less permissive for a set of authorizations and obligations is defined below.

Where “authorizations” is a list of authorizations and “obligations” is a list of obligations. This means that a policy (e.g. service policy P_S) is less restrictive than another policy (e.g. user preferences P_U) when the list of rights and the list of obligation are more restrictive. Matching function for lists of rights is:

> L:ListAuthorizations ⊴ R:ListAuthorizations ⇔ ∀ (i ∈ L) : ∃ (j ∈ R) where (i ⊴ j)

This means that for each authorization in the policy, there exists a more permissive authorization in the preferences.

Matching function for obligations is quite different:

> L:ListObligations ⊴ R:ListObligations ⇔ ∀ (j ∈ R) : ∃ (i ∈ L) where (i ⊴ j)

This means that for each obligation in the preference, there exists a less permissive obligation in the policy. Obligations are compared as following:

> L:Obligation ⊴ R:Obligation ⇔ (((L.action ⊴ R.action) ∧ (L.triggers ⊴ R.triggers)) ∧ (L.validity ⊴ R.validity))

Where “action” is the action resulting from the obligation, “triggers” is the list of triggers resulting in the execution of the action, and “validity” is the validity period of the obligation. Validities are compared as following:

$$L{:}Validity \trianglelefteq R{:}Validity \Leftrightarrow ((L.start \leq R.start) \wedge (L.end \geq R.end))$$

Matching function for a list of triggers is:

$$L{:}ListTriggers \trianglelefteq R{:}ListTriggers \Leftrightarrow \forall (b \in R) : \exists (a \in L) \text{ where } (a \trianglelefteq b)$$

In other words, for a given obligation, all triggers in the preferences must be in the policy, but the policy can specify other triggers. The matching functions for actions and for triggers are not yet define and must remain open to future extensions. As an example, we can define the following rules to match triggers:

$$L{:}TriggerAtTime \trianglelefteq R{:}TriggerAtTime \Leftrightarrow ((L.start \geq R.start) \wedge (L.start + L.maxDelay \leq R.start + R.maxDelay))$$

$$L{:}TriggerReadData \trianglelefteq R{:}TriggerReadData \Leftrightarrow (L.dataRef == R.dataRef)$$

$$L{:}TriggerReadForPurpose \trianglelefteq R{:}TriggerReadForPurpose \Leftrightarrow ((L.dataRef == R.dataRef) \wedge (L.purpose == R.purpose))$$

$$L{:}\ TriggerReadData \trianglelefteq R{:}TriggerReadForPurpose \Leftrightarrow (L.dataRef == R.dataRef)$$

And the following rules to match actions:

$$L{:}ActionDeletePersonalData \trianglelefteq R{:}ActionDeletePersonalData \Leftrightarrow (L.personalData == R.personalData)$$

$$L{:}ActionNotifyUser \trianglelefteq R{:}ActionNotifyUser \Leftrightarrow (L.userName == R.userName)$$

$$L{:}ActionNotifyByEmail \trianglelefteq R{:}ActionNotifyByEmail \Leftrightarrow ((L.userName == R.userName) \wedge (L.address == R.address))$$

$$L{:}ActionNotifyByEmail \trianglelefteq R{:}ActionNotifyUser \Leftrightarrow (L.userName == R.userName)$$

We are working on a small set of generic actions and generic triggers that can be combined and parameterized to cover most cases. It is however clear that supporting an exhaustive list of obligations is not realistic. We propose to enrich the framework with domain-specific obligations based on domain-specific actions and/or triggers. For instance, a healthcare-specific action could be “notify user’s doctor”. When specifying a new trigger or a new action, it is also necessary to define the corresponding matching rules. Depending on the

implementation, those rules could be directly interpreted by the matching engine or could be pre-defined, e.g. using a plug-in mechanism.

3.3.3 Sample Obligations

With the set of rules specified above, it is possible to compare the following preferences and policies:

	Policy	**Preferences**
High-level DSL	WILL delete “ref to personal data” within 30 days from 01/01/2010 WILL notify “Bob” at “bob@contoso.com” when “ref to personal data” is accessed	must delete "ref to personal data" within 365 days must notify "Bob" when "ref to personal data" is accessed for "statistics"
Detailed DSL (similar to XML serialization)	`Policy { authorizations : ListAuthorizations { }, obligations : ListObligations { n0 : Obligation { triggers : ListTriggers { t1 : TriggerAtTime { start : 01/01/2010, maxDelay : 30 days } }, action : ActionDeletePersonalData { personalData : "ref to personal data" }, validity : Validity { start : 9/14/2009, end : 9/14/2009 } }, n1 : Obligation { triggers : ListTriggers { t1 : TriggerReadData { dataRef : "ref to personal data" } }, action : ActionNotifyByEmail { userName : "Bob", address : "bob@contoso.com" }, validity : Validity { start : 9/14/2009, end : 9/14/2009 } } } }`	`Policy { authorizations : ListAuthorizations { }, obligations : ListObligations { n0 : Obligation { triggers : ListTriggers { t1 : TriggerAtTime { start : 10/14/2009, maxDelay : 365 days } }, action : ActionDeletePersonalData { personalData : "ref to personal data" }, validity : Validity { start : 9/14/2009, end : 9/14/2010} }, n1 : Obligation { triggers : ListTriggers { t1 : TriggerReadForPurpose { dataRef : "ref to personal data", purpose : "statistics" } }, action : ActionNotifyUser { userName : "Bob" }, validity : Validity { start : 9/14/2009, end : 9/14/2010 } } } }`

Figure 7: Sample matching obligations

In this example, Policy ⊴ Preference is true.

3.3.4 Enforcing Obligations

This section describes mechanism in place at Data Controller side to make sure that committed obligations (e.g. part of a sticky policy) are indeed enforced. More information on proposed obligation enforcement framework can be found in [20].

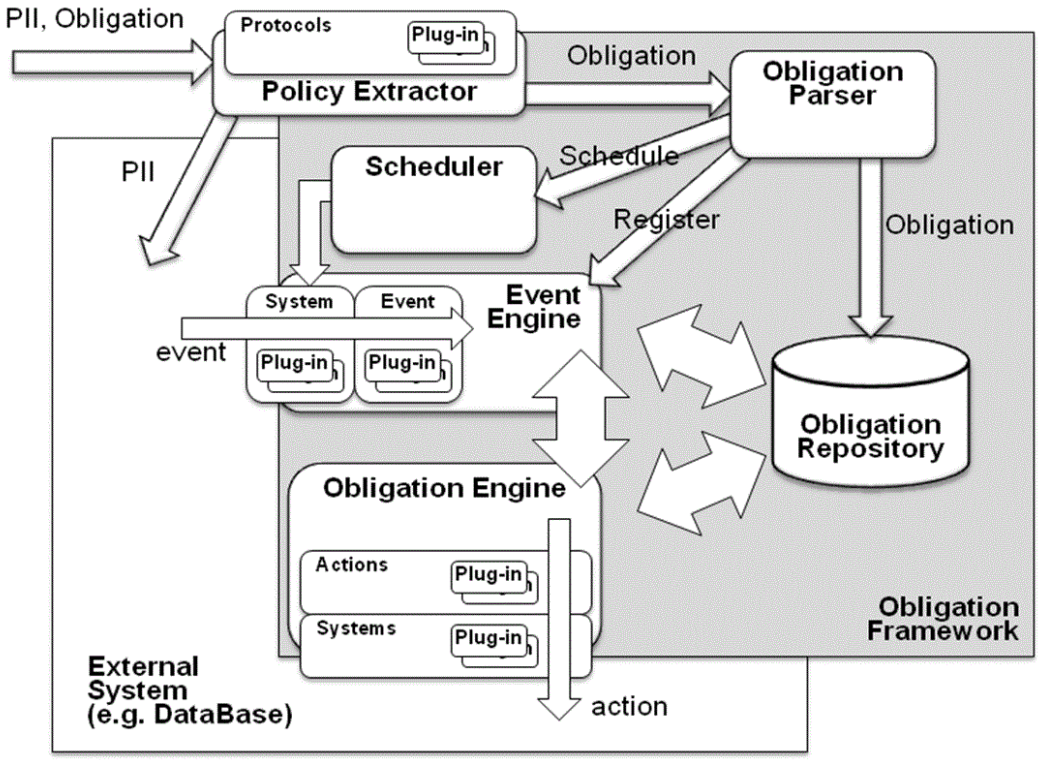


Figure 8: Sample matching obligations

Figure 8 gives an overview on the framework enforcing obligations. The key feature of this framework is its extensibility, which is achieved through plug-ins for triggers and actions. The framework relies on the following components:

Policy Extractor and plug-ins

The policy extractor is in charge of storing personal data in an external system and make sure that a reference to the personal data is part of the policy. Since the structure of incoming message may depend on the protocol, different plug-ins are used. When the obligation policy is embedded within a container message, the corresponding plug-in parses the message and forwards only the obligation policy part to the system.

Obligation Parser

The obligation parser is in charge of deserializing obligations, checking inconsistencies, scheduling deterministic triggers, registering to events, and storing the received obligations.

Scheduler

The scheduler is used to send events required by time-based triggers, which are scheduled by other components of the obligation enforcement framework.

Obligation Repository

The repository stores obligations. It may be a dedicated component or part of the policy repository that stores sticky policies.

Event Engine

Is in charge of triggering actions necessary to enforce obligations. It consumes internal events (e.g. from scheduler) and external events (e.g. read access on personal data). The major goal to have a single point of event receiving and distribution is to ensure integrity. All the external systems, scheduler and obligation engine communicate through the event engine. It behaves mainly like a queuing component keeping track of the received and processed messages. It ensures message reliability in case of system shut down or malfunctions.

Obligation Engine

The obligation engine is the main load processing component. It is triggered by the event engine and processes corresponding obligation rules. After the execution of actions, the obligation engine may change the state of the obligation rule and may throw outward events. The obligations contain actions with parameters attached to each obligation rule. Each of these actions must match to an available action plug-in within the obligation engine.

In the obligation engine component, we propose a two-layer action plug-in mechanism. The upper layer contains the plug-ins for specific actions e.g. delete, notify and the lower plug-in layer contains the implementations for different external systems supporting a set of actions. For instance, delete operation can operate on files or on data in a relational database. Notification to user could be sent via e-mail, fax, or postal mail.

The obligation enforcement engine may be extended with audit features. Keeping track of actions executed by the obligation enforcement engine would facilitate the work of Data Controller and external auditors by enabling automatic analysis of traces and check for conformance with policies.

3.4 Usual Obligation Triggers

This section briefly describes usual triggers. This is a *first draft that requires further refinement and validation*. Even if some aspects are underspecified, we hope that providing this draft specification helps with understanding what we are aiming at.

3.4.1 Trigger at Time

Name	TriggerAtTime		
Parameters	dateTime	start	Start time
	duration	maxDelay	Maximum delay before execution
Description	Time-based trigger that occurs only once between start and start + maxDelay		
Examples	Within x → TriggerAtTime(Now, x) Within x hours → TriggerAtTime(Now, x hours) Within x days → TriggerAtTime(Now, x days) Within x months → TriggerAtTime(Now, 30 * x days) Within x days from y → TriggerAtTime(y, x days) Between x and y → TriggerAtTime(x, y-x)		
Matching	L:TriggerAtTime ⊴ R:TriggerAtTime ⇔ ((L.start ≥ R.start) ⋀ (L.start + L.maxDelay ≤ R.start + R.maxDelay)) i.e. such a trigger is less permissive when it defines events within a shorter time window.		

3.4.2 Trigger Periodic

Name	TriggerPeriodic		
Parameters	dateTime	start	Validity start
	dateTime	End	Validity end
	duration	maxDelay	Maximum response time (maxDelay ≤ period)
	duration	Period	Periodicity
Description	Time-based trigger that occurs multiple times on a periodic basis between start and end.		
Examples	Once a day → TriggerPeriodic (Today at 00:00, dataEnd, 1 day, 1 day) Once a month → TriggerPeriodic (1st day of this month at 00:00, dataEnd, 30 days, 30 days) Once a day at noon ± 1 hour → TriggerPeriodic (Today at 11:00AM, dataEnd, 2 hour, 1 day) At beginning of each year → TriggerPeriodic (1st day of this year at 00:00, dataEnd, 7 days, 365 days)		
Matching	L:TriggerPeriodic ⊴ R:TriggerPeriodic ⇔ (L.start ≤ R.start) ⋀ (L.end ≥ R.end) ⋀ (R.period mod L.period == 0) ⋀ ((1 + round((R.start-L.start) / L.period)) * L.period + L.start ≥ R.start) ⋀ ((1 + round((R.start-L.start) / L.period)) * L.period + L.start +L.maxDelay ≤ R.start + R.maxDelay) i.e. such a trigger is less permissive if it reacts at least as often within a shorter time window.		

3.4.3 Trigger Personal Data Accessed for Purpose

Name	TriggerPersonalDataAccessedForPurpose		
Parameters	DataRef	personal data	Reference to the personal data concerned by the obligation
	Purpose[]	Purposes	Set of purposes that trigger the obligation. Any represents all possible purposes.
	duration	maxDelay	Maximum response time.
Description	Event-based trigger. This trigger occurs each time the personal data associated with the obligation is accessed for one of the specified purposes.		
Examples	When reading x → TriggerPersonalDataAccessedForPurpose(x, any, default) Within x hours after reading y → TriggerPersonalDataAccessedForPurpose(y, any, x hours) When reading x for purpose z → TriggerPersonalDataAccessedForPurpose(x, z, default) When reading x for purposes a, b, and c → TriggerPersonalDataAccessedForPurpose(x, {a,b,c}, default)		
Matching	L:TriggerPersonalDataAccessedForPurpose ⊴ R:TriggerPersonalDataAccessedForPurpose ⇔ ((L.personalData ⊇ R.personalData) ⋀ (L.maxDelay ≤ R.maxDelay) ⋀ (L.purposes ⊇ R.purposes)) i.e. such a trigger is less permissive if it reacts faster and on at least as much types of access.		

3.4.4 Trigger Personal Data Deleted

Name	TriggerPersonalDataDeleted		
Parameters	DataRef	personal data	Reference to the personal data concerned by the obligation
	Duration	maxDelay	Maximum response time.
Description	Event-based trigger. This trigger occurs when the personal data associated with the obligation is deleted.		
Examples	When deleting x → TriggerPersonalDataDeleted(x, default) Within x hours after deleting y → TriggerPersonaDataDeleted(y, x hours)		
Matching	:TriggerPersonalDataDeleted ⊴ R:TriggerPersonalDataDeleted ⇔ ((L.personalData ⊇ R.personalData) ⋀ (L.maxDelay ≤ R.maxDelay)) i.e. such a trigger is less permissive if it reacts faster and on at least as much deletions		

3.4.5 Trigger Personal Data Sent

Name	TriggerPersonalDataSent		
Parameters	DataRef	personal data	Reference to the personal data concerned by the obligation
	Id[]	Targets	List of third parties the PersonalData is sent to. Any represents all possible third parties.
	Duration	maxDelay	Maximum response time.
Description	Event-based trigger. This trigger occurs when the PersonalData associated with the obligation is shared with a third party (downstream Data Controller).		
Examples	When sending x → TriggerPersonalDataSent(x, any, default) When sending x to y → TriggerPersonalDataSent(x, y, default) Within x hours after sending y → TriggerPersonalDataSent(y, any, x hours)		
Matching	L:TriggerPersonalDataSent ⊴ R:TriggerPersonalDataSent ⇔ ((L.personalData ⊇ R.personalData) ⋀ (L.targets ⊇ R.targets) ⋀ (L.maxDelay ≤ R.maxDelay)) i.e. such a trigger is less permissive if it reacts faster and on at least as much data sharing.		

3.4.6 Trigger Data Subject Access

Name	TriggerDataSubjectAccess		
Parameters	DataRef	personal data	Reference to the personal data concerned by the obligation
	url	Targets	Endpoint to access data.
Description	Event-based trigger. This trigger occurs when the Data Subject tries to access its own personal data that has been collected by the Data Controller.		
Examples	When accessing x → TriggerDataSubjectAccess(x, any) When accessing x at y → TriggerDataSubjectAccess(x, y)		
Matching	L:TriggerDataSubjectAccess ⊴ R:TriggerDataSubjectAccess ⇔ ((L.personalData == R.personalData) ⋀ (L.url ⊆ R.url))		

3.4.7 Trigger Data Lost

Name	TriggerDataLost		
Parameters	Duration	maxDelay	Maximum response time.
Description	Event-based trigger. This trigger occurs when the Data Controller has lost control on collected data (e.g. data controller’s data base is compromised by insiders or outsiders).		
Examples	Within 1 day after data leakage → TriggerDataLost (1 day)		
Matching	L:TriggerDataLost ⊴ R:TriggerDataLost ⇔ (L.maxDelay ≤ R.maxDelay) i.e. such a trigger is less permissive if it reacts faster		

3.4.8 Trigger On Violation

Name	TriggerOnViolation		
Parameters	Duration	maxDelay	Maximum response time.
	Obligation[]	obligations	List of obligations.
Description	Event-based trigger. This trigger occurs when an obligation is violated (e.g. not fulfilled on due time).		
Examples	Within 1 day after violation → TriggerOnViolation(1 day, any) After violation of x → TriggerOnViolation(default, x)		
Matching	L:TriggerOnViolation ⊴ R:TriggerOnViolation ⇔ ((L.maxDelay ≤ R.maxDelay) ⋀ (L.obligations ⊇ R. obligations)) i.e. such a trigger is less permissive if it reacts faster and covers at least as many violations.		

The set of triggers is extensible amd more will be defined in future together with the rules to match them.

3.5 Usual Obligation Actions

This section briefly describes usual actions. This is a *first draft that requires further refinement and validation*. Even if some aspects are underspecified, we hope that providing this draft specification helps with understanding what we are aiming at.

3.5.1 Action Delete Personal Data:

Name	ActionDeletePersonalData		
Parameters	DataRef	personal data	Reference to the personal data to delete.
Description	This action deletes a specific piece of information, and is intended for handling data retention.		
Examples	Delete x → ActionDeletePersonalData(x)		
Matching	L:ActionDeletePersonalData ⊴ R:ActionDeletePersonalData ⇔ (L.personalData ⊇ R.personalData) i.e. such an action is less permissive if it results in deleting at least as much data.		

3.5.2 Action Anonymize Personal Data

Name	ActionAnonymizePersonalData		
Parameters	DataRef	personal data	Reference to the personal data to anonymize.
Description	This action anonymizes a specific piece of information.		
Examples	Anonymize x → ActionAnonymizePersonalData(x)		
Matching	L:ActionAnonymizePersonalData ⊴ R:ActionAnonymizePersonalData ⇔ (L.personalData ⊇ R.personalData) And relationship between delete personal data and anonymize personal data L: ActionDeletePersonalData ⊴ R:ActionAnonymizePersonalData ⇔ (L.personalData ⊇ R.personalData)		

3.5.3 Action Notify Data Subject

Name	ActionNotifyDataSubject		
Parameters	Media	media	The media used to notify the user (e-mail, SMS, etc.)
	Address	address	The corresponding address (e-mail address, phone number, etc.)

Name	ActionNotifyDataSubject
Description	This action notifies the Data Subject when triggered, i.e. send the trigger information to the Data Subject.
Examples	Notify by e-mail → ActionNotifyDataSubject(e-mail, any) Notify by e-mail at x → ActionNotifyDataSubject (e-mail, x)
Matching	L:ActionNotifyDataSubject ⊴ R:ActionNotifyDataSubject ⇔ ((L.media == R. media) ⋀ (L.address == R. address))

3.5.4 Action Log

Name	ActionLog		
Parameters			
Description	This information logs an event, e.g. write in a trace file the trigger information.		
Examples	Log → ActionLog()		
Matching	L: ActionLog ⊴ R: ActionLog ⇔ True		

3.5.5 Action Secure Log

Name	ActionSecureLog		
Parameters			
Description	This information logs an event and ensures integrity and authentication of origin of the event.		
Examples	Log Securely → ActionSecureLog()		
Matching	L: ActionSecureLog ⊴ R: ActionSecureLog ⇔ True And relationship between Log and Secure Log L: ActionSecureLog ⊴ R: ActionLog ⇔ True		

3.5.6 Action Give Access to Personal Data

Name	ActionGiveAccessToPersonalData		
Parameters	DataRef	personal data	Reference to the personal data to be accessed
Description	This action lets a party access a specific piece of information.		
Examples	Access x → ActionGiveAccessToPersonalData(x)		
Matching	L:ActionGiveAccessToPersonalData ⊴ R:ActionGiveAccessToPersonalData ⇔ (L.personalData ⊇ R.personalData)		

The set of actions is extensible amd more will be defined in future.

4 Specifying Authorizations

Data handling policies, preferences, and sticky policies contain, apart from the set of obligations described above, also a set of authorizations. While obligations specify actions that the Data Controller is required to perform on the transmitted information, authorizations specify actions that it is allowed to perform. Similarly to what we did for obligations, we recognize that it is impossible to define an exhaustive list of authorizations that covers all needs that may ever arise in the real world. Rather, we define a generic, user-extensible structure for authorizations so that new, possibly industry-specific authorization vocabularies can be added later on. We do provide however a basic authorization vocabulary for using data for certain purposes and for downstream access control, and we describe how these authorizations can be efficiently matched via the general strategy described in Section 3.3.

4.1 Generic Definition and Schema

The set of authorizations in the Data Controller's data handling policy, in the Data Subject's data handling preferences, or in the agreed-upon sticky policy are specified in an <AuthorizationsSet> element containing a list of <Authorization> elements. The <Authorization> element is abstract, however: it is up to the authorization vocabularies to extend the schema with new elements of type AuthorizationType. The schema of the elements is given below.

```
<xs:schema xmlns="http://www.primelife.eu"
    targetNamespace="http://www.primelife.eu"
    xmlns:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:cd-1"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:import schemaLocation="xacml-core-v3-schema-cd-1.xsd"
  namespace="urn:oasis:names:tc:xacml:3.0:core:schema:cd-1"/>

  <!-- List of Authorizations -->
  <xs:element name="AuthorizationsSet" type="AuthorizationsSetType"/>
  <xs:complexType name="AuthorizationsSetType">
    <xs:sequence>
      <xs:element ref="Authorization" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <!-- Authorization -->
  <xs:element name="Authorization" type="AuthorizationType" abstract="true"/>
  <xs:complexType name="AuthorizationType">
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:any namespace="##any" processContents="lax" />
    </xs:sequence>
    <xs:anyAttribute />
```

```
  </xs:complexType>
</xs:schema>
```

4.2 Usage Purposes Authorization

The first concrete authorization type that we define is the authorization to use information for a particular set of purposes. Purposes are referred to by standard URIs specified in agreed-upon vocabularies of usage purposes. These vocabularies of URIs may be organized as flat lists or as hierarchical OWL ontologies. The <UseForPurpose> element contains a list of <Purpose> elements, each containing one URI describing a purpose for which the data can be used.

```
<!-- Authorization: Use for purpose -->
<xs:element name="Purpose" type="xs:anyURI"/>
<xs:element name="AuthzUseForPurpose" substitutionGroup="Authorization">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="AuthorizationType">
        <xs:sequence minOccurs="1" maxOccurs="unbounded">
          <xs:element ref="Purpose"/>
        </xs:sequence>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

As the URIs are organized as a flat list, matching of purposes is simply done by testing equality of the URIs. If purposes occur in a hierarchical OWL ontology, we use the notation purpose1 ⊆ purpose2 to denote that purpose1 is equal to purpose2 or a sub-purpose of purpose2 (i.e., a descendant of purpose2 in the hierarchy defined by the OWL ontology). In the matching procedure described below we employ the hierarchical matching operator "⊆", but it is understood to be replaced with a URI equality operator for purposes that are organized in a flat list.

Name	**AuthzUseForPurpose**
Parameters	{anyURI}
Description	Authorization to use the data for the specified list of purposes.
Examples	Use for admin and telemarketing purposes as defined in P3P 1.0 ➔ AuthzUseForPurpose({http://www.w3.org/2002/01/P3Pv1/admin, http://www.w3.org/2002/01/P3Pv1/telemarketing})
Matching	L:AuthzUseForPurpose ⊴ R:AuthzUseForPurpose ⇔ ∀ p ∈ L.purposes ∃ q ∈ R.purposes : p ⊆ q i.e. the list of purposes match when for each purpose in L there is a purpose in q that is an ancestor of or equal to p.

As a basic purpose ontology, one can use the following flat list of purposes and primary purposes as defined in P3P 1.1. We refer to the specification of P3P 1.1 for a description of the purposes associated to the URIs below.

- http://www.w3.org/2002/01/P3Pv1/current
- http://www.w3.org/2002/01/P3Pv1/admin
- http://www.w3.org/2002/01/P3Pv1/develop
- http://www.w3.org/2002/01/P3Pv1/tailoring
- http://www.w3.org/2002/01/P3Pv1/pseudo-analysis

- http://www.w3.org/2002/01/P3Pv1/pseudo-decision
- http://www.w3.org/2002/01/P3Pv1/individual-analysis
- http://www.w3.org/2002/01/P3Pv1/individual-decision
- http://www.w3.org/2002/01/P3Pv1/contact
- http://www.w3.org/2002/01/P3Pv1/historical
- http://www.w3.org/2002/01/P3Pv1/telemarketing
- http://www.w3.org/2006/01/P3Pv11/account
- http://www.w3.org/2006/01/P3Pv11/arts
- http://www.w3.org/2006/01/P3Pv11/browsing
- http://www.w3.org/2006/01/P3Pv11/charity
- http://www.w3.org/2006/01/P3Pv11/communicate
- http://www.w3.org/2006/01/P3Pv11/custom
- http://www.w3.org/2006/01/P3Pv11/delivery
- http://www.w3.org/2006/01/P3Pv11/downloads
- http://www.w3.org/2006/01/P3Pv11/education
- http://www.w3.org/2006/01/P3Pv11/feedback
- http://www.w3.org/2006/01/P3Pv11/finmgt
- http://www.w3.org/2006/01/P3Pv11/gambling
- http://www.w3.org/2006/01/P3Pv11/gaming
- http://www.w3.org/2006/01/P3Pv11/government
- http://www.w3.org/2006/01/P3Pv11/health
- http://www.w3.org/2006/01/P3Pv11/login
- http://www.w3.org/2006/01/P3Pv11/marketing
- http://www.w3.org/2006/01/P3Pv11/news
- http://www.w3.org/2006/01/P3Pv11/payment
- http://www.w3.org/2006/01/P3Pv11/sales
- http://www.w3.org/2006/01/P3Pv11/search
- http://www.w3.org/2006/01/P3Pv11/state
- http://www.w3.org/2006/01/P3Pv11/surveys

Additionally we define one purpose

- http://www.primelife.eu/purposes/unspecified

to indicate that the data can or will be used for purposes that are not specified at the time of transmission.

4.3 Downstream Usage Authorization

The second concrete authorization type that we define is the authorization to forward the information to third parties, so-called downstream Data Controllers. In addition to merely stating the fact that forwarding the information to downstream data controllers is allowed, this authorization type allows to specify the policy under which this information will be made available to the downstream data controllers. This policy states the minimal policy that the (primary) data controller has to enforce when sharing the information with downstream data controllers. In case the downstream usage authorization allows sharing the information with third parties (i.e. the attribute 'allowed' as specified in the below schema is 'true') but does not specify a specific policy, no sharing restrictions are imposed on the the (primary) data controller and he can share the data with downstream data controllers at discretion.

The authorization to share the personal data in question with downstream Data Controllers is indicated by a <AuthzDownstreamUsage> element, of which the schema is given below.

```
<!-- Authorization: Downstream usage -->
<xs:element name="AuthzDownstreamUsage" substitutionGroup="Authorization">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="AuthorizationType">
        <xs:sequence minOccurs="0" maxOccurs="1">
          <xs:element ref="xacml:Policy"/>
        </xs:sequence>
        <xs:attribute name="allowed" type="xs:boolean" use="required"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

The <AuthzDownstreamUsage> element has an attribute allowed indicating whether downstream usage is allowed or not, and an optional <xacml:Policy> child element specifying the access control policy that has to be enforced by the data controller when it forwards the data to downstream data controllers. The <AuthzDownstreamUsage> should NOT contain an <xacml:Policy> child element when it occurs in the <AuthorizationsSet> of a <DataHandlingPolicy> element (This is because our current approach on handling downstream requirements when generating sticky policies is to merely copy the downstream usage preferences directly into the sticky policy. A more sophisticated but also more complex approach would be to match the downstream usage policy with the downstream usage preferences and copy the result of this matching into the sticky policy. For details on our current approach see Section 7.3.).

When occurring in the <AuthorizationsSet> of a <DataHandlingPrefrences> or <StickyPolicy> element, the optional <xacml:Policy> child element must contain an empty <xacml:Target/> element. The Data Controller will replace the empty <xacml:Target/> element with an <xacml:Target> element specifying a unique reference to the transmitted information as stored on the Data Controller's system; this is to avoid that a malicious Data Subject can modify the access control policy for other resources than the personal data that is being transmitted. We refer to Section 7.2 for more details on downstream access control.

Intuitively, a match occurs whenever the data handling policy forbids downstream usage, or whenever the data handling policy and the data handling preferences both allow it. In the description below, we assume that the authorization L is included in the data subject's <DataHandlingPreferences>, and that R is included in the Data Controller's <DataHandlingPolicy>.

Name	**AuthzUseForPurpose**		
Parameters	boolean	allowed	Downstream usage is allowed or not
	<xacml:Policy>	Policy	Policy to enforce for downstream usage.

Name	AuthzUseForPurpose
Description	Authorization to forward the data to downstream Data Controllers.
Examples	No downstream usage allowed or intended ➔ AuthzDownstreamUsage(false) Downstream usage allowed under policy P ➔ AuthzDownstreamUsage(true, P)
Matching	L:AuthzDownstreamUsage ⊴ R:AuthzDownstreamUsage ⇔ R.allowed=false OR (L.allowed=true AND R.allowed=true) i.e. R does not allow downstream usage, or both R and L allow downstream usage.

When a match occurs, the resulting sticky policy is derived according to the following procedure:

- The matching fails if allowed=”true” in the data controller’s data handling policy but allowed=”false” in the Data Subject’s preferences; otherwise, the matching succeeds.
- If allowed=”false” in the Data Controller’s data handling policy, then the resulting sticky policy contains a <AuthzDownstreamUsage> element with allowed=”false”.
- If allowed=”true” in the Data Controller’s data handling policy, then the resulting sticky policy contains a <AuthzDownstreamUsage> element with allowed=”true” and the <xacml:Policy> element specified in the Data Subject’s preferences.

5 Introduction to XACML

The eXtensible Access Control Markup Language (XACML) [eXt09] is an XML-based language for expressing and interchanging access control policies. The language offers the functionalities of most security policy languages and has standard extension points for defining new functions, data types, policy combination logic, and so on. In addition to the language, XACML defines both an architecture for the evaluation of policies and a communication protocol for message interchange. Some of the main functionalities offered by XACML can be summarized as follows.

- Policy combination. XACML provides a method for combining policies independently specified. Different entities can then define their policies on the same resource. When an access request on that resource is submitted, the system takes into consideration all the applicable policies.
- Combining algorithms. Since XACML supports the definition of positive and negative authorizations, there is the need for a method for reconciling independently specified policies when their evaluation is contradictory. XACML supports different combining algorithms, each representing a way of combining multiple decisions into a single decision.
- Attribute-based restrictions. XACML supports the definition of policies based on generic properties (attributes) associated with subjects (e.g., name, address, occupation) and resources (e.g., creation date, type). XACML includes some built-in operators for comparing attribute values and provides a method for adding non-standard functions.
- Policy distribution. Policies can be defined by different parties and enforced at different enforcement points. Also, XACML allows one policy to contain, or refer to, another.
- Implementation independence. XACML provides an abstraction layer that isolates the policy-writer from the implementation details. This layer guarantees that different implementations operate in a consistent way, regardless of the specific implementation.
- Obligations. XACML provides a method for specifying actions, called obligations, which must be fulfilled in conjunction with the policy enforcement, after the access decision has been taken.

XACML also supports multiple subjects specification in a single policy, multi-valued attributes, conditions on metadata of the resources, and policy indexing.

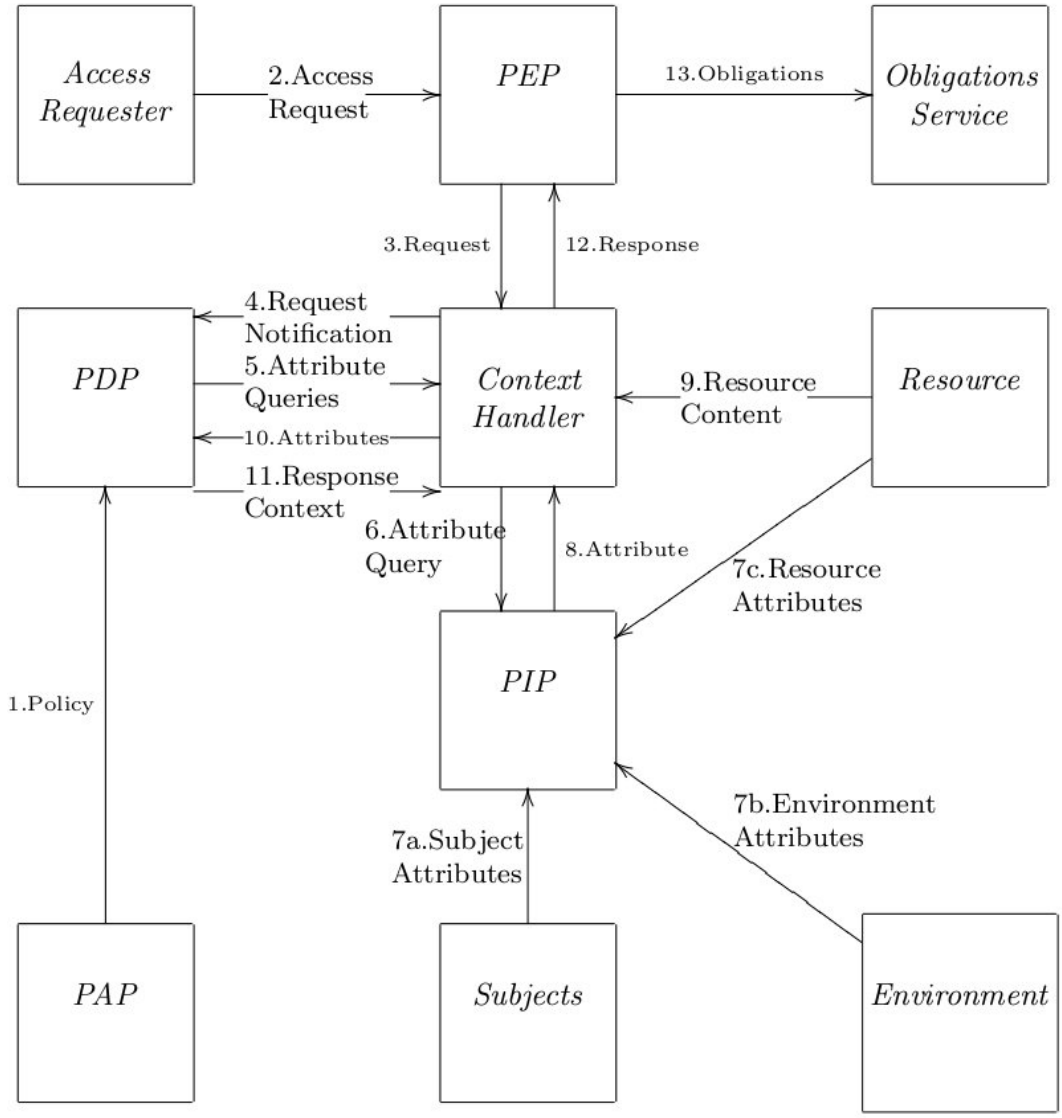


Figure 9: Overview of XACML dataflow [eXt09]

Figure 9 illustrates the XACML working and the data flow in the access control evaluation. Access control works as follows. The requester sends an access request to the Policy Enforcement Point (PEP) module (step 2) which in turn send it to the Context Handler. The Context Handler translates the original request into a canonical format, called XACML request context (step 3) and sends it to the Policy Decision Point (PDP) (step 4). The PDP identifies the applicable policies among the ones stored at the Policy Administration Point (PAP) and retrieves the attributes required for the evaluation through the Context Handler (steps 5-10). If some attributes are missing, the context handler queries the Policy Information Point (PIP) module for collecting them. The PIP provides attribute values about the subject, resource, and environment. To this purpose, the PIP interacts with the subjects, resource, and environment modules. The environment module provides a set of attributes that are relevant to take an authorization decision and are independent of a particular subject, resource, and action. The PDP evaluates the policies against the retrieved attributes, and returns the XACML response context to the Context Handler (step 11). The Context

Handler translates the XACML response context to the native format of the PEP and returns it to the PEP together with an optional set of obligations (step 12). The PEP fulfills the obligations (step 13), and grants or denies the request according to the decision in the response context.

5.1 Basic XACML Concepts

XACML relies on a model that provides a formal representation of access control policies and on mechanisms for their evaluation. An XACML policy contains one <Policy> or <PolicySet> root element, which is a container for other <Policy> or <PolicySet> elements. Element <Policy> consists of a Target, a set of Rule, an optional set of Obligation, an optional set of Advice, and a rule combining algorithm. A Target element includes a set of requests in the form of a logical expression on subjects, resources, and actions. If a request satisfies the requirements specified in the Target, the corresponding policy applies to the request. A Rule corresponds to a positive (permit) or negative (deny) authorization, depending on its effect, and may additionally include an element Condition specifying further restrictions on subjects, resources, and actions. As for element Policy, element Rule may contain a Target, Obligation, and Advice. Each condition can be defined through element Apply with attribute FunctionID denoting the XACML predicate (e.g., string-equal, integer-less-than) and with appropriate sub-elements denoting both the attribute against which the condition is evaluated and the comparison value. The rule's effect is then returned whenever the rule evaluates to true. The Obligation element specifies an action that has to be performed in conjunction with the enforcement of an authorization decision. The Advice element specifies supplemental information about a decision. Each element Policy has attribute RuleCombiningAlgID specifying how to combine the decisions of different rules to obtain a final decision of the policy evaluation (e.g., deny overrides, permit overrides, first applicable, only one applicable). According to the selected combining algorithm, the authorization decision can be permit, deny, not applicable (i.e., no applicable policies or rules can be found), or indeterminate (i.e., some information is missing for the completion of the evaluation process).

As an example of XACML policy, suppose that a hospital defines a high-level policy stating that "any user with role head physician can read the patient record for which she is designated as head physician". Figure 9 illustrates the XACML policy corresponding to this high-level policy. The policy applies to requests on the http://www.example.com/hospital/patient.xsd resource. The policy has one rule with a target that requires a read action, a subject with role head physician and a condition that applies only if the subject is the head physician of the requested patient.

5.2 XACML 3.0: Privacy Profile

The XACML v3.0 Privacy Policy Profile Version 1.0 is a standard issued by the OASIS group describing “a profile of XACML for expressing privacy policies” [OAS09]. This profile uses the following two attributes:

- urn:oasis:names:tc:xacml:2.0:resource:purpose, which indicates the purpose for which a data resource was collected, and
- urn:oasis:names:tc:xacml:2.0:action:purpose, which corresponds to the purpose for which access to a data resource was requested.

A standard rule is defined, according to which access to the requested resource is to be denied unless the two above-mentioned purposes match by regular-expression match, as shown below.

```
<Policy PolicyId="Pol1" RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
 rule-combining-algorithm:permit-overrides" . . . >
 <Target>
   <AnyOf>
     <AllOf>
       <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:stringmatch">
         <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
           urn:example:med:schemas:record
         </AttributeValue>
         <AttributeDesignator
           Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
           DataType="http://www.w3.org/2001/XMLSchema#string"
           AttributeId="urn:oasis:names:tc:xacml:1.0:resource:target-namespace"/>
       </Match>
     </AllOf>
   <AnyOf>
 </Target>
 <Rule RuleId="ReadRule" Effect="Permit">
   <Target>
     <AnyOf>
       <AllOf>
         <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
           <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
             head physician
           </AttributeValue>
           <AttributeDesignator
             Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
             AttributeId= "urn:oasis:names:tc:xacml:2.0:example:attribute:role"
             DataType="http://www.w3.org/2001/XMLSchema#string"/>
         </Match>
       </AllOf>
     </AnyOf>
     <AnyOf>
       <AllOf>
         <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
           <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
             read
           </AttributeValue>
           <AttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
             Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
             AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
         </Match>
       </AllOf>
     </AnyOf>
   </Target>
   <Condition>
     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <AttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
         Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
         AttributeId="urn:oasis:names:tc:xacml:1.0:subject:head-physicianID"/>
```

```
      <AttributeSelector RequestContextPath="/ctx:Request/ctx:Resource/ctx:
        ResourceContent/hospital:record/hospital:patient/hospital:
        patient-head-physicianID/text()"
        DataType="http://www.w3.org/2001/XMLSchema#string"
        Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"/ >
   </Condition>
 </Rule>
</Policy>
```

Figure 10: An example of XACML policy

```
<Rule xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-06"
 RuleId="urn:oasis:names:tc:xacml:2.0:matching-purpose" Effect="Permit">
 <Condition
   FunctionId="urn:oasis:names:tc:xacml:2.0:function:string-regexp-match">
   <AttributeDesignator
     category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
     AttributeId="urn:oasis:names:tc:xacml:2.0:resource:purpose"/>
     DataType="http://www.w3.org/2001/XMLSchema#string"
   <AttributeDesignator
     category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
     AttributeId="urn:oasis:names:tc:xacml:2.0:action:purpose"/>
     DataType="http://www.w3.org/2001/XMLSchema#string"
 </Condition>
</Rule>
```

Figure 11: Collection purpose and Access request purpose matching rule

Such rule must be used in the scope of rule-combining algorithm urn:oasis:names:tc:xacml:2.0:rule-combining-algorithm:deny-overrides. To conform to such specification, any implementation should, as an XACML request producer, make use of the attributes above, and, as an XACML policy processor, enforce the proposed rule, respectively. The profile deals with an important aspect that can be found also in the research context of the PrimeLife project, namely, purposes of data processing. Nevertheless, purposes are a very specific facet of Data Handling, and we consider the scope of this standard proposal too narrow if compared to the much more general concepts its name, “Privacy Policy Profile”, refers to.

6 Policy Language

6.1 Policy Language Model

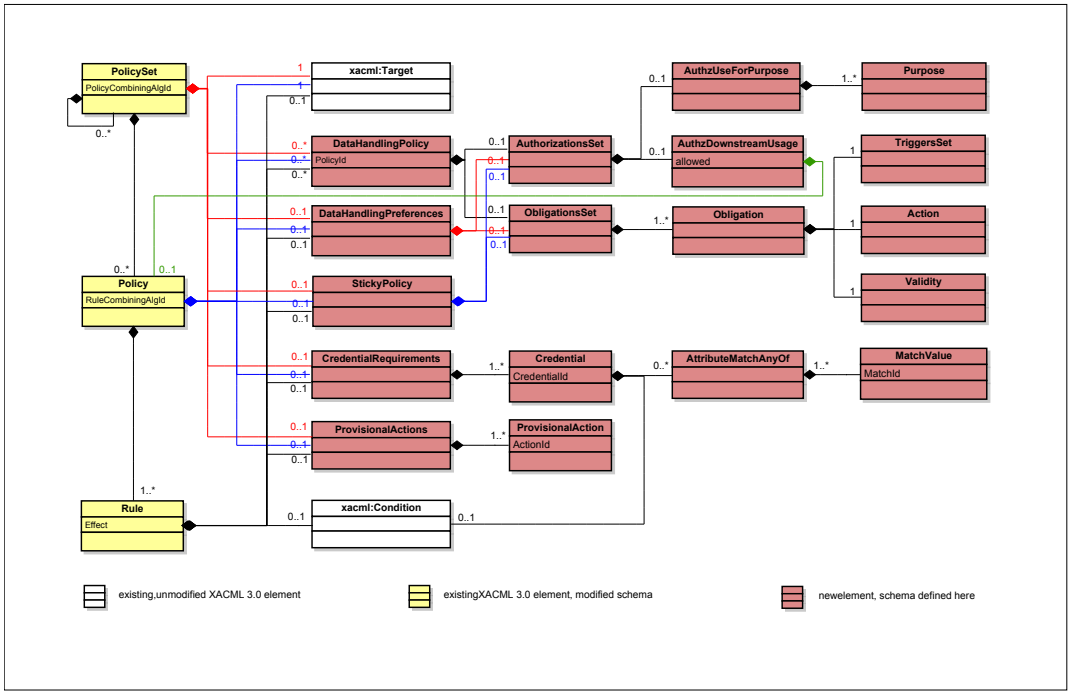


Figure 12: Model of our policy language

In this deliverable we extend XACML 3.0 [Rissanen] with a number of privacy-enhancing and credential-based features. Our language is intended to be used

- by the Data Controller to specify the access restrictions to the resources that he offers;

- by the Data Subject to specify access restrictions to her personal information, and how she wants her information to be treated by the Data Controller afterwards;

- by the Data Controller to specify how “implicitly” collected personal information (i.e., information that is revealed by the mere act of communicating, such as IP address, connection time, etc.) will be treated;

- and by the Data Subject to specify how it wants this implicit information to be treated.

6.1.1 Rules, Policies, and Policy Sets

We maintain the overall structure of the XACML language, but we introduce a number of new elements to support the advanced features that our language has to offer, and we also modify the schema of a number of existing elements.

As in XACML, the main components of our language are rules, policies, and policy sets. Each rule has an effect, either “Permit” or “Deny”, that indicates the consequence when all conditions stated in the rule have been satisfied. Rules are grouped together in policies. When a policy is evaluated, the rule combining algorithm of the policy (as stated in an XML attribute of the policy) defines how the effects of the applicable rules are combined to determine the effect of the policy. Policies, on their turn, are grouped together in policy sets; the effect of a policy set is determined by the effects of the contained policies and the stated policy combining algorithm. Finally, different policy sets can be further grouped together in parent policy sets.

The main components of a rule are:

a target, describing the resource, the subject, and the environment variables for which this rule is applicable;

credential requirements, describing the credentials that need to be presented in order to be granted access to the resource;

provisional actions, describing which actions (e.g., revealing attributes or signing statements) have to be performed by the requestor in order to be granted access;

a condition, specifying further restrictions on the applicability of the rule beyond those specified in the target and the credential requirements;

data handling policies, describing how the information that needs to be revealed to satisfy this rule will be treated afterwards;

and data handling preferences, describing how the information contained in the resource that is protected by this rule has to be treated.

The target is described by a <xacml:Target> element containing a number of nested <xacml:AnyOf> and <xacml:AllOf> elements. The schema of the <xacml:Condition> element is also left intact, in the sense that it contains an element of type <xacml:Expression> that evaluates to a Boolean, but we do introduce a new element <CredentialAttributeDesignator> of type <xacml:Expression> that can be used inside a <xacml:Condition> to retrieve an attribute value from a presented credential. The other elements are new; we briefly describe their functionality below.

6.1.2 Credential Requirements

Each rule can contain a <CredentialRequirements> element to specify the credentials that have to be presented in order to satisfy the rule. The <CredentialRequirements> element contains a separate <Credential> element for each credential that needs to be presented. Each <Credential> element contains a unique identifier CredentialId of type URI that is used to refer to the credential from elsewhere in the rule. No two credentials required by a rule should have the same identifier.

The <Credential> element can contain restrictions that apply to the credential. These restrictions can be expressed in one of two ways: either by means of a list of <AttributeMatchAnyOf> elements, allowing matching attributes of the credential against a list of candidate values, or by means of a generic <xacml:Condition> element, with the restriction that all <CredentialAttributeDesignator> elements contained in the expression can only refer to the CredentialId of this credential. The <AttributeMatchAnyOf> element is less expressive than the <xacml:Condition> element, but allows for more efficient matching of credentials for a very common class of restrictions, typically restricting the credential type or the issuer to a list of specified values.

The <CredentialRequirements> element can also occur in parent <Policy> and <PolicySet> elements. They follow a typical distributive semantics; namely, one should treat the <CredentialRequirements> element of a rule as if it contained all <Credential> elements specified within the rule itself, as well as those specified within all parent <Policy> and <PolicySet> elements.

6.1.3 Provisional Actions

The <ProvisionalActions> defined in this document is used to specify the provisional actions that a requestor must perform before being granted access to the resource. Currently supported actions include revealing of attributes (to the Data Controller or to a third party), signing a statement, and so-called “spending” of credentials, which allows to put restrictions on the number of times that the same credential is used to obtain access. Each action is described in a <ProvisionalAction> element; the schema is extensible so that new types of provisional actions can easily be added later on, in a way that is reminiscent of the way that XACML itself allows the definition of new functions for use in the <xacml:Apply> element.

Similarly to the <CredentialRequirements> element, <ProvisionalActions> elements contained in <Policy> and <PolicySet> elements follow the standard distributive semantics, i.e., it is as if the <Rule> contains the <ProvisionalAction> elements contained in the <Rule> itself and in all parent <Policy> and <PolicySet> elements.

6.1.4 Data Handling Policies

Each rule, policy, or policy set can contain a number of data handling policies, each of which is expressed within a <DataHandlingPolicy> element. A data handling policy can be referred to from anywhere in the rule by its unique PolicyId identifier. The main purpose of the data handling policies is for the Data Controller to express what will happen to the information about the Data Subject that is collected during an access request. The provisional action to reveal an attribute value (see previous subsection) therefore contains an optional reference to the applicable data handling policy. The data handling policies for generic information (i.e., IP address, connection information, etc.) are specified in <DataHandlingPolicy>s with dedicated PolicyId identifiers in the root <PolicySet>.

A data handling policy consists of a set of authorizations, contained in an <AuthorizationsSet> element, that the Data Controller wants to obtain on the collected information, and a set of obligations, expressed in a <ObligationsSet> element, that he promises to adhere to. Before the Data Subject reveals her information, these authorizations and obligations are matched against the Data Subject's data handling preferences (see next subsection) to see whether a matching sticky policy can be agreed upon.

Our language supports an extensible authorization vocabulary, but we predefine two concrete authorization types here. The first is the authorization to use the information for a list of purposes, enumerated inside an <AuthzUseForPurpose> element. We envisage purposes to be defined in hierarchical and user-extensible ontologies; a predefined list of purposes is given in Section reference X.2 Usage purposes authorization in authorizations.doc. The second predefined authorization type is the authorization to forward the information to third parties, also called downstream usage. The <AuthzDownstreamUsage> element contains a Boolean attribute allowed indicating whether downstream usage is allowed. The optional <Policy> child element can only be used in the data handling preferences, as described in the next subsection.

Obligations are specified inside <Obligation> elements, which on their turn contain a <TriggersSet> element describing the events that trigger the obligation, an <Action> element describing the action to be performed, and a <Validity> element describing the validity time frame of the obligation. In this document we provide a basic vocabulary of authorizations, triggers, and actions that can be used to describe a data handling policy, but the schema is left intentionally open so that new authorizations, triggers, and actions can be added in the future.

6.1.5 Data Handling Preferences

The data handling preferences of a rule, embedded in the <DataHandlingPreferences> element, specify how the information obtained from the resource protected by this rule is to be treated after access is granted. The preferences are expressed by means of a set of authorizations and obligations, just like data handling policies. When access to the resource is requested, the data handling preferences have to be matched against a proposed data handling policy to derive the applicable sticky policy – if a match can be found.

An important difference between data handling preferences and data handling policies is the resource that they pertain to: data handling preferences always describe how the resource protected by the rule itself has to be treated, while data handling policies pertain to information that a requester will have to reveal in order to be granted access to the resource.

The main use of data handling preferences that we envisage is for a Data Subject to specify how she wants her personal data to be treated by a Data Controller, i.e., which authorizations she grants to the Data Controller with respect to her personal data, and which obligations he will have to adhere to.

Optionally, if the data handling preferences contain a downstream usage authorization, the <AuthzDownstreamUsage> element can optionally include a <Policy> element specifying the downstream access control policy, i.e., the access control policy that has to be enforced on the downstream data controllers.

6.1.6 Sticky Policies

The sticky policy associated to a resource, meaning the agreed-upon sets of granted authorizations and promised obligations with respect to a resource, is expressed in the <StickyPolicy> element. The sticky policy is usually the result of an automated matching procedure between the Data Subject's data handling preferences and the Data Controller's data handling policy.

The main difference between the <StickyPolicy> and the <DataHandlingPreferences> is that the former contains the authorizations and obligations that the policy-hosting entity itself has to adhere to, while the latter contains authorizations and obligations that an eventual recipient has to adhere to. Typically a Data Subject will not impose on his or her self any authorizations or obligations concerning her own personal data, so her policy will not contain a <StickyPolicy> element. The Data Controller, on the other hand, will describe in the <StickyPolicy> the authorizations and obligations that he himself has to adhere to, while the <DataHandlingPreferences> contain those that a Downstream Data Controller has to adhere to. Usually, the Downstream Data Controller will be subject to the same or stronger restrictions than the Data Controller himself, meaning that the policy specified in the

<DataHandlingPreferences> will usually be at most as permissive as the policy specified in the <StickyPolicy>.

6.1.7 Relation to XACML Obligations

One may wonder why we didn’t choose to use the standard <xacml:Obligations> element to specify the obligations that we embed in the data handling policies or preferences. The reason is that <xacml:Obligations> can only be used to specify obligations that the PEP has to adhere to when an access request occurs for the resource that is protected by this rule. This cannot be used for our data handling policies, since the latter pertain to information that the requestor will have to reveal in order to obtain access, rather than to the resource being protected. It cannot be used for our data handling preferences either, since the latter specify obligations that the recipient of the resource has to adhere to, rather than the PEP that is protecting access to the resource. Meaning, by populating the <xacml:Obligations> element that protects her personal data, a Data Subject would impose obligations that she herself has to adhere to each time a Data Controller requests access to the personal data, rather than imposing obligations on the Data Controller.

The only use that we could have had for the <xacml:Obligations> element is to store and enforce those obligations that the Data Controller committed to in an agreed-upon sticky policy that are triggered by access requests. Since obligations triggered by access requests are only a small subclass of the obligations that we consider here, we chose to leave the storage and enforcement of obligations entirely up to the Obligation Engine, and let the PEP simply signal the Obligation Engine each time an access request occurs.

6.2 Extending XACML for Credential-Based Access Control

This section describes an extension to XACML 3.0 for credential-based access control.

In the following we explain the setting of credential-based access control, as it is the basis for the language extensions that we propose, as well as what exactly is meant by a "credential". Afterwards, we list a number of technologies that can be seen as instantiations of credentials, and hence can be modeled by our (technology-independent) extensions, and describe a number of functionalities that are supported by existing technologies and therefore have influenced the design of our language.

6.2.1 The Scenario

We consider a setting with three types of entities, namely Data Subjects, Data Controllers, and Issuers. Data Subjects hold trusted credentials that they have obtained from Issuers and want to access protected resources

(e.g., Web pages, databases, Web services, etc.) hosted by the Data Controllers.

We do not make assumptions on how the Data Subjects obtain their credentials; this could be on-line by visiting the Isser's website, or off-line by physically going to an issuing desk (e.g. the local town hall).

Servers restrict access to their resources by means of access control policies containing requirements in terms of the credentials that the Data Subject needs to own (and present) in order to be granted access.

In the interaction sketched here, we only consider the part that is relevant to credential-based access control, and not the full picture of our policy language sketched in (add reference to full interaction sketch). Typically, a Data Subject contacts a Data Controller to request access to a resource that she is interested in. The Data Controller responds with the applicable access control policy, containing the credential requirements expressing which conditions on which credential attributes have to hold, which attributes from which credentials have to be revealed, and whom the Data Controller trusts as issuers for these credentials. The policy that is transmitted to the Data Subject is derived from the policy that the Data Controller attached to the resource, but is not necessarily exactly equal to it. First, the policy sent to the Data Subject may be a partially evaluated version of the Data Controller's policy where for instance environment variables (e.g., current time) or previously revealed attributes have already been replaced by their respective values. Second, the Data Controller may not be willing to reveal all details of its access control policy, in which case a sanitized version of the policy is transmitted.

Upon receiving the policy, the Data Subject evaluates whether she is able to fulfill the given requirements with her credentials. If so, she produces a token that proves her fulfillment of the requirements and sends it, together with the attributes to reveal and a description of the claims about the credentials to the Data Controller. The exact format and content of this token depend on the underlying authentication technology. Finally, if the Data Controller successfully verifies the validity of the proof token, he grants the Data Subject access to the resource.

Depending on the technology, the Data Subject may be able to derive the proof token herself, or she may need to interact with the credential issuers. Likewise, the Data Controller may be able to independently verify the proof token, or may need to contact the issuers to assist in the verification or simply to confirm that the Data Subject satisfies the specified conditions.

6.2.2 Definition of Credentials

The requirements language that we present is geared towards enabling user-centric and privacy-friendly access control on the basis of certified credentials. While the language leverages the advanced privacy and

anonymity features offered by anonymous credential systems, it is designed to be technology-agnostic in the sense that it addresses general credential concepts without targeting one technology in particular. As the concept of a credential is a very abstract one that can be understood in different ways, we now describe how this concept is used here.

By a credential we mean an authenticated statement about attribute values made by an Issuer, where the statement is independent from a concrete mechanism for ensuring authenticity. The statement made by the issuer is meant to affirm qualification. A credential serves as means for proving qualification, i.e., it typically serves as proof of identity, proof of authority, or both proof of identity and authority at the same time. For example, national identity cards are proofs of identity, movie tickets are proofs of authorization to watch a particular movie from a particular seat, and driver's licenses are proofs of identity and of authorization to drive motor vehicles of a certain category at the same time.

6.2.3 Example Credential Technologies

While a policy author is allowed to express his policy in a technology-independent way, a Data Subject can freely choose the technology to fulfill the policy – to the extent that the same technology is supported by the Data Controller. This includes the possibility to use credentials of different technologies to fulfill one particular policy. We envision that a multitude of different credential technologies implements the generic concepts of credential-based access control. The only assumption that our language makes on the underlying technology is that credentials are abstract data structures in the form of certified attribute-value pairs. We highlight some candidate technologies below:

X.509 certificates: While the original purpose of X.509 certificates was merely to bind entities to their public signing and/or encryption keys, the latest version of X.509 v3 certificates also allows additional community-specific attributes to be included in the certificate. When used as a credential mechanism, the issuer essentially acts as a certification authority by signing certificates containing the bearer's list of attribute values as well as his public key. The bearer can prove ownership of the credential by proving knowledge of the underlying private key.

Anonymous credentials: Two main anonymous credential systems have been implemented today, namely Identity Mixer and UProve. Much like an X.509 certificate, an anonymous credential can be seen as a signed list of attribute-value pairs issued by an issuer. Unlike X.509 certificates, however, they have the advantage that the owner can reveal any subset of the attributes, or merely prove that they satisfy some condition, without revealing any more information. Also, they provide additional privacy guarantees like unlinkability, meaning that even with the help of the issuer a Data Controller cannot link multiple visits by the same Data Subject to each other, or link a visit to the issuing of a credential.

OpenID: In OpenID Data Subjects are identified by a URL that is authenticated by the Data Subject's OpenID provider. When logging on to a website the Data Subject authenticates with respect to this provider, rather than the website itself. The recent OpenID Attribute Exchange extension allows user-defined attributes to be exchanged. One could consider the OpenID provider to be the issuer of a single credential per Data Subject that contains all of her attributes. This gives a very basic form of credential-based access control; perhaps future extensions will allow grouping attributes into credentials and storing one Data Subject's credentials at different OpenID providers.

LDAP: One can also imagine credentials to be represented within an LDAP directory tree. A Data Subject's entire directory entry in an LDAP directory maintained by the credential issuer could be seen as a single credential, or the hierarchical structure of LDAP trees could be exploited to group together the attributes of a single credential. A server can verify a Data Subject's attributes simply by looking them up in the issuer's directory.

Kerberos: Kerberos tickets could also be seen as digital credentials, albeit very limited ones containing just the Data Subject's identity, the server that the ticket gives access to, and some validity information. In principle, one could imagine other attributes to be authenticated in the ticket as well, but this is not part of the current standard.

Functionality: We now describe a number of credential features that we leverage for credential-based access control, and sketch how these features could be supported in the different technologies. The language that we develop will be able to express all the concepts listed below.

6.2.4 Credential Functionality

Proof of ownership. An important aspect of credentials is their ownership. To bind a credential to its legitimate owner, authentication information may be tied to the credential. This could be, e.g., a picture of the Data Subject, the hash value of a PIN, or the public key of a cryptographic identification scheme. Proving credential ownership in our notion means that authentication is successfully performed with respect to the authentication information whenever such information is present. Depending on the employed authentication mechanisms, the successful authentication may further provide some liveness guarantees (and thereby prevent replay attacks).

The actual implementation of the ownership proof is technology dependent. For X.509 certificates, Data Subjects can prove ownership of the credential by signing a random nonce under the public key that is certified by the certificate. In anonymous credentials, Data Subjects execute a zero-knowledge proof of knowledge of an underlying master secret. OpenID and LDAP both work with a password-based approach. For OpenID the Data Subject authenticates to the OpenID provider directly, for LDAP the Data Subject sends the password to the server, who then checks it with the directory.

Selective attribute disclosure. Some technologies allow attributes within a credential to be revealed selectively, meaning that the server only learns the value of a subset of the attributes contained in the credential. Not all technologies support this feature. For example, verification of the issuer's signature on X.509 certificates requires all attribute values to be known. In LDAP directories the Data Subject obviously has no control over which values the server looks up. Anonymous credentials and OpenID, on the other hand, have native support for this feature, although the mechanisms are quite different. For OpenID the provider simply only reveals the requested attributes, while for anonymous credentials it is the cryptography that ensures that no information is leaked about non-disclosed attributes.

Proving conditions on attributes. Certain credential technologies allow for proving conditions over attributes without revealing their actual values. Obviously, for technologies such as X.509 certificates and LDAP directory, the only way to prove that an attribute satisfies a condition is by revealing its value, so here this distinction is not very important. Anonymous credentials, however, do support this feature by using zero-knowledge proofs. OpenID supports this too, as the provider can simply confirm to the server that the condition holds, without revealing anything more.

Attribute disclosure to third parties. Usually attributes are revealed to the server that enforces the policy, but the policy could also require certain attributes to be revealed to an external third party. For example, the server may require that the Data Subject reveals her full name to a trusted escrow agent, so that she can be de-anonymized in case of fraud, thereby adding accountability to otherwise anonymous transactions. As another example, an online shop could require the Data Subject to reveal her address to the shipping company directly, rather than disclosing it to the shop.

The Idemix anonymous credential system elegantly supports this feature using verifiable encryption. Here, the Data Subject hands to the server a ciphertext containing the relevant attribute encrypted under the third party's public key, and proves in zero-knowledge that the correct attribute was encrypted. As an additional feature, a data handling policy describing, e.g., for what purpose the ciphertext can be decrypted, can be bound to the ciphertext via the so-called decryption label. The server therefore cannot lie about the intended data handling policy when forwarding the ciphertext to the third party, as if the policy was tied together inseparably with the data.

In OpenID this feature could be supported by letting the provider reveal the attribute to the third party directly, rather than to the server. For X.509 certificates one could change the communication pattern and let the Data Subject send the certificate containing the relevant attribute directly to the third party, obtain a receipt in exchange, and show this receipt to the server. A similar approach would work for LDAP directories: the Data Subject sends her LDAP password to the third party, who can verify its correctness, fetch the attribute and send back a receipt.

Preventing credential mixing. Credentials are atomic collections of attributes, in the sense that a Data Subject cannot ``mix-and-match'' attributes from different credentials in order to satisfy a policy. For example, if the policy requires the Data Subject to reveal the number and expiration date of a credit card, then the Data Subject should not be able to satisfy the policy by revealing the number of one credit card and the expiration date of another. On the other hand, if the policy requires the Data Subject to have two credit cards, then the language should be able to express which card is being referred to.

Signing of statements. The server may require the Data Subject's explicit consent to some statement, e.g., the terms of service or the privacy policy of the site. The signature acts as transferable evidence that this statement was agreed to by a Data Subject fulfilling the policy in question. There are various ways in which the Data Subject can express her consent; our language does not impose a particular technology. For example, the Data Subject could simply click an OK-button at the bottom of the statement, or the Data Subject could digitally sign the statement using her X.509 credential. Anonymous credentials allow to sign statements while maintaining maximal privacy: anyone can verify that the signature was placed by a Data Subject satisfying the access control policy, but only a trusted opening authority can tell who exactly the Data Subject was.

Limited spending. The server may want to impose limitations on the number of times that the same credential can be used (or ``spent'') to access a resource, e.g., to specify that each Data Subject can only vote once in an online poll. The policy language should be able to express the amount, i.e., the number of units that have to be spent to obtain access, and the spending limit, i.e., the maximum number of units that can be spent until access is refused. To express more complex usage limitation rules, e.g., multiple resources that can be accessed at most n times total per month, one can specify the spending scope on which the units have to be spent. This is a URI defining the “scope” of the usage limitation; overspending occurs if more units than the limit are spent from the same credential on the same spending scope. For example, to prevent two resources A and B from being accessed more than n times per month, the scope URI could be

append (“urn:scope:AorB:”, currMonth()), “/”,currYear())

Some anonymous credential systems support this type of limited spending natively in the underlying cryptography. With X.509 certificates or LDAP directories, the server could simply keep track which credential was used how many times for which spending scope URI, and refuse access when the spending limit is reached. For OpenID the same principle could be used, but it would be the OpenID provider who maintains the list.

6.3 Policy Sanitization

Privacy issues, which are usually associated with the protection of a Data Subject's personally identifiable information (personal data), apply also to servers, in particular to the release of their access control policies. In fact, there is a trade-off between the privacy of the Data Subject requesting access to a resource, and that of the Data Controller managing the resource with the relevant access control policy, as shown by the following example.

Consider a policy that grants access to a resource only to Swiss citizens, by requesting that the Data Subject should show a passport credential that includes a nationality attribute with value 'Switzerland'.

The Data Subject has a different level of freedom depending on how such a policy is communicated to her. Let the original requirement in the Data Controller's access control policy be specified as

passport.nationality = 'Switzerland' .

If the requirement is also presented in this way to the Data Subject, then she learns the policy managing the requested resource, and she can check whether she satisfies it even before responding to the server. In case she isn't (e.g., the Data Subject only owns a French passport), she has the option to interrupt the interaction, which then is aborted without the server acquiring any information from the Data Subject.

With sanitization we mean the technique of processing a policy before presenting it to the user, so that not all information on the authorization conditions is revealed. For instance, a very simple way to sanitize the condition above would be by transmitting

passport.nationality = ----------

as the access control policy. This requirement specifies that only citizens of a particular nationality are allowed, but doesn't specify of which nationality. Clearly, when presented with such a partially obfuscated requirement, the Data Subject has less knowledge about what property she is expected to fulfill to get access to the requested resource. No policy evaluation on the Data Subject side is possible in this case, so that the user has no choice but to reveal her nationality to the Data Controller, hoping to be allowed access to the resource.

The loss of privacy on the Data Subject's side obviously increases privacy on the Data Controller's side, as the latter no longer has to reveal his full policy. The choice between privacy for the Data Subject's personal data and for the Data Controller's policy may look arbitrary, but in some cases is driven by actual necessity, as in the following example:

passport.nationality $\notin$ Country-Blacklist

Revealing the name of the countries that are on the server's blacklist can lead to bigger issues than privacy trade-offs between the server and the user. Sometimes the exact access control policies can even contain trade secrets of the Data Controller, for example when they encode the application requirements for obtaining a credit card.

Sanitization can be of great help, but it must be noticed that such process can be performed on different levels in the policy, with different effects on the user/server interaction, as illustrated below.

If we sanitize the condition with the blacklist as we have done before, we obtain:

passport.nationality ∉ ----------

If a Data Subject submits her nationality and access is denied, even though the condition is sanitized in a way to protect the server's blacklist, the user will acquire partial knowledge about the list. It is easy to see how the whole list can be inferred by means of a series of trials. Let us then take the sanitization one step further, by concealing not only the last item in the condition, but also the symbol that comes before:

passport.nationality X ----------

With such a sanitized condition, the Data Subject only learns that to get access to the requested resource she must provide her nationality as certified by a passport, and that such information will have to satisfy a specific (unknown) property.

Let us remark that this condition is different from a simple request for the user's nationality, as in the latter case providing the information is sufficient to obtain access to the resource, whereas in the former situation further conditions apply.

Our policy language allows the policy designer to indicate which parts of the policy can be communicated to the user and which cannot. Namely, we allow for the specification of the disclosure level for each element in a condition, by inserting an optional Disclose attribute in the <AttributeMatchAnyOf>, <MatchValue>, and <Apply> elements. We refer to the syntax of these elements for more details.

It is up to the policy designer to make sure that he obtains the necessary information from the Data Subject in order to evaluate the policy, e.g. by adding explicit requirements to reveal sanitized attributes, by making sure that sanitized attributes are part of credentials that do not allow selective revealing of attributes, or by making sure that the attribute is already part of the XACML environment by some other means.

6.4 Attribute Types and Credential Types

6.4.1 General Approach

For a Data Controller to make access control decisions, he needs to distinguish the different kinds of credentials and attributes that he processes, as their meanings may differ depending on their type. For example, consider a university issuing digital student IDs and diploma certificates that have the same basic format (e.g., both contain the student's name and field of study). These credentials do of course have different purposes and therefore must be distinguished. This can be achieved by giving them different types. Indeed, relying only on the fact of who issued a credential may not be sufficient for determining its purpose and trustworthiness.

Both attributes and credential types are defined in ontologies and referred to by URIs. For attributes, an ontology defines the meaning associated to an attribute URI and its basic data type[1]; for credential types, an ontology defines the meaning associated to a credential type URI and the list of attributes that it contains. For example, the United Nations could design an ontology that standardizes the attribute http://www.un.org/Nationality as specifying the nationality of a credential bearer, encoded as a two-character ISO 3166 country code, and the credential type http://www.un.org/Passport as a digital equivalent to real-world passports, including amongst others a http://www.un.org/Nationality attribute.

The credential type may include attributes defined in the same ontology, as in the passport example above, or may borrow attributes from a different ontology for cross-compatibility. For example, if a national government defines a credential type for its its national ID cards, it could borrow the http://www.un.org/Nationality attribute from the United Nation’s ontology to encode the nationality of a citizen.

We allow inheritance among credential types. A subtype B of a credential type A contains all the attributes of A, but possibly restricts the allowed values of certain attributes, and possibly adds new attributes. Multiple inheritance is allowed, meaning that the subtype contains all the attributes of all supertypes.

All attributes are assumed to refer to the bearer of the credential, unless otherwise specified in the ontology. For example, if Facebook would issue friendship credentials, then the first name of the bearer of the credential could be encoded in an attribute http://facebook.com/FirstName, but the first name of a friend would have to be a different attribute, e.g. http://facebook.com/FriendFirstName, and cannot reuse the same URI http://facebook.com/FirstName. This is to avoid you from unintentionally revealing your http://facebook.com/FirstName by

1. See for example www.axschema.org for an existing community effort to standardize an ontology for basic identity attributes.

satisfying a policy requiring to you to reveal your friend's first name from a Facebook credential.[2]

We do not distinguish between credential meta-data and normal credential attributes. The credential issuer, the credential type, the authentication mechanism, etc. are treated as normal attributes. Those that are common to all credentials can be made mandatory attributes of a credential supertype, e.g. http://www.primelife.eu/Credential.

An access control policy can optionally specify the type of a credential that needs to be presented; if no type is specified, any credential containing the necessary attributes can be used to satisfy the policy.

6.4.2 Defining Credential Type Ontologies in OWL

A credential type is modeled as an OWL class (the class URI is used as credential type URI). This approach gives us full flexibility to define credential type as we piggyback on the very expressive OWL class definition mechanism. In particular, this class definition mechanism allows for defining unambiguously which attributes a certain credential type class has, the cardinality of the attributes (e.g., 'exactly one'), etc. We refer to the OWL documentation (most importantly the part on owl:Restriction) for more details on this.

For example, a movie ticket credential type could be modeled as OWL class with the URI 'http://movie.org/ticket'. Further, that type defines instances of that credential must have exactly one attribute specifying a row number (http://movie.org/rowNo).

To model inheritance among credential types we make use of OWL's subclassing mechanism (rdfs:subClassOf). To expres that credential type B extends type A, the class representing type B is modeled as subclass of A.

We define a root credential type with URI http://www.primelife.eu/Credential. It is the parent of all other credential types and contains one mandatory attribute, namely the credential's issuer (http://www.primelife.eu/issuer). We allow inheritance among credential types. A subtype B of credential type A contains all the attributes of A, plus possibly defines some additional attributes. Multiple inheritance is allowed, meaning that the subtype contains all the attributes of the super types.

All credentials have a type. This type is either the root credential type or any subtype of that root type.

Credentials are modeled in OWL as ordinary 'individuals', i.e., as RDF resources. In order to specify the credential type of a credential instance

2. For the same reason, the axschema.org ontology defines the home address and business address as different URIs (http://axschema.org/contact/postalAddress/home and http://axschema.org/contact/postalAddress/business, respectively), even though both are street addresses.

we utilize OWL’s standard typing mechanism (rdf:type). For example, to express that a credential instance is a movie ticket, then the credential instance has the rdf-type http://movie.org/ticket.

The attributes of a particular credential are modeled as OWL properties of the corresponding credential instance. For example, to express that a specific movie ticket is for seat number 5, then a specific movie ticket instance has the OWL property http://movie.org/row with value 5.

6.4.2.1 An Example OWL Ontology

We illustrate the above described approach on modeling credentials on the following concrete example:

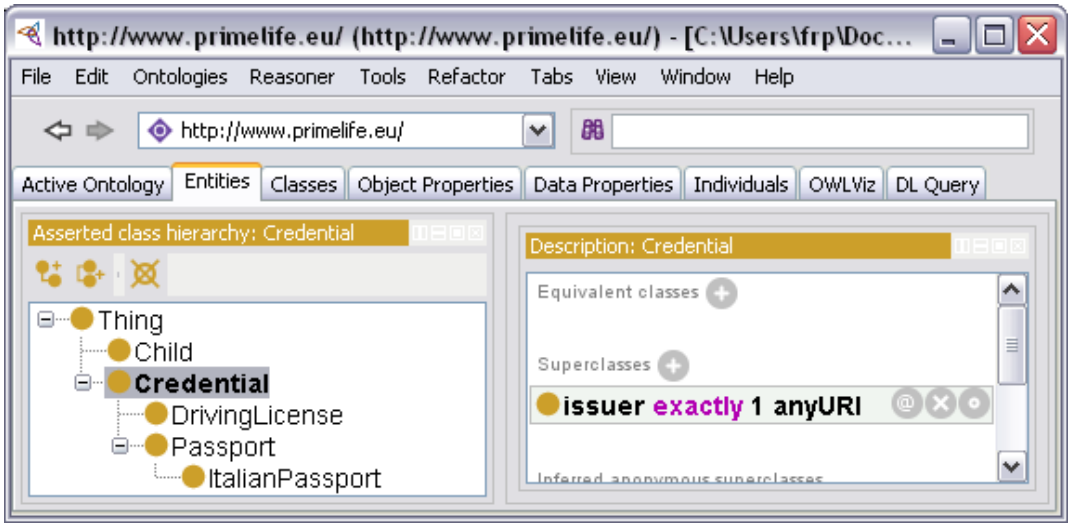
The United Nations publishes an ontology for passport credentials (http://www.un.org/Passport) that extends the root credential type (http://www.primelife.eu/Credential). It contains a citizen’s first name (http://www.un.org/firstName), last name (http://www.un.org/lastName), date of birth (http://www.un.org/dateOfBirth), and nationality (http://www.un.org/nationality).

In Italy children do not have their own passports and are therefore registered within their parents passports, therefore the Italian government publishes an ontology for passport credentials (http://www.governo.it/ItalianPassport) that extends the UN passport (http://www.un.org/Passport) by adding any number of child (http://www.governo.it/Child) attributes, which are composed attributes containing a child’s first name (http://www.governo.it/childFirstName) and last name (http://www.governo.it/childLastName). The reason for distinguishing between http://www.un.org/firstName and http://www.governo.it/childFirstName is to make sure that a parent can not pretend to have the name of its own child. Furthermore, to prevent mixing of children's attributes, we assume a mechanism in place that makes sure that only attributes from within the same child node can be released. Alternatively one would have to consider the ordering of children and model their attributes as http://www.governo.it/ firstChildFirstName, http://www.governo.it/firstChildLastName, http://www.governo.it/secondChildFirstName, etc.'

Finally, the European Union publishes an ontology for driving licences (http://ec.europa.eu/transport/DrivingLicence) that extends the root credential type (http://www.primelife.eu/Credential). It contains the owner’s first name (http://www.un.org/firstName), last name (http://www.un.org/lastName) and date of birth (http://www.un.org/ dateOfBirth), borrowed from the United Nation’s ontology, and any number of vehicle categories (http://ec.europa.eu/transport/ VehicleCategory).

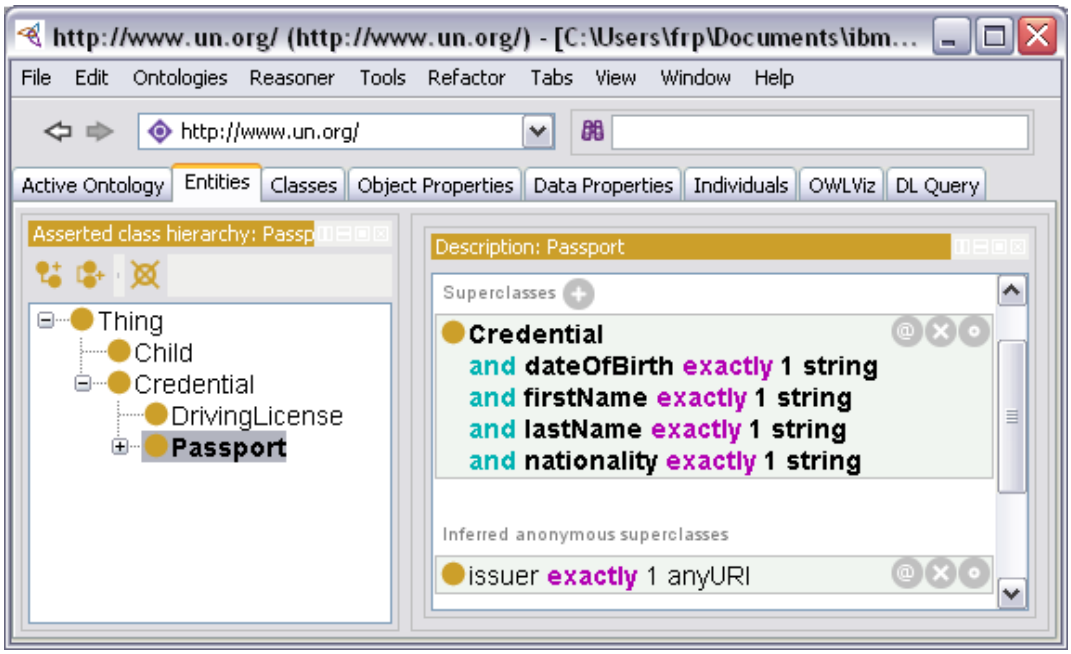
Now we show how the credential types are modeled with OWL. We will always give the screenshot of the Protégé OWL editor as well as the underlying RDF/XML syntax (as produced by the Protégé editor).

The following shows the definition of the PrimeLife root credential type as given in the ontology http://www.primelife.eu/:




```
<owl:Class rdf:about="Credential">
  <rdfs:subClassOf>
    <owl:Restriction>
        <owl:onProperty rdf:resource="issuer"/>
        <owl:qualifiedCardinality
          rdf:datatype="&xsd;nonNegativeInteger"> 1</owl:qualifiedCardinality>
        <owl:onDataRange rdf:resource="&xsd;anyURI"/>
    </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

The passport definition in the ontology http://www.un.org/ (where &pl; is a macro for the namespace http://www.primelife.eu/):



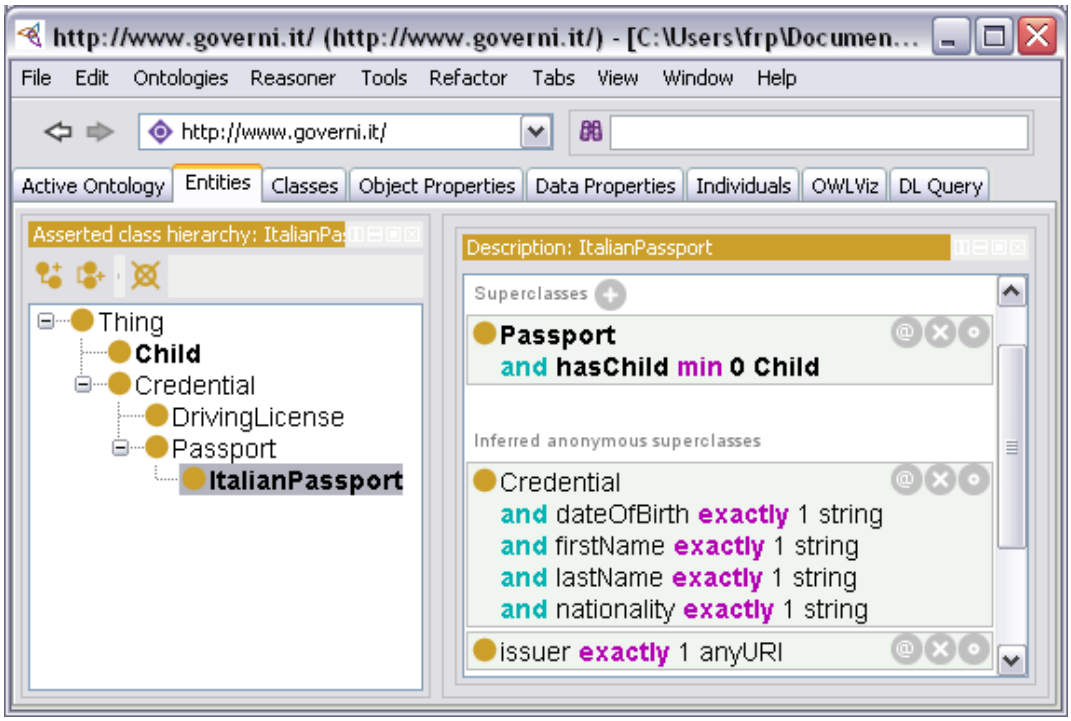

```
<owl:Class rdf:about="Passport">
  <rdfs:subClassOf>
    <owl:Class>
```

```
      <owl:intersectionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="&pl;Credential"/>
        <owl:Restriction>
          <owl:onProperty rdf:resource="dateOfBirth"/>
          <owl:qualifiedCardinality
            rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
          <owl:onDataRange rdf:resource="&xsd;string"/>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty rdf:resource="firstName"/>
          <owl:qualifiedCardinality
            rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
          <owl:onDataRange rdf:resource="&xsd;string"/>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty rdf:resource="lastName"/>
          <owl:qualifiedCardinality
            rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
          <owl:onDataRange rdf:resource="&xsd;string"/>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty rdf:resource="nationality"/>
          <owl:qualifiedCardinality
            rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
          <owl:onDataRange rdf:resource="&xsd;string"/>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </rdfs:subClassOf>
</owl:Class>
```

The definitions of the Italian passport and its child entry in the ontology http://www.governi.it/ (where &un; is a macro for http://www.un.org/):

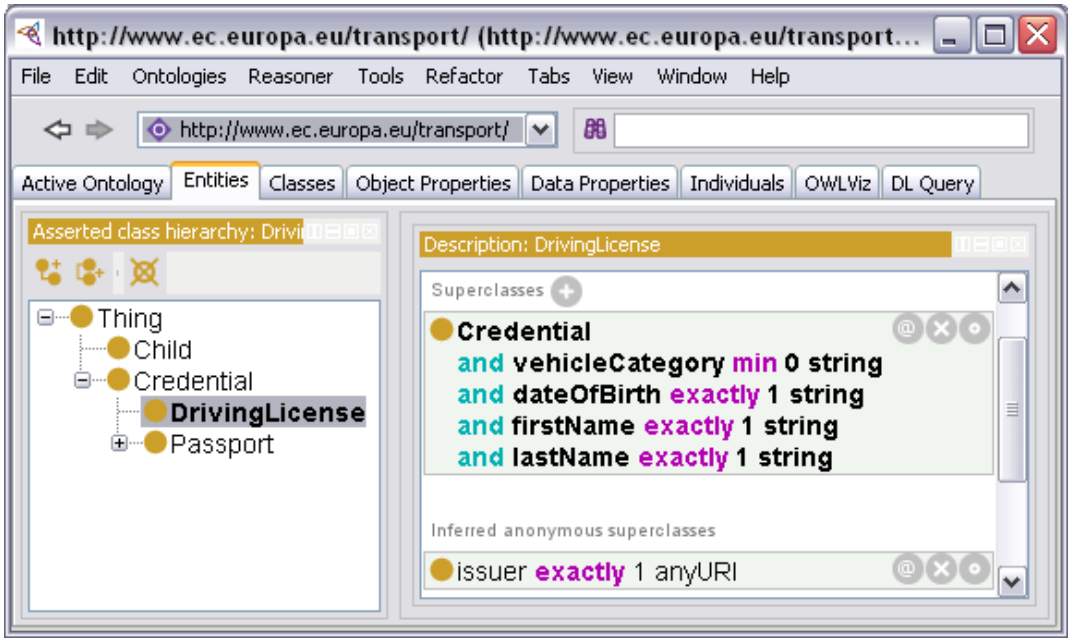

```
<owl:Class rdf:about="Child">
  <rdfs:subClassOf>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:onProperty rdf:resource="childFirstName"/>
          <owl:qualifiedCardinality
            rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
          <owl:onDataRange rdf:resource="&xsd;string"/>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty rdf:resource="childLastName"/>
          <owl:qualifiedCardinality
            rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
          <owl:onDataRange rdf:resource="&xsd;string"/>
         </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
```

```
  </rdfs:subClassOf>
</owl:Class>
```




```
<owl:Class rdf:about="ItalianPassport">
  <rdfs:subClassOf>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="&un;Passport"/>
          <owl:Restriction>
            <owl:onProperty rdf:resource="hasChild"/>
            <owl:onClass rdf:resource="Child"/>
            <owl:minQualifiedCardinality
              rdf:datatype="&xsd;nonNegativeInteger">0</owl:minQualifiedCardinality>
          </owl:Restriction>
        </owl:intersectionOf>
      </owl:Class>
  </rdfs:subClassOf>
</owl:Class>
```

The drivers license definition in the ontology http://www.ec.europa.eu/transport/ (where &pl; is a macro for http://www.primelife.eu/):



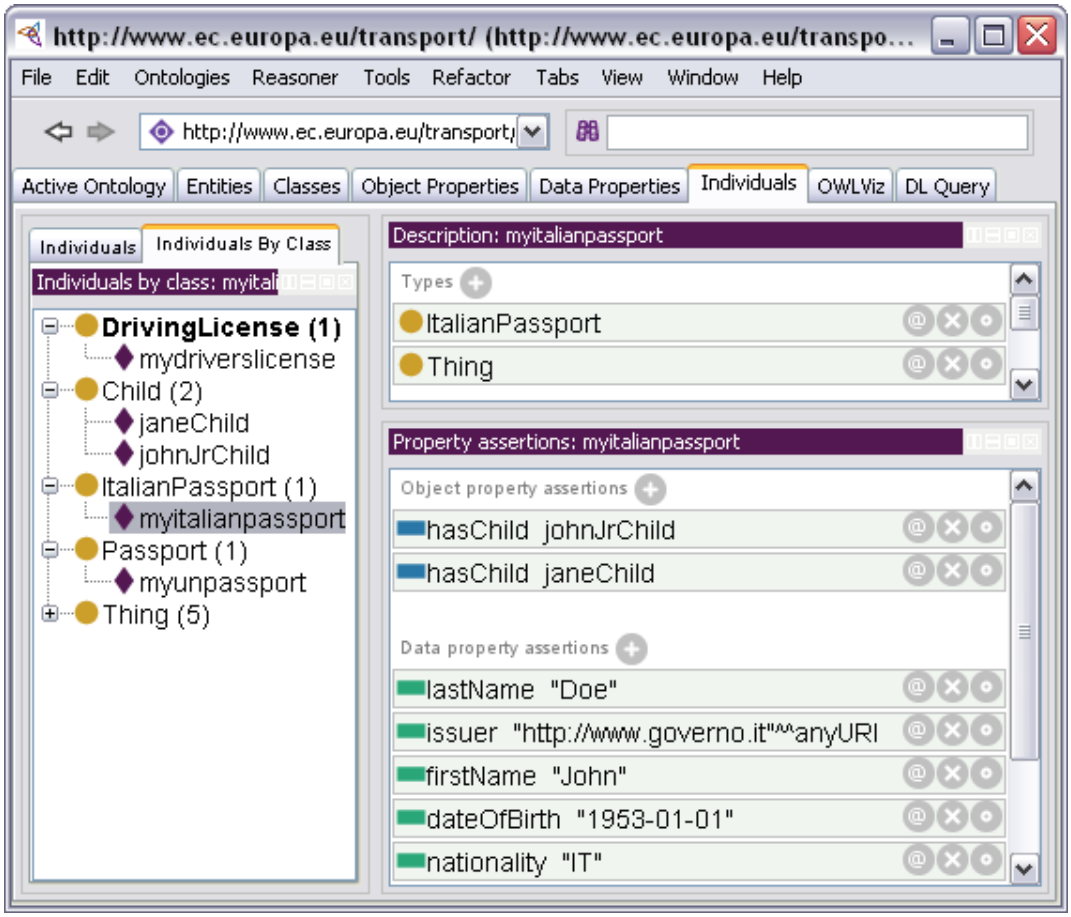

```
<owl:Class rdf:about="DrivingLicense">
  <rdfs:subClassOf>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="&pl;Credential"/>
        <owl:Restriction>
          <owl:onProperty rdf:resource="vehicleCategory"/>
          <owl:minQualifiedCardinality
            rdf:datatype="&xsd;nonNegativeInteger">0</owl:minQualifiedCardinality>
          <owl:onDataRange rdf:resource="&xsd;string"/>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty rdf:resource="&www;dateOfBirth"/>
          <owl:qualifiedCardinality
            rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
          <owl:onDataRange rdf:resource="&xsd;string"/>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty rdf:resource="&www;firstName"/>
          <owl:qualifiedCardinality
            rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
          <owl:onDataRange rdf:resource="&xsd;string"/>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty rdf:resource="&www;lastName"/>
          <owl:qualifiedCardinality
            rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
          <owl:onDataRange rdf:resource="&xsd;string"/>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </rdfs:subClassOf>
</owl:Class>
```

6.4.2.2 Example Credentials

The following shows an example wallet that contains concrete instances of a UN passport, an Italian passport with two children and a drivers license (where &govit;=http://www.governi.it, &un;=http://www.un.org/, &plife;=http://www.primelife.eu/, &ectp;=http://www.ec.europa.eu/transport/).

```
<owl:Thing rdf:about="janeChild">
  <rdf:type rdf:resource="&govit;Child"/>
  <govit:childLastName rdf:datatype="&xsd;string">Doe</govit:childLastName>
  <govit:childFirstName rdf:datatype="&xsd;string">Jane</govit:childFirstName>
</owl:Thing>

<owl:Thing rdf:about="johnJrChild">
  <rdf:type rdf:resource="&govit;Child"/>
  <govit:childLastName rdf:datatype="&xsd;string">Doe</govit:childLastName>
  <govit:childFirstName rdf:datatype="&xsd;string">John Jr.</govit:childFirstName>
</owl:Thing>

<ectp:DrivingLicense rdf:about="mydriverslicense">
  <rdf:type rdf:resource="&owl;Thing"/>
  <un:dateOfBirth rdf:datatype="&xsd;string">1953-01-01</un:dateOfBirth>
  <ectp:vehicleCategory rdf:datatype="&xsd;string">A</ectp:vehicleCategory>
  <ectp:vehicleCategory rdf:datatype="&xsd;string">B</ectp:vehicleCategory>
  <un:lastName rdf:datatype="&xsd;string">Doe</un:lastName>
  <un:firstName rdf:datatype="&xsd;string">John</un:firstName>
  <plife:issuer rdf:datatype="&xsd;anyURI">http://www.motorizzazioneroma.it</plife:issuer>
</ectp:DrivingLicense>

<owl:Thing rdf:about="myitalianpassport">
  <rdf:type rdf:resource="&govit;ItalianPassport"/>
  <un:dateOfBirth rdf:datatype="&xsd;string">1953-01-01</un:dateOfBirth>
  <un:lastName rdf:datatype="&xsd;string">Doe</un:lastName>
  <un:nationality rdf:datatype="&xsd;string">IT</un:nationality>
  <un:firstName rdf:datatype="&xsd;string">John</un:firstName>
  <plife:issuer rdf:datatype="&xsd;anyURI">http://www.governo.it</plife:issuer>
  <govit:hasChild rdf:resource="janeChild"/>
  <govit:hasChild rdf:resource="johnJrChild"/>
</owl:Thing>

<owl:Thing rdf:about="myunpassport">
  <rdf:type rdf:resource="&un;Passport"/>
  <un:dateOfBirth rdf:datatype="&xsd;string">1953-01-01</un:dateOfBirth>
  <un:nationality rdf:datatype="&xsd;string">BE</un:nationality>
  <un:lastName rdf:datatype="&xsd;string">Doe</un:lastName>
  <un:firstName rdf:datatype="&xsd;string">John</un:firstName>
  <plife:issuer rdf:datatype="&xsd;anyURI">http://www.fgov.be</plife:issuer>
</owl:Thing>
```

Here a screenshot of the OWL editor showing the individual ‘myitalianpassport’:



6.5 Example Policy

We first illustrate our proposed extensions through an example XACML policy that uses our extensions. The following policy requires that to access the resource (that is left unspecified here) the requestor needs to

- possess a passport or driver's license issued by the Swiss or Belgian governments, or any of their delegates;
- possess a Visa or American Express credit card;
- reveal the first name from the passport or driver's license;
- reveal the credit card number;
- spend the passport or driver's license credential once to a maximum of 10 on the domain http://www.mysite.com;
- sign the statement "I agree to the terms of service" using the passport or driver's license;
- be older than 18 according to the date of birth on the passport or driver's license;

- not use a Russian credit card to obtain access. This last requirement is considered confidential and will be sanitized from the policy that is transmitted to the Data Subject.

```
<?xml version="1.0" encoding="utf-8"?>
<PolicySet
  xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-11"
  xmlns:pl="http://primelife.eu"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  PolicySetId="PrimeLifePolicySet"
  PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable">

  <Target>
    <!-- ... -->
  </Target>

  <Policy PolicyId="PrimeLifePolicy"
    RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
    <Target />

    <pl:Rule RuleId="PrimeLifeRule" Effect="Permit">
      <Condition>
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
          <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-less-than-or-equal">
            <pl:CredentialAttributeDesignator CredentialId="#pp"
              AttributeId="urn:BithDate" DataType="http://www.w3.org/2001/XMLSchema#date" />
            <Apply
              FunctionId="urn:oasis:names:tc:xacml:3.0:function:date-subtract-yearMonthDuration">
              <EnvironmentAttributeDesignator
                AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-date"
                DataType="http://www.w3.org/2001/XMLSchema#date" />
              <AttributeValue DataType="xs:duration">P18Y</AttributeValue>
            </Apply>
          </Apply>
          <pl:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:not"
            Disclose="attributes-only">
            <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
              <pl:CredentialAttributeDesignator CredentialId="#pp"
                AttributeId="http://www.bankingstandards.org/Country" DataType="xs:string" />
              <AttributeValue DataType="xs:string">RU</AttributeValue>
            </Apply>
          </pl:Apply>
        </Apply>
      </Condition>

      <pl:CredentialRequirements>
        <pl:Credential CredentialId="#pp">
          <pl:AttributeMatchAnyOf AttributeId="pl:Iusser">
            <pl:MatchValue MatchId="pl:delegatee-of"
              DataType="xs:anyURI">http://www.admin.ch</pl:MatchValue>
            <pl:MatchValue MatchId="pl:delegatee-of"
              DataType="xs:anyURI">http://www.fgov.be</pl:MatchValue>
          </pl:AttributeMatchAnyOf>
          <pl:AttributeMatchAnyOf AttributeId="pl:Type">
            <pl:MatchValue MatchId="pl:subclass-of"
              DataType="xs:anyURI">http://www.un.org/Passport</pl:MatchValue>
            <pl:MatchValue MatchId="pl:subclass-of"
              DataType="xs:anyURI">http://ec.europa.eu/transport/DrivingLicence</pl:MatchValue>
          </pl:AttributeMatchAnyOf>
        </pl:Credential>

        <pl:Credential CredentialId="#cc">
          <Condition>
            <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
              <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:or">
                <Apply FunctionId="pl:delegatee-of">
                  <pl:CredentialAttributeDesignator CredentialId="#cc"
                  AttributeId="pl:Iusser" DataType="xs:anyURI" />
                  <AttributeValue DataType="xs:anyURI">http://www.visa.com</AttributeValue>
                </Apply>
                <Apply FunctionId="pl:delegatee-of">
```

```
                  <pl:CredentialAttributeDesignator CredentialId="#cc"
                    AttributeId="pl:Iusser" DataType="xs:anyURI" />
                  <AttributeValue
                    DataType="xs:anyURI">http://www.americanexpress.com</AttributeValue>
                </Apply>
              </Apply>
              <Apply FunctionId="pl:subclass-of">
                <pl:CredentialAttributeDesignator CredentialId="#cc"
                  AttributeId="pl:Type" DataType="xs:anyURI" />
                <AttributeValue
                  DataType="xs:anyURI">http://www.bankingstandards.org/Creditcard</AttributeValue>
              </Apply>
            </Apply>
          </Condition>
        </pl:Credential>
      </pl:CredentialRequirements>

      <pl:ProvisionalActions>
        <pl:ProvisionalAction ActionId="pl:RevealUnderDHP">
          <AttributeValue
            DataType="xs:anyURI">http://www.un.org/FirstName</AttributeValue>
          <AttributeValue DataType="xs:anyURI">#dhpl</AttributeValue>
          <AttributeValue DataType="xs:anyURI">#pp</AttributeValue>
        </pl:ProvisionalAction>
        <pl:ProvisionalAction ActionId="pl:RevealUnderDHP">
          <AttributeValue
            DataType="xs:anyURI">http://www.bankingstandards.org/CreditcardNumber</AttributeValue>
          <AttributeValue DataType="xs:anyURI">#dhp2</AttributeValue>
          <AttributeValue DataType="xs:anyURI">#cc</AttributeValue>
        </pl:ProvisionalAction>
        <pl:ProvisionalAction ActionId="pl:Spend">
          <AttributeValue DataType="xs:anyURI">#pp</AttributeValue>
          <AttributeValue DataType="xs:integer">1</AttributeValue>
          <AttributeValue DataType="xs:integer">10</AttributeValue>
          <AttributeValue DataType="xs:anyURI">http://www.ysite.com</AttributeValue>
        </pl:ProvisionalAction>
        <pl:ProvisionalAction ActionId="pl:Sign">
          <AttributeValue
            DataType="xs:string">I agree to the terms of service.</AttributeValue>
        </pl:ProvisionalAction>
      </pl:ProvisionalActions>
    </pl:Rule>

  </Policy>
</PolicySet>
```

6.6 Syntax and Description

In the following we describe a number of extensions to XACML 3.0 to enable the expression of credential-based access control. We use the prefix xacml: to denote the XACML 3.0 namespace urn:oasis:names:tc:xacml:3.0:core:schema:wd-11.

The extensions we make are mainly in the elements <xacml:Rule>, <xacml:Policy>, and <xacml:PolicySet> elements. In particular,

(1)the elements <CredentialRequirements> and <ProvisionalActions> are introduced as child elements to the <xacml:Rule> element, and

(2)a <CredentialAttributeDesignator> element is introduced in the <xacml:Expression> element substitution group, which enables this new element to be used within the <xacml:Condition> element.

However, to allow for short policies, elements <CredentialRequirements> and <ProvisionalActions> can also be

included within <xacml:Policy> or <xacml:PolicySet>. This is a shorthand notation that is equivalent to repeating these credential requirements in each descendant <xacml:Rule> element. The <CredentialRequirements> and <ProvisionalActions> elements within <xacml:Rule> must therefore be considered in a logical conjunction with the <CredentialRequirements> and <ProvisionalActions> that are possibly specified in antecedent <xacml:Policy> or <xacml:PolicySet> elements.

Therefore, the xacml:PolicySetType, xacml:PolicyType and xacml:RuleType are extended with the following elements:

<CredentialRequirements> [Optional]

Contains conjunctive declarations of and requirements on the credentials that the Data Subject has to present. The credential requirements must be fulfilled in conjunction with the rule's condition for the rule to be assigned its Effect value.

If the access requester did not already provide a proof for fulfillment of these requirements, then they are communicated to the access requester such that she can create and provide such proof.

<ProvisionalActions> [Optional]

The actions that an access requester has to perform prior to being granted access. Typical actions include revealing of credential attributes, signing of statements, or spending of credentials.

6.6.1 Element <CredentialRequirements>

The <CredentialRequirements> element is the container for a sequence of credential declarations which are subsequently needed for specifying conditions on those credentials. Each individual credential is declared within a <Credential> element.

An XACML rule with an embedded <CredentialRequirements> element is only satisfied (i.e., will only have its stated Effect) in case in addition to the standard <xacml:Target> and <xacml:Condition> statements, all requirements in the <CredentialRequirements> element have been satisfied. There can be at most one <CredentialRequirements> child element per <xacml:Rule>, <xacml:Policy>, or <xacml:PolicySet> element. In order to combine different sets of credential requirements, they have to be embedded in different <xacml:Rule> elements which are then combined with the standard XACML combining algorithms.

The <CredentialRequirements> element contains the following attributes and elements:

<Credential> [Required, any number]

A conjunctive sequence of <Credential> elements describing the credentials that the Data Subject is required to present.

6.6.2 Element <ProvisionalActions>

This element is a container for all the provisional actions that an access requester must fulfill prior to being granted access. Depending on the type of action, the requestor may have to provide some form of proof that the provisional action was actually performed.

A first list of supported action types is provided in this profile, but the policy mechanism is designed for new action types to be added later without changing the schema, much like the extensible functions in the core of XACML.

The <ProvisionalActions> element contains the following attributes and elements:

<ProvisionalAction> [Required, any number]

A conjunctive sequence of <ProvisionalAction> elements each of which describes a provisional action to fulfill.

6.6.3 Element <Credential>

This element is used to (1) declare a credential that has to be held by an access requester and optionally to (2) specify conditions that concern only this particular credential.

When a credential is declared, a credential identifier CredentialId is defined. This credential identifier can subsequently be used in <CredentialAttributeDesignator> elements within the rule's <Condition> element to specify further conditions on the attribute values of the credential. The idea is that the conditions specified within the rule's <Condition> element concern mainly cross-requirements between credential attributes out of different declared credentials, whereas the conditions specified directly within the <Credential> element concern requirements that are specific to the one credential that is being declared.

For every declared credential, an access requester has to prove possession of an atomic credential satisfying both the conditions specified directly within the <Credential> element itself and those specified in the rule's <Condition> element.

To specify conditions directly within the <Credential> element we specify two possible ways of doing so, of which one should be chosen for each <Credential> element:

1. By including a standard <xacml:Condition> element inside <Credential> element, including the <CredentialAttributeDesignator> element that we describe in this document, but with the restriction that only attributes from this credential can be referred to, i.e., the CredentialId attribute of each <CredentialAttributeDesignator> element inside the

<xacml:Condition> is equal to the CredentialId of the parent <Credential> element.

2. By including a dedicated <AttributeMatchAnyOf> element within which conditions on the credential can be expressed in a specific pattern that may often be needed.

The first option offers the highest expressivity by allowing essentially any combination of requirements on the attributes of the credential in question. The second option can only express requirements that adhere to a specific structure, namely matching attributes of the credential against a list of candidate values, but possibly allows for more efficient parsing and more efficient credential selection (i.e., testing which credentials can be used to satisfy the policy) at the Data Subject's side. Clearly, any set of requirements that can be expressed using an <AttributeMatchAnyOf> element can equivalently be expressed in a <xacml:Condition> element, much like any condition in the <xacml:Target> element could equivalently be specified in the <xacml:Condition> element. We also mention that comparing a single attribute to a list of values can be done more compactly using XACML's bag functionalities, at least as long as the same matching algorithm is used for the whole list.

The <Credential> element contains the following attributes and elements:

CredentialId [Required]

This attribute specifies an identifier for the declared credential. The credential identifier has to be unique within the given <xacml:PolicySet>.

<xacml:Condition> [Optional] (1st option)

A predicate on this credential's attributes that has to be satisfied in conjunction with the conditions specified in the <xacml:Condition> element of the parent <xacml:Rule>. However, all <CredentialAttributeDesignator> elements appearing within it have to use the same CredentialId attribute as the parent <Credential> element, i.e., only conditions on attributes of this credential can be expressed here.

<AttributeMatchAnyOf> [Optional, any number] (2nd option)

A conjunctive sequence of conditions on this credential's attributes which have to be satisfied together with the rule's conditions in <Condition>. Since the sequence is conjunctive, all <AttributeMatchAnyOf> elements have to evaluate to true for the requirements to be fulfilled.

6.6.4 Element <AttributeMatchAnyOf>

An <AttributeMatchAnyOf> element is used for matching a given attribute with a list of values, whereby for every list element an individual matching algorithm is used. This element evaluates to true if at least one list element successfully matches against the given value.

Although in principle any attribute can be matched, the <AttributeMatchAnyOf> construction is particularly useful for providing lists of accepted credential types or issuers. Clearly, if no credential types are explicitly specified, then any credential type that contains the necessary attributes can be used to satisfy the policy. If no issuers are satisfied, then credentials by any issuer are accepted (including self-stated credentials/attributes).

The element <AttributeMatchAnyOf> contains the following attributes and elements:

AttributeId [Required]

The name of the attribute in this credential that is matched against the list of values.

Disclose [Optional]

The type of policy disclosure used for this element when this policy is sent to the Data Subject. Possible values are “yes”, “no”, and “attributes-only”. When the attribute is omitted, the default value “yes” is assumed.

When set to “yes”, this <AttributeMatchAnyOf> element is sent unmodified to the Data Subject.

When set to “no”, this <AttributeMatchAnyOf> element is sanitized by means of the following substitutions:

- the value of AttributeId is replaced with “undisclosed”, and
- each <MatchValue> child element within this <AttributeMatchAnyOf> element is replaced with an <UndisclosedExpression> element.

When set to “attributes-only”, then only the latter substitution is performed, i.e., all <MatchValue> child elements are replaced with an <UndisclosedExpression> element.

See Section 6.3 for more details on policy sanitization.

<MatchValue> [Required, any number]

A disjunctive sequence of values that are individually matched against the attribute specified by AttributeId using the matching algorithm specified within the <MatchValue> element itself.

6.6.5 Element <UndisclosedExpression>

This element can only occur in a policy that is sent to the Data Requestor. It acts as a placeholder to indicate that a credential condition was omitted due to policy sanitization. See Section 2 for more information regarding policy sanitization.

6.6.6 Element <MatchValue>

This element contains a literal value against which the given attribute (specified with AttributeId in the parent <AttributeMatchAnyOf> element) is matched as well as the matching algorithm that is used.

The element <MatchValue> contains the following attributes and elements:

MatchId [Required]

The name of the matching algorithm that is used to match the attribute with the literal. Here, one can use one of the functions defined in Section 4, or any function defined in the XACML 3.0 standard that takes two arguments of the correct datatypes (the first argument of the same type of the attribute, the second of the type specified in the DataType argument) and that evaluates to a Boolean.

DataType [Required]

The data type of the literal value against which the attribute will be matched. Any of the built-in data types of XACML 3.0 (see Section 10.2.7 of the specification) can be used here.

Disclose [Optional]

The type of policy disclosure used for this element when this policy is sent to the Data Subject. Possible values are “yes” and “no”. When the attribute is omitted, the default value “yes” is assumed. Note that the value “attributes-only” is not allowed here.

When set to “yes”, this <AttributeMatchAnyOf> element is sent unmodified to the Data Subject. When set to “no”, this <AttributeMatchAnyOf> element is replaced with an <UndisclosedExpression> element before the policy is sent to the Data Subject.

See Section 2 for more details on policy sanitization.

6.6.7 Element <ProvisionalAction>

This element describes a single provisional action that needs to be fulfilled in order to satisfy a rule. The schema of the <ProvisionalAction> element consciously kept independent of the particular provisional

action that needs to be fulfilled, in order to keep the door open to possible future extensions with new provisional action types. Its schema is reminiscent of the <xacml:Apply> element, which is also kept independent of the particular function being applied to allow extensions with new function definitions.

The <ProvisionalAction> element contains the following attributes and elements:

ActionId [Required]

The identifier (URI) of the action to be performed. See below for a list of built-in actions.

<xacml:Expression> [Optional, any number]

Arguments of the action, which may include other functions. The semantics of the argument depend on the particular action being performed.

This profile supports the following list of provisional actions:

http://www.primelife.eu/Reveal

This action requires the Data Subject to reveal an attribute. The attribute could be part of one of her credentials, or could be a self-stated, uncertified attribute. The action takes one or two arguments of data-type http://www.w3.org/2001/XMLSchema#anyURI. The first (mandatory) argument is the URI of the attribute to be revealed. The second (optional) argument is a URI referring to the CredentialId of the credential that contains the attribute. If the parent <ProvisionalActions> element occurs within a <xacml:Rule>, then the URI could refer to the CredentialId of any <Credential> defined within either the <xacml:Rule> itself or within the parent <xacml:Policy> or <xacml:PolicySet>. If it occurs inside a <xacml:Policy> or <xacml:PolicySet> but not inside a <xacml:Rule>, then it can only refer to a <Credential> appearing within the same <xacml:Policy> or <xacml:PolicySet> element.

http://www.primelife.eu/RevealUnderDHP

This action requires the Data Subject to reveal an attribute while specifying the data handling policy that will be applied to the attribute after it is revealed. The attribute could be part of one of her credentials, or could be a self-stated, uncertified attribute. The action takes two or three arguments of data-type http://www.w3.org/2001/XMLSchema#anyURI. The first (mandatory) argument is the URI of the attribute to be revealed. The second (mandatory) argument is a URI referring to the data handling policy under which the attribute has to be revealed. *To do: specify exactly where (element, attribute) this DHP URI links to.* The third (optional) argument is a URI referring to the CredentialId of the credential that contains the attribute. If the parent <ProvisionalActions> element

occurs within a <xacml:Rule>, then the URI could refer to the CredentialId of any <Credential> defined within either the <xacml:Rule> itself or within the parent <xacml:Policy> or <xacml:PolicySet>. If it occurs inside a <xacml:Policy> or <xacml:PolicySet> but not inside a <xacml:Rule>, then it can only refer to a <Credential> appearing within the same <xacml:Policy> or <xacml:PolicySet> element.

http://www.primelife.eu/RevealTo

This action requires the requestor to reveal an attribute to an external third party. The attribute could be part of one of her credentials, or could be a self-stated, uncertified attribute. The action takes two or three arguments of data-type http://www.w3.org/2001/XMLSchema#anyURI. The first (mandatory) argument is the URI of the attribute to be revealed. The second (mandatory) argument is the URI defining the third party to whom the attribute will be revealed. The third (optional) argument is a URI referring to the CredentialId of the credential that contains the attribute. If the parent <ProvisionalActions> element occurs within a <xacml:Rule>, then the URI could refer to the CredentialId of any <Credential> defined within either the <xacml:Rule> itself or within the parent <xacml:Policy> or <xacml:PolicySet>. If it occurs inside a <xacml:Policy> or <xacml:PolicySet> but not inside a <xacml:Rule>, then it can only refer to a <Credential> appearing within the same <xacml:Policy> or <xacml:PolicySet> element.

http://www.primelife.eu/RevealToUnderDHP

This action requires the Data Subject to reveal an attribute to an external third party while specifying the data handling policy that will be applied to the attribute after it is revealed. The attribute could be part of one of her credentials, or could be a self-stated, uncertified attribute. The action takes three or four arguments of data-type http://www.w3.org/2001/XMLSchema#anyURI. The first (mandatory) argument is the URI of the attribute to be revealed. The second (mandatory) argument is the URI defining the third party to whom the attribute will be revealed. The third (mandatory) argument is a URI referring to the data handling policy under which the attribute has to be revealed. *To do: specify exactly where (element, attribute) this DHP URI links to.* The fourth (optional) argument is a URI referring to the CredentialId of the credential that contains the attribute. If the parent <ProvisionalActions> element occurs within a <xacml:Rule>, then the URI could refer to the CredentialId of any <Credential> defined within either the <xacml:Rule> itself or within the parent <xacml:Policy> or <xacml:PolicySet>. If it occurs inside a <xacml:Policy> or <xacml:PolicySet> but not inside a <xacml:Rule>, then it can only refer to a <Credential> appearing within the same <xacml:Policy> or <xacml:PolicySet> element.

http://www.primelife.eu/Sign

This action requires the requestor to sign a statement before accessing the resource. How the signature is implemented depends on the underlying technology, but carries the semantics that a verifier can check later that some Data Subject satisfying the policy explicitly agreed to the statement. The action takes a single argument of data-type http://www.w3.org/2001/XMLSchema#string describing the statement that needs to be signed.

http://www.primelife.eu/Spend

This action requires the requestor to “spend” one of her credentials, thereby imposing restrictions on now many times the same credential can be used in an access request. The action takes four mandatory arguments. The first is of data-type http://www.w3.org/2001/XMLSchema#anyURI and contains the CredentialId of the credential that has to be spent. If the parent <ProvisionalActions> element occurs within a <xacml:Rule>, then the URI could refer to the CredentialId of any <Credential> defined within either the <xacml:Rule> itself or within the parent <xacml:Policy> or <xacml:PolicySet>. If it occurs inside a <xacml:Policy> or <xacml:PolicySet> but not inside a <xacml:Rule>, then it can only refer to a <Credential> appearing within the same <xacml:Policy> or <xacml:PolicySet> element.

The second and third arguments are of data-type http://www.w3.org/2001/XMLSchema#integer. The second argument is the number of units that have to be spent for this access; the latter is the spending limit, i.e., the maximum number of units that can be spent with this credential on the same scope.

The fourth argument is of data-type http://www.w3.org/2001/XMLSchema#string and defines the scope on which the credential has to be spent. If spending is to be limited globally, the fourth argument should be set to the fixed URI http://www.primelife.eu/Spend/GlobalScope.

6.6.8 Element <CredentialAttributeDesignator>

This element is similar to the <xacml:AttributeDesignator> element, which inserts the value of a given attribute from the context, but with the major difference that it also contains a reference to the specific credential from which the attribute has to be taken. The <CredentialAttributeDesignator> element evaluates to the value of a specific attribute (referred to by the AttributeId) from a specific credential (referred to by the CredentialId). This element is intended to be used within a <xacml:Condition> element to express requirements on the attribute values of specific credentials an access requester must hold. To enable the <CredentialAttributeDesignator> element to be used within the <xacml:Condition> element, it is within the <xacml:Expression> element substitution group.

In addition to the standard attributes of the <xacml:AttributeDesignator> element, the element <CredentialAttributeDesignator> contains the following attribute:

CredentialId [Required]

This attribute specifies the identifier of the credential from which the attribute should be taken. The identifier refers to the CredentialId attribute of one of the <Credential> elements within the same <xacml:Rule> element or in one of the ancestor <xacml:Policy> or <xacml:PolicySet> elements.

6.6.9 Element <Apply>

This element inherits the scheme of the <xacml:Apply> element, which is used to apply a function to a list of arguments, but takes one additional attribute Disclose to allow for policy sanitization:

Disclose [Optional]

The type of policy disclosure used for this element when this policy is sent to the Data Subject. Possible values are “yes”, “no”, and “attributes-only”. When the attribute is omitted, the default value “yes” is assumed.

When set to “yes”, this <Apply> element is sent unmodified to the Data Subject.

When set to “no”, this <Apply> element is replaced with an empty <UndisclosedExpression> element.

When set to “attributes-only”, this <Apply> element is sanitized by means of the following substitutions:

- the value of FunctionId is replaced with the value “undisclosed”, and
- the arguments are removed and replaced with the list of <xacml:AttributeDesignator> and <CredentialAttributeDesignator> elements that they contain. The list is encoded as a flat sequence of elements, even hiding the nesting structure in case the arguments contain further <Apply> elements.

See Section 2 for more details on policy sanitization.

6.7 Matching Functions

We introduce two new functions for matching credential types and credential issuers. These functions can be used as a FunctionId within any <xacml:Apply> element or as a MatchId in any <MatchValue> element.

http://www.primelife.eu/delegatee-of

This function shall take two arguments, the first of data-type http://www.w3.org/2001/XMLSchema#string and the second of data-type http://www.w3.org/2001/XMLSchema#anyURI and shall return an http://www.w3.org/2001/XMLSchema#boolean. The first argument shall encode a comma-separated list of URIs. The function shall return "True" if and only if the URI in the second argument occurs in the list given by the first argument. Otherwise, it shall return “False”. The main purpose of this function is to check whether a given issuer appears in the list of hierarchical issuers of a credential, as in a credential chain.

http://www.primelife.eu/subclass-of

This function shall take two arguments of data-type http://www.w3.org/2001/XMLSchema#anyURI and shall return an http://www.w3.org/2001/XMLSchema#boolean. The function shall return "True" if and only if according to the trusted ontologies the URI in the first argument refers to a credential type that is a subtype of the credential type defined by the second argument. Otherwise, it shall return “False”.

7 Protocols and Message Flows

This report sketches the message flow between Data Subject and Data Controller, and between a Data Controller and a downstream Data Controller. It doesn't define bindings to protocols such as HTTP, which is left for future work. Finally, there is description for how SAML 2.0 can be extended for use as protocol language in credential-based access control.

7.1 Generic Policy

As soon as a user agent starts an HTTP session with a server, the latter is able to collect some information that can be considered as private (PIIs). Here is a non exhaustive list of the information that can be collected by a server receiving an HTTP GET request:

1. User client hostname. The hostname and the IP address of the user agent requesting access to this site. From this IP address the server is able to deduce the location of the requester (country, city, street …) and the domain (company name, ISP, operator …).

2. HTTP header, "user agent." The user agent information includes the type of browser (IE, Mozilla, Opera …), its version, and the operating system on which that the browser is running.

3. HTTP header “content-language” The Content-Language entity-header field describes the natural language(s) of the intended audience for the enclosed entity. Note that this might not be equivalent to all the languages used within the entity-body.

4. HTTP header, "referrer." The referrer specifies the previous web pages from which the user accessed the current web page. It gives information about the navigation history of the user agent.

5. Query string of the URI. Anything after the question mark in a URI. If the user agent accessed the web page through a search engine (Google, Yahoo …) the query string of the URI can contains the key words typed by the user in the search engine.

What is the advantage of matching privacy policies, if the data controller already has some PIIs related to the user agent?

For this reason we propose to use what we call a generic policy GP that expresses a detailed data handling policy on the navigation information (“implicit” PII) that can be collected by the Data Controller on the beginning the interaction with the user agent. This GP is available in a well known location called "safe zone". Every server has it's own safe zone that can be accessed by any user agent that wants to control his navigation information during the interaction with a server. Server's who offer such safe zone implicitly agree on not using and storing the data collected for accessing the safe zone. To circumvent contacting the servers directly for obtaining their GPs, a central certified repository, e.g., the PrimeLife website, could cache the server's GPs and then users obtain a server's GP from the PrimeLife repository rather than the server itself. Clearly, this central repository would then learn about a user's interest in transacting with the servers whose GPs she obtained.

- We identified the list of URI related to the navigation information that can be targeted by a GP:
- Hostname and IP address : primelife.navigation.host
- “User Agent” header
    - Browser : primelife.navigation.user_agent.browser
    - Version: primelife.navigation.user_agent.version
    - Operating system : primelife.navigation.user_agent.operating_system
- Content language : primelife.navigation.content_language
- Referrer : primelife.navigation.referrer.URI
- Query string : primelife.navigation.referrer.URI.query

7.2 Data Subject and Data Controller

In the scenario described in Section 6.2.1 on credential-based access control a party (called Data Controller) asks for personal data from a Data Subject in exchange for granting her access to one of his resources. To account for cases where the resource is on it's turn personal data for the Data Controller we generalize the scenario described in 6.2.1. In this generalized scenario we consider the resource of interest as potential personal data for the Data Controller who may protect this data not only with an access control policy but may also have data handling preferences attached to it. Note that the names Data Subject and Data Controller may be confusing in situations where personal data is requested in exchange for access permission to other personal data. These therms should be considered as dynamic roles rather than static names for interaction partners: a party providing personal data is usually referred to as Data Subject; a party consuming/processing/controlling is referred to as Data Controller.

Our generalized scenario is applicable in both of the following cases: 1) the case where a party A wants to access some resource of party B and therefore has to provide personal data to B in exchange for the access 2) the case where the personal data of A stored at B is the resource of interest for a party C and C wants to access this resource at B - and possibly therefore has to provide (personal) data to B in exchange for the access

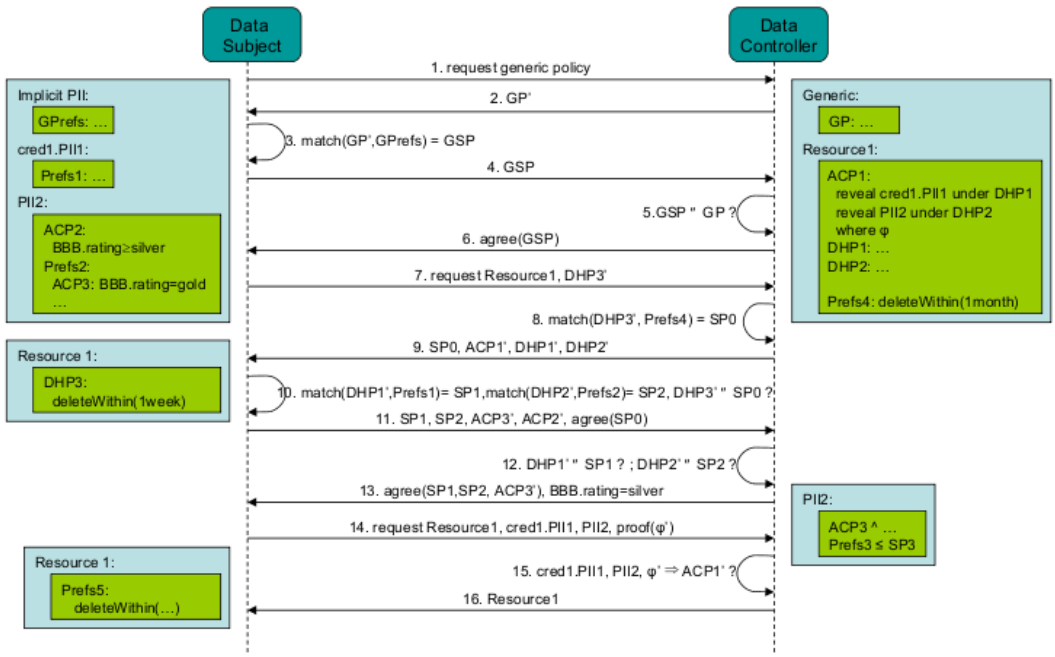


Figure 13: Message flow between Data Subject and controller

In Figure 12 we sketch the typical interaction between a Data Subject who wants to access a resource Resource1 hosted at the Data Controller. The former party is called Data Subject as it has to provide her personal data in exchange for being granted access to the resource, and the latter party is called Data Controller as it subsequently processes/controls the data. However, note that in this example the Data Controller is also providing (personal) data to the Data Subject: a certification for having a Better Business Bureau label. With respect to this BBB label the Data Subject acts as Data Controller and the Data Controller as Data Subject. Both sides run a PrimeLife policy engine: the Data Collector to protect access to the resources that its hosts and the Data Subject to protect his or her personal data. This personal data could be certified information contained in one of the Data Subject's credentials (e.g., the data of birth on her passport), or could be any other type of uncertified information that the Data Subject considers sensitive (e.g., pictures, videos).

For the sake of illustration, the Data Subject's policy in Figure 13 is a rather advanced one containing an access control policy ACP2 and a downstream access control policy ACP3 over its personal data. In many practical cases the Data Subject's policy will not contain (downstream) access control restrictions, which may simplify the interaction.

Furthermore, one could reduce the number of roundtrips in the interaction by not waiting for explicit agreement with sticky policies, which should follow automatically if the policy matching step was done properly. In this case, one could piggy-back flows 4 and 7 together, as well as flows 10 and 15, reducing the total number of roundtrips from 5 to 3.

Step 1. The Data Subject sends a request message to a well-defined location, the so-called “safe zone”, to obtain the Data Controller’s generic policy. The generic policy GP expresses a detailed data handling policy on “implicit” personal data, meaning personal data that is revealed by the mere interaction with the Data Collector, such as the Data Subject’s IP address, the connection time, or the list of resources that it accesses.

Step 2. The Data Controller sends back GP’, which may be identical to his generic policy GP, but may also be a partially evaluated version thereof, for example after replacing certain attributes that appear in GP (such as the current date) with their values at the moment of evaluation. The generic policy is expressed by means of the same obligation language as for “explicit” personal data.

Obviously, the Data Subject already reveals a considerable amount of implicit personal data by simply setting up a connection with the Data Controller. Alternatively, the Data Subject may therefore obtain the generic policy from a third party, e.g., a search engine. The third party can crawl safe zones of multiple Data Collectors and cache their generic policies as a service to its users.

Step 3. The Data Subject matches the Data Controller’s generic policy GP’ against its own generic preferences GPrefs. Her policy contains the generic preferences as preferences associated to abstract resources that represent her implicit personal data, expressed in the same obligation language as for explicit personal data. The matching procedure is identical to that for explicit personal data. We refer to the resulting set of matching obligations as the generic sticky policy (GSP), if a match is found.

Step 4. The Data Subject sends the resulting generic sticky policy to the Data Controller.

In the scenario sketched here, it is the Data Controller who first sends his generic policy to the Data Subject, and the Data Subject who performs the matching with respect to her own preferences. Alternatively, these roles could be inverted: the Data Subject could append her preferences GPrefs to her request in Step 1, so that the Data Controller can match these against his own policy GP to derive the generic sticky policy GSP. A possible advantage of this approach is that, by making the first move in the policy negotiation, the Data Subject may have a stronger influence on the final generic sticky policy. (Whether this is the case depends on the matching mechanism.) A possible disadvantage is that the Data Subject’s preferences may be very specific, so that revealing her preferences makes her visits easily linkable.

Step 5. The Data Controller checks whether the proposed GSP is at most as restrictive as the generic policy GP' that he proposed. This should always be the case if the matching was performed honestly, but the check is necessary so that a cheating Data Subject cannot impose obligations on the Data Controller that he doesn't want to, or simply cannot, adhere to.

Step 6. The Data Controller explicitly agrees to the proposed generic sticky policy.

Step 7. The Data Subject requests the access control and data handling policies associated to the resource that she is interested in.

Step 8. The Data Controller transmits to the Data Subject the access control policy ACP' associated to the resource, as well as the data handling policies DHP1' and DHP2' for the personal data that the Data Subject has to reveal. These policies are partially evaluated variants (for example by filling in values of known attributes) of the policies ACP, DHP1, and DHP2 as contained in the Data Controller's policy file.

In the example sketched in Figure 13, the access control policy requires the Data Subject to reveal personal data1 as certified by one of her credentials, reveal personal data2 that is not part of a credential, and prove that a condition φ over the attributes is satisfied.

Step 9. The Data Subject matches the proposed data handling policies DHP1' and DHP2' against the preferences Prefs1 and Prefs2 that her policy associates to personal data1 and personal data2, respectively. If a match is found, then the resulting set of obligations are called the sticky policies SP1 and SP2.

Step 10. In the example of Figure 13, there is an access control policy ACP2 associated to personal data2 specifying that it can only be revealed to Data Controllers with at least a silver privacy rating as issued by the Better Business Bureau. Also, the preferences Prefs2 associated to personal data2 contain a (downstream) access control policy ACP3 insisting that personal data2 can thereafter only be shared with downstream data controllers that have a gold privacy rating. The Data Subject transmits the sticky policies SP1 and SP2 together with the downstream access control policies ACP3', as well as the access control policy ACP2'. (For the simple cases displayed here these are probably identical to ACP3 and ACP2, respectively.)

Step 11. The Data Controller checks whether the sticky policies SP1 and SP2 are at most as restrictive as the policies DHP1' and DHP2' that he proposed in Step 8. Again, if the Data Subject performed the matching correctly, this will always be the case.

Step 12. The Data Controller explicitly agrees to adhere to the sticky policies SP1 and SP2, and to enforce access control policy ACP3' when making personal data2 available to third parties. He also sends a proof of his silver privacy rating by the Better Business Bureau.

Step 13. The Data Subject submits a request for Resource1, and sends along the required personal data and a proof that she satisfies conditions φ'. Here φ' could be different from φ as stated in ACP1, not only because of the partial evaluation that took place in Step 8, but also because the Data Subject may choose to prove a stronger statement than required by φ, for example by revealing an additional attribute rather than proving that a condition on it holds. This may be for reasons of efficiency, or because the underlying technology simply doesn't allow proving conditions over attributes without revealing their values.

Step 14. The Data Controller checks that the revealed personal data and the statement φ' satisfy the access control policy ACP1'. At this point, the Data Controller stores personal data1 and personal data2 into its own records, attaching the obligations agreed upon in sticky policies SP1 and SP2 via its obligation enforcement engine. Moreover, it may make personal data2 more widely available to third parties, but only by enforcing access control policy ACP3 imposed by the Data Subject, possibly enhanced with additional restrictions.

Step 15. If the check succeeds, access to the resource is granted.

7.3 Data Controller and Downstream Data Controller

We speak of downstream usage when a Data Controller wants to make a Data Subject's personal data available to third parties, so-called downstream Data Controllers. For example, a travel service mash-up may want to forward the Data Subject's driver license information to a car rental service to book a car, or a hospital may want to make patients' medical records available to its researchers.

Our policy language allows the Data Subject to control to whom and under which conditions its personal data can be further forwarded. Figure 14 illustrates how the Data Subject specifies her restrictions in her local policy and how these, via the sticky policy, make their way into the Data Controller's local policy.

The following figure displays a Data Subject who accesses a service theservice hosted by the Data Controller. The access control policy of the service requires the Data Subject to reveal a piece of personal data (mypersonaldata). In the data handling policy #theDHP associated to this reveal request, the data controller expresses that he may forward this piece of personal data to other downstream Data Controllers. The Data Subject's data handling preferences associated to mypersonaldata contain an <AuthzDownstreamUsage> authorization specifying that downstream usage is allowed to recipients satisfying the downstream access control policy myDACP.

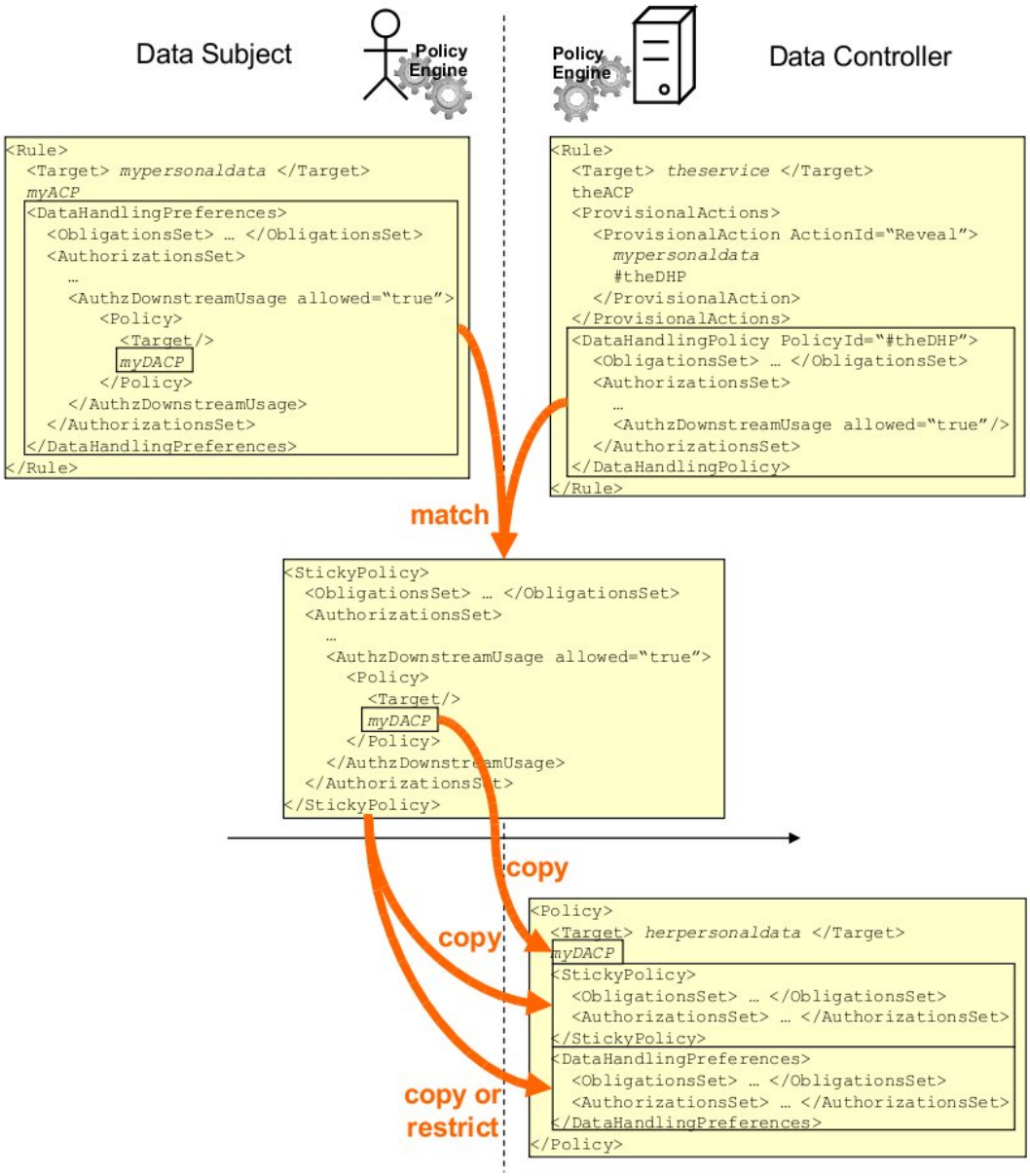


Figure 14: Data flow of downstream usage restrictions

If the other obligations and authorizations can also be successfully matched, then the sticky policy that is the result of matching the Data Controller's data handling policy against the Data Subject's preferences also contains a downstream usage authorization specifying myDACP as the downstream access control policy.

The Data Controller stores the personal data on his own system, assigning it a unique local resource identifier herpersonaldata. He now has to associate the sticky policy to the resource to allow the obligation engine to correctly implement the agreed-upon obligations, and has to make sure that at each access to herpersonaldata the access control policy myDACP is enforced. By lack of a suitable combination algorithm in XACML1, this policy cannot be easily inserted into the Data Controller's overall policy that governs access to its services, but needs to be inserted in a separate policy engine whose rules are enforced

on top of any rules that may be expressed in the main policy, thereby providing an additional layer of protection for the downstream use of collected data.2

To see why the downstream access control policy cannot be simply inserted into the main policy with an existing combination algorithm, consider the case that the Data Controller's overall policy already contains a general rule that would apply to herpersonaldata. For example, suppose that herpersonaldata is the Data Subject's email address, and that the main policy already contains a rule imposing access control restrictions on all stored email addresses. We would need a combining algorithm so that both the existing policy and myDACP have to return a Permit decision in order to obtain access. Semantically this seems like a "conjunctive" combination of policies. The only existing combining algorithm that comes close to conjunctive semantics is deny-overrides, but the outcome of a deny-overrides combination is insensitive to policies with Permit decisions. This seems to be a quite intricate issue in how XACML combines rules and policies; we leave it to further research whether a more elegant solution can be found than the one with two separate engines as sketched above.

In fact, keeping policies designed by the Data Subject strictly separate from those defined by the Data Controller seems like a good engineering decision anyway. Namely, blindly inserting user-authored policies into the overall policy could open the door to "policy insertion attacks", where a malicious Data Subject tries to circumvent the Data Controller's access control restrictions by providing a malformed downstream access control policy that not only applies to her own personal data, but to the services hosted by the Data Controller as well. This is also for the reason for which we previously insisted that the <xacml:Target/> element contained in the downstream access control policy in the Data Subject's preferences is empty, and for which we insist here that the Data Controller himself replaces it with a <xacml:Target> element that restricts the applicability to the personal data that are being transmitted.

The Data Controller inserts a <Policy> element containing the access control restrictions specified by myDACP, but with as <Target> element the resource identifier herpersonaldata, thereby restricting the applicability of the new policy to the piece of personal data that was just transmitted. Additionally, it includes a <StickyPolicy> element containing the agreed-upon sticky policy associated to myDACP (both the authorizations and the obligations), as well as a <DataHandlingPreferences> element containing authorizations and obligations that are less permissive than or equal to the agreed-upon sticky policy. The reason for splitting up the <StickyPolicy> from the <DataHandlingPreferences> element is that the former specifies the authorizations and obligations that the Data Controller himself has to adhere to, while the latter contains those that an eventual downstream data controller has to adhere to. It is quite thinkable that a data controller may want to further restrict downstream data usage, for example allowing only a subset of the usage purposes, insisting on even earlier deletion of the data, or disallowing further downstream usage.

When later a downstream Data Controller requests the data controller to reveal herpersonaldata, the interaction proceeds perfectly symmetrically to when a (primary) Data Controller requests the personal data from the Data Subject directly. Namely, the downstream Data Controller proposes a data handling policy that has to match the Data Controller's preferences associated to herpersonaldata, and proves that he satisfies the (downstream) access control policy myDACP that the Data Controller enforces on herpersonaldata. If allowed by the new sticky policy, the downstream Data Controller can store herpersonaldata on his own system and attach the necessary policies exactly as the (primary) Data Controller did in the scenario described above. Our strategy thereby enables unlimited downstream usage of personal data within the restrictions imposed by the Data Subject herself.

7.4 SAML 2.0 Credential Profile

7.4.1 Introduction

This document describes how SAML 2.0 6 can be extended such that it can be used as protocol language in credential-based access control. In particular, SAML 2.0 shall eventually carry the messages that are exchanged in the protocol flow between a data subject and a Data Controller as described in Section 7.2. We only use the defined extension points of SAML 2.0 and do not make any changes that are not conform the SAML's base XML schema.

The following lists how we use SAML for the individual steps in the protocol flow:

Step	Policy
1	SAML XACML Policy Query for policy with special URI 'uri:pl:genericpolicy'
2	SAML XACML Policy Response
4+6	*Sticky Policy Statement will be piggybacked onto the Policy Query in Step 7*
7	SAML XACML Resource Query (as specified in Section 7.4.3)
9	SAML XACML Policy Response containing a SAML StickyPolicy statement (as specified in Section 7.4.3.8)
11	SAML XACML Policy Response containing a SAML StickyPolicy Statement (as specified in Section 7.4.3.8)
13	SAML Response containing Credential-, Declaration- and CrossCredential Assertions (as specified in Sections 7.4.3.2, 7.4.3.3 and 7.4.3.4)
14	SAML Resource Query (as specified in Section 7.4.3)
16	SAML Resource Response (as specified in Section 7.4.3)

The formats for the SAML XACML Policy Query and the SAML XACML Policy Response are defined in the SAML 2.0 Profile for XACML 6. In addition we assume the SAML XACML Policy Response to carry policies and policy-sets that are conform to the XACML 3.0 Credential Profile 6.

7.4.2 Namespaces

Prefix	XML Namespace	Description
saml	urn:oasis:names:tc:SAML:2.0:assertion	SAML
pl	http://www.primelife.eu	PrimeLife
samlp	urn:oasis:names:tc:SAML:2.0:protocol	SAML protocol

7.4.3 SAML Resource Query

Of major interest in this profile is an element based on SAML that allows for querying an XACML resource. It is represented by the element <ResourceQuery>. In this Profile, we call this a SAML Resource Query (or short Resource Query).

A Resource Query mainly serve two main purposes. First, it indicates an XACML resource that is requested. Second, it provides verifiable evidence that the requirements for being granted access to the requested resource are fulfilled.

It is assumed that the requirements for being granted access are already known at the time of querying the resource. The requirements are typically learned by performing a SAML XACML Policy Query 6 for the resource in question. Note that the Policy Query and Response are independent from the Resource Query. This may be of advantage in certain situations (e.g., when the policy is already known, when the resource is queried frequently while the policy does not change, etc.).

7.4.3.1 Element <ResourceQuery>

The <pl:ResourceQuery> element contains the following elements:

<xacml:Request> [Required]
An XACML request that must contain a specific resource in it's target.

<saml:Assertion> [Optional, any number]
May either be a Credential Assertion, a Declaration Assertion, or a CrossCredential Assertion. However, there may be at most one Declaration Assertion, and at most one CrossCredential Assertion.

<DataHandlingPolicy> [Optional, at most one]
The data handling policy that describes how the resource(s) which are specified in the target of the <xacml:Request> are intended to be used by the requestor.

7.4.3.2 Element <saml:Assertion>: Credential Assertion

An instance of a <saml:Assertion> element is called a Credential Assertion in this Profile if the assertion contains any of the following information:

- Verifiable evidence for owning a credential.
- Verifiable evidence that certain conditions holds on credential attributes.
- Verifiable evidence that certain provisional actions have been successfully performed.

Verifiable evidence typically consists of two parts: (1) a technology dependent representation of the evidence itself (also referred to as a proof), and (2) some metadata describing the evidence (also referred to as a proof description) which is used to verify the validity of the evidence. The proof description is independent of the proof and is typically a textual description of what is actually proved in the proof. The proof itself is represented by proof objects. Depending on the underlying credential technology, such proof objects may contain evidence for multiple of the above mentioned information. In particular, a certain proof object may contain evidence for ownership, conditions, provisional actions, or multiple of those together.

To include verifiable evidence for ownership in an assertion, Evidence Statements (c.f. Section 7.4.3.7) are used.

To include verifiable evidence for conditions in an assertion, Condition Statements (c.f. Section 7.4.3.5) are used. Condition Statements contain the proof description (i.e., a description of the condition that is supposed to hold on the credential attribute) and the proof itself or a reference to the proof.

To include verifiable evidence for provisional actions in an assertion, ProvisionalAction Statements (c.f. Section 7.4.3.6) may be used. ProvisionalAction Statements contain the proof description (i.e., a description of the provisional action that was supposedly successfully performed) and the proof itself or a reference to the proof.

A special case is the very common provisional action of revealing the values of credential attributes. For this kind of provisional action it is recommended to make use of the built in SAML Attribute Statements (<saml:AttributeStatement> elements which contain <saml:Attribute> elements). In this profile, however, we allow for attaching data handling policies to these Attribute Statements. This is done by first specifying the data handling policy in a StickyPolicy Statement (c.f. Section 7.4.3.8) within an assertion in the ResourceQuery, and then referencing the identifier of this policy in the <saml:Attribute> element by using the optional attribute stickyPolicyReference (Note: according to SAML 6 the <saml:Attribute> may contain arbitrary attributes). Additionally, we allow for revealing of attributes to third parties. This is done by referencing an identifier of the third party in the optional attribute revealTo. For this special provisional action, the Attribute Statement accounts for the proof description part of the evidence. The proof itself needs then to be given within the ownership proof object (i.e., within the Evidence Statement). Clearly, this is only possible if the technology supports such ownership proof. In case this is not supported, instead of the Attribute Statements, individual Provisional Action Statements have to be employed as described above.

Beyond the restrictions specified in SAML 2.0, this Profile imposes that the following elements are contained in a <saml:Assertion> when used as an Credential Assertion:

<saml:Statement> [Optional, any number]
May either be a <saml:AttributeStatement>, a StickyPolicy Statement, a Condition Statement, or a ProvisionalAction Statement.

7.4.3.3 Element <saml:Assertion>: Declaration Assertion

An instance of a <saml:Assertion> element whose ID attribute and <Issuer> element have the value ‘uri:pl:xacml:assertion:declaration’ is called a Declaration Assertion in this Profile.

Declaration Assertions are used to contain statements that are not associated with a specific credential. In particular, uncertified data or uncertified conditions that are required by policy but which are merely declared rather than contained in a certified credential are stated within a Declaration Assertion.

The same restrictions that are imposed on a Credential Assertion are imposed on a Declaration Assertion.

7.4.3.4 Element <saml:Assertion>: CrossCredential Assertion

An instance of a <saml:Assertion> element whose ID attribute and <Issuer> element have the value ‘uri:pl:xacml:assertion:crosscredential’ is called a CrossCredential Assertion in this Profile.

CrossCredential Assertions are used to contain statements that are associated to multiple credentials. For example, StickyPolicy Statements embedded in a CrossCredential Assertion indicate that these sticky policies are referenced from within different Credential Assertions. Condition Statements embedded in a CrossCredential Assertion typically reference multiple credentials (with the <CredentialAttributeDesignator> element) and state cross-credential requirements. Typically, CrossCredential Assertions do not contain Attribute Statements.

The same restrictions that are imposed on a Credential Assertion are imposed on a CrossCredential Assertion.

7.4.3.5 Element <saml:Statement>: Condition Statement

We introduce a new SAML statement type that contains an XACML condition. To express that an instance of a <saml:Statement> element is of this new type, the xsi:type attribute is set to pl:ConditionStatement. Such an instance of a <saml:Statement> is called Condition Statement in this profile.

A Condition Statement contains an XACML condition that represents a predicate that holds over attributes out of one or multiple credentials. A credential attribute in a condition is referenced with the <CredentialAttributeDesignator> element. In the XACML Credential Profile this designator element uses the CredentialId attribute to reference a particular credential instance (represented as <Credential> element). As in a <ResourceQuery> the credentials are represented by Credential Assertions (c.f. Section 5), the CredentialId attribute is now (re-)used to reference instances of such assertions, rather than <Credential> elements. A <CredentialAttributeDesignator> element contained within a Credential Assertion shall only reference an attribute out of the credential that is represented by this assertion (i.e., the CredentialId attribute of the <CredentialAttributeDesignator> elements shall be equal to the ID attribute of the containing Credential Assertion). A <CredentialAttributeDesignator> element contained within a CrossCredential Assertion or a Delegation Assertion must reference an attribute out of any credential that is represented by a Credential Assertion.

A Credential Assertion that contains multiple Condition Statements has the same semantics as if all <xacml:Condition> elements were contained within a single Condition Statement combined with the XACML function urn:oasis:names:tc:xacml:1.0:function:and. In some cases separating the condition into multiple parts may be required by the underlying technology.

The conditions in the Condition Statements specified across Credential Assertions, Declaration Assertions and CrossCredential Assertions are all together considered as single condition with conjunction semantics between them.

In case a Credential Assertion contains a Condition Statement that constrains the attribute pl:Issuer to a concrete value, then the Credential Assertion’s <saml:Issuer> element has to have the same value.

Beyond the restrictions specified in SAML 2.0, this Profile imposes that the following elements are contained in a <saml:Statement> when used as a Condition Statement:

<xacml:Condition> [Optional, at most one]
A predicate that holds on the credentials that are represented by Credential Assertions and referenced by <CredentialAttributeDesignator> elements.

<Evidence> [Required, exactly one]
The proof that the predicate specified in the <xacml:Condition> holds, or a reference to an evidence that contains this proof.

7.4.3.6 Element <saml:Statement>: ProvisionalAction Statement

We introduce a new SAML statement type that contains a provisional action and the corresponding evidence (or the reference to it) for having successfully performed the action. To express that an instance of a <saml:Statement> element is of this new type, the xsi:type attribute is set to pl:ProvisionalActionStatement. Such an instance of a <saml:Statement> is called ProvisionalAction Statement in this profile.

Beyond the restrictions specified in SAML 2.0, this Profile imposes that the following elements are contained in a <saml:Statement> when used as a ProvisionalAction Statement:

<ProvisionalAction> [Required, exactly one]
This element describes the provisional action for which the given evidence is provided. See 6 for a description of the <ProvisionalAction> element.

<Evidence> [Required, exactly one]
The proof for having successfully performed the <ProvisionalAction> that is described in the enclosing <ProvisionalActionStatement>, or a reference to an evidence that contains this proof.

7.4.3.7 Element <saml:Statement>: Evidence Statement

We introduce a new SAML statement type that contains some concrete evidence (i.e., a proof object), or the reference to it. To express that an instance of a <saml:Statement> element is of this new type, the xsi:type attribute is set to pl:EvidenceStatement. Such an instance of a <saml:Statement> is called Evidence Statement in this profile.

The contained <Evidence> element is used to prove that certain credential requirements (e.g., conditions, provisional actions) are fulfilled. The content of the element is dependent on the underlying technology.

Beyond the restrictions specified in SAML 2.0, this Profile imposes that the following elements are contained in a <saml:Statement> when used as an Evidence Statement:

<Evidence> [Required, exactly one]
The proof that a certain fact is true, or a reference to an evidence that contains this proof.

7.4.3.8 Element <saml:Statement>: StickyPolicy Statement

We introduce a new SAML statement type that represents a concrete sticky policy. To express that an instance of a <saml:Statement>

element is of this new type, the xsi:type attribute is set to pl:StickyPolicyStatement. Such an instance of a <saml:Statement> is called StickyPolicy Statement in this profile.

Sticky policies are referenced from within <saml:Attribute> elements that are contained in <saml:AttributeStatement> elements.

The <StickyPolicyStatement> element contains the following attributes and elements:

ID [Required]
An identifier that is used within <saml:Attribute> elements to reference this sticky policy.

<AuthorizationSet> [Optional]
A set of authorizations.

<ObligationSet> [Optional]
A set of obligations.

7.4.3.9 Element <Evidence>

This element is used as container for or reference to the actual evidence (i.e., a proof object) that certain facts are true. In particular, the contained evidence may prove credential ownership, that certain conditions holds on credential attributes, that certain provisional actions have been successfully performed, or multiple of those facts at the same time (c.f. Section 7.4.3.2). The content of the element is dependent on the underlying technology. This element may reference to other <Evidence> elements that are specified within the <ResourceQuery>. Likewise it may be referenced from other <Evidence> elements within the <ResourceQuery>. In case this element does not contain the evidence itself, it must reference to another <Evidence> element.

The <Evidence> element contains the following attributes and elements:

ID [Optional, at most one]
An identifier that may be used within other <Evidence> elements to reference this element.

referenceID [Optional, at most one]
In case the element is used as reference to some evidence rather than containing the evidence itself, this attribute is used to specify which evidence instance contains the evidence.

Arbitrary Elements
These elements shall contain the technology-dependent evidence.

7.4.4 SAML Resource Response

The SAML Resource Response is used to convey the resource that was queried by a previous SAML Resource Query.

7.4.4.1 Element <saml:Statement>: Resource Statement

We introduce a new SAML statement type to transport resources. To express that an instance of a <saml:Statement> element is of this new type, the xsi:type attribute is set to pl:ResourceStatement. Such an instance of a <saml:Statement> is called Resource Statement in this profile.

Beyond the restrictions specified in SAML 2.0, this Profile imposes that the following elements are contained in a <saml:Statement> when used as a Condition Statement:

Arbitrary Elements
The resources intended to be transported may have arbitrary form.

7.4.4.2 Element <saml:Assertion>: Resource Assertion

A <saml:Assertion> instance may contain a Resource Statement. An instance of a <saml:Assertion> element containing a Resource Statement is called a Resource Assertion in this profile. When an Resource Assertion is part of a response to a SAML Resource Query, then the Resource Assertion must contain exactly one Resource Statement.

7.4.4.3 Element <samlp:Response>: Resource Response

A <samlp:Response> instance may contain a Resource Assertion. An instance of a <samlp:Response> element containing a Resource Assertion is called a Resource Response in this Profile.

7.4.5 Creating Resource Queries

This section shall give an intuition on how a <ResourceQuery> element has to look like with respect to an XACML policy that protects the queried resource. The goal is that the evidence given in the <ResourceQuery> element is sufficient for showing that the stated policy requirements (i.e., conditions, provisional actions) are fulfilled and thus sufficient for being granted access to the resource in question.

Not that it is strongly dependent on the underlying credential technology how proofs are embedded in the <ResourceQuery>. Thus, in the following we only describe how the proof descriptions have to be specified with respect to a given XACML policy. A description on how the proofs are embedded needs to be specified in individual profiles for the particular technologies.

For provisional actions that are required by a policy, it is straightforward to describe how the proof description in the <ResourceQuery> has to look like. This is because evidence for successful performance has to be given for all provisional actions prescribed by the policy. However, for conditions this is different as a condition may contain nested sub-conditions combined with logical operators (e.g., disjunction). Thus, a user may have a multitude of possibilities on how to fulfill a concrete condition (e.g., to prove the condition 'A and (B or C)', he could prove A and B, or A and C), therefore it depends on the user's choice for which (sub-)conditions evidence is eventually provided. Note that the user's choice has strong implications on her privacy.

Now we describe how a user can create a SAML Resource Query with respect to a given XACML policy. (Recall that a SAML XACML Policy Response is sent in response to a SAML XACML Policy Query 6 for a particular resource. The Policy Query contains an <xacml:Request> element and the Policy Response contains a number of <xacml:PolicySet> elements. Out of those policy sets, the user then chooses the one that she wants to fulfill and that is used as basis for her proofs. In the following, we refer to the chosen PolicySet simply as policy.) The requirements stated in the XACML policy typically contain (1) credential declarations, (2) conditions, and (3) provisional actions.

The resource that shall be queried is given in the <xacml:Request> element of the Resource Query. For every credential that is declared by <Credential CredentialId='c'> in the policy and subsequently referenced in any condition or provisional action, a Credential Assertion with the ID attribute being c is created. Sections 7.4.5.1 and 7.4.5.2 describe how conditions and provisional actions have to be treated, respectively.

7.4.5.1 Conditions

As mentioned earlier, a user has to choose for which (sub-)conditions, which are required in the policy, evidence shall be provided in the Resource Query. However, for the (sub-)conditions that are chosen there are multiple possibilities on where to embed those in the Resource Query.

Conditions may be credential specific, they may concern multiple credentials, or no credentials at all. In general, conditions are embedded together with their technology-dependent proof (in an <Evidence> element) in Condition Statements. In a Condition Statement a <xacml:Condition> element accounts for the proof description and an <Evidence> element accounts for the proof. The conditions may be split across multiple Condition Statements or combined within a single Condition Statement. In case a condition concerns a specific credential, the corresponding Condition Statement shall be embedded in the respective Credential Assertion. In case a condition concerns multiple credentials, the statement shall be embedded in a CrossCredential Assertion. In case a condition does not concern any credential, the statement shall be embedded in a Declaration Assertion.

7.4.5.2 Provisional Actions

A policy may require verifiable evidence that certain provisional actions have successfully be performed. In general, this evidence is given in a Provisional Action Statement where a <ProvisionalAction> element accounts for the proof description and the <Evidence> element accounts for the proof. Just like Conditions, Provisional actions may be credential specific, they may concern multiple credentials, or no credentials at all. In case a provisional action concerns a specific credential, the corresponding Provisional Action Statement shall be embedded in the respective Credential Assertion. In case a provisional action concerns multiple credentials, the statement shall be embedded in a CrossCredential Assertion. In case a condition does not concern any credential, the statement shall be embedded in a Declaration Assertion.

However, it is not mandatory to use Provisional Action Statements for providing evidence for a successful performance. It depends on the type of provisional action and on the implementation if such a Provisional Action Statement is used. For example, the common provisional action of revealing the values of credential attributes may be handled differently, i.e., without Provisional Action Statements.

In the following sections we describe in detail how verifiable evidence may be given for the provisional actions defined the XACML Credential Profile 6.

7.4.5.2.1 Reveal

In the XACML Credential profile the revealing of attributes may be done in different ways by using different provisional actions. Attributes may be revealed (1) from a specific credential or independent of a specific credential, (2) under a specific data handling policy (DHP) or without such policy, and (3) to a specific third party. To this end, the profile defines provisional actions with the ActionsIds ‘Reveal’, ‘RevealUnderDHP’, ‘RevealTo’ and ‘RevealToUnderDHP’. They take arguments for an attribute A, a credential C, a third party T and a data handling policy H, respectively:

- Reveal: Attribute, Credential (optional)
- RevealUnderDHP: Attribute, DHP, Credential (optional)
- RevealTo: Attribute, Third-Party, Credential (optional)
- RevealToUnderDHP: Attribute, Third-Party, DHP, Credential (optional)

For every <ProvisionalAction> with the ActionId ’Reveal’ an Attribute Statement within the Credential Assertion with ID C has to contain an <Attribute> element with the Name attribute set to A.

For every <ProvisionalAction> with the ActionId ‘RevealUnderDHP’ an Attribute Statement within the Credential Assertion with ID C has to contain an <Attribute> element with the Name attribute set to A and the stickyPolicyReference attribute set to S. Accordingly, a

StickyPolicyReference Statement with ID S has to be present within any assertion in the Resource Query where the sticky policy S is less permissive than the data handling policy H (c.f. Section 3.3 for explanation on permissiveness).

For every <ProvisionalAction> with the ActionId ‘RevealTo’ an Attribute Statement within the Credential Assertion with ID C has to contain an <Attribute> element with the Name attribute set to A and the revealTo attribute set to T.

For every <ProvisionalAction> with the ActionId ‘RevealToUnderDHP’ follow both the instructions given for the ActionIds ‘RevealTo’ and ‘RevealUnderDHP’ at the same time.

In case the last optional argument representing the Credential is omitted in any of the provisional actions, the attribute is self stated and therefore not concerning a specific credential. In this case, the corresponding Attribute Statements are embedded into the Declaration Assertion, rather than a Credential Assertion.

The proof object for these provisional actions must be given within an Evidence Statement embedded in a CrossCredential Assertion.

7.4.5.2.2 Sign

For every <ProvisionalAction> with the ActionId ‘Sign’ a Provisional Action Statement is created that contains a <ProvisionalAction> element that is a copy of the one required by the policy. This statement is embedded within all Credential Assertions. Depending on the underlying credential technology, the <Evidence> elements in these Provisional Action Statement contain the proof directly or reference any other <Evidence> element that contains a proof. In case the technology uses a single proof object for the signature this shall be given within an Evidence Statement embedded in a CrossCredential Assertion (and subsequently referenced from the individual Provisional Action Statements that are embedded in the individual Credential Assertions).

7.4.5.2.3 Spend

The provisional action with the ActionId ‘Spend’ takes, among others, an argument for a credential C. For every <ProvisionalAction> with the ActionId ‘Spend’ a Provisional Action Statement within the Credential Assertion with ID C is created. This statement contains a <ProvisionalAction> element that is a copy of the one required by the policy. Depending on the underlying credential technology, the <Evidence> elements in these Provisional Action Statement contain the proof directly or reference any other <Evidence> element that contains a proof.

7.4.6 Example

Now we show an example Resource Query inspired by the example policy given in 6. The query contains evidence for the following requirements:

- Ownership of a drivers license issued by the Swiss government (or any delegates).
- Ownership of a (valid) credit card issued by Visa (or any delegates).
- Revealing the credit card number for the purpose ‘payment’.
- Sign the statement ‘I agree to the terms of service.’.
- The credit card’s expiration date must lie in the future.
- The names on the credit card and the drivers license must be equal.

Concerning the technology dependent proof we assume that the technology that was used to create the proof allows for incorporating the evidence for above requirements in a single proof object. Therefore all <Evidence> elements reference a single <Evidence> element that contains this proof.

```
<pl:ResourceQuery>
  <saml:Assertion ID='#cc'>
    <saml:Subject>??</saml:Subject>
    <saml:Issuer>http://www.visa.com</saml:Issuer>

    <saml:AttributeStatement>
      <saml:Attribute Name='http://bankingstandards.org/CreditcardNumber'
                      stickyPolicyReference='#sp01'>1234 5678 9012</saml:Attribute>
    </saml:AttributeStatement>

    <saml:Statement type='pl:ConditionStatement'>
      <xacml:Conditon>
        <Apply FunctionId='pl:delegatee-of'>
          <CredentialAttributeDesignator CredentialId='#cc' AttributeId='pl:Issuer'/>
          <AttributeValue DataType='xs:anyURI'>http://www.visa.com</AttributeValue>
        </Apply>
      </xacml:Conditon>
      <pl:Evidence referenceID='evd01'/>
    </saml:Statement>

    <saml:Statement type='pl:ConditionStatement'>
      <xacml:Conditon>
        <Apply FunctionId='pl:subclass-of'>
          <CredentialAttributeDesignator CredentialId='#cc' AttributeId='pl:Type' />
          <AttributeValue DataType='xs:anyURI'>
            http://www.bankingstandards.org/Creditcard
          </AttributeValue>
        </Apply>
      </xacml:Conditon>
      <pl:Evidence referenceID='evd01'/>
    </saml:Statement>

    <saml:Statement type='pl:ConditionStatement'>
      <xacml:Conditon>
       <Apply FunctionId='urn:oasis:names:tc:xacml:1.0:function:date-less-than-or-equal'>
            <CredentialAttributeDesignator CredentialId='#cc'
               AttributeId='http://bankingstandards.org/ExpirationDate'/>
            <Apply FunctionId='pl:xacml:1.0:function:date:today'/>
       </Apply>
      </xacml:Conditon>
      <pl:Evidence referenceID='evd01'/>
    </saml:Statement>
```

```
  <saml:Statement type='pl:ProvisionalActionStatement'>
    <pl:ProvisionalAction ActionId='pl:Sign'>
        <AttributeValue DataType='xs:string'>
          I agree to the terms of service.
        </AttributeValue>
    </pl:ProvisionalAction>
    <pl:Evidence referenceID='evd01'/>
  </saml:Statement>

  <saml:Statement type='pl:StickyPolicyStatement' ID='#sp01'>
    <pl:AuthorizatonSet>
      <pl:Purpose>payment</pl:Purpose>
    </pl:AuthorizatonSet>
  </saml:Statement>
</saml:Assertion>



<saml:Assertion ID='#dl'>
  <saml:Subject>??</saml:Subject>
  <saml:Issuer>http://www.admin.ch</saml:Issuer>

  <saml:Statement type='pl:ConditionStatement'>
    <xacml:Conditon>
      <Apply FunctionId='pl:delegatee-of'>
        <CredentialAttributeDesignator CredentialId='#dl' AttributeId='pl:Issuer'/>
        <AttributeValue DataType='xs:anyURI'>http://www.admin.ch</AttributeValue>
      </Apply>
    </xacml:Conditon>
    <pl:Evidence referenceID='evd01'/>
  </saml:Statement>

  <saml:Statement type='pl:ConditionStatement'>
    <xacml:Conditon>
      <Apply FunctionId='pl:subclass-of'>
        <CredentialAttributeDesignator CredentialId='#dl' AttributeId='pl:Type' />
        <AttributeValue DataType='xs:anyURI'>
          http://ec.europa.eu/transport/DrivingLicence
        </AttributeValue>
      </Apply>
    </xacml:Conditon>
    <pl:Evidence referenceID='evd01'/>
  </saml:Statement>

  <saml:Statement type='pl:ProvisionalActionStatement'>
    <pl:ProvisionalAction ActionId='pl:Sign'>
        <AttributeValue DataType='xs:string'>
          I agree to the terms of service.
        </AttributeValue>
    </pl:ProvisionalAction>
    <pl:Evidence referenceID='evd01'/>
  </saml:Statement>
</saml:Assertion>



<saml:Assertion ID='uri:pl:xacml:assertion:crosscredential'>
  <saml:Statement type='pl:ConditionStatement'>
    <xacml:Conditon>
      <Apply FunctionId='urn:oasis:names:tc:xacml:1.0:function:string-equal'>
          <CredentialAttributeDesignator CredentialId='#cc'
             AttributeId='http://bankingstandards.org/CreditCardName'/>
          <CredentialAttributeDesignator CredentialId='#dl'
             AttributeId='http://somestandard.org/Name'/>
      </Apply>
    </xacml:Conditon>
    <pl:Evidence referenceID='evd01'/>
  </saml:Statement>

  <saml:Statement type='pl:EvidenceStatement'>
    <pl:Evidence ID='evd01'>
      <Technology>someTechnology</Technology>
      <ProofBloB>
```

```
          12345678901234567890123456789012345678901234567890123456789012345678901234567890
          12345678901234567890123456789012345678901234567890123456789012345678901234567890
          12345678901234567890123456789012345678901234567890123456789012345678901234567890
          12345678901234567890123456789012345678901234567890123456789012345678901234567890
          12345678901234567890123456789012345678901234567890123456789012345678901234567890
          12345678901234567890123456789012345678901234567890123456789012345678901234567890
        </ProofBloB>
      </pl:Evidence>
    </saml:Statement>
  </saml:Assertion>

</pl:ResourceQuery>
```

8 Policy Language Schema

Implementors will need a complete schema for the policy language and obligation preferences, including a core vocabulary for credentials, purposes, recipients, events, notifications, etc. A complete schema has yet to be finished.

Section 5.2 includes an XML schema for the XACML v3.0 Privacy Policy Profile Version 1.0, as defined by OASIS. Section 6.6 defines extensions to XACML 3.0 to enable the expression of credential-based access control. The XML schema for these extensions is not yet available.

Section 3.2 defines a schema for an obligation language. This defines the container for triggers and actions, but omits the definition of specific triggers and actions, although some examples are given. Further work is needed to provide schemas for the set of triggers in section 3.4 and the set of actions in section 3.5. Note that this report only defines a draft specification for triggers and actions, and the details may change in future work.

9 References

[ACDS08]
C.A. Ardagna, M. Cremonini, S. De Capitani di Vimercati, and P. Samarati. A privacy-aware access control system. Journal of Computer Security, 16(4):369-392, 2008.

[ACK 09]
C.A. Ardagna, J. Camenisch, M. Kohlweiss, R. Leenes, G. Neven, B. Priem, P. Samarati, D. Sommer, and M. Verdicchio. Exploiting cryptography for Journal of privacy-enhanced access control: A result of the PRIME project. Computer Security, 2009. (to appear).

[Agrawal]
Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Implementing P3P using database technology. In Proceedings of the 19th International Conference on Data Engineering, March 2003.

[AHKS02]
P. Ashley, S. Hada, G. Karjoth, and M. Schunter. E-P3P privacy policies and privacy authorization. In Proc. of the ACM workshop on Privacy in the Electronic Society (WPES 2002), Washington, DC, USA, November 2002.

[Ali]
M. Ali, L. Bussard, and U. Pinsdorf.: Obligation Language and Framework to Enable Privacy-aware SOA. To appear in DPM'09: workshop on Data Privacy Management (2009).

[Ardagna]
Ardagna, C.A., Cremonini, M., De Capitani di Vimercati, S., Samarati, P.: A privacy-aware access control system. J. Comput. Secur. 16(4) (2008) 369-397

[Art.29 Opinion 100 Annex]
Article 29 Working Party, Opinion 100: Annex, accessed 10 December 2008.
http://ec.europa.eu/justice_home/fsj/privacy/docs/wpdocs/2004/wp100a_en.pdf

[Art.29 Opinion 100]
Article 29 Working Party, Opinion 100: Opinion on More Harmonised Information Provisions, accessed 10 December 2008.
http://ec.europa.eu/justice_home/fsj/privacy/docs/wpdocs/2004/wp100_en.pdf

[Art.29 Opinion 136]
Article 29 Working Party, Opinion 136: Opinion on the concept of personal data, accessed 10 December 2008.
http://ec.europa.eu/justice_home/fsj/privacy/docs/wpdocs/2007/wp136_en.pdf

[AT02]
A. Arsenault and S. Turner. Internet x.509 public key infrastructure: Roadmap. Internet Draft, Internet Engineering Task Force, 2002.

[BCC05]
E. F. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In Proc. of the 11th ACM Conference on Computer and Communications Security (CCS 2005), Alexandria, VA, November 2005.

[BCS05]
M. Backes, J. Camenisch, and D. Sommer. Anonymous yet accountable access control. In Proc. of the 2005 ACM Workshop on Privacy in the Electronic Society, pages 40-46, ACM New York, NY, USA, 2005.

[BFIK98]
M. Blaze, J. Feigenbaum, J. Ioannidis, and A.D. Keromytis. The role of trust management in distributed systems security. Secure Internet Programming: Issues in Distributed and Mobile Object Systems, 1998.

[BFL96]
M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In Proc. of 1996 IEEE Symposium on Security and Privacy, Oakland, CA, USA, May 1996.

[Boag]
S. Boag, D. Chamberlin, M. F. Fernandez, D. Florescu, J. Robie, J. Simeon, and M. Stefanescu, editors. XQuery 1.0: An XML Query Language. W3C Working Draft, April 2002.

[BS02]
P. Bonatti and P. Samarati. A unified framework for regulating access and information release on the web. Journal of Computer Security, 10(3):241-272, 2002.

[Casassa]
Casassa, M., Beato, F.: On parametric obligation policies: Enabling privacy-aware information lifecycle management in enterprises. In Eighth IEEE International Workshop on Policies for Distributed Systems and Networks, 2007. POLICY'07. (June 2007) 51-55

[CD00]
J. Camenisch and I. Damgård. Verifiable encryption, group encryption, and their applications to separable group signatures and signature sharing schemes. In Proc. of the 6th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT 2000), Kyoto, Japan, September 2000.

[CFL 97]
Y-H. Chu, J. Feigenbaum, B. LaMacchia, P. Resnick, and M. Strauss. Referee: Trust management for web applications. Computer Networks and ISDN Systems, 29(8-13):953-964, 1997.

[Cha85]
D. Chaum. Security without identification: Transaction systems to make big brother obsolete. ommunications of the ACM, 28(10):1030-1044, October 1985.

[Cholvy]
Cholvy, L., Garion, C.: Deriving individual obligations from collective obligations. In AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems, New York, NY, USA, ACM (2003) 962-963

[CL01]
J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Proc. of the Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 2001.

[Cra02]
L.F. Cranor. Web Privacy with P3P. O'Reilly & Associates, 2002.

CV02
J. Camenisch and E. Van Herreweghen. Design and implementation of the idemix anonymous credential system. In Proc. of the 9th ACM Conference on Computer and Communications Security (CCS 2002), Washingtion, DC, USA, November 2002.

[Damianou]
Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The ponder policy specification language. In POLICY '01: Proceedings of the International Workshop on Policies for Distributed Systems and Networks, London, UK, Springer-Verlag (2001) 18-38

[Directive 2002/58/EC]
Directive 2002/58/EC on Privacy and Electronic Communications, accessed 10 December 2008.
http://eur-lex.europa.eu/pri/en/oj/dat/2002/l_201/l_20120020731en00370047.pdf

[Directive 95/46/EC]
Directive 95/46/EC of the European Parliament and of the Council on the protection of individuals with regard to the processing of personal data and on the free movement of such data, accessed 10 December 2008.
http://ec.europa.eu/justice_home/fsj/privacy/docs/95-46-ce/dir1995-46_part1_en.pdf

[Ell99]
C. Ellison. Spki requirements. Request For Comments 2692, Internet Engineering Task Force, 1999.

[EPAL]
IBM: Enterprise privacy authorization language (EPAL 1.2)

[eXt05]
eXtensible Access Control Markup Language (XACML) Version 2.0, February 2005. See http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf.

[Fin09]
PrimeLife. Final requirements and state-of-the-art for next generation policies, 2009. https://trac.ercim.org/primelife/browser/docs/deliverables/D5.1.1-Requirements_and_state-of-the-art_for_next_generation_policies-Final.pdf.

[FK92]
D. Ferraiolo and R. Kuhn. Role-based access control. In Proc. of the 15th NIST-NCSC National Computer Security Conference, 1992.

[GD06]
S. Gevers and B. De Decker. Automating privacy friendly information disclosure. Technical Report CW441, K.U. Leuven, Dept. of Computer Science, April 2006.

[Hig09]
Higgins: Open source identity framework, 2009. http://www.eclipse.org/higgins/.

[Hilty]
Manuel Hilty, D.B., Pretschner, A.: On obligations. Computer Security ESORICS 2005 (2005) 98-117

[IDE]
IDEntity MIXer (IDEMIX). http://www.zurich.ibm.com/security/idemix/.

[IE6]
http://en.wikipedia.org/wiki/Internet_Explorer_6

[Int]
International security, trust, and privacy alliance (istpa). http://www.istpa.org/.

[Irwin]
Irwin, K., Yu, T., Winsborough, W.H.: On the modeling and analysis of obligations. In CCS '06: Proceedings of the 13th ACM conference on Computer and communications security, New York, NY, USA, ACM (2006) 134-143

[IY05]
K. Irwin and T. Yu. Preventing attribute information leakage in automated trust negotiation. In Proc. of the 12th ACM Conference on Computer and Communications Security (CCS 2005), Alexandria, VA, USA, November 2005.

[Kagal]
Kagal, L., Finin, T., Joshi, A.: A policy language for a pervasive computing environment. In POLICY '03: Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks, Washington, DC, USA, IEEE Computer Society (2003)

[Katt]
Katt, B., Zhang, X., Breu, R., Hafner, M., Seifert, J.P.: A general obligation model and continuity: enhanced policy enforcement engine for usage control. In SACMAT '08: Proceedings of the 13th ACM symposium on Access control models and technologies, New York, NY, USA, ACM (2008) 123-132

[Kojima]
Takao Kojima, Yukio Itakura. Proposal of privacy policy matching engine. Digital Identity Management October 31, 2008, Fairfax, Virginia, USA: p.9-14

[KSW02]
G. Karjoth, M. Schunter, and M. Waidner. Privacy-enabled services for enterprises. In Proc. of the 13th International Conference on Database and Expert Systems Applications (DEXA'02), Aix-en-Provence, France, September 2002.

[LGF00]
N. Li, B.N. Grosof, and J. Feigenbaum. A practically implementable and tractable delegation logic. In Proc. of the IEEE Symposium on Security and Privacy, Oakland, CA, USA, June 2000.

[Lib]
Liberty alliance project. http://www.projectliberty.org/.

[LMW05]
N. Li, J.C. Mitchell, and W.H. Winsborough. Beyond proof-of-compliance: Security analysis in trust management. Journal of the ACM, 52(3):474-514, 2005.

[Moses]
Moses, T.: OASIS eXtensible Access Control Markup Language (XACML) Version 2.0. OASIS Standard oasis-access control-xacml-2.0-core-spec-os, OASIS (February 2005)

[Netscape 7]
http://en.wikipedia.org/wiki/Netscape_(version_7)

[Ni]
Ni, Q., Bertino, E., Lobo, J.: An obligation model bridging access control policies and privacy policies. In SACMAT '08: Proceedings

of the 13th ACM symposium on Access control models and technologies, New York, NY, USA, ACM (2008) 133-142

[NLW05]
J. Ni, N. Li, and W.H. Winsborough. Automated trust negotiation using cryptographic credentials. In Proc. of the 12th ACM Conference on Computer and Communications Security (CCS 2005), Alexandria, VA, USA, November 2005.

[OAS09]
OASIS XACML v3.0 Privacy Policy Profile Version 1.0, Committee draft 1, April 2009. http://www.oasis-open.org/committees/document.php?document_id=32425

[P3P]
L. Cranor, M. Langheinrich, M. Marchiori, M. Presler-Marshall, and J. Reagle. The Platform for Privacy Preferences 1.0 (P3P1.0) Specification. W3C Recommendation, April 2002. See http://www.w3.org/TR/P3P/

[P3P Prefs]
L. Cranor, M. Langheinrich, and M. Marchiori. A P3P Preference Exchange Language 1.0 APPEL1.0). W3C Working Draft, April 2002. See http://www.w3.org/TR/P3P-preferences

[PLING]
W3C Policy Language Interest Group, Use Cases, accessed 24 October 2008.
http://www.w3.org/Policy/pling/wiki/UseCases

[Pretschner]
Hilty, M., Pretschner, A., Basin, D., Schaefer, C., Walter, T.: A policy language for distributed usage control. In 12th European Symposium on Research in Computer Security (ESORICS 2007). Volume 4734 of LNCS, Springer-Verlag (2007) 531-546

[Pri]
Privacy and Identity Management for Europe (PRIME). http://www.prime-project.eu.org/.

[Privacy Bird]
AT&T privacy bird, see http://www.w3.org/2002/p3p-ws/pp/privacybird.pdf

[Rakaiby]
El Rakaiby, Y Cuppens, F., Cuppens-Boulahia, N.: Formalization and management of group obligations. In Proceedings of IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY'09). (2009)

[Rea]
Reasoning on the web (rewerse). http://www.pms.ifi.lmu.de/rewerse-wga1/index.html.

[RFC2119]
Key words for use in RFCs to Indicate Requirement Levels, S. Bradner, editor, IETF RFC 2119, March 1997.
http://www.ietf.org/rfc/rfc2119.txt

[Rissanen]
Rissanen, E.: OASIS eXtensible Access Control Markup Language (XACML) Version 3.0. OASIS working draft 10, OASIS (March 2009)

[SAML2a]
Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0, OASIS Standard, 15 March 2005

[SAML2b]
SAML 2.0 Profile of XACML, Version 2.0, Committee Draft 1, 16 April 2009

[SCFY96]
R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C. E. Youman. Role-based access control models. IEEE Computer, 29(2):38-47, 1996.

[Schütz]
Pretschner, A., Schütz, F., Schaefer, C., Walter, T.: Policy evolution in distributed usage control. In 4th Intl. Workshop on Security and Trust Management. Elsevier (June 2008)

[SWW97]
K. E. Seamons, W. Winsborough, and M. Winslett. Internet credential acceptance policies. In Proc. of the Workshop on Logic Programming for Internet Applications, Leuven, Belgium, July 1997.

[TCPA]
TCG: Trusted Computing Platform Alliance (TCPA). Main Specification Version 1.1b, Trusted Computing Group, Inc. (February 2002)

[Thibadeau]
R. Thibadeau, Privacy Server Protocol Project, http://yuan.ecom.cmu.edu/psp/

[Trombetta]
Ni, Q., Trombetta, A., Bertino, E., Lobo, J.: Privacy-aware role based access control. In: SACMAT '07: Proceedings of the 12th ACM symposium on Access control models and technologies, New York, NY, USA, ACM (2007) 41-50

[Gama]
Gama, P., Ferreira, P.: Obligation policies: An enforcement platform. In POLICY'05: Proceedings of the Sixth IEEE International Workshop on Policies for Distributed Systems and Networks, Washington, DC, USA, IEEE Computer Society (2005) 203-212

[Tru]
Truste. http://www.truste.org/about/index.php.

[U-P07]
Credentica. U-Prove SDK overview: A Credentica white paper, 2007. http://www.credentica.com/files/U-ProveSDKWhitepaper.pdf.

[W3C02]
W3C. Platform for privacy preferences (P3P) project, April 2002. http://www.w3.org/TR/P3P/.

[WCJS97]
M. Winslett, N. Ching, V. Jones, and I. Slepchin. Assuring security and privacy for digital library transactions on the web: Client and server security policies. In Proc. of the 4th International Forum on Research and Technology Advances in Digital Libraries (ADL '97), Washington, DC, USA, May 1997.

[Web06]
Web services policy framework. See http://www.ibm.com/developerworks/webservices/library/specification/ws-polfram/?S_TACT=105AGX04&S_CMP=LP, March 2006.

[Win]
Windows cardspace. http://cardspace.netfx3.com/.

[WNR05]
P. Wang, P. Ning, and D.S. Reeves. Network access control for mobile adhoc networks. In Proc. of the 7th International Conference on Information and Communications Security (ICICS '05), pages 350-362, Beijing, China, December 2005.

[WSJ00]
W. Winsborough, K. E. Seamons, and V. Jones. Automated trust negotiation. In Proc. of the DARPA Information Survivability Conference & Exposition (DISCEX 2000), Hilton Head Island, South Carolina, USA, January 2000.

[XACML3 Credentials]
XACML 3.0 Credential Profile. PrimeLife: IBM, University of Milano and University of Bergamo.

[XACML3]
eXtensible Access Control Markup Language (XACML) Version 3.0, April 2009. See http://www.oasis-open.org/committees/document.php?document_id=32425.

[YWS03]
T. Yu, M. Winslett, and K.E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust. ACM Transactions on Information and System Security (TISSEC), 6(1):1-42, February 2003.