Report on design and implementation

Editors:	Slim Trabelsi (SAP)
	Gregory Neven (IBM)
	Dave Raggett (W3C)
Reviewers:	Stuart Short
	Uli Pinsdorf

Identifier:	D5.3.4
Type:	Deliverable
Class:	Public
Date:	May 20, 2011

Abstract

This Delivrable is a final specification document describing the PrimeLife Privacy Policy Langauge (PPL) and engine. The PPL language is defined to express Access and usage control rules. The PPL final engine is designed to handle and enforce the policies written in PPL

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 research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 216483 for the project PrimeLife.

SEVENTH FRAMEWORK PROGRAMME

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 VIMERCATI (UNIMI), Fatih Gey (EMIC), Aleksandra Kuczerawy (KUL), Sebastian Meissner (ULD), Gregory Neven (IBM), Akram Njeh (SAP), Stefano Paraboschi (UNIBG), Eros Pedrini (UNIMI), Sara Foresti (UNIMI), Ulrich Pinsdorf (EMIC), Franz-Stefan Preiss (IBM),Jakub Sendor (SAP), 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).

Executive Summary

This Deliverable describes the final specification document of the PrimeLife Policy language and engine. This document results is addressing most of the requirements raised in the D5.1.1 delivrable [Fin09]. It also materializes and concretizes the huge research work done in the Research Delivrable D5.2.3 [D5.2.3]. The results presented in this document demonstrate the transition from a theoretical research work into a real implementation of the innovative concepts braught in this poroject. It also reflects the collaboration between 3 different activities of the project: Activity 1 that is using the engine to handle delegation and access control in the secure backup demonstrator, Activity4 that contributed actively in the implementation of the UI elements of the engine (PolicyUI browser plug-in and the Privacy Tuner) and Activity 6 that is using the PrimeLife policy language and engine in their eCV use case.
The specification presented in this deliverable was partially initiated in the H5.3.2 internal heartbeat. Most of the concepts presented in this specification are implemented in the Final release of the policy engine [D5.3.3], except some optional mechanisms (due to the limitation of the integration resources).

The main sections describing the innovative concepts introduced in PrimeLife are Chapters 2, 3 and 5.

Chapters 6 and 7 describing the implementation design of the prototype are more technical. They cannot be considered as a specification, because they define one possible implementation approach.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

WEB 2.0 takes now major place in terms of Internet users, the biggest social network Facebook[1]. For example, counts more than 500 million active users. Blogging, or publishing on the Web 2.0 media not only impacts our virtual life, but has important consequences on our daily life; starting from a successful business media to a powerful political tool. In another hand web 2.0 relies essentially on the personal information input, and this lead to a non controlled leek of personal data. Users are asked to provide various kinds of personal information, starting from basic contact information (addresses, phone, email) to more complex data as preferences, friends' list, and photos. Service providers describe how users' data are handled using privacy policy, which is, more or less explicitly, presented to users during the data collection phase. Privacy policies are typically composed by a long text written in legal terms, and they rarely fully understood, or even read, by the users. As a result, most of the users creating account on web 2.0 applications are not aware about the conditions under their data are handled.

Therefore there is a need to support the user in this process, providing an as-automatic-as-possible mean to process privacy policy and compare them with user privacy preferences. In this context we propose in this report the specification documentation for the PrimeLife Privacy Policy Language (PPL) designed for handling access control and privacy. The design is independent of the underlying transport protocol and bindings that would be possible for HTTP and SOAP (Web Services), although such bindings are left to future work.. This includes the 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. This document also describes the technical specification of the proof of concept policy engine in charge of handling and enforcing the access and usage control rules described using the PPL language. This technical specification gives some hints and recommendation about the implementation design but it should not be a constraint for implementing the PPL engine.

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 information and credentials. The website needs to comply with the requirements of European data protection laws when it comes to the handling of personal data. An introduction and definitions relating to the

[1] http://www.facebook.com

protection of personal data can be found on the website of the Data Protection Office of the European Commission[2]. Further information can be found on the European Commission's Data Protection website[3]. With the PPL language and engine we propose an automated solution to define, declare and enforce such privacy requirements.

This deliverable consists of 8 Chapters. The terminology and the recommended architecture is described in this chapter. Chapter 2 and 3 present the new concepts related to data usage policies introduced in the PPL language. These new concepts are Obligation and the Authorization. Chapter 4 is a reminder to the XAML access control language that is extended by PPL. Chapter 5 details the language specification. The different protocols and message flows recommended for PPL are described in chapter 6. The implementation model is explained in chapter 7. Finally the chapter 8 or the appendix of the document contains all the schemas related to the PPL language and some examples written in XML.

1.1 Terminology

Access Control

This 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

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.)*

---

[2]Regulation (EC) 45/2001
http://ec.europa.eu/dataprotectionofficer/index.cfm?TargetURL=D_INTRO%20EUROPA
[3] EU Data Protection http://ec.europa.eu/justice/policies/privacy/index_en.htm

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 "downstream 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 expectations 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) define 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.

DHP

This term refers to an acronym of Data Handling Policy/Preference. We define it as a policy configuration file stating the usage condition and handling of a targeted data. In the case of Policy it refers to the description of how the Data Controller will handle the data collected. In the case of Preference, the Data Subject specifies how his data should be handled after being collected.

1.2 High Level Architecture Components

The high level architecture present an abstract overview of the PPL architecture and the interaction between the different entities; DS, DC and third party.

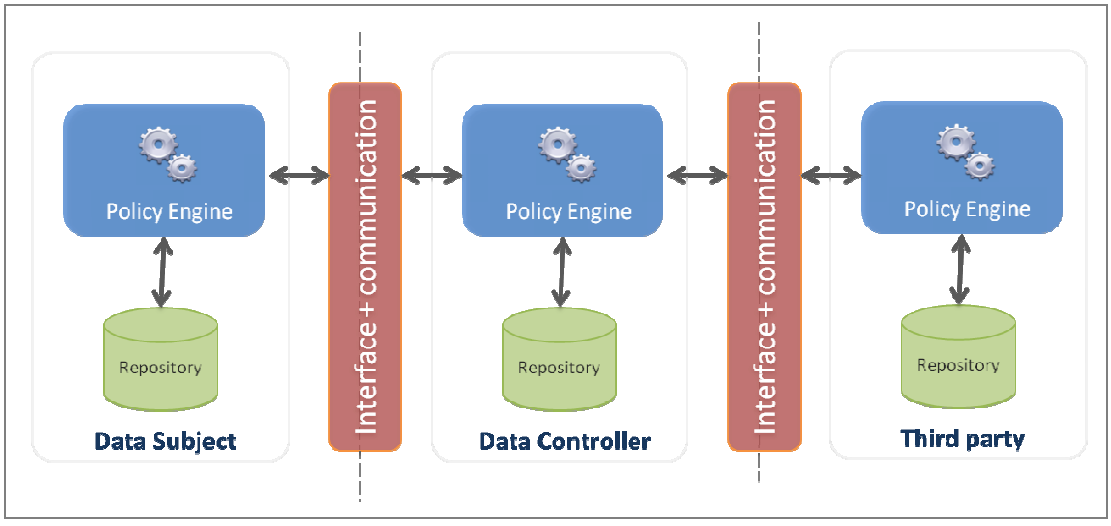

Figure 1. High Level Architecture Components

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, credential, etc). For this reason this module is replicated on the Data Controller side and the third party side.

Repository: represent the Data and policy repositories. It's a database containing data owned by the Data Subject. This data could be composed of personal data, credentials, certifications, and other information that should be used during the interaction with the Data Controller application. It contains also the policy files representing his privacy preferences.

Interface & communication: This interface represents a communication interface with the Data Controller implementing the message exchange protocol.

1.2.2 Data Controller

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

Repository: this repository represents a database that contains all the information collected from the Data Subject during an interaction session. This data represents personal data, credentials, certificates, and other information provided by the user. Also, this database contains the privacy policies related to the different resources and services that the Data controller held. This repository does not contain any information about the navigation history of a Data Subject (like IP addresses, page visited, etc.)

Interface & communication: 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.

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 Detailed architecture

The entire architecture can be represented as a three-layer design. The first layer is the *user interface layer* which informs a user about the policy matching result. The second - *business layer* - represents the main processing unit of the PPL Engine, which is Composed of different subcomponents that will be described in more detail below. The last layer represented by the *persistence layer* is the personal data (PII) and policies repository.

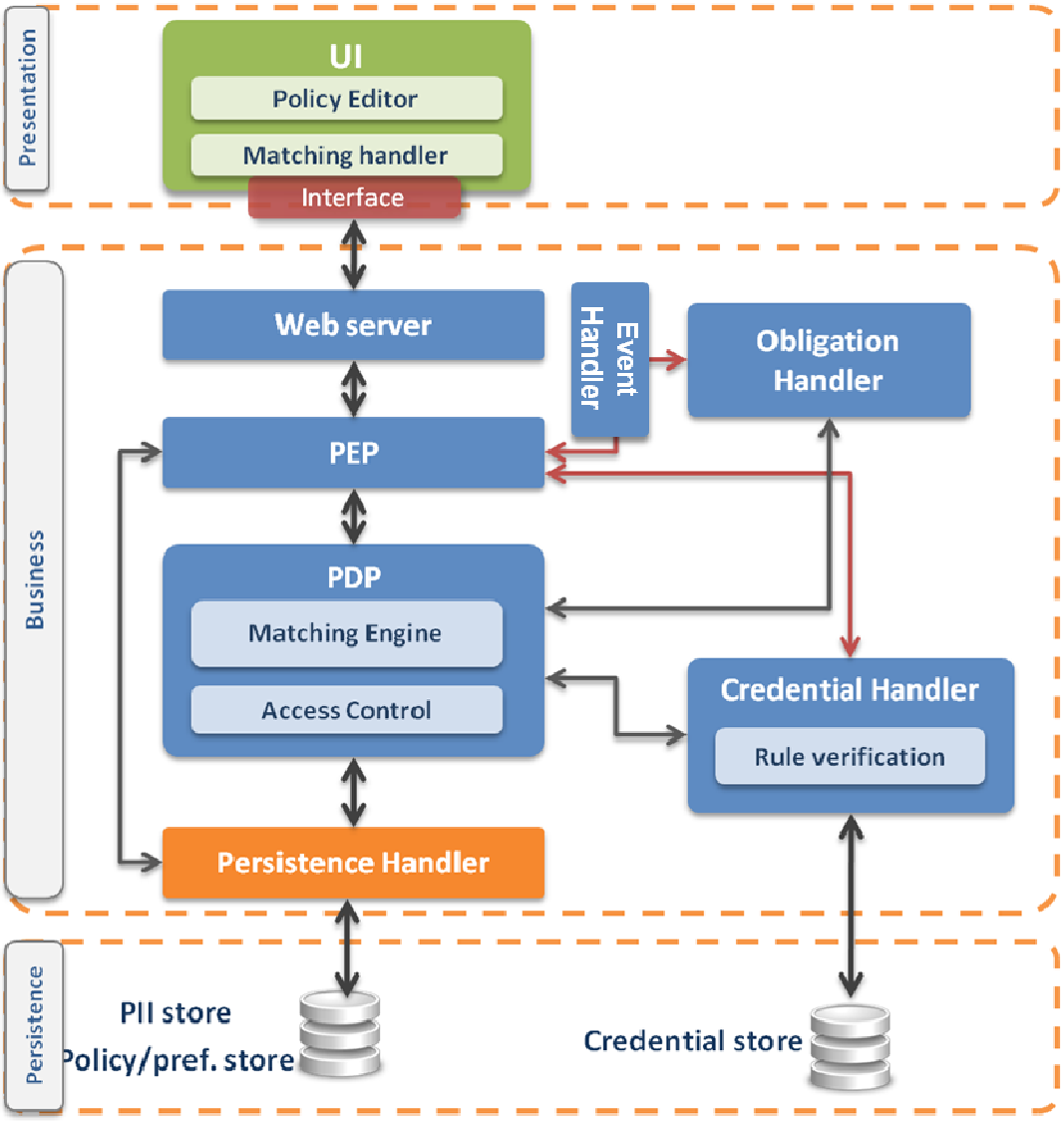

Figure 2: Detailed Architecture

1.3.1 Presentation Layer

The presentation layer is responsible for the display to the end user. The presentation layer contains two types of components:

- Policy editor: displays and provides a way to manage all the information related to the Data Subject, Data Controller and the third party. This information can be the personal data (PIIs, the credentials, etc), the privacy policy/preference, the information involved during a transaction between the different entities, …
- Matching handler: display to the user the matching result, by notifying a mismatch in case it occurs, and provide a set of tools that allow him to manage this mismatching (e.g., accepting a mismatch, updating local preferences).

The UI layer should be independent from the business layer. For that, an interface component might be present between these two layers to provide an abstraction level.

1.3.2 Business Layer

The business layer is composed of four main components that are implementing the concepts introduced within the PrimeLife project. These components are:

- **Policy Enforcement Point (PEP)**, which formats and then dispatches the messages to the associated components according to the state of the execution process. The decision made by the PDP is enforced in the PEP, meaning that if the PDP decided to provide data or enforce the access of a resource, this data/resource is collected, formatted and sent to the receiver through the PEP.
- **Policy Decision Point (PDP)** is the core of the PPL engine. All the decisions regarding processing of policies are taken in this component. It has two functionalities:
    - **Matching engine**, which performs comparison and matching between the preferences of the data subject and the data handling policy of the data controller. The matching is done to verify if the intentions of the data controller in terms of the data usage and obligations are compliant with the data subject preferences.
    - **Access control engine**, which is in charge of the application of the access control rules related to the local resources. It analyses a resource query, checks the access control policy of the requested resource and decides whether or not the requestor satisfies the rule.
- **Event handler** is a kind of monitoring component that tracks all the events that are related to the PIIs in the database. Such events are then reported to the obligation handler in order to check if there are triggers that are related to it. The event handler is also essential for the logging feature of the obligation handler.
- **Obligation handler**, which is responsible for handling the obligations that should be satisfied by the data controller/third party. This engine executes two main tasks:
    - Set up the triggers related to the actions required by the privacy preferences of the data subject.
    - Executes the actions specified by the data subject whenever it is required.

1.3.3 Persistence Layer

The other components displayed in Figure 2 play a secondary role in the concept introduced by the PPL engine:

- **Web server**, which is an embedded HTTP server that represents the entry point to the PPL engine. It can be seen as an interface facilitating access to the PEP.
- **Persistence handler**, which can be described as an interface between the business layer and persistence layer. It encapsulates access to the storage intermediate business objects. It creates a transparent way in which the business layer could access persisted data regardless of the location and storage model of the data it manipulates. In general, this layer is supported by a persistence framework. The defined object in this layer is generic DAO (Data Access Object). The persistence handler provides management functions to handle the DAO calls, which are basic CRUD (create, read, update, delete) operations performed on objects stored in the database.

1.4 Relationship to Existing Work (State of the Art)

Below we discuss existing related work and give an overview of the state of the art in the field of privacy policy languages.

1.4.1 Policy Matching

In 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 of 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 data. The server expresses its policy with a summarized version of P3P. This summary includes a bulleted list 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 makes this approach limited in terms of expressiveness and matching scope.

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

Table 1: Policy matching strategies

1.4.2 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 of 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.

1.4.3 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 unlikable 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].

1.4.4 Credential-Based Identity Management

We refer to Section 5.2.3 for an overview of the different technologies that are covered under our abstract notion of credentials.

The Simple Public Key Infrastructure (SPKI) [Ell99] was born as a joint effort to overcome the complication and scalability problems of the 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 an 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 for the proof of properties about encrypted values, and limited spending [BCC05] that allows for putting restrictions on how often the same credential can be used to access a service, without compromising anonymity.

Besides 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 mark-up languages aiming at unified mark-up 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 associated identity information (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 fulfil 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.

1.4.5 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.

1.5 Contributions of the Proposed Language

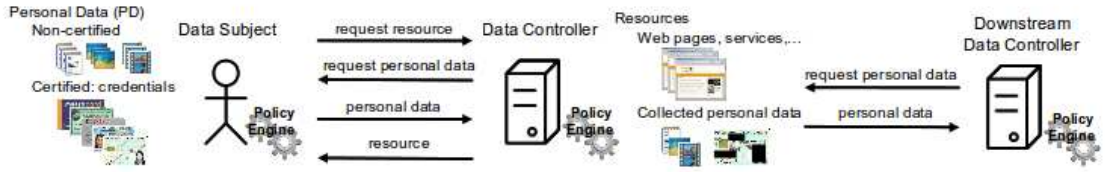

Figure 3. Downstream usage

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.
- Event based obligations: Obligation actions are no more defined statically. With the combination Trigger/Action events can be attached to the execution of an obligation. These events can be enabled through any contextual information captured in the system like for example: time, actions on date, system modification etc.

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.

Chapter 2

Usage Control : 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 fulfil the promise by executing and/or preventing a specific action after a particular event, e.g. time, and optionally under certain conditions".

This chapter presents the implementation of obligations in PrimeLife Policy Language (PPL). Two main components are presented: the Obligation matching Engine (OME), which is in charge of comparing obligations, and the Obligation Enforcement Engine (OEE), which is in charge of enforcing obligations.

2.1 Introduction

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 1.4.5 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

misbehaviour 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 and the second aspect by providing mechanisms to match obligations (see Section 2.4). The third aspect is also covered in Section 2.4. The fourth challenge is addressed by assuming a simple trust model: audit and reputation mechanisms.

2.2 Key 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 fulfilment 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.

2.3 Obligation Language

Based on the different aspects of obligations, in this section we define the structure of an obligation. 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:

Do ***Action*** when ***Trigger***

For instance, we can define "Do {Notify User} when {User's personal data is read}". Obligations are thus defined as a set of triggers and an action.

The language and schema for defining obligations 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.

2.4 Obligation Matching Engine (OME)

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.

2.4.1 Overview

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.

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). In addition, the obligation matching engine supports a second procedure call that also requires both obligations as in the latter and provides the full obligations part of the sticky policy. It will return a sticky policy in matching and not matching cases. Optionally, when the policy does not match the preferences, more details on the mismatch are provided. For the schema of the sticky policy and other details about obligation mismatches, please refere to section 1.1.

2.4.1.1 Matching Rules

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

$$L{:}Policy \trianglelefteq R{:}Policy \Leftrightarrow ((L.authorizations \trianglelefteq R.authorizations) \wedge (L.obligations \trianglelefteq R.obligations))$$

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 PS) is less restrictive than another policy (e.g. user preferences PU) when the list of rights and the list of obligation are more restrictive. Matching function for lists of rights is:

$$L{:}ListAuthorizations \trianglelefteq R{:}ListAuthorizations \Leftrightarrow \forall (i \in L) : \exists (j \in R) \text{ where } (i \trianglelefteq j)$$

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) )
```

Where “action” is the action resulting from the obligation, and “triggers” is the list of triggers resulting in the execution of the action.

Matching function for a list of triggers is:

```
L:ListTriggers ⊴ R:ListTriggers ⇔ ∀ (b ∈ R) : ∃ (a ∈ L) where (a ⊴ 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 ⊴ R:TriggerAtTime ⇔ ( (L.start ≥ R.start) ∧
                            (L.start + L.maxDelay ≤ R.start + R.maxDelay) )

L:TriggerDataAccessedForPurpose ⊴ R:TriggerDataAccessedForPurpose ⇔
    ( ( L.dataRef == R.dataRef ) ∧ ( L.purpose ⊇ R.purpose ) )
```

And the following rules to match actions:

```
L:ActionDeletePersonalData ⊴ R:ActionDeletePersonalData ⇔
                                        (L.personalData == R.personalData)

L:ActionNotifyDataSubject ⊴ R: ActionNotifyDataSubject ⇔
                                                  (L.Address == R.Address)
```

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.

2.4.1.2 Usual Obligation Triggers

This section briefly describes usual triggers. Even if some aspects are underspecified, we hope that providing this draft specification helps with understanding what we are aiming at.

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.

Table 2: Trigger at Time

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.		

Table 3: Trigger Personal Data Accesses for Purpose

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	L: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		

Table 4: Trigger Personal Data Deleted

Name	TriggerPersonalDataSent		
Parameters	DataRef	personal data	Reference to the personal data concerned by the obligation
	anyUri	Id	Third party 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.		

Table 5: Trigger Personal Data Sent

Name	TriggerDataSubjectAccess		
Parameters	DataRef	personal data	Reference to the personal data concerned by the obligation
	anyUri	url	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))		

Table 6: Trigger Data Subject Access

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

2.4.1.3 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.

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.		

Table 7: Action Deleted 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)		

Table 8: Action Anonymize Pesonal data

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.)
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))		

Table 9: Action Notify Data Subject

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		

Table 10: Action 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		

Table 11: Action secure Log

2.4.1.4 Normalisation of Obligations

In the obligation language, one can express the same obligations in multiple ways. In order to be able to provide the same matching-result on the same obligations, the obligation matching engine (OME) does a normalization of obligations before running the matching process. OME provides a configurable flag to normalize a given obligation by splitting it up into multiple obligations, if this is needed. That is, if there is an obligation with multiple triggers, then it is transformed in multiple obligations with one trigger and the same action.

2.4.2 Implementation of OME

The obligation matching engine is implemented as a “Windows service”. It starts a Web Service with an address specified in the config file.

2.4.2.1 Web Service API

The web service provides these methods:

- ```
  Boolean Match(  ObligationsSet preferenceObligationSet,
                  ObligationsSet policyObligationSet);
  ```

  This method takes a preference an a policy obligation and returns a single boolean value, indicating whether the provided obligations do or do not match.

- ```
  Boolean MatchAsString(  string preferenceObligationSetAsString,
                          string policyObligationSetAsString);
  ```

  Same as above, but parameters are XML-serialized representations of the objects above. This function is deprecated and should no longer be used.

- ```
  PrimeLifeObligationMismatch.ObligationsSetMm
              GetStickyPolicy(ObligationsSet preferenceObligationSet,
                              ObligationsSet policyObligationSet);
  ```

  This method takes a preference and a policy obligation and will provide the full obligations part of the sticky policy. It is supposed to return a sticky policy in matching and not matching cases. Please have a look at the examples provided in Chapter 8 for more details on how sticky policies will look like in different cases.
  ```

- `string GetStickyPolicyAsString( string preferenceObligationSetAsString, string policyObligationSetAsString);`

  Same as above, but parameters are XML-serialized representations of the objects above. This function is deprecated and should no longer be used.

2.4.2.2 Configuration

Starting and Stopping the Service: As OME is a Windows service, it is registered as a service during installation process and then listed in the services section at the Windows management console. You will find OME listed as “PrimeLife OME Service”.

Traces and Debug Outputs: OME is a Windows service and no console or graphical interface can be used to output some information. Some basic information about the start and about received events as well as warnings and errors are sent to the Windows application event log. The easiest way to access those traces is by using the start menu named “Obligation Matching Engine Outputs (Event Viewer)”. Moreover, there is also a debug log which is far more informative. It is a text file which is located on \LocalAppData\EMIC\PrimelifeOEE of the LocalService User.

Configuration File: The configuration file of OME can be found in the installation directory of OEE. It is named “OMEWindowsService.exe.config”. There are several configurable parameters and flags that influence the behaviour of OME. These can be found in the “applicationSetting” element of the configuration file and are described next:

- Parameter ServiceAddress: This parameter sets the URL, where OME should provide its web service. It is mandatory as there is no hardcoded default value given for it.
- Parameter RawMismatchMode: The raw mismatch mode can be turned on and off with the equal-named parameter. It is designed to handle a certain mismatch edge case in a non-ordinary way in order to provide a useful output. Such an edge cases is if a pair of comparable obligations (that is, a preference and a policy obligation have at least some similarities) do have at least one not comparable trigger or action (that is, the triggers or actions do not have any similarities).

  In such a situation, the regular handling, which is, comparing every pair of triggers and actions separately and summing up the resulting similarity score, usually do not make sense. It is then better to give up comparing obligation properties separately, stating that these obligations do not match entirely. The resulting sticky policy would be as follows. The policy obligation is listed under obligations with the matching attribute to false at the top obligation level and the preference and policy obligation are given, as a whole, as mismatch details. This is the default value, which is raw mismatch mode in on. Setting this parameter to false would let OME behave regularly in these edge cases.
- Parameter ProcessObligationsNormalized: This parameter is a switch for the normalization of obligations before they are processed. Please see section 2.4.2.3 for details on normalized processing of obligations.

Sample Client Application: The client application (“OMEClient.exe”) calls the Web Service. The address of the WS can be changed in its configuration file, i.e. “OMEClient.exe.config”. The Client calls Match() and GetStickyPolicy() with preference-side and policy-side obligation sets from files preferenceObligationSet.xml and policyObligationSet.xml respectively.

2.4.2.3 Normalized Processing of Obligations

In PPL, one can express the same obligations in multiple ways. In order to be able to provide the same matching-result on the same obligations, there have to be some kind of normalization. OME

provides a configurable flag “ProcessObligationsNormalized“ in its configuration file to normalize a given obligation by splitting it up into multiple obligations, if this is needed. That is, if there is an obligation with multiple triggers, then it is transformed in multiple obligations with one trigger and the same action.

2.4.2.4 Mismatch Description with Respect to Similarity

A sticky policy contains, in addition to a set of obligations, one or more mismatch elements which are nested in the mismatches element. A mismatch element is designed to carry information about a particular mismatch, pointing at the according element in the preference and at the policy obligation, which are supposed to be the origin of the mismatch. This is the regular behaviour.

However, there are also some special cases:

- Comparing two trigger or action elements, one may be from preference and the other from policy, assume they have no similarity with respect to the similarity estimation algorithm, then both obligations, each as a whole, are taken as the origin of mismatch and are shown in a mismatch element accordingly.
- Assume two different obligations have no similarity with respect to the similarity estimation algorithm, then
    - no mismatch element will appear referring to that mismatch,
    - the obligation will not be listed in the obligation set of the sticky policy and
    - the overall obligation set element of the sticky policy will have the attribute matching set to false and the attribute infinite set to true.

The mismatch element provides two kinds of refer mechanisms. First, the mismatching elements of the obligations and the according mismatch element are labelled with a mismatchId attribute, which have the same sticky-policy-wide, unique identifier. Second, if two elements, e.g. obligation elements or some child elements, that mismatches against each other provide the elementId attribute, which should be unique inside a preference or policy, then these attributes are used inside the mismatch element for pointing to the mismatch originating elements. Please also have a look the examples in the following sections to get better understanding of this mechanism.

2.4.2.5 Examples

1. Matching Example	
Obligations in user’s preferences	• use of PII for purpose contact and pseudo-analysis needs be to logged within 5 minutes • collected PII has to be deleted within 7 days
Obligations in service’s policy	• will log any use of PII for purpose … o … contact within 5 minutes o … delivery within 15 minutes o … pseudo-analysis within 30 seconds • will delete collected PII within 5 days
Result	Obviously, these obligations do match. The resulting sticky policy will include all obligations which are part of the preference and of the policy.

2. Mismatching Example: Deletion of collected PII will be done in a longer period than expected by user	
Obligations in user's preferences	• use of PII for purpose contact and pseudo-analysis needs be to logged within 5 minutes • collected PII has to be deleted within 7 days
Obligations in service's policy	• will log any use of PII for purpose … o … contact within 5 minutes o … delivery within 15 minutes o … pseudo-analysis within 30 seconds • will delete collected PII within **10** days
Result	This example will lead to a mismatch.

The corresponding sticky policy will (still) lists two obligations, labelling the latter as a mismatch with attribute *matching* set to *false* on all elements down the path from the mismatch origin to the obligation element. Furthermore a mismatchId attribute will be added to the mismatch origin. This mismatchId, which is unique for a sticky policy, is used to refer to within a second part of the sticky policy, the mismatches element.

The mismatches element will list all mismatches in the document as long as the pairs of mismatching content has at least some kind of similarity. In this example, there is a mismatch at the delay before deleting a collected PII, which is shown next as listings of the TriggerAtTime elements.

Preference-side:

```
<ob:TriggerAtTime>
  <ob:Start>
    <ob:StartNow/>
  </ob:Start>
  <ob:MaxDelay elementId="myIdA">
    <ob:Duration>
      P0Y0M7DT0H0M0S
    </ob:Duration>
  </ob:MaxDelay>
</ob:TriggerAtTime>
```

Policy-side:

```
<ob:TriggersSet>
  <ob:TriggerAtTime>
    <ob:Start>
      <ob:StartNow/>
    </ob:Start>
    <ob:MaxDelay elementId="myId1">
      <ob:Duration>
        P0Y0M10DT0H0M0S
      </ob:Duration>
    </ob:MaxDelay>
  </ob:TriggerAtTime>
```

This mismatch would result in a mismatch element at the sticky policy, which points at the two MaxDelay properties of the TriggerAtTime elements using their elementIds.

```
<Mismatch mismatchId="mismatchId_4">
  <Similarity>0.5714285714285714</Similarity>
  <Preference elementId="myIdA">
    <Duration xmlns="http://www.primelife.eu/ppl/obligation">
      <Duration>P0Y0M7DT0H0M0S</Duration>
    </Duration>
  </Preference>
  <Policy elementId="myId1">
    <Duration xmlns="http://www.primelife.eu/ppl/obligation">
      <Duration>P0Y0M10DT0H0M0S</Duration>
    </Duration>
  </Policy>
</Mismatch>
```

3. Mismatching Example: The User provides two obligations to delete PII within 7 days and within 10 days, whereas the service offers to delete within 13 days	
Obligations in user's preferences	• use of PII for purpose contact and pseudo-analysis should be logged within 5 minutes • **collected PII should be deleted within 10 days** • collected PII should be deleted within 7 days NB:Such preferences are not expected but well formatted and thus handled.
Obligations in service's policy	• will log any use of PII for purpose … o … contact within 5 minutes o … delivery within 15 minutes o … pseudo-analysis within 30 seconds • will delete collected PII within **13** days
Result	As one can see, the second obligation will lead to a mismatch. But having two preferences not fulfilled, there need to be two mismatches listed.

This means, two obligations are shown in the obligation-set-part of the sticky policy, differing only in mismatch labels.

```
 <Obligation matching="false">
  <TriggersSet matching="false">
    <TriggerAtTime matching="false">
      <Start>
        <DateAndTime>2010-08-19T11:50:37.1130248+02:00</DateAndTime>
      </Start>
      <MaxDelay mismatchId="mismatchId_5" matching="false">
        <Duration>P0Y0M13DT0H0M0S</Duration>
      </MaxDelay>
    </TriggerAtTime>
  </TriggersSet>
  <ActionDeletePersonalData />
</Obligation>

<Obligation matching="false">
  <TriggersSet matching="false">
    <TriggerAtTime matching="false">
      <Start>
        <DateAndTime>2010-08-19T11:50:37.1130248+02:00</DateAndTime>
      </Start>
      <MaxDelay mismatchId="mismatchId_9" matching="false">
        <Duration>P0Y0M13DT0H0M0S</Duration>
      </MaxDelay>
    </TriggerAtTime>
  </TriggersSet>
  <ActionDeletePersonalData />
</Obligation>
```

Also there are two mismatch elements pointing to the two different policy obligations and the same policy obligation.

```
<Mismatch mismatchId="mismatchId_5">
  <Similarity>0.7</Similarity>
  <Preference elementId="myIdA">
    <Duration xmlns="http://www.primelife.eu/ppl/obligation">
      <Duration>P0Y0M10DT0H0M0S</Duration>
    </Duration>
  </Preference>
  <Policy elementId="myId1">
    <Duration xmlns="http://www.primelife.eu/ppl/obligation">
      <Duration>P0Y0M13DT0H0M0S</Duration>
    </Duration>
  </Policy>
</Mismatch>
<Mismatch mismatchId="mismatchId_9">
```

```
  <Similarity>0.14285714285714291</Similarity>
  <Preference elementId="myIdB">
    <Duration xmlns="http://www.primelife.eu/ppl/obligation">
      <Duration>P0Y0M7DT0H0M0S</Duration>
    </Duration>
  </Preference>
  <Policy elementId="myId1">
    <Duration xmlns="http://www.primelife.eu/ppl/obligation">
      <Duration>P0Y0M13DT0H0M0S</Duration>
    </Duration>
  </Policy>
</Mismatch>
```

4. Mismatching Example: Policy does not log usage of PII for all purposes	
Obligations in user's preferences	• use of PII for purpose contact and pseudo-analysis should be logged within 5 minutes • collected PII should be deleted within 7 days
Obligations in service's policy	• will log any use of PII for purpose … o … contact within 5 minutes o … delivery within 15 minutes • will delete collected PII within 5 days
Result	The mismatching part of these obligations is that the policy is not willing to log the use of PII for purpose delivery.

As this detail level is not handled by the obligation matching engine by now, the mismatch description of the sticky policy would list both obligations as a whole. This behavior might change in the next version, depicting the difference in detail.

```
<Mismatch mismatchId="mismatchId_17">
  <Similarity>0.5</Similarity>
  <Preference>
    <Obligation xmlns="http://www.primelife.eu/ppl/obligation">
      <TriggersSet>
        <TriggerPersonalDataAccessedForPurpose>
          <Purpose xmlns="http://www.primelife.eu/ppl">
              http://www.w3.org/2002/01/P3Pv1/pseudo-analysis</Purpose>
          <MaxDelay>
            <Duration>P0Y0M0DT0H5M0S</Duration>
          </MaxDelay>
        </TriggerPersonalDataAccessedForPurpose>
      </TriggersSet>
      <ActionLog />
    </Obligation>
  </Preference>
  <Policy>
    <Obligation xmlns="http://www.primelife.eu/ppl/obligation">
      <TriggersSet>
        <TriggerPersonalDataAccessedForPurpose>
          <Purpose xmlns="http://www.primelife.eu/ppl">
              http://www.w3.org/2006/01/P3Pv11/delivery</Purpose>
          <MaxDelay>
            <Duration>P0Y0M0DT0H15M0S</Duration>
          </MaxDelay>
        </TriggerPersonalDataAccessedForPurpose>
      </TriggersSet>
      <ActionLog />
    </Obligation>
  </Policy>
</Mismatch>
```

5. Mismatching Example: Preference and Policy did not match on a Single Obligation	
Obligations in user's preferences	• use of PII for purpose contact and pseudo-analysis should be logged within 5 minutes
Obligations in service's policy	• will delete collected PII within 5 days
Result	As these obligations have nothing in common, the obligations matching algorithm will neither match exactly nor match for similarity.

The resulting sticky policy would consist of an empty obligation set and no mismatch detail descriptions, as shown below.

```
<?xml version="1.0"?>
<ObligationsSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" matching="false"
xmlns="http://www.primelife.eu/ppl/obligation/mismatch">
  <ObligationsSet matching="false" infinit="true"
xmlns="http://www.primelife.eu/ppl/obligation" />
```

6. Matching Example: Preference has several separated Obligations which will be met by a single Policy	
Obligations in user's preferences	• use of PII for purpose contact should be logged within 5 minutes • use of PII for purpose delivery should be logged within 15 minutes • use of PII for purpose pseudo-analysis should be logged within 30 seconds
Obligations in service's policy	• will log any use of PII for purpose … o … contact within 5 minutes o … delivery within 15 minutes o … pseudo-analysis within 30 seconds
Result	Obviously the policy matches each preference. By processing the obligations in a normalized way, OME is able to detect these as a match.

Chapter 3

Usage Control: 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.

3.1 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>
```

3.2 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>
```

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.

Table 12: Authorization to use the data for purpose

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 purpose-1 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.

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.

3.3 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 (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: 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>
```

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 5.1.6).

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

Table 13: Authorization to forward Data

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.

3.4 Authorization Matching Exceptions

In some particular cases the matching between authorizations become unclear, especially when some elements are set to empty or null. We decided to covers these situations using exceptions in the result of the matching. The 5.1.2.2 defines how the authorization matching should result when some elements are missing or empty.

Table 14: Authorization matching exception strategies

Policy AuthzSet	**Preferences AuthzSet**	**Matching Result**	**Content of Sticky Policy**
no AuthzSet nothing is defined => could be any value	no AuthzSet	✘	empty AuthSet no MM
	emptyAuthzSet	✘	empty AuthSet no MM
empty AuthzSet nothing is allowed => always match	no AuthzSet	✓	empty AuthSet no MM
	empty AuthzSet	✓	empty AuthSet no MM
no AuthPurp	no AuthPurp	✘	no Auth for SP no MM
AuthPurp	no AuthPurp	✘	policy AuthPurp MM: empty pref
no AuthPurp	AuthPurp	✓/✘	preference AuthPurp MM: empty policy
AuthPurp	AuthPurp	✓/✘ pref.containsAll(polPurp)	policy AuthPurp MM: pref, pol
no AuthDSU	no AuthDSU	✘	no Auth for SP no MM
authDSU	no AuthDSU	✓/✘ match if policy does not allow DSU	policy AuthDSU MM: empty pref
no AuthzDSU	AuthDSU	✓/✘ match if pref allow DSU	preference AuthDSU MM: empty policy

Chapter 4

Access Control: 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.

As one of the main requirements of the PPL language is to express Access control beside the usage control, we decided to adopt the XACML language as the basis for expressing access control in stead of specifying a new one. This decision was done after a comparative study done with other access control languages. The arguments explained previously were strong enough to rely on this language. In the section 5.1.1 we explain more in details how the XACML elements are extended in PPL.

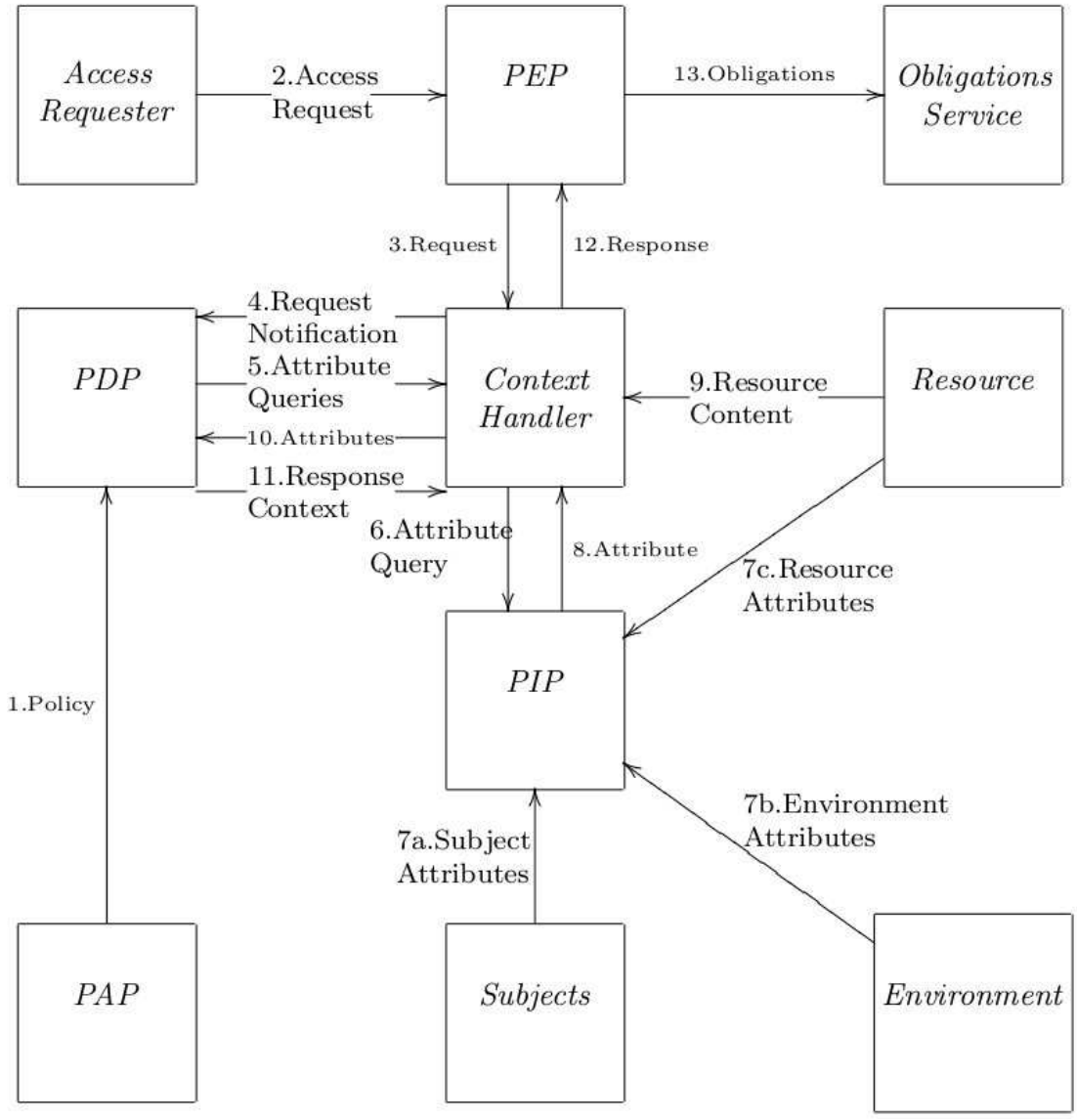

Figure 4. Overview of XACML dataflow [eXt09]

Figure 4 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.

4.1 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". The policy below 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.

```
<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>
```

4.2 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.

```
<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>
```

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.

Chapter 5

The PrimeLife Policy Language

5.1 Policy Language Model

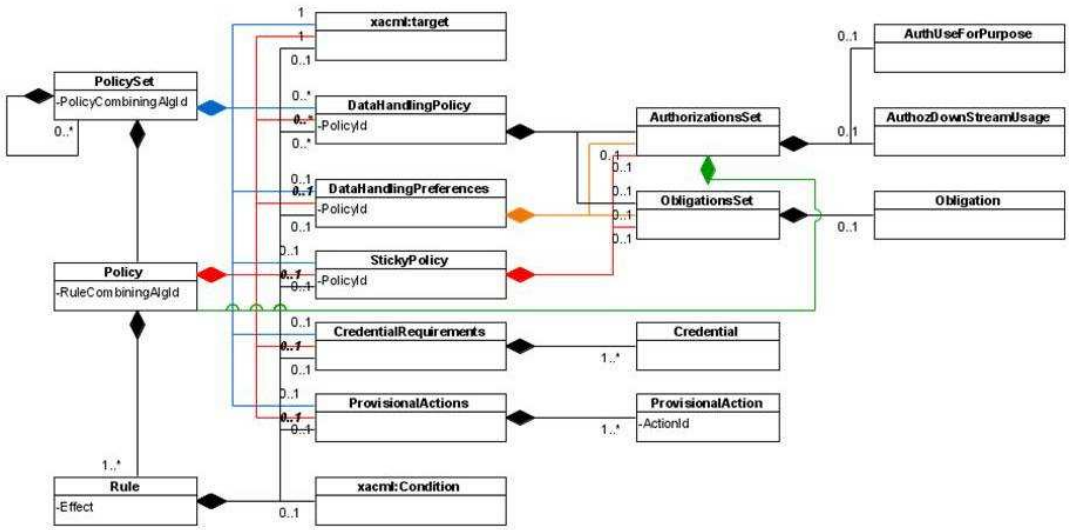

Figure 5. Model of our policy language

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.

5.1.1 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, are also 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.

5.1.2 Authorization Element

Data handling policy, preferences policy and sticky policy contain a set of authorization elements. Authorizations specify actions that data controller is allowed to perform on the collected data. PPL language creators found it impossible to define an exhaustive list of authorizations that could cover all the needs that may ever arise in the real world. Rather than that, a generic, user-extensible structure for authorizations was defined, so that new, possibly industry-specific

terminology can be added later on. Current specification provides a basic authorization vocabulary that covers usage of the data for certain purposes and for downstream access control (forwarding the information to third parties).

5.1.2.1 Authorization use for purpose

Authorization use for purpose is defined as the authorization to use the information for a particular set of purposes. They are referred to by standard URIs specified in agreed-upon vocabulary. That vocabulary of URIs may be organized as a flat list or a hierarchical ontology. PPL specification proposes to use the list of primary purposes defined in P3P 1.1 (see section 3.2)

5.1.2.2 Authorization for downstream usage

Element <AuthzDownstreamUsage> gives possibility to permit forwarding the information protected by this policy to the third party, so-called downstream data controller. Optionally, this authorization enables the data subject to specify the policy under which the information will be made available, i.e., the minimal access control policy that the (primary) data controller has to enforce when sharing the information with downstream data controllers. This is the reason why there is a possibility to nest <Policy> element inside the <AuthzDownstreamUsage> element. That indicates that the third party access to the data will be ongoing by the rules specified inside this nested policy, which allows putting additional restrictions and obligations that were not imposed on the primary data controller.

5.1.3 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, plus a <Condition> element describing the conditions that the attributes of these credentials have to satisfy. 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 must have the same identifier.

The <Credential> element can contain restrictions that apply to the credential. These restrictions can be expressed two ways: by means of a list of <AttributeMatchAnyOf> elements within the <Credential> element, allowing matching attributes of the credential against a list of candidate values, or by means of the generic <xacml:Condition> element that is a sibling of the list of <Credential> elements, and that contains all other conditions related to the credential attributes, including conditions that affect attributes from multiple credentials. 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.

5.1.4 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.

5.1.5 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.

5.1.6 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>.

5.1.7 Provisional Actions

This element describes a set of provisional actions that needs to be fulfilled in order to satisfy an access control rule for certain server-side resource. The schema of the <ProvisionalAction> element is consciously kept independent of the particular provisional action that needs to be fulfilled, in order to allow the possibility for the future extensions with the new provisional action types.

In the current version of PPL specification the list of possible provisional actions includes revealing of users' (requestor) attributes, signing a statement, "spending" the credential which is understood as a mean of restricting number of times that the same credential can be used.

The vocabulary for those terms was defined as a set unique identifiers in the form of URI:

- `http://www.primelife.eu/Reveal` requires access requestor to reveal one of his attributes (piece of personal information) like an e-mail address or a name
- `http://www.primelife.eu/RevealUnderDHP` requires revealing the attribute with the promise that it will be treated under certain data handling policy
- `http://www.primelife.eu/RevealTo` requires revealing the attribute to the third party (entity which is external to the resource owner)
- `http://www.primelife.eu/RevealToUnderDHP` requires revealing the attribute to the third party with the promise that it will be treated under certain data handling policy
- http://www.primelife.eu/Sign signing a statement (requires a mechanisms that will allow verification of the signature)
- http://www.primelife.eu/Spend spending a credential, interpreted as a way to limit use of this credential to a specific number of times

List of the possible provisional actions could be extended in the future to facilitate the new requirements.

5.1.8 Relation to XACML Obligations

As an alternative to adding new elements to the XACML schema, we considered to use the standard <xacml:Obligations> element to specify the obligations that we embed in the data handling policies or preferences. The reason that we didn't pursue this option is that <xacml:Obligations> can only be used to specify obligations that the local PEP and 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.

5.2 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 modelled 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.

5.2.1 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 Issuer'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 fulfil the given requirements with her credentials. If so, she produces a token that proves her fulfilment 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.

5.2.2 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.

5.2.3 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 fulfil 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 fulfil 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.

5.2.4 Credential 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.

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.

5.3 Policy Sanitization

An important issue is how the server should communicate its access control policy to the requester. For instance, suppose that an authorization imposes that attribute nationality should be equal to "US". Should the server communicate such a condition to the requester? Or should it just inform the requester that it has to state the nationality? Clearly there is no unique response to whether one option is better than the other, and which one is to be preferred depends on the specific context and information involved. We can notice that communicating the complete policy (i.e., the fact that the policy will grant access if the nationality is US) favors the privacy of the requester. In fact, a requester can know, before releasing credentials or information to the server, whether the release will be sufficient to acquire access to the service. In particular, a client associated with a non-US user can avoid disclosing the nationality of the user. By contrast, communicating only part of the policy favors the privacy of the server. As a matter of fact, the access control policy, and the information on which it evaluates, can be considered sensitive too and as such needs to be protected. For instance, while the server might not mind disclosing the fact that access to a service is restricted to US citizens, it might not want to disclose other conditions (or values against which properties are evaluated) as they are considered sensitive. As an example, consider an authorization allowing access to a service to those users who work for an organization that does not appear in a Secret Black List (SBL) kept by the server. The corresponding subject expression is: $c \triangleright type = \text{employment} \wedge c.employer \notin \text{SBL}.$

Communicating the complete policy to the requester (and allowing its evaluation by the requester) would imply releasing the subject expression, together with the state of black list SBL. Also, assuming the context of SBL is not released, the requester will know, in case it will not be granted access that its employer is black listed. This is clearly an information the server does not wish to disclose; rather the server will want to maintain confidential the condition and simply state that the employment certificate is required. Among the two extremes of the current XACML approach of simply returning indeterminate, on one side, and of completely disclosing the policy, on the other side, there are therefore other options offering different degrees of protection to the server policy

and of information communicated to the requester. Each condition appearing in the policy can then be subject to a different disclosure policy, regulating the way the presence of such a condition should be communicated to the requester. We can distinguish five different disclosure policies, with each one potentially used independently in any condition appearing in an expression. In terms of our formal notation, we denote the disclosure policy by including the portion of a condition to not be disclosed in square brackets. For concreteness of the discussion, we assume a condition $c \triangleright M\ \pi\ m$; the case of a condition on an attribute (i.e., of the form $c.A\pi\upsilon$ or $A\pi\upsilon$) is analogous. The following disclosure policies can be associated with the condition.

- None. Nothing can be disclosed about the condition. It corresponds to the XACML approach as only the information that the outcome of the policy is indeterminate is communicated, since there are conditions that cannot be evaluated. Formally, the condition will appear in the expression completely included in square brackets, that is, $[c \triangleright M\ \pi\ m]$
- Credential. Only the information that there is a condition imposed on some metadata about a credential (or on some attributes of the credential) can be disclosed. The metadata (or attributes) on which the conditions are evaluated are not released. Formally, the condition will appear in the expression as $c \triangleright [M\ \pi\ m]$.
- Property. Only the information that a property (metadata or attributes of a credential, or uncertified statements) needs to be evaluated can be released; no information can be released on the control that will be enforced on the property. Formally, the condition will appear in the expression as $c \triangleright M\ [\pi\ m]$.
- Predicate. Only the information that a property (metadata or attributes of a credential, or uncertified statements) needs to be evaluated and the predicate with which it is evaluated can be released; no information can be released on the value/s against which the evaluation is performed. Formally, the condition will appear in the expression as $c \triangleright M\ \pi\ [m]$
- Condition. The condition can be fully disclosed as it is. Formally, the condition will appear in the expression with no square brackets, signalling that no component is subject to disclosure restriction, that is, $c \triangleright M\ \pi\ m$.

Disclosure policy	Condition in expression	Communication to the client
none	$[c \triangleright M\ \pi\ m]$ $[c.A\ \pi\ v]$	[ ] [ ]
credential	$c \triangleright [M\ \pi\ m]$ $c.\ [A\ \pi\ v]$	$c \triangleright$ [ ] $c.$ [ ]
property	$c \triangleright M\ [\pi\ m]$ $c.A\ [\pi\ v]$	$c \triangleright M$ [ ] $c.A$ [ ]
predicate	$c \triangleright M\ \pi\ [m]$ $c.A\ \pi\ [v]$	$c \triangleright M\ \pi$ [ ] $c.A\ \pi$ [ ]
condition	$c \triangleright M\ \pi\ m$ $c.A\ \pi\ v$	$c \triangleright M\ \pi\ m$ $c.A\ \pi\ v$

Table 1 Disclosure policies and their effect on conditions [ACP11]

Table 1 summarizes the different disclosure policies reporting the formal notation with which they appear in the expression and the consequent communication to the client in the dialog.

Note that the disclosure policies of the server, affecting the information released to the requester about the conditions appearing in the policy, also impact the way the requester can satisfy the conditions. In particular, the credential policy implies that the requester will not know which information in the credential is needed and therefore will have to release the credential in its entirety (assuming that the credential to which the condition refers is known by other conditions in the policy, else the requester will have to disclose all its credentials). The property policy implies that the requester can selectively disclose the property in the credential (or utter it, in case of a condition on uncertified properties). The same for the predicate policy, where the requester however knows also against what predicate the property will be evaluated. Finally, in the case of the condition policy, the requester can either provide the property (but it can assess, before submitting, whether such a release will satisfy the condition) or provide a proof that the property is satisfied.

Example 1. Consider a policy stating that "a user can access a service if her nationality is Italian, her city of birth is Milan, and her year of birth is earlier than 1981". Suppose that all attributes mentioned in the policy must be certified by an X.509 identity card or by a SAML passport both released by IT Gov. The policy is formally stated as:

$$((c_1 \triangleright type = \text{identity\_card} \wedge c_1 \triangleright method = \text{X.509}) \vee$$
$$(c_1 \triangleright type = \text{passport} \wedge c_1 \triangleright method = \text{SAML})) \wedge$$
$$c_1 \triangleright issuer\ [= \text{IT\_Gov}] \wedge c_1.nationality\ [= \text{Italy}] \wedge$$
$$c_1.city\_of\_birth = \text{Milan} \wedge c_1.year\_of\_birth < [1981]$$

Here, the square brackets representing the disclosure policies implicitly state that: i) conditions on metadata *type* and method and attribute *city_of_birth* can be eventually disclosed as they are; conditions on metadata *issuer* and attribute *nationality* need to be protected by hiding the control that will be enforced on them; and iii) condition on attribute *year_of_birth* needs to be protected by hiding the value against which the evaluation will be performed. If the above policy applies to a request submitted by a requester for which the server has no information, the following conditions are communicated to the requester.

$$((c_1 \triangleright type = \text{identity\_card} \wedge c_1 \triangleright method = \text{X.509}) \vee$$
$$(c_1 \triangleright type = \text{passport} \wedge c_1 \triangleright method = \text{SAML})) \wedge$$
$$c_1 \triangleright issuer\ [] \wedge c_1.nationality\ [] \wedge$$
$$c_1.city\_of\_birth = \text{Milan} \wedge c_1.year\_of\_birth < [].$$

The requester can satisfy such conditions by releasing either an identity card or a passport containing the requested attributes.

Further details can be found in [ACP11].

5.4 Attribute Types and Credential Types

5.4.1 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[4]; 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 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 inheritances are allowed, meaning that the subtype contains all the attributes of all super types.

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 satisfying a policy requiring to you to reveal your friend's first name from a Facebook credential.[5]

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 super type, 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.

5.4.2 Defining Credential Type Ontologies in OWL

A credential type is modelled 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.

---

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

[5] 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.

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 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.

5.4.3 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: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>
```

5.4.4 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: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>
```

5.4.5 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:MatchV
alue>
          </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</AttributeVa
lue>
              </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</Attri
buteValue>
          <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>
```

5.4.6 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.

5.4.6.1 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.

5.4.6.2 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.

5.4.6.3 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.

5.4.6.4 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 5.3 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.

5.4.6.5 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.

5.4.6.6 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 4 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 5.3 for more details on policy sanitization.

5.4.6.7 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.

5.4.6.8 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.

5.4.6.9 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.

5.4.6.10 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”.

Chapter 6

Protocols and Message Flows

This section sketches the message flow between Data Subject and Data Controller, and between a Data Controller and a downstream Data Controller. These protocols are used in the proof of concept implementation section described in sec 7.This section contains for example the description for how SAML 2.0 can be extended for use as protocol language in credential-based access control.

6.1 Generic Policy

As soon as a user agent starts an HTTP session with a server, the latter is able to collect some information that may impact the user's privacy. Here is a non exhaustive list of the information that can be collected by a server upon receiving an HTTP GET request:

1- User's client IP address. This is either the IP address of the user agent requesting access to this site, or the IP address of a proxy server. 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 page 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 typically contains the key words typed by the user in the search engine.
6- What and when. The time of each request and the path for the resource being requested. Dynamically generated URIs may include information that can be used to track users.

7- HTTP Cookies. These provide a means for websites to track users over time by setting information that the user agent records and sends back to the server with future requests.

The very act of asking a server for its privacy policy leaks information about the client. For this reason, we propose to use what we call a generic policy (GP) to cover the information disclosed "implicitly" through HTTP request headers. Data Controllers should provide safe access to the generic policy without using or storing information gathered as a side effect of that access. The user agent can then check to see if the generic policy matches the user's preferences before proceeding to retrieve the specific policy for a given web page or resource.

Our implementation of this involves performing an HTTP HEAD request on the server's root path, and examining the HTTP response for an HTTP Link header with the appropriate link relationship type, for example:

```
Link: <http://example.com/w3c/policy.json>;
   rel="http://primelife.eu/generic-privacy-policy"
```

Where "http://primelife.eu/generic-privacy-policy" is the link relationship and "http://example.com/w3c/policy.json" is the URI for the server's generic policy. User agents would be expected to provide the minimum set of headers for such requests, e.g.

```
HEAD / HTTP/1.1
Connection: close
Host: example.com
```

This limits the implicitly disclosed information to the user agent's IP address and the time of the request. If the user wants to avoid the risk of a server logging this information, the user agent could be configured to make the request via an anonymising proxy. A further possibility would be for the user agent to request the server's generic policy from a third party, such as a search engine that has previous collected generic policies during the course of indexing websites. This would allow search engines to offer users a means to identify privacy friendly websites in the list of sites returned for a given search, i.e. those that provide generic privacy policies that match the user's preferences.

Rather than starting from scratch, we chose to base generic policies on the vocabulary developed for P3P (Platform for Privacy Preferences). P3P provides a rich set of terms for describing privacy policies, but this flexibility is also a significant drawback as it makes it impractical to provide a user interface for defining user preferences, and likewise, to automatically generate a human readable description of conflicts between the user's preferences and the server's policy.

As a result, we restricted generic policies to a flat list of properties for which the user preferences could be trivially realized as a set of grouped checkboxes. We further chose to use P3P's data categories rather than the taxonomy of data items as this was found to be a much better fit to the needs for describing the kinds of data collected from HTTP requests. Our approach is a significant enhancement in expressivity compared to Compact P3P policies which are limited to dealing with information collected in HTTP Cookies.

This left open the question of how to represent generic policies. One choice would have been to use the XML format defined by P3P, but this would have required the definition of a new XML schema to constrain the flexibility. Another choice would have been to define a restricted subset of the PPL policy language. That would have complicated the format without any benefits from a shared implementation of the matching engine and user interface code. In the end, a very simple format was chosen using JSON [JSO] (JavaScript Object Notation), a lightweight data-interchange

format that is easy to process via web page scripts, and as a result is gaining widespread adoption. Here is an example policy in JSON:

```
{
  "fullURI": null,
  "optURI": null,
  "name": "ACME widgets online inc.",
  "purposes": ["current", "admin", "tailoring", "individual-analysis"
],
  "recipients": [ "ours", "delivery", "same" ],
  "retention": "business-practices",
  "categories": [ "computer", "navigation", "interactive" ]
}
```

The terms used are borrowed directly from the P3P 1.1 specification, along with the longer descriptions of these terms. The generic policy mechanism was implemented as a Firefox extension and presented at the W3C Workshop on Privacy and data usage control, held 4-5 October 2010 in Cambridge MA, USA. Note that if this work were to be taken further, it would be appropriate to consider adding versioning information, either as part of the policy or via dereferencing the URI for the relationship type.

6.2 Data Subject and Data Controller Protocol

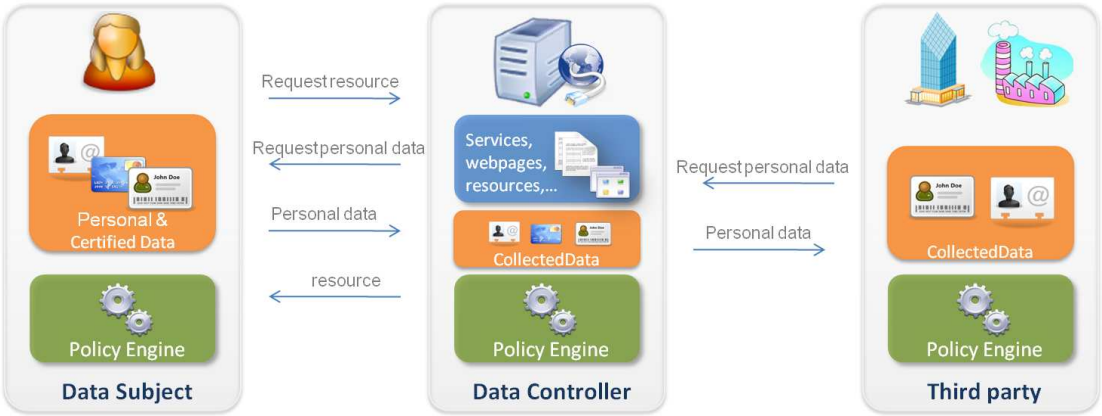

Figure 6 PrimeLife PPL collaboration scenario.

The collaboration scenario depicted in Figure 6 shows data subject (user) who wants to access the resource hosted by the data controller. This resource is protected by the access control policy that also contains PPL privacy-enhancing elements. So in the response to the initial user's request data controller sends back his requirements as one of the steps the user is asked to perform (provisional actions) in order to gain access to the resource. For example, data controller can ask the user to reveal his personal data. Additional information sent back from the data controller also contains statements on how this data will be treated once it is collected (specified in attached data handling policy).

Data subject will then try to match data handling policy and its own data handling preferences for the requested personal data types. If policies are successfully matched against each other, then data subject repeats the resource request, this time with additional personal data accompanied by the sticky policy, which is the result of matching.

Message exchange format is defined as an extension to the Security Assertion Mark-up Language (SAML). The root element of the data controller's request with provisional actions is SAML <Assertion> element. It contains the <Issuer> element that describes policy author and also an <AttributeStatement> which contains data controller identifier (specified as the attribute with the name http://www.primelife.eu/ppl/DataControllerID). Data handling policy is nested inside the XACML policy that itself is a part of a <Statement> element. For that the original SAML <Statement> element was extended and thus a new (inherited) type was created - *PPLPolicyStatementType*. Example of this message format can be found below:

```
< samla:Assertion xmlns =" urn:oasis:names:tc:SAML:2 .0 :assertion "
      xmlns:ppl =" http: // www. primelife .eu/ppl"
      xmlns:samla =" urn:oasis:names:tc:SAML:2 .0 :assertion ">
      < samla:Issuer >http: // store . example .com </ samla:Issuer >
      < samla:AttributeStatement >
            < samla:Attribute
                  Name =" http: // www. primelife .eu/ppl/
                  DataControllerID ">
            < samla:AttributeValue >http: // store . example .com
            </ samla:AttributeValue >
      </ samla:Attribute >
</ samla:AttributeStatement >
< samla:Statement xsi:type =" cl:PPLPolicyStatementType ">
      <ppl:Policy >
      < ppl:DataHandlingPolicy PolicyId ="# DHP1 ">
            ..
      </ ppl:DataHandlingPolicy >
      < ppl:DataHandlingPolicy PolicyId ="# DHP2 ">
            ..
      </ ppl:DataHandlingPolicy >
      < ppl:ProvisionalActions >
            < ppl:ProvisionalAction
              ActionId="http://www.primelife.eu/ppl/RevealUnderDHP ">
            < xacml:AttributeValue DataType =" xs:anyURI ">
              http: // www. example .org/ names # display_name
            </ xacml:AttributeValue >
            < xacml:AttributeValue DataType =" xs:anyURI ">
              # DHP1
            </ xacml:AttributeValue >
      </ ppl:ProvisionalAction >
      < ppl:ProvisionalAction
         ActionId ="http://www. primelife .eu/ ppl/ RevealUnderDHP ">
            < xacml:AttributeValue DataType =" xs:anyURI ">
              http: // www. example .org/ names # user_name
            </ xacml:AttributeValue >
            < xacml:AttributeValue DataType =" xs:anyURI ">
              # DHP1
            </ xacml:AttributeValue >
      </ ppl:ProvisionalAction >
      <ppl:ProvisionalAction
ActionId="http://www.primelife.eu/ppl/RevealUnderDHP ">
            < xacml:AttributeValue DataType =" xs:anyURI ">
              http: // www.w3.org /2006/ vcard /ns# email
            </ xacml:AttributeValue >
            < xacml:AttributeValue DataType =" xs:anyURI ">
              # DHP2
            </ xacml:AttributeValue >
```

```
      </ ppl:ProvisionalAction >
      </ ppl:ProvisionalActions >
</ ppl:Policy >
</ samla:Statement >
</ samla:Assertion >
```

The data subject's response is based on custom element that was introduced in the PPL language - <Claims> (see below). It is a list of nested <Claim> elements. Each claim contains many SAML <Assertion> elements. They either contain: (1) the revealed PII value (<AttributeType> element), (2) the access control decision (<ResponseType> element) or (3) the sticky policies (<StickyPolicyStatementType> element) that are linked to the PIIs by the unique identifiers.

```
<cl:Claims xmlns:cl =" http: // www . primelife .eu/ppl / claims "
  xmlns:sp =" http: // www. primelife .eu/ppl/ stickypolicy "
  xmlns:ppl =" http: // www. primelife .eu/ppl"
  xmlns:ob =" http: // www. primelife .eu/ppl/ obligation "
  xmlns:samla =" urn:oasis:names:tc:SAML:2 .0 :assertion ">
  <cl:Claim >
    < samla:Assertion >
      < samla:Issuer >http://www.primelife.eu/claims/self-issued21
      </ samla:Issuer >
      < samla:Statement xsi:type="cl:ResponseType"Decision="Access">
        < cl:MissingPII />
        <cl:DenyPII />
      </samla:Statement>
    </ samla:Assertion >
    < samla:Assertion >
      <samla:Issuer> http://www.primelife.eu/claims/self-issued
      </ samla:Issuer >
      < samla:AttributeStatement >
        < samla:Attribute xsi:type =" cl:AttributeType "
          StickyPolicyId =" SP1869111054 "
          Name =" http: // www .w3.org /2006/ vcard /ns# email ">
          <samla:AttributeValue> bob@example.com
          </samla:AttributeValue >
        </ samla:Attribute >
      </ samla:AttributeStatement >
    </ samla:Assertion >
    < samla:Assertion >
      <samla:Issuer> http://www.primelife.eu/claims/self- issued
      </ samla:Issuer >
      < samla:Statement xsi:type =" cl:StickyPolicyStatementType ">
        < sp:Attribute ID=" SP1869111054 " matching =" true "
          AttributeURI ="http://www.w3.org /2006/ vcard /ns# email ">
          < ppl:AuthorizationsSet matching =" true ">
            ..
          </ ppl:AuthorizationsSet >
          < ob:ObligationsSet matching =" true ">
            ..
          </ ob:ObligationsSet >
        </ sp:Attribute >
      </ samla:Statement >
    </ samla:Assertion >
  </ cl:Claim >
</ cl:Claims >
```

The message flow between data subject and data controller instances including SAML assertion and PPL claims documents is illustrated in Figure 7.

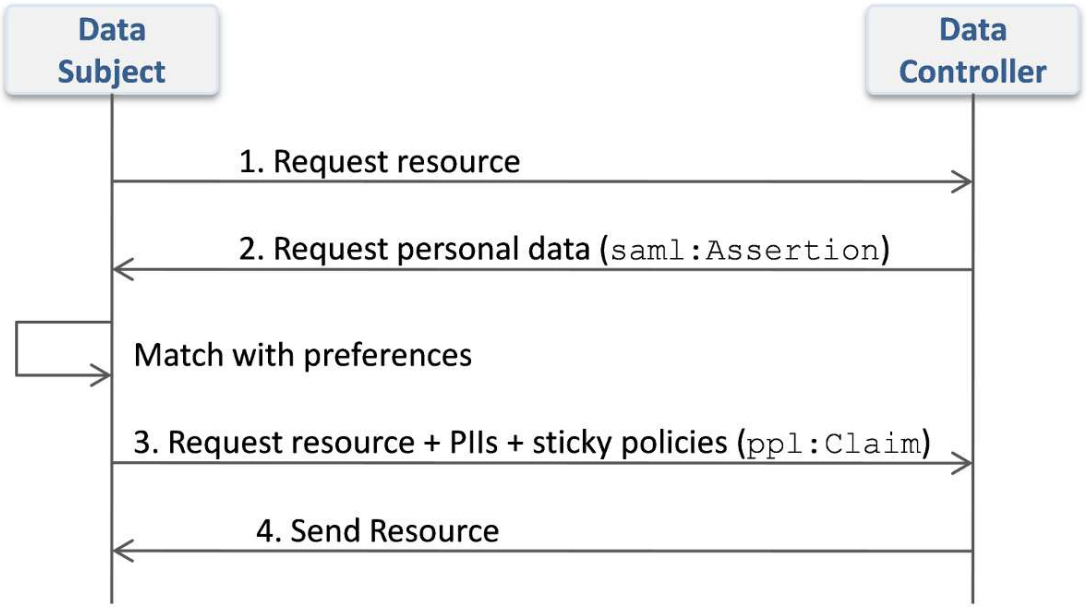

Figure 7 Message flow between data subject and data controller

Within the downstream usage scenario message formats stay the same. Here data controller acts in a similar way as the data subject in the previous message flow. When third party requests personal data stored on the data controller side (using SAML <Assertion> element), the latter tries to match the policy under which the third party wants to use the data with the persisted sticky policy. If the result of the matching is positive, the data is then shared with the third party under the derived sticky policy (sent inside <Claims> element).

6.3 Data Subject and Data Controller Protocol Including the UI dialog box

For usability purpose we proposed to use a graphical interface showing the content of the request and the privacy policy coming from the Data Controller. The Data Subject can then select which PII he can send. After this selection the UI will show the matching/mismatching result between user's preferences and Data Controllers privacy policy. The user can finally decide to agree on the result of the matching and send the data or disagree and discard the transaction. Figure 8 describes the entire interaction.
This UI is implemented as a browser plug-in communicating with the DS and DC through HTTP. More details of the specification of the plug-in can be found in [D4.3.2].

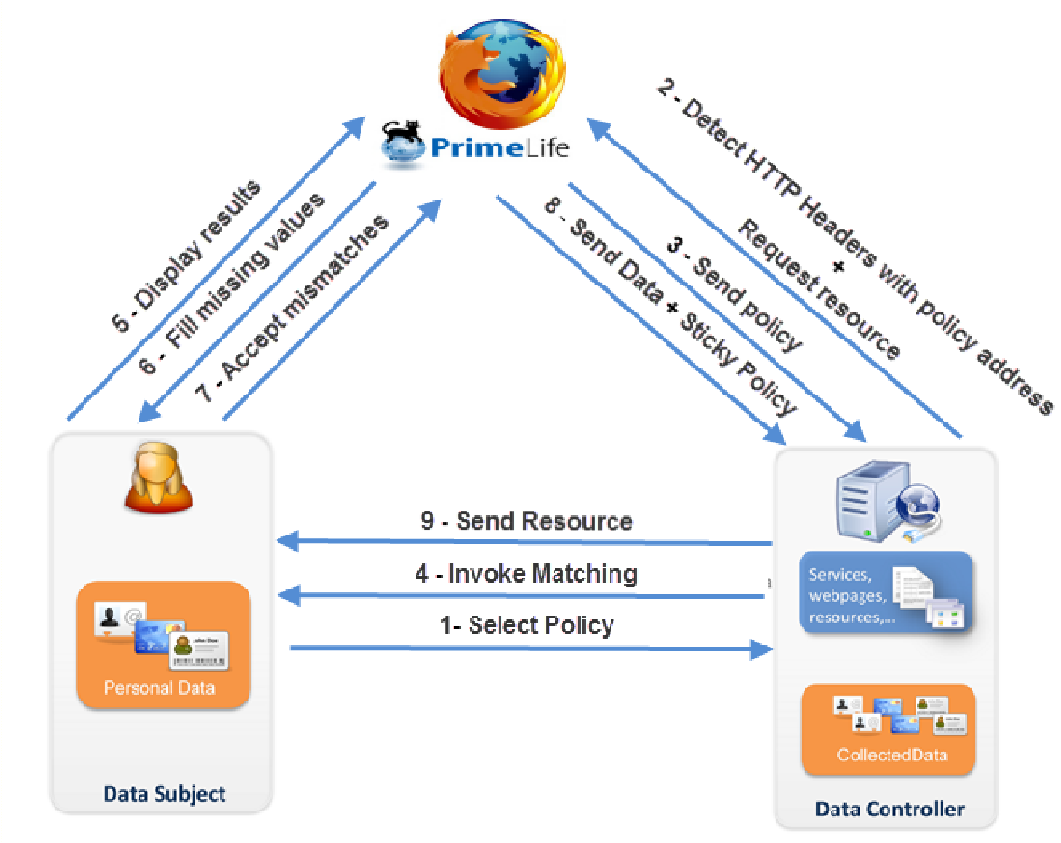

Figure 8 Interaction between Data Subject, data Controller and the Browser plug-in

For communication, HTTP headers are used to execute a special protocol. In the following table, the http://localhost:8082 stands for the DC Server, the http://localhost:9477 stands for the DS Server. The following table describes this HTTP protocol

Step	HTTP headers/parameters	Values
GET http://localhost:8082/clique/register (Invoked by User)		
1 Req	-	-
1 Resp	HTTP HEADERS: X-PrimeLife-PPL-ControllerInfo X-PrimeLife-PPL-ControllerPolicy X-PrimeLife-PPL-ResourcePolicy	 Example.com's store subscription (store.example.com, contact@example.com) http://localhost:8082/clique/policy http://localhost:8082/clique/policy
GET http://localhost:8082/clique/policy (Invoked by browser plugin)		
2 Req	-	-
2 Resp	-	Policy in content
POST http://localhost:9477/dataSubject/accessResource (Invoked by browser plugin)		
3 Req	Parameters: resource group	 [policy XML] “NameOfThePreferenceGroup”
3 Resp		Calim in content
Claims content is displayed by browser plugin and the User clicks on send and accepts mismatches.		
4 Req	Parameter: claim	 [claims of 3rd response]
4 Resp		[claims in content]

POST http://localhost:8082/clique/register (Invoked by browser plugin)		
5 Req	HTTP Header: X-PrimeLife-PPL-Proof Parameter: proof	 any value [claims]
5 Resp		[show result page]

Chapter 7

Implementation

The first version of the PPL Engine [D.5.3.2] provided the foundations that were used in the project continuation. The main elements already included in the first engine implementation were mechanisms for the XML policies marshalling and persistence - the two aspects that are filling the gap between the data repositories and the rest of the engine are described in section 7.1
The initial implementation covered only the data subject elements, such as policy matching and basic access control mechanism. We extended further the engine elements (PDP, PEP) by refactoring the existing solution to support fully the data subject/data controller scenario as well as the downstream usage. New processing model implemented in the PDP, based mainly on the strategy pattern, is introduced in section. 7.2.
These changes also allowed very smooth integration of the new functionalities into the engine, like the obligations handling (Sect. 7.3) or the preference group support (section 7.6).
Further, we have defined the PPL Engine interface that could be used from the external applications. We exposed the functionality of the engine by providing the API described in section 7.7.

7.1 Marshalling and Persistence

7.1.1 Marshalling Java objects with JAXB

To be able to analyze and process PPL language policies that are created in XML format we need to transform them into Java data types. One of the available solutions is to use the JAXB API.

Java Architecture for XML Binding (JAXB) [JAXB] provides an API and tools to map Java classes into theirs XML representations. JAXB provides two main features: the capability to marshal (transform) Java objects into XML data structure, and the vice versa: unmarshall the XML data structure into Java objects. To use this mapping, it's necessary to compile an XML schema into Java classes hierarchy. The latter are generated as Java beans and contain the specific JAXB annotations to allow the marshalling and the unmarshalling process, as seen in Figure 9

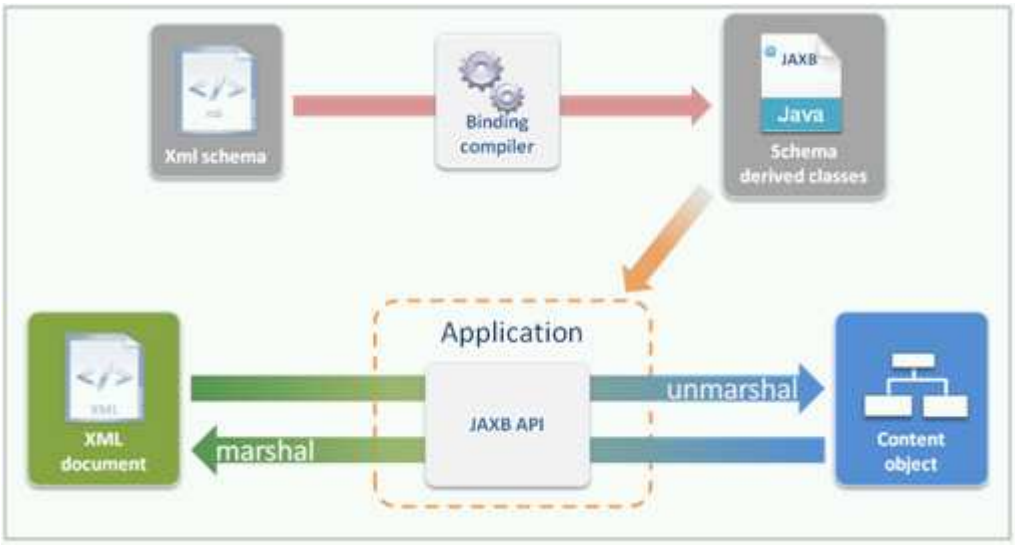

Figure 9 JAXB class generation and marshalling/unmarshalling process.

7.1.2 Persistence with JPA

On the other hand, the engine has to be able to persist the policies in a data base. The well established Java API for the object persistence is JPA.

JPA is a Java programming interface that bridge the gap between object oriented domain models (Java classes hierarchy) and relational database systems, technique known as ORM (Object Relational Mapping).

JPA is based on: (1) a set of interfaces and classes that separates the user of a persistence service (the application) and the provider of a persistence service (the data base engine), (2) a set of annotations to specify the mapping between Java classes and relational data base tables, (3) a persistence provider (like Hibernate [HIB] or Toplink [TOP]), or an implementation of the JPA specification, (4) an XML file describing the persistence configuration (provider, datasource, etc.).

7.1.3 Hyperjaxb3

JAXB provides us the way to map between the XML policies files and Java beans, and JPA provides us the way to map between Java classes and the relational data base. So we need to gather these two API to have an extendible way to go from one object representation to another. The problem is when we generate a class hierarchy with JAXB, the classes contain only JAXB annotations. To introduce the persistence capabilities into those classes we need to add JPA annotations as well (Figure 10).

Adding manually the JPA annotations into the JAXB generated Java classes is too difficult and complex, especially when handling a big number of classes (such as the ones generated from the PPL language schema).

The solution here is to use HyperJaxb3 [HYP]. Hyperjaxb3 uses the JAXB framework to generate the Java class hierarchy from the XML schema files and also adds the appropriate JPA annotations. The generation process provided by HyperJaxb is illustrated in Figure 11.

Figure 10 JPA/JAXB mappings.

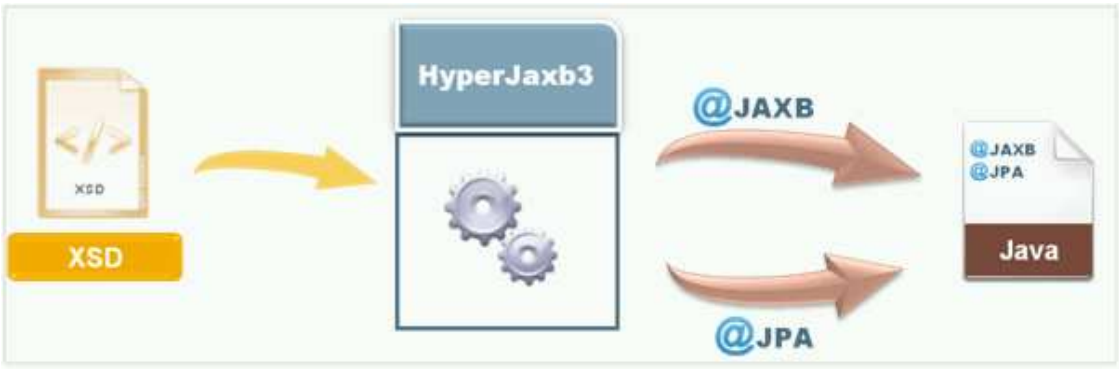

Figure 11 HyperJaxb3 class generation.

7.2 Policy Decision Point

The major part of the data controller request processing is implemented inside the PDP. Above all steps involved in the request processing the two seemed to be most important from the final decision perspective: access control and policy matching.

Because of the different possible requests and response requirements we defined the set of strategies that may alter the policy processing and the final result. The three use cases that were considered for the PII access are defined below.

Simple Scenario

In the simple scenario, when data subject is analyzing data controller request, we assume that preferences policies are shared among all users PIIs. When the PII requested by the data controller is missing, that information must be passed to the user interface. Also the PIIs that preference policy was mismatched against data controller's policy, are returned to the web browser extension to allow user accepting a mismatch.

Batch downstream Usage Scenario

In the downstream usage context all PIIs o certain type (PII's attribute name) are selected from the PII store. If there is no PII of that type, then empty set is returned to the third party. The preferences used for matching are this time retrieved from the content of <AuthzDownstreamUsage> element inside preferences policy.

Downstream Usage for Specific PII Scenario

If the third party requests access for the single PII using its PII store identifier, most of the steps are the same as in previous scenario. The only difference is that PII store is queried for the single object.

After analyzing the differences that are distinguishing those cases, we defined four strategy groups that are covering the steps in algorithm where behavior is changing.Each of the point is illustrated as a separate column in the Figure 12. The specific strategies are presented inside each column.

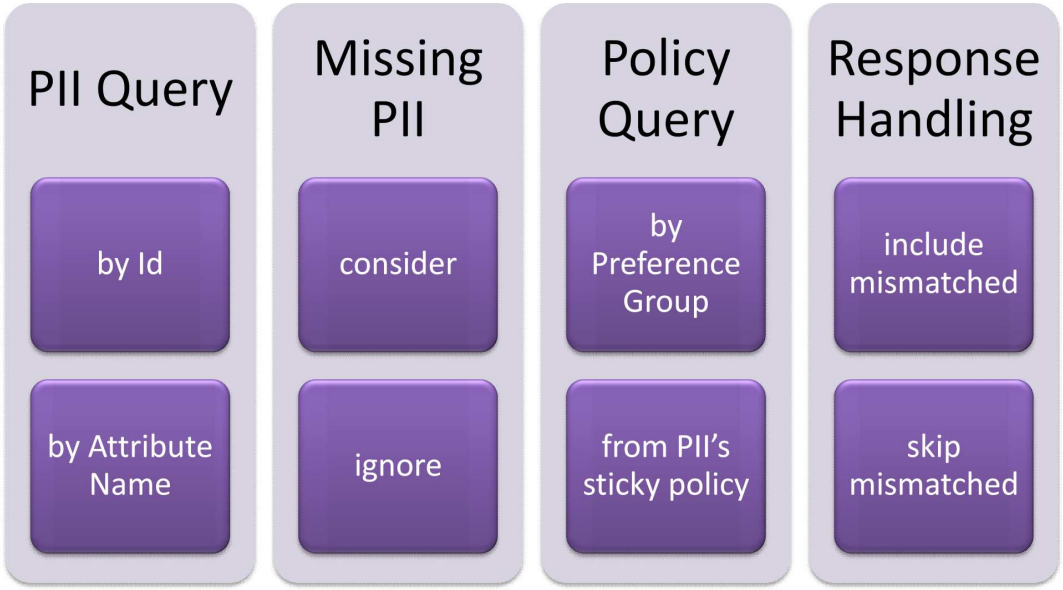

Figure 12 Processing strategies.

PII Query group represents the strategy that defines the criteria after which PIIs are looked up in the store. The case of forwarding or ignoring the information about missing PII is labelled as Missing PII. For the decision which of the preference policies should be chosen for matching we have defined the Policy Query strategy that either takes the policy from policy store according to the preference group or downstream usage policy stored in sticky policy associated directly with PII. The last group, Response Handling, refers to the interpretation of the outcome of a single action response. In downstream usage case PIIs with the other value of the access control response than "Permit" and the positive matching result are just skipped whilst in the simple data subject/data controller scenario they affect the overall decision, changing it either to "Indeterminate" or "Deny", depending on the policy mismatch or the denial of access coming from HERAS.

Strategies were transformed into the Java class hierarchy. The four groups are represented as the interfaces that are named *IPiiQueryStrategy*, *IMissingPiiStrategy*, *IPolicyQueryStrategy* and *IResponseHandlingStrategy* respectively. Diagrams in the Figure 13, Figure 14, Figure 15 and Figure 16 are presenting how the ideas behind this pattern were mapped to the Java classes.

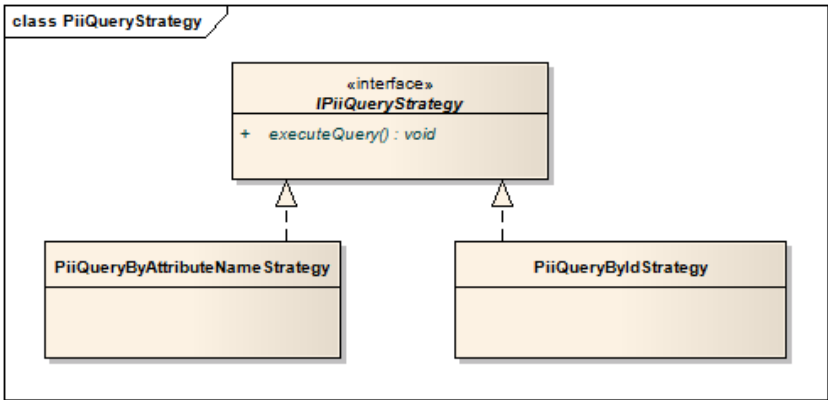

Figure 13 PII query strategies

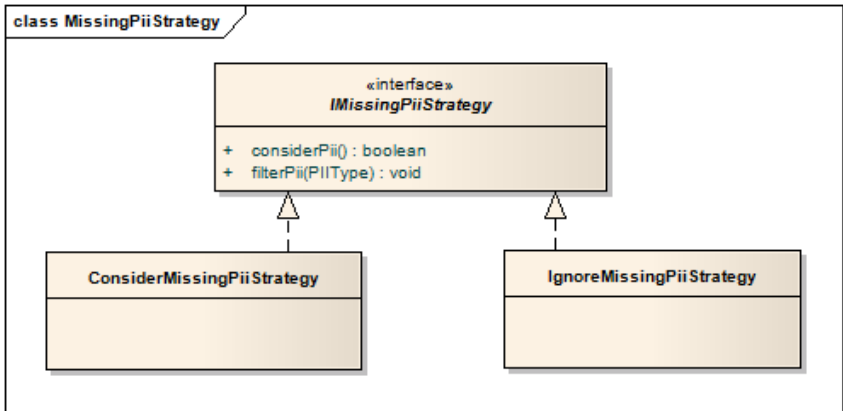

Figure 14 Missing PII strategies

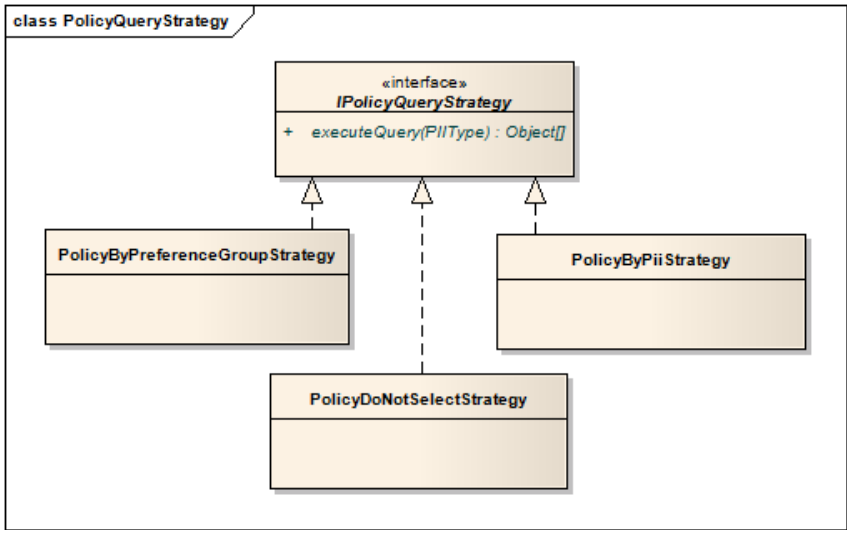

Figure 15 Policy query strategies

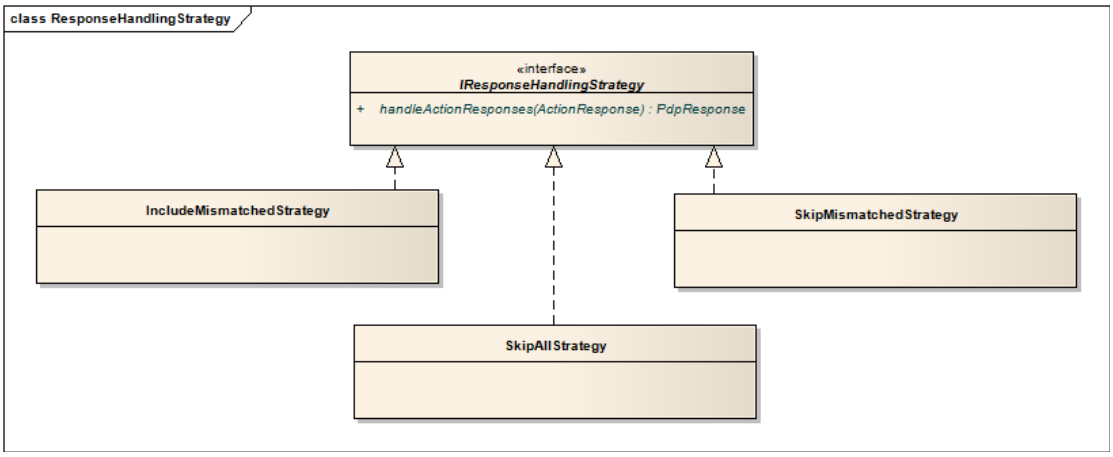

Figure 16 response handling strategies

Two additional strategies (PolicyDoNotSelectStrategy in the Figure 15 and SkipAllStrategy in the Figure 16) are also considered there. They are not part of the three scenarios defined here and are used for the update preference group functionality, described later in section 7.6.

7.2.1 PDP Request

The canonical form of the PDP request is represented by the *PDPRequest* class. It is constructed inside the PEP, transforming the incoming SAML Assertion to the request object. Assertion is passed to the request object constructor where it is marshaled to Java object representing XML element. Information that is retrieved from the SAML message are stored as the request object fields:

- properties describing requestor (in XACML terminology referred to as subject) are in the attributes field
- the map of the data handling policies that came in the request (dhpMap field)
- list of the Java class representation of the <ProvisionalAction> elements (provisional Actions field)

Each of the three scenarios presented before is represented by the concrete class that extends *PDPRequest*. The classes implements getter methods for the processing strategies, which were defined as abstract in the parent class. The class hierarchy is shown in the Figure 17.

DataSubjectPDPRequest class is used as a request in the simple data subject/data controller scenario. PIIs query strategy in this scenario is to select by PII attribute name (*PiiQueryByAttributeNameStrategy* class). List of the missing PIIs is considered in the response (*ConsiderMissingPiiStrategy*). Preference policy chosen for matching is determined by the user's current preference group (*PolicyByPreferenceGroupStrategy*). Finally, the mismatched PIIs are also forwarded within the response (*IncludeMismatchedStrategy*).

The batch downstream usage scenario is implemented using *DownstreamPDPRequestForNames* class. In this case the difference is that we do not consider the missing (*IgnoreMissingPiiStrategy*) or the mismatched (*SkipMismatchedStrategy*) PIIs in the response. Also instead of using preference policy in matching the sticky policy's downstream usage policy is used (*PolicyByPiiStrategy*).

The last scenario, where the single PII is requested is using same strategies as previous downstream usage scenario except instead of requesting whole batch of PIIs identified by their attribute name the PII identifier is used for quering PII store (*PiiQueryByIdStrategy*).

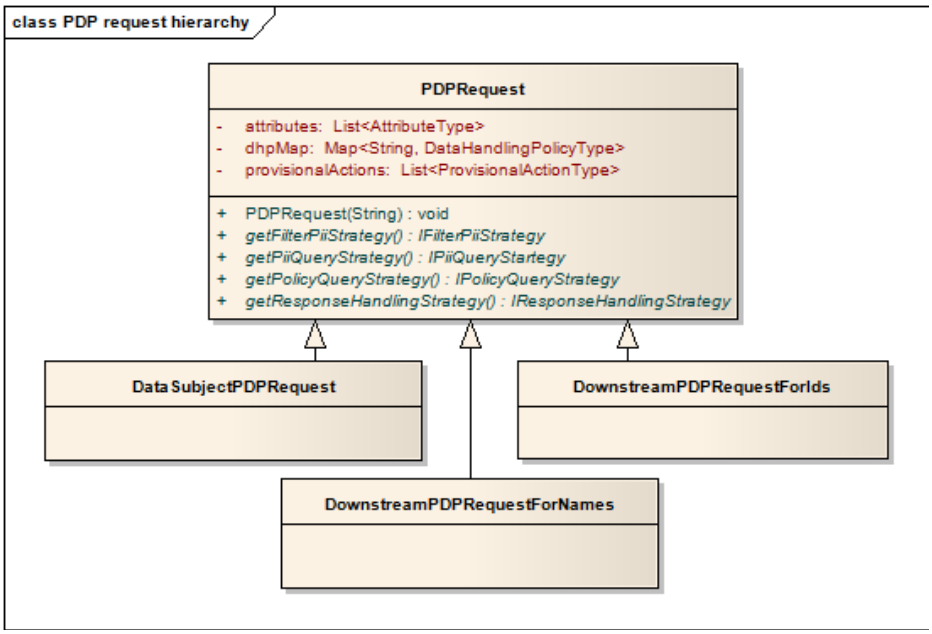

Figure 17 PDP request class hierarchy

7.2.2 Provisional Actions

For each <ProvisionalAction> element present in the data controller's resource policy PDP creates specific handler object representing one of the six possible actions (*Reveal*, *RevealUnderDhp*, *RevealTo*, *RevealToUnderDhp*, *Sign*, *Spend*) that will execute the processing phases. Creation of the action handling objects is the job of *ProvisionalActionsFactory*, which follows the factory design pattern. The class hierarchy representing provisional action handler elements is depicted in the Figure 18.

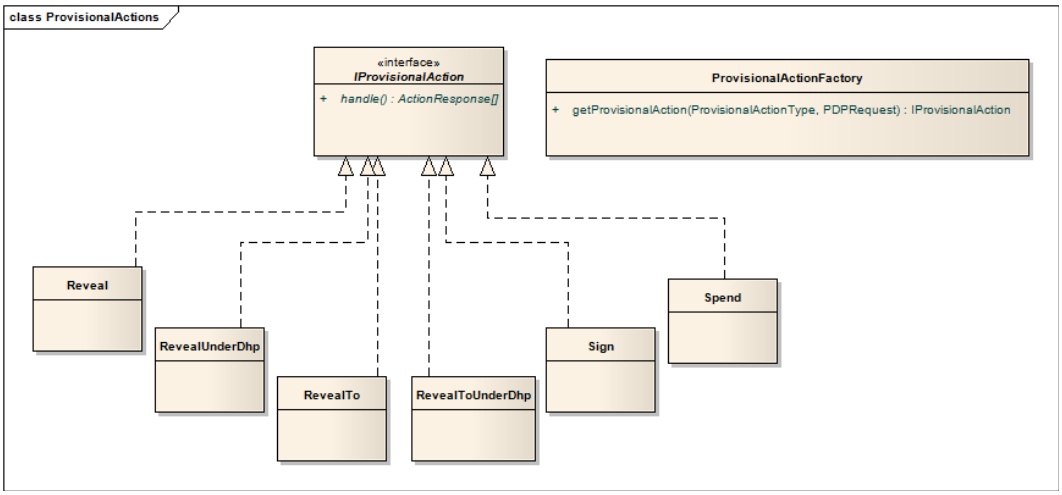

Figure 18 Provisional action factory class diagram.

All the burden associated with the configuration of the specific action handler is in this case left to the factory object. The creation of the provisional action handler could take several simple initializtion steps depending on the action in subject:

1. The provisional action object is populated with the data handling policies map that comes along with the request.

2. XACML request is created, taking the data controller attributes (like *DataControllerID*) that will be used for creation of the XACML subject element and PII type (PII Attribute Name) that will be used as the requested resource attribute (see Figure 19).

3. Last step involves passing the strategies (PII query, policy query, filter) that were set in the *PDPRequest* to the action object.

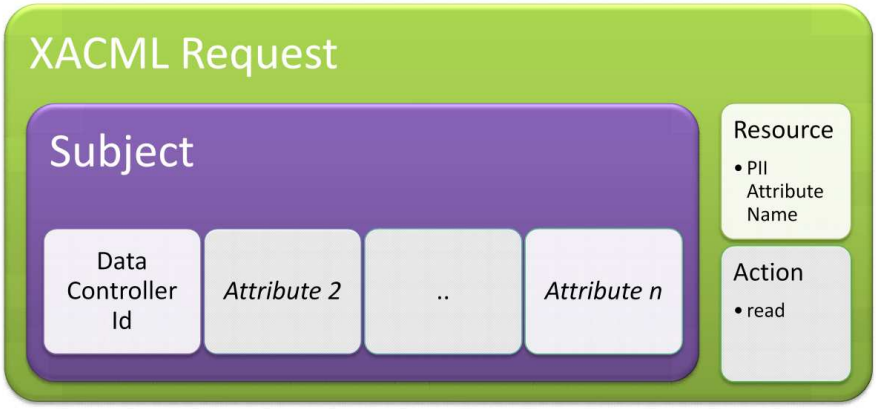

Figure 19 Access control request structure.

After all provisional action handlers are created, the processing step begins by invoking handle() method on each of them.

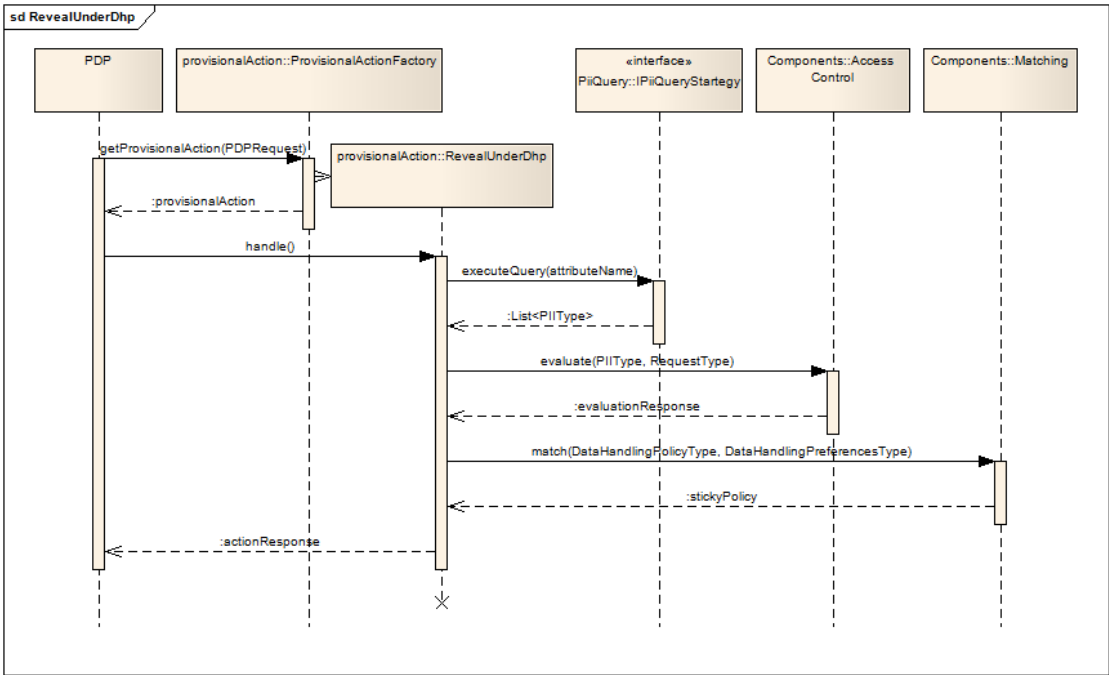

Figure 20 Provisional action sequence diagram.

The scenario considering provisional action *RevealUnderDhp* is illustrated in Figure 20. In the case of this provisional action, the overall PDP response relies on standard XACML access control decision and the policy matching result.

7.2.3 Access Control Engine

For the evaluation of the XACML-based part of the policy (which is ignoring all the new elements introduced by the PPL language specification) we are using HERAS [HER] AF implementation.

The order of the method calls is illustrated in Figure 21. In the first phase of the access control process implemented in the *AccessControlUtils* class we obtain PDP provided by the HERAS library using the *SimplePDPFactory* class. HERAS *SimplePDP* exposes a getter method for the policy repository. This repository is used by the engine to evaluate the incoming request. Preference policies that will be used in the process are deployed there. But before it is possible the policies are converted to the HERAS XACML class stack because those are the types expected by the method in by the PolicyRepository class. In brief the convertion process filters out all the PPL-related elements from the policy and marshals it again to the HERAS format.

Now that the policies are deployed successfully to the HERAS repository, the XACML access control request visualized in the Figure 19 is evaluated in the PDP and in the result to that method call we receive the decision whether access to the PII is permitted or denied.

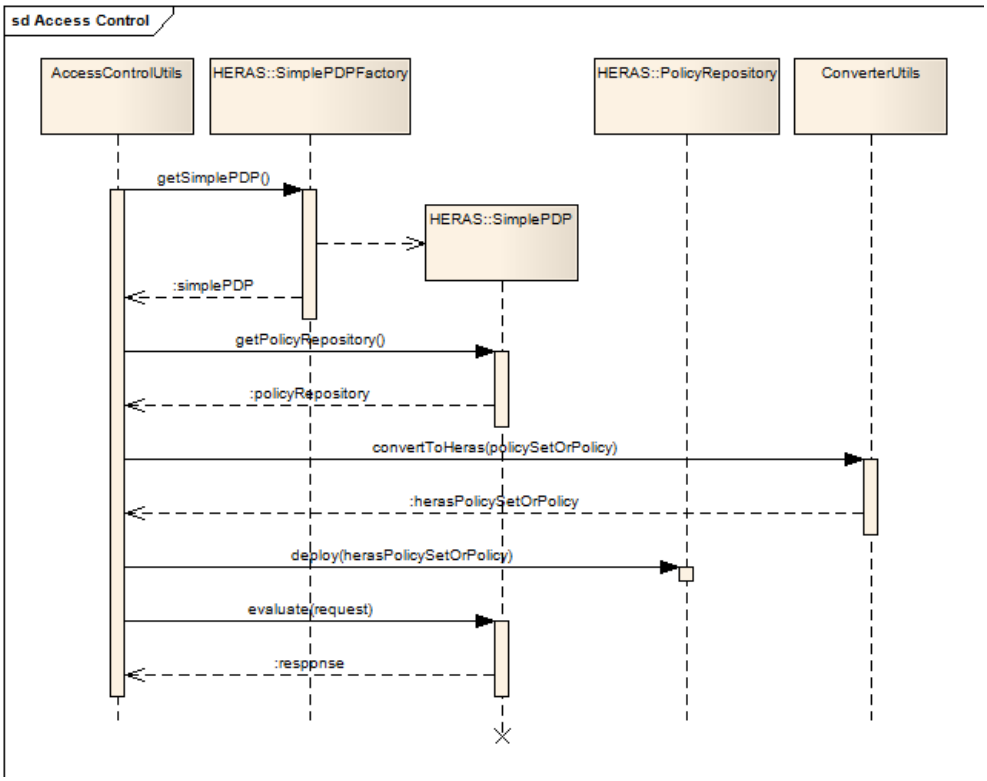

Figure 21 PII access control sequence diagram.

Later in this scenario we also use HERAS to determine which <DataHandlingPreferences> element from the user's preferences policy should we use for the matching. To obtain the element, there is an algorithm presented as pseudo-code in Algorithm 1 for finding the proper rule element containing data handling preferences from the policy and policy sets hierarchy.

The *findApplicableRule* method iterates through the policy structure in depth-first search manner. Each policy set, policy and rule element's target is tested (matched in HERAS vocabulary, not to be mistaken with policy matching) against the request target, to check if it is applicable for that rule (policy set or policy respectively). That means that request's target and elements' target's properties (*subject*, *resource* and *action*) are compared with each other to obtain the information if such rule (*policy* or *policyset*) is referring to the current request or not. If current element's target is applicable then algorithm invokes itself recursively on the children elements.

The final data handling preferences element (the one returned with the access control response) is always taken from the last applicable rule found by the algorithm.

Algorithm 1 *findApplicableRule* method that is returning the last applicable rule

```
rule ← null
for policySetOrPolicy in policyList do
  if policySetOrPolicy instanceof PolicySetType then
    nestedPolicyList ← getNestedPolicies(policySetOrPolicy)
    ruleTemp ← findApplicableRule(nestedPolicyList, request) {recursive invocation}
    if ruleTemp ≠ null then
      rule ← ruleTemp
    end if
  else
    ruleList ← getRules(policySetOrPolicy)
    for r in ruleList do
      if matchTarget(request, r.getTarget()) then
        rule ← r
      end if
    end for
  end if
end for
return rule
```

7.2.4 Matching Engine

After obtaining the positive access control result the policies part related to the data handling (<DataHandlingPreferences> and <DataHandlingPolicy> elements) are a subject to the policy matching process. Each data handling policy consists of two separate parts - authorizations and obligations. The matching process of these two elements is performed independently.

Matching is in charge of deciding whether the policy is less premissive than the user's preferences. Generally, when policy requires more authorizations than the user has allowed in his preferences or the policy obligations do not cover all the obligations specified by user than the mismatch occurs and information about which elements are not conformant is supplied to the sticky policy.

Authorizations matching is implemented in the AuthorizationsMatcher class and provides two methods: match, that produces the authorizations set part of the sticky policy and getAuthMismatch, that returns mismatches list, if any, that is also later included in the sticky policy.

Obligations matching implementation is encapsulated in form of the Obligations Matching Engine (OME) web service. The class that invokes service's client’s method is the ObligationsMatcher. The single method is provided here, getStickyPolicy. Possible mismatches are in this case a part of the sticky policy obligations set element.

The matching sequence is depicted in Figure 22.

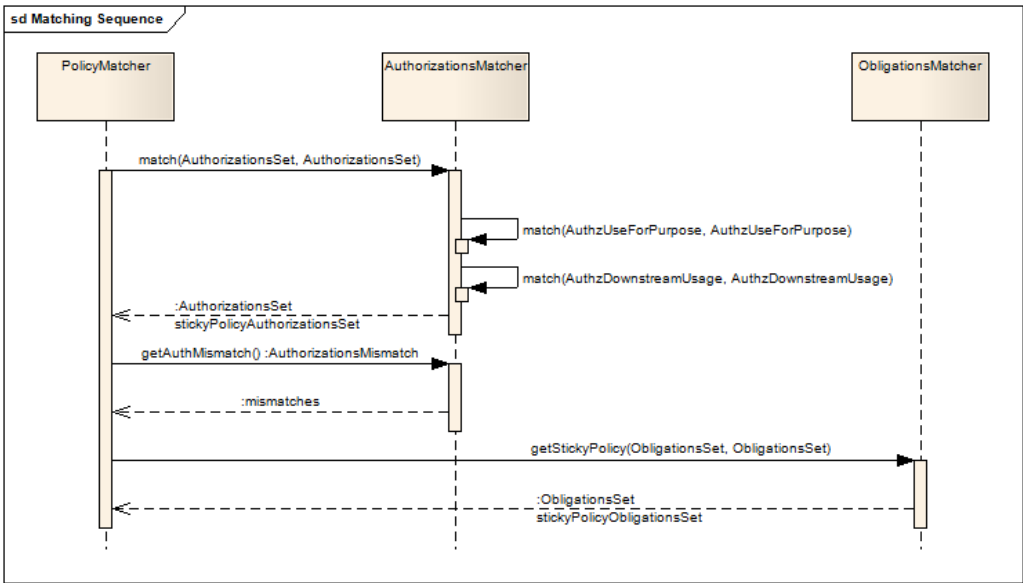

Figure 22 Policy matching sequence diagram

7.3 Obligation Enforcement Engine (OEE)

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.

7.3.1 Overview

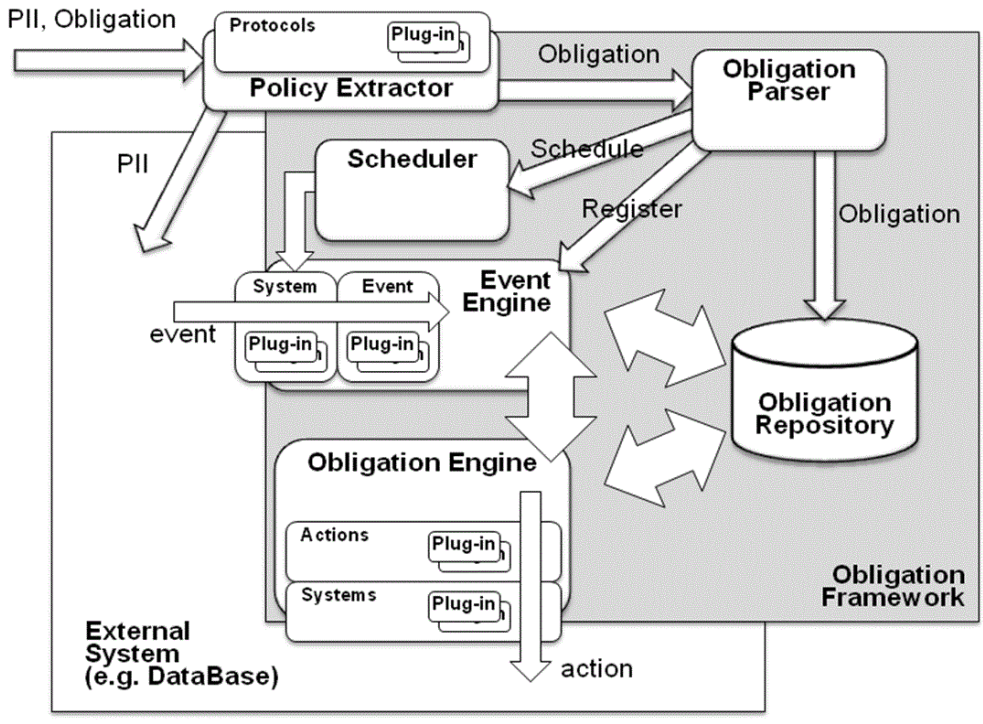

Figure 23:. Architecture diagram

Figure 23 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. In order to be able to fire the scheduled event right in time, it will schedule the event a configurable delta time t the actual event time. A value for the delta time may be 10 seconds.

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.

This section gives an overview of the obligation enforcement engine. There are three components:

1. OEE Service: the engine

2. OEE Client: a sample client
3. Action Handler: dummy front end for legacy system to execute “actions”, e.g. delete data. This will be replaced by SAP’s database frontend.

The client loads a sticky policy (see StickyPolicy.xml) associated with a reference to a Pii. The client sends events related to the PII.

7.3.2 Implementation of OEE

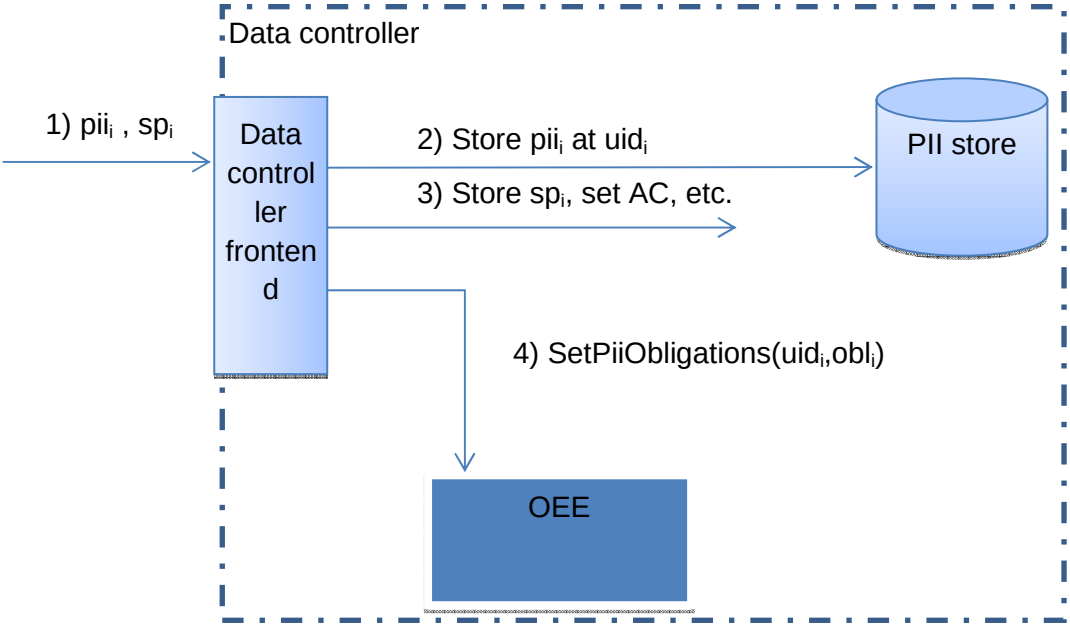

Figure 24 - Obligation Engine: Loading Obligation

Figure 24 presents the first API of the Obligation Enforcement Engine (OEE) that is used to upload new obligations.

- Pii$_i$: one piece of personal data
  - Uid$_i$: the local (i.e. at data controller side) reference to Pii$_i$.
- Sp$_i$: the sticky policy associated with Pii$_i$.
  - Obl$_i$: the obligation part of Sp$_i$, i.e. an ObligationsSet

Please refer to section 7.3.2.1 for a full listing API function calls.

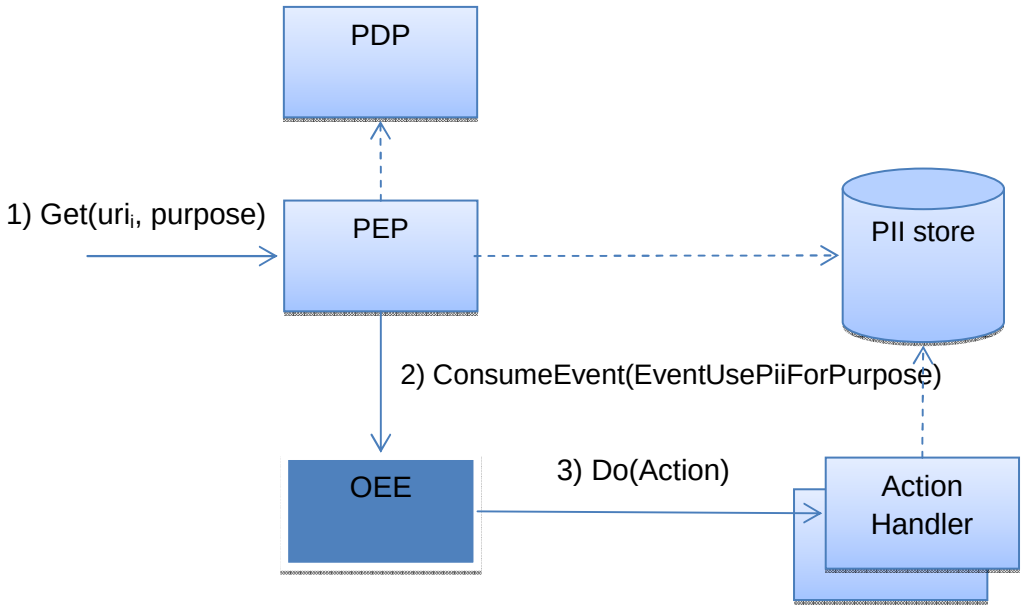

Figure 25 - Obligation Engine: External Event

Figure 25 presents the second API of the Obligation Enforcement Engine (OEE) that is used to receive relevant events. Please refer to section 7.3.2.1 for a full listing obligation API function calls.

If the event is relevant, i.e. there is an obligation associated with the PII that is triggered by this event, the corresponding action is executed. Actions are “executed” by calling appropriated action handler with a set of parameters.

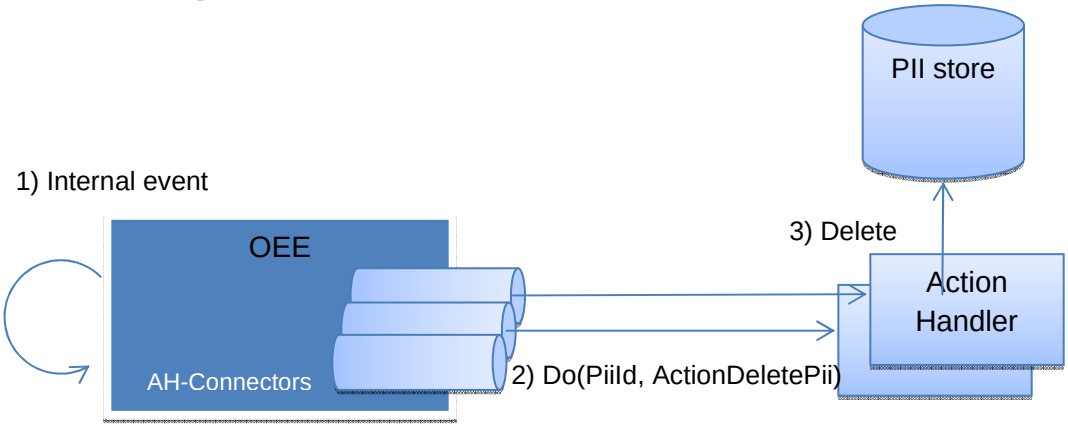

Figure 26 – Obligation Engine: Internal Event

7.3.2.1 Obligation Management Web Service APIs

This section will give an overview to the web service API for Registering and Removing Obligations. This API offers three methods:

- `void SetPiiObligations(PiiUniqueId piiId, ObligationsSet stickyObligationsSet);`

This method sets internal scheduler and “register” to relevant events in order to enforce obligations specified in a sticky policy.
The first parameter is the unique id of the PII in this system. For instance when the PII is stored on a database, the unique id could be server + database + table + cell.
The second parameter is an obligation set, i.e. the obligation part of the sticky policy.

- `void SetPiiObligations(PiiUniqueId piiId, string stickyObligationsSet);`

  Same as above, but second parameter is a XML-serialized string of the former ObligationsSet type. This method is deprecated and should not be used anymore.

- `void RemovePiiObligations(PiiUniqueId piiId);`

  This method remove obligations related to some PII. The parameter is the unique id of this PII.

7.3.2.2 API for Triggering Event

This API offers one method:

- `void ConsumeEvent(Event piiEvent);`

  This method takes an object of a derived class of Event, which will OEE lead to enforcement according actions on registered obligations. For the moment, we support two types of events: EventUsePiiForPurpose and EventDeletePii.

7.3.2.3 Working with OEE Service

Starting and Stopping the Service: As OEE is a Windows service, it is registered as a service during installation process and then listed in the services section at the Windows management console.

Log and Debug Outputs: some basic information about the start and about received events as well as warnings and errors are sent to the Windows application event log. Moreover, there is also a debug log which is far more informative. It is a text file which is located on %LocalAppData%\EMIC\PrimelifeOEE of the LocalService User.

Persistence of OEE’s State: The main task of OEE is to store event-triggered actions up to a long time (several day, months or even years) and wait for an event to trigger the according action. Considering that a computer may have to restart or have to be exchanged in hardware during that period, there’s a need to store the data on non-volatile memory like hard disk.

OEE will store its events, actions, and even its state about what time-dependent event should have been triggered but have not been triggered yet. The OEE has been implemented as a proof of concept and does not provide strong persistence mechanisms.

Configuration File: The configuration file of OEE can be found in the installation directory of OEE. It is named “OEEWindowsService.exe.config”. There are several configurable parameters and flags that influence the behavior of OEE. These can be found in the “applicationSetting” element of the configuration file and are described next:

- Parameter ServiceAddress: This parameter sets the URL, where OEE should provide its web service. It is mandatory as there is no hardcoded default value given for it.
- Parameter DebugMode: The DebugMode-flag will set the OEE in debug mode. In debug mode, OEE will behave less aggressive on incorrect user inputs, print out a warning

telling about the error and try to recover from that irregular state automatically by guessing closest the right input value.

- Parameter SpareDeltaTimeForTimedEvents: On Obligations, that require an action to be executed within a certain time, OEE will schedule the action right at the end of the given time period. But as there could be a workload on the OEE running machine right on that time, where an action needs to be executed, the action is scheduled t seconds before the end of the time period.

New trigger/action handlers can be defined and hooked to the OEE with a plug-in mechanism.

7.4 Action Handler

Another web service that completes the obligations handling process is the Action Handler that resides on the data controller (it is not part of the mentioned before Obligation Handler, rather than that it should be consider as integral part of the data controller implementation). Its main task is to fulfill actions coming from OEE that were results either of PII events or time-based actions triggered by a scheduler.

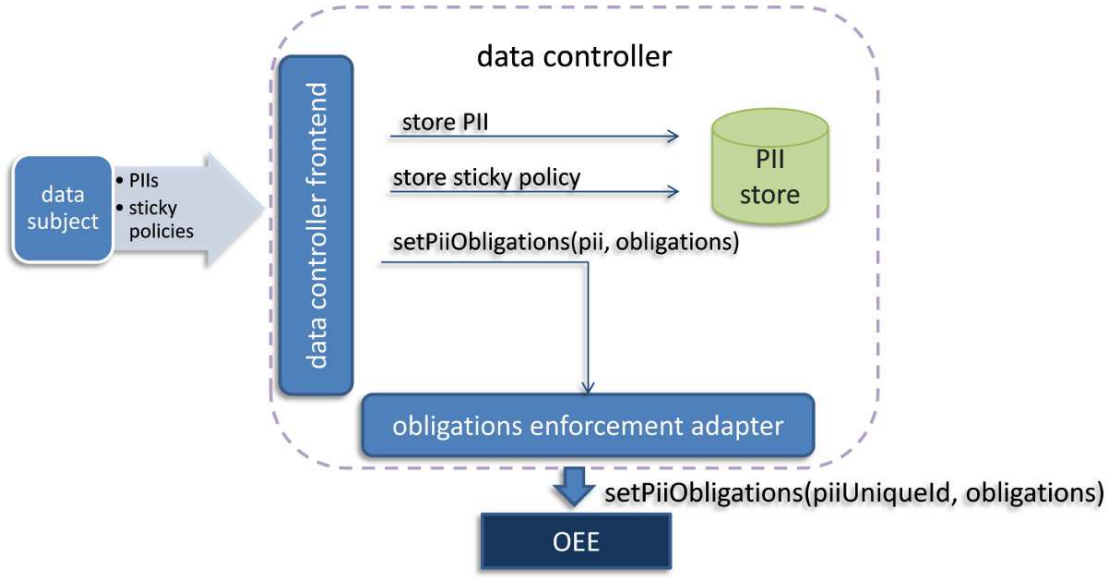

Figure 27: Setting Obligation

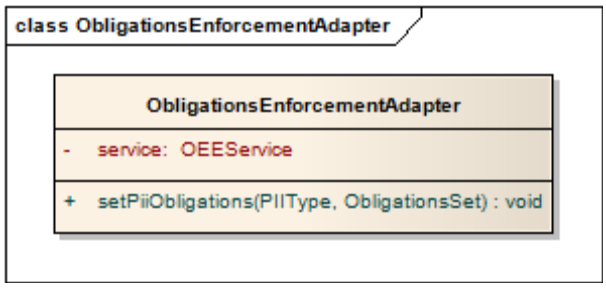

Figure 28: ObligationsEnforcementAdapter class

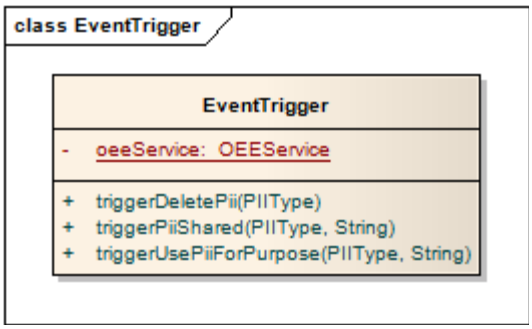

Figure 29: EventTrigger class

Action Handler is implemented in form of the web service that is integrated into the data controller interface. That allows manipulation of the PII store required by some of the considered actions (e.g., log "PII used for marketing purpose"). The interface is meant to be used by the Obligation Handler (OEE service) to execute the actions triggered by the events associated with the obligation (Figure 30)

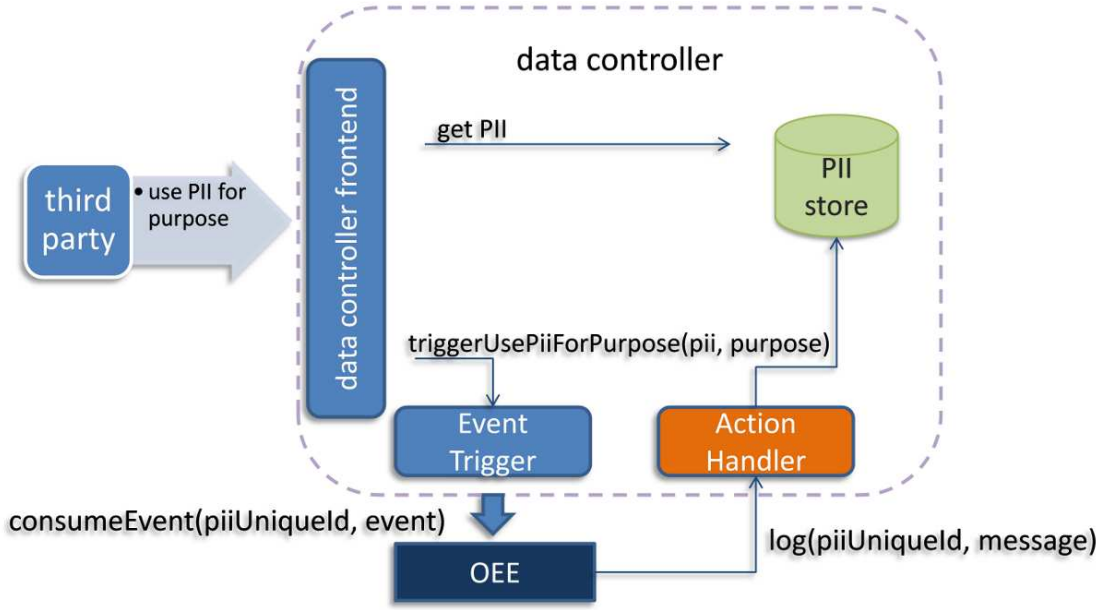

Figure 30: Executing the event

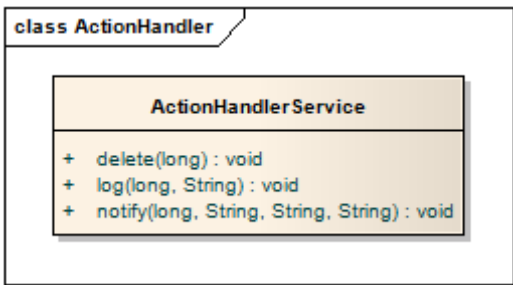

Figure 31: Action handler service

Current implementation provides three methods (Figure 31):

- `delete(long piiUniqueId)` - that facilitates Action Delete Personal Data
- `log(long piiUniqueId, String message)` - that facilitates Action Log
- `notify(long piiUniqueId, String media, String address, String message)` - that facilitates Action Notify Data Subject

7.5 Policy enforcement Point

Policy Enforcement Point (PEP) is defined in our architecture as the interface between data subject and data controller policy engine (or data controller and third party). The interface is different in the data subject and the data controller so this component is kept separate for both entities.

The main PEP task is to transform the policy request (that could be either in string representation or already marshalled object) to the common *PDPRequest* object that is used to invoke the PDP. On the data subject it contains a single method process, which is processing data controller request (provisional actions) and returns the claims document with the access control decision, revealed PII values and theirs sticky policies (Figure 32). The *DataSubjectPDPRequest* class is used here as the PDP request.

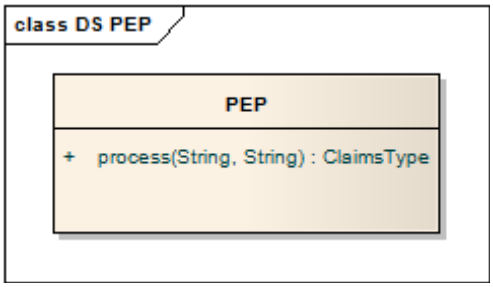

Figure 32: Data Subject PEP interface

PEP interface on the data controller side is more complex as it supports data subject and other (third party) data controller requests Figure 33. Here the three methods are provided:

1. `List<PIIType> processPolicy(String resourceName, ClaimsType claims, boolean setObligations)`

2. `ClaimsType processDownstreamUsage(String policyQueryResponse, String defaultServerPolicy)`

3. `ClaimsType processRequestForSpecificPii(String policyQueryResponse, String defaultServerPolicy)`

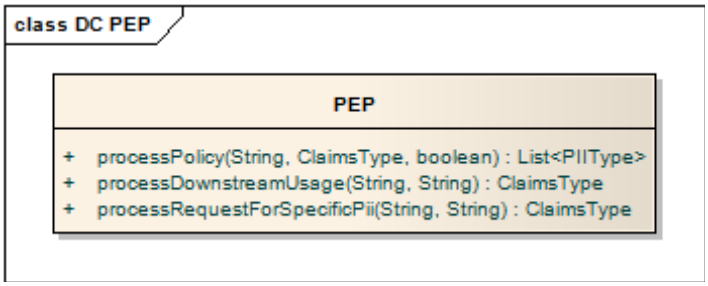

Figure 33: Data Controller PEP interface

processPolicy method is invoked as a consequence of the data subject request for the data controller resources. It analyses the claims document returned by the data subject.
The following steps are taken here:
1. List of the sticky policies is retrieved from the claims.
2. List of the PIIs is retrieved from the claims.
3. Each PII is associated with the correct sticky policy (according to the sticky policy's identifier).
4. Obligation Enforcement Engine is called for each obligations set contained in the sticky policies.

The two other methods in the PEP interface are used in the downstream usage scenario. processDownstreamUsage is a request for the list of PIIs stored in the data controller's PII Store. The PDP request created here is the *DownstreamPDPRequestForNames* object. *processRequestForSpecificPii* is a request for the single specific PII and it uses *DownstreamPDPRequestForIds* PDP request object.

7.6 Preference Groups

To give the users possibility to easily change their privacy preferences we introduced the preference groups. They allow users to maintain multiple browsing profiles with a different privacy settings (e.g., the "secure" profile which is not allowing any PII to be revealed or "trusted" profile which contains a list of trusted websites that are allowed to collect user's data).

The preference groups support means that user can now store multiple preference policies, which identifiers are recognized as the group names. User will be able in the future to select his current browsing profile in the Firefox extension. The information about under which profile user is browsing will be passed to the engine as the parameter of the data subject's PEP only method.

Additional functionality associated with the preference groups is a possibility to update the current preferences if there was a policy mismatch (either by creating a new group or updating the current one) so the next when the user is requesting the data controller's resource there will be no mismatch.

7.7 API

To expose the engine functionality to allow other developers reuse our engine, we decided to create the API for the management of data subject and data controller instances. The interface is prepared in form of the RESTful HTTP methods.

7.7.1 Data Subject API

For managing PIIs and PreferenceGroups on the data subject we provide the following API:

Managing PII

- Create a PII

```
PUT /api/pii?name=attributeName&value=attributeValue HTTP/1.1
```

Parameters:

- name - attribute name of the PII (the type).
- value - attribute value of the PII.

- Update a PII

```
POST /api/pii?name=attributeName&value=attributeValue HTTP/1.1
```

Updates the value of the PII with that attribute name.

Parameters:

- name - attribute name of the PII (the type).
- value - new attribute value of the PII.

- Delete a PII

```
DELETE /api/pii?name=attributeName HTTP/1.1
```

Parameters:

- name - attribute name of the PII (the type).

- Get all PIIs

```
GET /api/pii HTTP/1.1
```

Returns list of all PIIs stored in the database.

Managing Preference Groups

- Create a preference group

```
PUT /api/groups HTTP/1.1
```

Creates one or more preference groups.

Request content:

String representation of the resource policy in the XML format. The root element of the policy document must be http://www.primelife.eu/ppl:Policy or `http://www.primelife.eu/ppl:PolicySet`.

- Update a preference group

```
POST /api/groups/group id[?new pref group=new pref group id] HTTP/1.1
```

Updates the preference group by merging the mismatches from the sticky policy, so the next time when the matching is performed (between the policy from specified preference group and the same the data controller's resource policy) there will be no mismatch.

Parameters:

- group id - the policySetId or policyId of the preference group.
- new pref group - (optional) identifier of the new preference group if the result shall be stored as a new preference group.
- policy { the policy of the resource the user is trying to gain access to. Note: this is the same input as given to the *accessResource*() method.

- Delete a preference group

```
DELETE /api/groups/pref group id HTTP/1.1
```

Deletes a preference group.

Parameters:

- pref group id - the identifier of a preference group.

- Get all preference groups identifiers

`GET /api/groups HTTP/1.1`
Gets a list of all preference group names stored in the database.
Response:
A JSON list:
[PrefGroupName1, PrefGroupName2]

- Get a preference group policy

`GET /api/groups/pref group id HTTP/1.1`
Gets the XML of a specific preference group.
Parameters:
  - pref_group_id - the identifier of a preference group,

7.7.2 Data Controller API

Direct management of data controller's resource and PII store is now possible by calling RESTful interface methods.

- Uploading resource data and policy

  `PUT /api/resource/name HTTP/1.1`

  Parameters:

  - Name – resourse name

  Request content:

  String representation of the resource policy in the XML format. The root element of the policy document must be `urn:oasis:names:tc:SAML:2.0:assertion:Assertion`.

  Response:

  Resource id

- Uploading PII

  `PUT /api/pii HTTP/1.1`

  Parameters:

  - Name - PII's attribute name (e.g. `http://www.w3.org/2006/vcard/ns#email`).
  - Value - PII's attribute value (e.g. `john.doe@example.com`).

  Request Content:

  String representation of the sticky policy in the XML format. The root element of the sticky policy document must be `http://www.primelife.eu/ppl/stickypolicy:Attribute`.

  Response:

  PIIid

- PII downstream usage request for a single PII

  `POST /api/pii HTTP/1.1`

  Request content:

String representation of the request policy in the XML format. The root element of the policy document must be `urn:oasis:names:tc:SAML:2.0:assertion:Assertion`.

Response

String representation of the claims in the XML format. The root element of the document is `http://www.primelife.eu/ppl/claims:Claims`.

Chapter 8

Appendix

8.1 Privacy Policy schema

Implementers 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.

8.1.1 PrimeLife root schema

The root schema includes the reference to all the principal elements of the language like for example the XACML schema, the Obligation schema, Credential schema…

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="http://www.primelife.eu/ppl" xmlns:ppl="http://www.primelife.eu/ppl"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xacml="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
xmlns:ob="http://www.primelife.eu/ppl/obligation"
xmlns:cr="http://www.primelife.eu/ppl/credential"
targetNamespace="http://www.primelife.eu/ppl" elementFormDefault="qualified"
attributeFormDefault="unqualified">
        <xs:import namespace="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
schemaLocation="access_control-xacml-2.0-policy-schema-os.xsd"/>
        <xs:import namespace="http://www.primelife.eu/ppl/credential"
schemaLocation="PrimeLifeCredential.xsd"/>
        <xs:import namespace="http://www.primelife.eu/ppl/obligation"
schemaLocation="PrimeLifeObligation.xsd"/>
        <!-- PolicySet -->
        <xs:element name="PolicySet" type="ppl:PolicySetType"
substitutionGroup="xacml:PolicySet"/>
        <xs:complexType name="PolicySetType">
                <xs:complexContent>
                        <xs:extension base="xacml:PolicySetType">
                                <xs:sequence>
                                        <xs:element ref="ppl:DataHandlingPolicy" minOccurs="0"
maxOccurs="unbounded"/>
                                        <xs:element ref="ppl:DataHandlingPreferences"
minOccurs="0"/>
                                        <xs:element ref="ppl:StickyPolicy" minOccurs="0"/>
                                        <xs:element ref="cr:CredentialRequirements"
minOccurs="0"/>
                                        <xs:element ref="ppl:ProvisionalActions"
minOccurs="0"/>
```

```
                                </xs:sequence>
                        </xs:extension>
                </xs:complexContent>
        </xs:complexType>
        <!-- -->
        <!-- Policy-->
        <xs:element name="Policy" type="ppl:PolicyType" substitutionGroup="xacml:Policy"/>
        <xs:complexType name="PolicyType">
                <xs:complexContent>
                        <xs:extension base="xacml:PolicyType">
                                <xs:sequence>
                                        <!-- <xs:element ref="ppl:Rule"
maxOccurs="unbounded"/> -->
                                        <xs:element ref="ppl:DataHandlingPolicy" minOccurs="0"
maxOccurs="unbounded"/>
                                        <xs:element ref="ppl:DataHandlingPreferences"
minOccurs="0"/>
                                        <xs:element ref="ppl:StickyPolicy" minOccurs="0"/>
                                        <xs:element ref="cr:CredentialRequirements"
minOccurs="0"/>
                                        <xs:element ref="ppl:ProvisionalActions"
minOccurs="0"/>
                                </xs:sequence>
                        </xs:extension>
                </xs:complexContent>
        </xs:complexType>
        <!-- -->
        <!-- Rule -->
        <xs:element name="Rule" type="ppl:RuleType" substitutionGroup="xacml:Rule"/>
        <xs:complexType name="RuleType">
                <xs:complexContent>
                        <xs:extension base="xacml:RuleType">
                                <xs:sequence>
                                        <xs:element ref="ppl:DataHandlingPolicy" minOccurs="0"
maxOccurs="unbounded"/>
                                        <xs:element ref="ppl:DataHandlingPreferences"
minOccurs="0"/>
                                        <xs:element ref="ppl:StickyPolicy" minOccurs="0"/>
                                        <xs:element ref="cr:CredentialRequirements"
minOccurs="0"/>
                                        <xs:element ref="ppl:ProvisionalActions"
minOccurs="0"/>
                                </xs:sequence>
                        </xs:extension>
                </xs:complexContent>
        </xs:complexType>
        <!-- -->
        <!-- Common Type of DHP, DHPreferences and StickyPolicy DHPSP; Data Handling
Policy/Pref Sticky Policy-->
        <xs:complexType name="CommonDHPSPType">
                <xs:sequence>
                        <xs:element ref="AuthorizationsSet" minOccurs="0"/>
                        <xs:element ref="ob:ObligationsSet" minOccurs="0"/>
                </xs:sequence>
        </xs:complexType>
        <!-- -->

        <!-- DataHandlingPolicy -->
        <xs:element name="DataHandlingPolicy" type="ppl:DataHandlingPolicyType"/>
        <xs:complexType name="DataHandlingPolicyType">
                <xs:complexContent>
                        <xs:extension base="ppl:CommonDHPSPType">
                                <xs:attribute name="PolicyId" type="xs:anyURI"
use="required"/>
                        </xs:extension>
                </xs:complexContent>
        </xs:complexType>
        <!-- -->

        <xs:element name="DataHandlingPreferences" type="ppl:DataHandlingPreferencesType"/>
        <xs:complexType name="DataHandlingPreferencesType">
                <xs:complexContent>
                        <xs:extension base="ppl:CommonDHPSPType"/>
                </xs:complexContent>
```

```
        </xs:complexType>
        <!-- -->

        <xs:element name="StickyPolicy" type="ppl:StickyPolicyType"/>
        <xs:complexType name="StickyPolicyType">
                <xs:complexContent>
                        <xs:extension base="ppl:CommonDHPSPType"/>
                </xs:complexContent>
        </xs:complexType>
        <!-- -->
        <!-- List of Authorization -->
        <xs:element name="AuthorizationsSet" type="ppl:AuthorizationsSetType"/>
        <xs:complexType name="AuthorizationsSetType">
                <xs:sequence>
                        <xs:element ref="Authorization" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:attribute name="matching" type="xs:boolean" default="true"
use="optional"/>
        <xs:attribute name="mismatchId" type="xs:IDREF" use="optional"/>
        </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:attribute name="matching" type="xs:boolean" default="true"
use="optional"/>
        <xs:attribute name="mismatchId" type="xs:IDREF" use="optional"/>
                <xs:anyAttribute/>
        </xs:complexType>
        <!-- -->

        <!-- Purposes -->
        <xs:element name="Purpose" type="xs:anyURI"/>
        <!-- -->

        <!-- Authorization: Use for purpose -->
        <xs:element name="AuthzUseForPurpose" substitutionGroup="Authorization">
                <xs:complexType>
                        <xs:complexContent>
                                <xs:restriction base="AuthorizationType">
                                        <xs:sequence maxOccurs="unbounded">
                                                <xs:element ref="ppl:Purpose"/>
                                        </xs:sequence>
                                </xs:restriction>
                        </xs:complexContent>
                </xs:complexType>
        </xs:element>
        <!-- -->
        <!-- Authorization: Downstream usage -->
        <xs:element name="AuthzDownstreamUsage" substitutionGroup="Authorization">
                <xs:complexType>
                        <xs:complexContent>
                                <xs:restriction base="AuthorizationType">
                                        <xs:sequence minOccurs="0">
                                                <xs:element ref="ppl:Policy"/>
                                        </xs:sequence>
                                        <xs:attribute name="allowed" type="xs:boolean"
use="optional"/>
                                </xs:restriction>
                        </xs:complexContent>
                </xs:complexType>
        </xs:element>
        <!-- -->

        <!-- ProvisionalAction extension -->
        <!-- ProvisionalActions -->
        <xs:element name="ProvisionalActions" type="ppl:ProvisionalActionsType"/>
        <xs:complexType name="ProvisionalActionsType">
                <xs:sequence>
                        <xs:element ref="ppl:ProvisionalAction" maxOccurs="unbounded"/>
                </xs:sequence>
```

```
        </xs:complexType>
        <!-- -->
        <!-- ProvisionalAction -->
        <xs:element name="ProvisionalAction" type="ppl:ProvisionalActionType"/>
        <xs:complexType name="ProvisionalActionType">
                <xs:sequence>
                        <xs:element ref="xacml:AttributeValue" maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:attribute name="ActionId" type="xs:anyURI" use="required"/>
        </xs:complexType>
        <!-- -->

</xs:schema>
```

8.1.2 PrimeLife Claim Schema

This schema describes how the claims between Data Subject and Data Controller should be formatted. These claims re-use some elements of the SAML 2 schema.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xs:schema
        xmlns:ppl="http://www.primelife.eu/ppl"
        xmlns:pplc="http://www.primelife.eu/ppl/claims"
        xmlns:ob="http://www.primelife.eu/ppl/obligation"
        xmlns:obmm="http://www.primelife.eu/ppl/obligation/mismatch"
        xmlns:sp="http://www.primelife.eu/ppl/stickypolicy"
        xmlns:samla="urn:oasis:names:tc:SAML:2.0:assertion"
        xmlns:xacml="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
        xmlns:xacmlc="urn:oasis:names:tc:xacml:2.0:context:schema:os"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        attributeFormDefault="unqualified" elementFormDefault="qualified"
        targetNamespace="http://www.primelife.eu/ppl/claims">

        <xs:import namespace="http://www.primelife.eu/ppl"
schemaLocation="PrimeLifeSchema.xsd" />
        <xs:import namespace="http://www.primelife.eu/ppl/obligation"
schemaLocation="PrimeLifeObligation.xsd" />
        <xs:import namespace="http://www.primelife.eu/ppl/obligation/mismatch"
schemaLocation="PrimeLifeObligationMismatch.xsd" />
        <xs:import namespace="http://www.primelife.eu/ppl/stickypolicy"
schemaLocation="StickyPolicy.xsd" />
        <xs:import namespace="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
                schemaLocation="access_control-xacml-2.0-policy-schema-os.xsd" />
        <xs:import namespace="urn:oasis:names:tc:xacml:2.0:context:schema:os"
                schemaLocation="access_control-xacml-2.0-context-schema-os.xsd" />
        <xs:import namespace="urn:oasis:names:tc:SAML:2.0:assertion"
                schemaLocation="http://docs.oasis-open.org/security/saml/v2.0/saml-schema-
assertion-2.0.xsd" />
        <xs:element name="ResourceQuery" type="pplc:ResourceQueryType" />
        <xs:complexType name="ResourceQueryType">
                <xs:sequence>
                        <xs:element ref="xacmlc:Request" />
                        <xs:element maxOccurs="unbounded" minOccurs="0"
                                ref="samla:Assertion" />
                        <xs:element minOccurs="0" ref="ppl:DataHandlingPolicy" />
                </xs:sequence>
        </xs:complexType>
        <xs:complexType mixed="false" name="ConditionStatementType">
                <xs:complexContent mixed="false">
                        <xs:extension base="samla:StatementAbstractType">
                                <xs:sequence>
                                        <xs:element ref="xacml:Condition" />
                                </xs:sequence>
                                <xs:attribute name="EvidenceId" type="xs:anyURI" />
                        </xs:extension>
                </xs:complexContent>
        </xs:complexType>
        <xs:complexType mixed="false" name="ProvisionalActionStatementType">
                <xs:complexContent mixed="false">
```

```
                        <xs:extension base="samla:StatementAbstractType">
                                <xs:sequence maxOccurs="unbounded">
                                        <xs:element ref="pplc:ProvisionalAction" />
                                </xs:sequence>
                                <xs:attribute name="EvidenceId" type="xs:anyURI" />
                        </xs:extension>
                </xs:complexContent>
        </xs:complexType>
        <xs:element name="ProvisionalAction" type="pplc:ProvisionalActionType" />
        <xs:complexType name="ProvisionalActionType">
                <xs:complexContent>
                        <xs:extension base="ppl:ProvisionalActionType">
                                <xs:attribute name="EvidenceId" type="xs:anyURI" />
                        </xs:extension>
                </xs:complexContent>
        </xs:complexType>
        <xs:complexType mixed="false" name="AttributeStatementType">
                <xs:complexContent mixed="false">
                        <xs:extension base="samla:AttributeStatementType">
                                <xs:attribute name="EvidenceId" type="xs:anyURI" />
                        </xs:extension>
                </xs:complexContent>
        </xs:complexType>
        <xs:complexType mixed="false" name="AttributeType">
                <xs:complexContent mixed="false">
                        <xs:extension base="samla:AttributeType">
                                <xs:attribute name="StickyPolicyId" type="xs:anyURI" />
                                <xs:attribute name="EvidenceId" type="xs:anyURI" />
                        </xs:extension>
                </xs:complexContent>
        </xs:complexType>
        <!--  -->

        <xs:complexType mixed="false" name="EvidenceStatementType">
                <xs:complexContent mixed="false">
                        <xs:extension base="samla:StatementAbstractType">
                                <xs:sequence>
                                        <xs:element maxOccurs="unbounded" ref="pplc:Evidence"
/>
                                </xs:sequence>
                        </xs:extension>
                </xs:complexContent>
        </xs:complexType>
        <!--  -->

        <xs:element name="Evidence" type="pplc:EvidenceType" />
        <xs:complexType name="EvidenceType">
                <xs:sequence maxOccurs="unbounded" minOccurs="0">
                        <xs:any maxOccurs="unbounded" minOccurs="0" namespace="##any"
                                processContents="lax" />
                </xs:sequence>
                <xs:attribute name="ID" type="xs:anyURI" use="optional" />
        </xs:complexType>
        <!--  -->

        <xs:complexType mixed="false" name="StickyPolicyStatementType">
                <xs:complexContent mixed="false">
                        <xs:extension base="samla:StatementAbstractType">
                                <xs:sequence minOccurs="0" maxOccurs="unbounded">
                                        <xs:element ref="sp:Attribute" />
                                </xs:sequence>
                        </xs:extension>
                </xs:complexContent>
        </xs:complexType>
        <!--  -->

        <xs:element name="PPLPolicyStatement" type="pplc:PPLPolicyStatementType" />
        <xs:complexType name="PPLPolicyStatementType">
                <xs:complexContent>
                        <xs:extension base="samla:StatementAbstractType">
                                <xs:choice minOccurs="0" maxOccurs="unbounded">
                                        <xs:element ref="ppl:Policy" />
                                        <xs:element ref="ppl:PolicySet" />
                                </xs:choice>
```

```
                        </xs:extension>
                </xs:complexContent>
        </xs:complexType>
        <!--  -->

        <xs:element name="Claim" type="pplc:ClaimType" />
        <xs:complexType name="ClaimType">
                <xs:sequence>
                        <xs:element ref="samla:Assertion" minOccurs="0"
maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
        <!--  -->

        <xs:element name="Claims" type="pplc:ClaimsType" />
        <xs:complexType name="ClaimsType">
                <xs:sequence>
                        <xs:element ref="pplc:Claim" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>

        <xs:element name="Response" type="pplc:ResponseType" />
        <xs:complexType mixed="false" name="ResponseType">
                <xs:complexContent mixed="false">
                        <xs:extension base="samla:StatementAbstractType">
                                <xs:sequence>
                                        <xs:element ref="pplc:MissingPII" minOccurs="0"/>
                                        <xs:element ref="pplc:AccessPII" minOccurs="0"/>
                                        <xs:element ref="pplc:DeneyPII" minOccurs="0"/>
                                        <xs:element ref="pplc:MissingCredential"
minOccurs="0"/>
                                </xs:sequence>
                                <xs:attribute name="Decision" type="pplc:Decision"
use="required" />
                        </xs:extension>
                </xs:complexContent>

        </xs:complexType>
        <!--  -->

        <xs:simpleType name="Decision">
          <xs:restriction base="xs:string">
            <xs:enumeration value="Access" />
            <xs:enumeration value="Deny" />
            <xs:enumeration value="Indeterminate" />
          </xs:restriction>
        </xs:simpleType>

        <xs:element name="MissingPII" type="pplc:ListPIIType" />
        <xs:element name="AccessPII" type="pplc:ListPIIType" />
        <xs:element name="DeneyPII" type="pplc:ListPIIType" />
        <xs:element name="MissingCredential" type="pplc:ListPIIType" />

        <xs:complexType mixed="false" name="ListPIIType">
                <xs:sequence minOccurs="0" maxOccurs="unbounded">
                        <xs:element name="value" type="xs:string"/>
                </xs:sequence>
        </xs:complexType>

</xs:schema>
```

8.1.3 PrimeLife Credenatial Handling Schema

This schema specifies how credential requirement should be declared

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xacml="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
xmlns:cr="http://www.primelife.eu/ppl/credential" xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.primelife.eu/ppl/credential" elementFormDefault="qualified"
attributeFormDefault="unqualified">
```

```
        <xs:import namespace="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
schemaLocation="access_control-xacml-2.0-policy-schema-os.xsd"/>
        <!-- CredentialRequirements extension (note: MatchValueType is modeled on
xacml:AttributeValueType) -->
        <xs:complexType name="MatchValueType" mixed="true">
                <xs:sequence>
                        <xs:any namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:attribute name="MatchId" type="xs:anyURI" use="required"/>
                <xs:attribute name="DataType" type="xs:anyURI" use="required"/>
                <xs:attribute name="Disclose" type="cr:DiscloseType" use="optional"
default="yes"/>
        </xs:complexType>
        <xs:complexType name="AttributeMatchAnyOfType">
                <xs:sequence maxOccurs="unbounded">
                        <xs:choice>
                                <xs:element ref="cr:MatchValue" minOccurs="0"/>
                                <xs:element ref="cr:UndisclosedExpression" minOccurs="0"/>
                        </xs:choice>
                </xs:sequence>
                <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>
                <xs:attribute name="Disclose" type="cr:DiscloseType" use="optional"/>
        </xs:complexType>
        <xs:complexType name="CredentialType">
                <xs:sequence maxOccurs="unbounded">
                        <xs:choice>
                                <xs:element ref="cr:AttributeMatchAnyOf" minOccurs="0"/>
                                <xs:element ref="cr:UndisclosedExpression" minOccurs="0"/>
                        </xs:choice>
                </xs:sequence>
                <xs:attribute name="CredentialId" type="xs:anyURI" use="required"/>
        </xs:complexType>
        <xs:complexType name="CredentialRequirementsType">
                <xs:sequence>
                        <xs:element ref="cr:Credential" maxOccurs="unbounded"/>
                        <xs:element ref="cr:Condition" minOccurs="0"/>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="ConditionType">
                <xs:sequence>
                        <xs:element ref="xacml:Expression"/>
                </xs:sequence>
        </xs:complexType>
        <xs:element name="MatchValue" type="cr:MatchValueType"/>
        <xs:element name="AttributeMatchAnyOf" type="cr:AttributeMatchAnyOfType"/>
        <xs:element name="Credential" type="cr:CredentialType"/>
        <xs:element name="CredentialRequirements" type="cr:CredentialRequirementsType"/>
        <xs:element name="Condition" type="cr:ConditionType"/>

        <xs:simpleType name="DiscloseType">
                <xs:restriction base="xs:string">
                        <xs:enumeration value="yes"/>
                        <xs:enumeration value="no"/>
                        <xs:enumeration value="attributes-only"/>
                </xs:restriction>
        </xs:simpleType>
        <xs:complexType name="UndisclosedExpressionType" mixed="false">
                <xs:complexContent mixed="false">
                        <xs:extension base="xacml:ExpressionType">
                                <xs:sequence minOccurs="0" maxOccurs="unbounded">
                                        <xs:element name="AttributeId" type="xs:anyURI"/>
                                </xs:sequence>
                        </xs:extension>
                </xs:complexContent>
        </xs:complexType>
        <xs:complexType name="CredentialAttributeDesignatorType" mixed="false">
                <xs:complexContent mixed="false">
                        <xs:extension base="xacml:ExpressionType">
                                <xs:attribute name="CredentialId" type="xs:anyURI"
use="required"/>
                                <xs:attribute name="AttributeId" type="xs:anyURI"
use="required"/>
```

```
                                <xs:attribute name="DataType" type="xs:anyURI"
use="required"/>
                        </xs:extension>
                </xs:complexContent>
        </xs:complexType>
        <xs:element name="UndisclosedExpression" type="cr:UndisclosedExpressionType"
substitutionGroup="xacml:Expression"/>
        <xs:element name="CredentialAttributeDesignator"
type="cr:CredentialAttributeDesignatorType" substitutionGroup="xacml:Expression"/>
        <!-- Redefined XACML elements -->
        <xs:complexType name="PrimelifeApplyType" mixed="false">
                <xs:complexContent mixed="false">
                        <xs:extension base="xacml:ApplyType">
                                <xs:attribute name="Disclose" type="cr:DiscloseType"
use="optional" default="yes"/>
                        </xs:extension>
                </xs:complexContent>
        </xs:complexType>
        <xs:element name="Apply" type="cr:PrimelifeApplyType"
substitutionGroup="xacml:Apply"/>
</xs:schema>
```

8.1.4 PrimeLife Obligation Schema

This schema specifies how triggers and actions related to obligation should be formatted.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema targetNamespace="http://www.primelife.eu/ppl/obligation"
elementFormDefault="qualified" attributeFormDefault="unqualified"
           xmlns="http://www.primelife.eu/ppl/obligation"
           xmlns:ob="http://www.primelife.eu/ppl/obligation"
           xmlns:ppl="http://www.primelife.eu/ppl"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:import namespace="http://www.primelife.eu/ppl" schemaLocation="PrimeLifeSchema.xsd" />

  <xs:attribute name="match" type="xs:boolean" default="true"/>

  <!-- List of Obligations -->
  <xs:element name="ObligationsSet" type="ob:ObligationsSet"/>
  <xs:complexType name="ObligationsSet">
    <xs:sequence>
      <xs:element ref="ob:Obligation" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="matching" type="xs:boolean" default="true" use="optional"/>
    <xs:attribute name="infinit" type="xs:boolean" default="false" use="optional"/>
    <xs:attribute name="mismatchId" type="xs:string" use="optional"/>
    <xs:attribute name="elementId" type="xs:string" use="optional" />
  </xs:complexType>

  <!-- Obligation -->
  <xs:element name="Obligation" type="ob:Obligation" />
  <xs:complexType name="Obligation">
    <xs:sequence>
      <xs:element ref="ob:TriggersSet" minOccurs="1" maxOccurs="1" />
      <xs:element ref="ob:Action" minOccurs="1" maxOccurs="1" />
    </xs:sequence>
    <xs:attribute name="matching" type="xs:boolean" default="true" use="optional"/>
    <xs:attribute name="mismatchId" type="xs:string" use="optional"/>
    <xs:attribute name="elementId" type="xs:string" use="optional" />
  </xs:complexType>

  <!-- List of Triggers -->
  <xs:element name="TriggersSet" type="ob:TriggersSet" />
  <xs:complexType name="TriggersSet">
    <xs:sequence minOccurs="1" maxOccurs="unbounded">
      <xs:element ref="ob:Trigger" />
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="optional"/>
    <xs:attribute name="matching" type="xs:boolean" default="true" use="optional"/>
```

```
    <xs:attribute name="mismatchId" type="xs:string" use="optional"/>
    <xs:attribute name="elementId" type="xs:string" use="optional" />
  </xs:complexType>

  <!-- Trigger (abstract) -->
  <xs:element name="Trigger" abstract="true" type="ob:Trigger" />
  <xs:complexType name="Trigger">
    <xs:sequence/>
    <xs:attribute name="name" type="xs:string" use="optional"/>
    <xs:attribute name="matching" type="xs:boolean" default="true" use="optional"/>
    <xs:attribute name="mismatchId" type="xs:string" use="optional"/>
    <xs:attribute name="elementId" type="xs:string" use="optional" />
  </xs:complexType>

  <!-- Action (abstract) -->
  <xs:element name="Action" abstract="true" type="ob:Action"/>
  <xs:complexType name="Action">
    <xs:sequence/>
    <xs:attribute name="name" type="xs:string" use="optional"/>
    <xs:attribute name="matching" type="xs:boolean" default="true" use="optional"/>
    <xs:attribute name="mismatchId" type="xs:string" use="optional"/>
    <xs:attribute name="elementId" type="xs:string" use="optional" />
  </xs:complexType>

  <!-- TriggerAtTime -->
  <xs:element name="TriggerAtTime" type="ob:TriggerAtTime" substitutionGroup="ob:Trigger"/>
  <xs:complexType name="TriggerAtTime">
    <xs:complexContent>
      <xs:extension base="ob:Trigger">
        <xs:sequence>
          <xs:element name="Start" type="ob:DateAndTime" minOccurs="1" maxOccurs="1"/>
          <xs:element name="MaxDelay" type="ob:Duration" minOccurs="1" maxOccurs="1"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <!-- DateAndTime -->
  <xs:element name="DateAndTime" type="DateAndTime" />
  <xs:element name="Duration" type="Duration" />

  <xs:complexType name="DateAndTime">
    <xs:choice>
      <xs:element name="DateAndTime" type="xs:dateTime" minOccurs="1" maxOccurs="1" />
      <xs:element name="StartNow" />
    </xs:choice>
    <xs:attribute name="matching" type="xs:boolean" default="true" use="optional"/>
    <xs:attribute name="mismatchId" type="xs:string" use="optional"/>
    <xs:attribute name="elementId" type="xs:string" use="optional" />
  </xs:complexType>


  <!-- Duration -->
  <xs:complexType name="Duration">
    <xs:sequence>
      <xs:element name="Duration" type="xs:duration" minOccurs="1" maxOccurs="1" />
    </xs:sequence>
    <xs:attribute name="matching" type="xs:boolean" default="true" use="optional"/>
    <xs:attribute name="mismatchId" type="xs:string" use="optional"/>
    <xs:attribute name="elementId" type="xs:string" use="optional" />
  </xs:complexType>

  <!-- TriggerPeriodic -->
  <xs:element name="TriggerPeriodic" type="ob:TriggerPeriodic"
substitutionGroup="ob:Trigger"/>
  <xs:complexType name="TriggerPeriodic">
    <xs:complexContent>
      <xs:extension base="ob:Trigger">
        <xs:sequence>
          <xs:element name="Start" type="ob:DateAndTime" minOccurs="1" maxOccurs="1" />
          <xs:element name="End" type="ob:DateAndTime" minOccurs="1" maxOccurs="1" />
          <xs:element name="MaxDelay" type="ob:Duration" minOccurs="1" maxOccurs="1" />
          <xs:element name="Period" type="ob:Duration" minOccurs="1" maxOccurs="1" />
        </xs:sequence>
```

```
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>


  <!-- TriggerPersonalDataAccessedForPurpose -->
  <xs:element name="TriggerPersonalDataAccessedForPurpose"
type="ob:TriggerPersonalDataAccessedForPurpose" substitutionGroup="ob:Trigger"/>
  <xs:complexType name="TriggerPersonalDataAccessedForPurpose">
    <xs:complexContent>
      <xs:extension base="ob:Trigger">
        <xs:sequence>
          <xs:element ref="ppl:Purpose" minOccurs="0" maxOccurs="unbounded" />
          <xs:element name="MaxDelay" type="ob:Duration" minOccurs="1" maxOccurs="1" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>


  <!-- TriggerPersonalDataDeleted -->
  <xs:element name="TriggerPersonalDataDeleted" type="ob:TriggerPersonalDataDeleted"
substitutionGroup="ob:Trigger"/>
  <xs:complexType name="TriggerPersonalDataDeleted">
    <xs:complexContent>
      <xs:extension base="ob:Trigger">
        <xs:sequence>
          <xs:element name="MaxDelay" type="ob:Duration" minOccurs="1" maxOccurs="1" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>


  <!-- TriggerPersonalDataSent -->
  <xs:element name="TriggerPersonalDataSent" type="ob:TriggerPersonalDataSent"
substitutionGroup="ob:Trigger"/>
  <xs:complexType name="TriggerPersonalDataSent">
    <xs:complexContent>
      <xs:extension base="ob:Trigger">
        <xs:sequence>
          <xs:element name="Id" type="xs:anyURI" minOccurs="1" maxOccurs="1" />
          <xs:element name="MaxDelay" type="ob:Duration" minOccurs="1" maxOccurs="1" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>


  <!-- TriggerDataSubjectAccess -->
  <xs:element name="TriggerDataSubjectAccess" type="ob:TriggerDataSubjectAccess"
substitutionGroup="ob:Trigger"/>
  <xs:complexType name="TriggerDataSubjectAccess">
    <xs:complexContent>
      <xs:extension base="ob:Trigger">
        <xs:sequence>
          <xs:element name="url" type="xs:anyURI" minOccurs="1" maxOccurs="1" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>


  <!-- TriggerDataLost -->
  <xs:element name="TriggerDataLost" type="ob:TriggerDataLost"
substitutionGroup="ob:Trigger"/>
  <xs:complexType name="TriggerDataLost">
    <xs:complexContent>
      <xs:extension base="ob:Trigger">
        <xs:sequence>
          <xs:element name="MaxDelay" type="ob:Duration" minOccurs="1" maxOccurs="1" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
```

```
  </xs:complexType>


  <!-- TriggerOnViolation -->
  <xs:element name="TriggerOnViolation" type="ob:TriggerOnViolation"
substitutionGroup="ob:Trigger"/>
  <xs:complexType name="TriggerOnViolation">
    <xs:complexContent>
      <xs:extension base="ob:Trigger">
        <xs:sequence>
          <xs:element name="MaxDelay" type="ob:Duration" minOccurs="1" maxOccurs="1" />
          <xs:element ref="ob:Obligation" minOccurs="0" maxOccurs="unbounded" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>


  <!-- ActionDeletePersonalData -->
  <xs:element name="ActionDeletePersonalData" type="ob:ActionDeletePersonalData"
substitutionGroup="ob:Action"/>
  <xs:complexType name="ActionDeletePersonalData">
    <xs:complexContent>
      <xs:extension base="ob:Action">
        <xs:sequence>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>


  <!-- ActionAnonymizePersonalData -->
  <xs:element name="ActionAnonymizePersonalData" type="ob:ActionAnonymizePersonalData"
substitutionGroup="ob:Action"/>
  <xs:complexType name="ActionAnonymizePersonalData">
    <xs:complexContent>
      <xs:extension base="ob:Action">
        <xs:sequence>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>


  <!-- ActionNotifyDataSubject -->
  <xs:element name="ActionNotifyDataSubject" type="ob:ActionNotifyDataSubject"
substitutionGroup="ob:Action"/>
  <xs:complexType name="ActionNotifyDataSubject">
    <xs:complexContent>
      <xs:extension base="ob:Action">
        <xs:sequence>
          <xs:element name="Media" type="xs:string" minOccurs="1" maxOccurs="1" />
          <xs:element name="Address" type="xs:string" minOccurs="1" maxOccurs="1" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>


  <!-- ActionLog -->
  <xs:element name="ActionLog" type="ob:ActionLog" substitutionGroup="ob:Action"/>
  <xs:complexType name="ActionLog">
    <xs:complexContent>
      <xs:extension base="ob:Action">
        <xs:sequence>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>


  <!-- ActionSecureLog -->
  <xs:element name="ActionSecureLog" type="ob:ActionSecureLog" substitutionGroup="ob:Action"/>
  <xs:complexType name="ActionSecureLog">
```

```
    <xs:complexContent>
      <xs:extension base="ob:Action">
        <xs:sequence>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

</xs:schema>
```

8.1.5 Mismatching Schema

In order to automatically express what are les elements that are mismatching between the Data controller preferences and the data subject privacy policy, we generated a XML schema describing all the elements that could not be compatible in the authorization and obligation policies. This schema can be used with the UI interface to show to the user what are the differences between his preferences and the privacy policy of the data controller. This information can help him to make a decision.

8.1.5.1 Authorization Mismatch Schema

```
<?xml version="1.0" encoding="utf-8"?>

<xs:schema
                        xmlns="http://www.primelife.eu/ppl/authorization/mismatch"
                        xmlns:aumm="http://www.primelife.eu/ppl/authorization/mismatch"

                xmlns:ppl="http://www.primelife.eu/ppl"
                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                targetNamespace="http://www.primelife.eu/ppl/authorization/mismatch"
                elementFormDefault="qualified"
                attributeFormDefault="unqualified">

  <xs:import namespace="http://www.primelife.eu/ppl" schemaLocation="PrimeLifeSchema.xsd" />

  <!-- Authorization Mismatch -->
  <xs:element name="AuthorizationsMismatch" type="aumm:AuthorizationsMismatchType"/>
  <xs:complexType name="AuthorizationsMismatchType">
    <xs:sequence>
      <xs:element ref="aumm:AuthorizationsSet" minOccurs="0" maxOccurs="1" />
      <xs:element ref="aumm:AuthzUseForPurpose" minOccurs="0" maxOccurs="1" />
      <xs:element ref="aumm:AuthzDownstreamUsage" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
    <xs:attribute name="mismatchId" type="xs:ID" use="optional"/>
  </xs:complexType>
  <!--  -->

  <xs:element name="AuthorizationsSet" type="aumm:AuthorizationsSetMismatchType"/>
  <xs:complexType name="AuthorizationsSetMismatchType">
    <xs:sequence>
        <xs:element name="Policy" type="ppl:AuthorizationsSetType" />
        <xs:element name="Preference" type="ppl:AuthorizationsSetType" />
    </xs:sequence>
    <xs:attribute name="mismatchId" type="xs:ID" />
  </xs:complexType>


  <xs:element name="AuthzUseForPurpose" type="aumm:AuthzUseForPurposeMismatchType"/>
  <xs:complexType name="AuthzUseForPurposeMismatchType">
    <xs:sequence>
        <xs:element name="Policy" type="aumm:PurposeListType" />
        <xs:element name="Preference" type="aumm:PurposeListType" />
    </xs:sequence>
    <xs:attribute name="mismatchId" type="xs:ID" />
  </xs:complexType>
```

```
  <xs:complexType name="PurposeListType">
    <xs:sequence>
        <xs:element ref="ppl:Purpose" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <!--  -->

  <xs:element name="AuthzDownstreamUsage" type="aumm:AuthzDownstreamUsageMismatchType"/>
  <xs:complexType name="AuthzDownstreamUsageMismatchType">
    <xs:sequence>
        <xs:element name="Policy" type="ppl:AuthorizationType"/>
        <xs:element name="Preference" type="ppl:AuthorizationType" />
    </xs:sequence>
    <xs:attribute name="mismatchId" type="xs:ID" />
  </xs:complexType>

</xs:schema>
```

8.1.5.2 Obligation Mismatching Schema

```
<?xml version="1.0" encoding="utf-8"?>

<xs:schema
                        xmlns="http://www.primelife.eu/ppl/authorization/mismatch"
                        xmlns:aumm="http://www.primelife.eu/ppl/authorization/mismatch"

                xmlns:ppl="http://www.primelife.eu/ppl"
                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                targetNamespace="http://www.primelife.eu/ppl/authorization/mismatch"
                elementFormDefault="qualified"
                attributeFormDefault="unqualified">

  <xs:import namespace="http://www.primelife.eu/ppl" schemaLocation="PrimeLifeSchema.xsd" />

  <!-- Authorization Mismatch -->
  <xs:element name="AuthorizationsMismatch" type="aumm:AuthorizationsMismatchType"/>
  <xs:complexType name="AuthorizationsMismatchType">
    <xs:sequence>
      <xs:element ref="aumm:AuthorizationsSet" minOccurs="0" maxOccurs="1" />
      <xs:element ref="aumm:AuthzUseForPurpose" minOccurs="0" maxOccurs="1" />
      <xs:element ref="aumm:AuthzDownstreamUsage" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
    <xs:attribute name="mismatchId" type="xs:ID" use="optional"/>
  </xs:complexType>
  <!--  -->

  <xs:element name="AuthorizationsSet" type="aumm:AuthorizationsSetMismatchType"/>
  <xs:complexType name="AuthorizationsSetMismatchType">
    <xs:sequence>
        <xs:element name="Policy" type="ppl:AuthorizationsSetType" />
        <xs:element name="Preference" type="ppl:AuthorizationsSetType" />
    </xs:sequence>
    <xs:attribute name="mismatchId" type="xs:ID" />
  </xs:complexType>


  <xs:element name="AuthzUseForPurpose" type="aumm:AuthzUseForPurposeMismatchType"/>
  <xs:complexType name="AuthzUseForPurposeMismatchType">
    <xs:sequence>
        <xs:element name="Policy" type="aumm:PurposeListType" />
        <xs:element name="Preference" type="aumm:PurposeListType" />
    </xs:sequence>
    <xs:attribute name="mismatchId" type="xs:ID" />
  </xs:complexType>

  <xs:complexType name="PurposeListType">
    <xs:sequence>
        <xs:element ref="ppl:Purpose" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
```

```
  <!--  -->

  <xs:element name="AuthzDownstreamUsage" type="aumm:AuthzDownstreamUsageMismatchType"/>
  <xs:complexType name="AuthzDownstreamUsageMismatchType">
    <xs:sequence>
        <xs:element name="Policy" type="ppl:AuthorizationType"/>
        <xs:element name="Preference" type="ppl:AuthorizationType" />
    </xs:sequence>
    <xs:attribute name="mismatchId" type="xs:ID" />
  </xs:complexType>

</xs:schema>
```

8.1.6 Sticky Policy Schema

The sticky policy schema inherits from the data handling schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.primelife.eu/ppl/stickypolicy"
                xmlns="http://www.primelife.eu/ppl/stickypolicy"
                xmlns:sp="http://www.primelife.eu/ppl/stickypolicy"
                xmlns:ppl="http://www.primelife.eu/ppl"
                xmlns:ob="http://www.primelife.eu/ppl/obligation"
                xmlns:obmm="http://www.primelife.eu/ppl/obligation/mismatch"
                xmlns:aumm="http://www.primelife.eu/ppl/authorization/mismatch"

                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                elementFormDefault="qualified" attributeFormDefault="unqualified">

        <xs:import namespace="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
schemaLocation="access_control-xacml-2.0-policy-schema-os.xsd"/>
        <xs:import namespace="http://www.primelife.eu/ppl"
schemaLocation="PrimeLifeSchema.xsd"/>
        <xs:import namespace="http://www.primelife.eu/ppl/obligation"
schemaLocation="PrimeLifeObligation.xsd"/>
        <xs:import namespace="http://www.primelife.eu/ppl/obligation/mismatch"
schemaLocation="PrimeLifeObligationMismatch.xsd"/>
        <xs:import namespace="http://www.primelife.eu/ppl/authorization/mismatch"
schemaLocation="PrimeLifeAuthorizationMismatch.xsd"/>


        <xs:element name="StickyPolicy" type="sp:StickyPolicy"/>
        <xs:complexType name="StickyPolicy">
                <xs:sequence>
                        <xs:element ref="sp:Attribute" maxOccurs="unbounded"/>

                </xs:sequence>
                <xs:attribute name="matching" type="xs:boolean" default="true"
use="optional"/>
        </xs:complexType>

        <xs:element name="Attribute" type="sp:AttributeType"/>
        <xs:complexType name="AttributeType">
                <xs:sequence>
                        <xs:element ref="ppl:AuthorizationsSet"/>
                        <xs:element ref="ob:ObligationsSet"/>

                        <xs:element ref="sp:Mismatches" minOccurs="0" maxOccurs="1" />
                        <!--  <xs:element ref="obmm:ObligationsSet" minOccurs="0"/> -->

                </xs:sequence>
                <xs:attribute name="AttributeURI" type="xs:anyURI" use="optional"/>
                <xs:attribute name="matching" type="xs:boolean" default="true"
use="optional"/>
                <xs:attribute name="ID" type="xs:anyURI" use="optional"/>
        </xs:complexType>

        <xs:element name="Mismatches" type="sp:MismatchesType"/>
        <xs:complexType name="MismatchesType">
                <xs:sequence>
```

```
                    <xs:element ref="aumm:AuthorizationsMismatch" minOccurs="0"/>
                    <xs:element name="ObligationsMismatch" type="obmm:Mismatches"
minOccurs="0"/>
                </xs:sequence>
        </xs:complexType>

</xs:schema>
```

8.2 Example Policies

This appendinx section presents some emaples of policies and preferences related what was defined in this document. These exemples are valid compared to the XML shcema described before and can be useful for any developer that wants to test or implement the PrimeLife policy engine.

8.2.1 XACML

This example is presenting the policy that is securing the access to the resources that target namespace attribute (urn:oasis:names:tc:xacml:1.0:resource:target-namespace) has a value urn:example:med:schemas:record. Further the single rule inside the policy is permitting read access to the subjects that role (urn:oasis:names:tc:xacml:2.0:example:attribute:role) attribute is equal to "head physician".

```
<Policy PolicyId =" Pol1 " RuleCombiningAlgId =" urn:oasis:names:tc:xacml:1 .0:
rule - combining - algorithm:permit - overrides "  >
  <Target >
   <Resources >
     <Resource >
       < ResourceMatch 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 >
         < ResourceAttributeDesignator
         AttributeId =" urn:oasis:names:tc:xacml:1 .0 :resource:target - namespace "
         DataType =" http: // www .w3.org /2001/ XMLSchema # string "/>
       </ ResourceMatch >
     </ Resource >
   </ Resources >
  </ Target >
  <Rule RuleId =" ReadRule " Effect =" Permit ">
  <Target >
    <Subjects >
      <Subject >
        < SubjectMatch MatchId =" urn:oasis:names:tc:xacml:1 .0 :function:string - equal ">
        < AttributeValue DataType =" http: // www.w3. org /2001/ XMLSchema # string ">
        head physician </ AttributeValue >
        < SubjectAttributeDesignator
        AttributeId =" urn:oasis:names:tc:xacml:2 .0 :example:attribute:role "
        DataType =" http: // www .w3.org /2001/ XMLSchema # string "/>
        </ SubjectMatch >
      </ Subject >
    </ Subjects >
    <Actions >
       <Action >
        < ActionMatch
        MatchId =" urn:oasis:names:tc:xacml:1 .0 :function:string - equal ">
        < AttributeValue
        DataType =" http: // www .w3.org /2001/ XMLSchema # string ">
        read </ xacml:AttributeValue >
        < ActionAttributeDesignator
        DataType =" http: // www .w3.org /2001/ XMLSchema # string "
        AttributeId =" urn:oasis:names:tc:xacml:1 .0 :action:action -id"/>
        </ ActionMatch >
```

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

8.2.2 Data Controller to Data Subject Claim

In the example below we define a claim containing the PII request from the data controller to the data subject. Claims contain the list of required PII, the identity of the data controller *http://store.example.com* and the list of data handling policies stating how the collected PIIs will be treated.

```
<?xml version="1.0" encoding="UTF-8"?>
        <!--            This is a sample SAML assertion that store.example.com sends back to
                the data subject after the first access attempt. It contains the
                policy for accessing the subscription page at store.example.com, plus
                the information about store.example.com itself (ID and privacy seals).
        -->

<samla:Assertion xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
        xmlns:cl="http://www.primelife.eu/ppl/claims"
xmlns:cr="http://www.primelife.eu/ppl/credential"
        xmlns:ob="http://www.primelife.eu/ppl/obligation"
xmlns:ppl="http://www.primelife.eu/ppl"
        xmlns:xacml="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
        xmlns:samla="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        Version="2.0" ID="assertion1af6c003518cd91ba8832c" IssueInstant="2010-08-31T23:59:59">

  <samla:Issuer>http://store.example.com</samla:Issuer>

    <samla:AttributeStatement>
        <samla:Attribute Name="http://www.primelife.eu/ppl/DataControllerID">
          <samla:AttributeValue>http://store.example.com</samla:AttributeValue>
            </samla:Attribute>

                <samla:Attribute
                        Name="http://www.european-privacy-seal.eu/hasEuropeanPrivacySeal">
                        <samla:AttributeValue>true</samla:AttributeValue>
                </samla:Attribute>

             </samla:AttributeStatement>

            <samla:Statement xsi:type="cl:PPLPolicyStatementType">
              <ppl:Policy>
                <ppl:DataHandlingPolicy PolicyId="#DHP1">
                  <ppl:AuthorizationsSet>
                    <ppl:AuthzUseForPurpose>

        <ppl:Purpose>http://www.w3.org/2002/01/P3Pv1/individual-analysis</ppl:Purpose>

        <ppl:Purpose>http://www.w3.org/2002/01/P3Pv1/admin</ppl:Purpose>

        <ppl:Purpose>http://www.w3.org/2006/01/P3Pv11/marketing</ppl:Purpose>
                  </ppl:AuthzUseForPurpose>
                  <ppl:AuthzDownstreamUsage allowed="false" />
                </ppl:AuthorizationsSet>

                 <ob:ObligationsSet>

                <!-- Obligation to log within 5 minutes  use of PII for purpose "contact"
                     within 15 minutes use of PII for purpose "delivery"
                     within 30 seconds use of PII for purpose "pseudo-analysis" -->

                   <ob:Obligation>
                     <ob:TriggersSet>
                        <ob:TriggerPersonalDataAccessedForPurpose
xmlns="http://www.primelife.eu/PPL/obligation">
        <ppl:Purpose>http://www.w3.org/2002/01/P3Pv1/contact</ppl:Purpose>
```

```
                <ob:MaxDelay>
                  <ob:Duration>
                    P0Y0M0DT0H5M0S
                  </ob:Duration>
                </ob:MaxDelay>
                </ob:TriggerPersonalDataAccessedForPurpose>
                  <ob:TriggerPersonalDataAccessedForPurpose
xmlns="http://www.primelife.eu/PPL/obligation">

        <ppl:Purpose>http://www.w3.org/2006/01/P3Pv11/delivery</ppl:Purpose>
                <ob:MaxDelay>
                  <ob:Duration>
                  P0Y0M0DT0H15M0S
                  </ob:Duration>
                </ob:MaxDelay>
                </ob:TriggerPersonalDataAccessedForPurpose>
                <ob:TriggerPersonalDataAccessedForPurpose
xmlns="http://www.primelife.eu/PPL/obligation">

        <ppl:Purpose>http://www.w3.org/2002/01/P3Pv1/pseudo-analysis</ppl:Purpose>
                <ob:MaxDelay>
                  <ob:Duration>
                  P0Y0M0DT0H0M30S
                  </ob:Duration>
                </ob:MaxDelay>
                </ob:TriggerPersonalDataAccessedForPurpose>
                        </ob:TriggersSet>
                          <ob:ActionLog/>
                          </ob:Obligation>
                  <!-- Obligation to delete collected PII within 5 days -->
                       <ob:Obligation>
                         <ob:TriggersSet>
                          <ob:TriggerAtTime>
                                <ob:Start>
                                  <ob:StartNow/>
                                </ob:Start>
                                <ob:MaxDelay>
                                  <ob:Duration>
                                  P0Y0M5DT0H0M0S
                                  </ob:Duration>
                                </ob:MaxDelay>
                                </ob:TriggerAtTime>
                                  </ob:TriggersSet>
                                    <ob:ActionDeletePersonalData/>
                                  /ob:Obligation>
                                </ob:ObligationsSet>

                        </ppl:DataHandlingPolicy>

                        <ppl:DataHandlingPolicy PolicyId="#DHP2">
                                <ppl:AuthorizationsSet>
                                        <ppl:AuthzUseForPurpose>

        <ppl:Purpose>http://www.w3.org/2006/01/P3Pv11/payment</ppl:Purpose>
                        </ppl:AuthzUseForPurpose>
                          <ppl:AuthzDownstreamUsage allowed="true" />
                        </ppl:AuthorizationsSet>

                        <ob:ObligationsSet>
                                <ob:Obligation>
                                        <ob:TriggersSet>
                                          <ob:TriggerPersonalDataAccessedForPurpose
xmlns="http://www.primelife.eu/PPL/obligation">

        <ppl:Purpose>http://www.w3.org/2002/01/P3Pv1/contact</ppl:Purpose>
                        <ob:MaxDelay>
                          <ob:Duration>
                                P0Y0M0DT0H5M0S
                          </ob:Duration>
                        </ob:MaxDelay>
                         </ob:TriggerPersonalDataAccessedForPurpose>
                          <ob:TriggerPersonalDataAccessedForPurpose
xmlns="http://www.primelife.eu/PPL/obligation">
```

```
        <ppl:Purpose>http://www.w3.org/2006/01/P3Pv11/delivery</ppl:Purpose>
                        <ob:MaxDelay>
                          <ob:Duration>
                                P0Y0M0DT0H15M0S
                          </ob:Duration>
                        </ob:MaxDelay>
                         </ob:TriggerPersonalDataAccessedForPurpose>
                          <ob:TriggerPersonalDataAccessedForPurpose
xmlns="http://www.primelife.eu/PPL/obligation">

        <ppl:Purpose>http://www.w3.org/2002/01/P3Pv1/pseudo-analysis</ppl:Purpose>
                        <ob:MaxDelay>
                          <ob:Duration>
                                P0Y0M0DT0H0M30S
                          </ob:Duration>
                        </ob:MaxDelay>
                           </ob:TriggerPersonalDataAccessedForPurpose>
                        </ob:TriggersSet>
                        <ob:ActionLog/>
                          </ob:Obligation>

                 <!-- Obligation to delete collected PII within 5 days -->
                          <ob:Obligation>
                        <ob:TriggersSet>
                          <ob:TriggerAtTime>
                                <ob:Start>
                                  <ob:StartNow/>
                                </ob:Start>
                                <ob:MaxDelay>
                                  <ob:Duration>
                                        P0Y0M10DT0H0M0S
                                  </ob:Duration>
                                </ob:MaxDelay>
                          </ob:TriggerAtTime>
                        </ob:TriggersSet>
                        <ob:ActionDeletePersonalData/>
                        </ob:Obligation>
                        </ob:ObligationsSet>
                        </ppl:DataHandlingPolicy>

                        <ppl:DataHandlingPolicy PolicyId="#DHP3">
                <ppl:AuthorizationsSet>
                        <ppl:AuthzUseForPurpose>
                                <ppl:Purpose>http://www.w3.org/2002/01/P3Pv1/individual-
analysis</ppl:Purpose>
        <ppl:Purpose>http://www.w3.org/2002/01/P3Pv1/admin</ppl:Purpose>
        <ppl:Purpose>http://www.w3.org/2006/01/P3Pv11/marketing</ppl:Purpose>
                        </ppl:AuthzUseForPurpose>

                        <ppl:AuthzDownstreamUsage allowed="true"/>
                </ppl:AuthorizationsSet>

                <ob:ObligationsSet>
<!--
                Obligation to log (within 5 minutes) use of PII for purpose "contact" and
"pseudo-analysis"
                -->
                        <ob:Obligation>
                                <ob:TriggersSet>
                                  <ob:TriggerPersonalDataAccessedForPurpose
xmlns="http://www.primelife.eu/PPL/obligation">

        <ppl:Purpose>http://www.w3.org/2002/01/P3Pv1/contact</ppl:Purpose>
                                <ob:MaxDelay>
                                  <ob:Duration>
                                        P0Y0M0DT0H5M0S
                                  </ob:Duration>
                                </ob:MaxDelay>
                                  </ob:TriggerPersonalDataAccessedForPurpose>
                                  <ob:TriggerPersonalDataAccessedForPurpose
xmlns="http://www.primelife.eu/PPL/obligation">

        <ppl:Purpose>http://www.w3.org/2006/01/P3Pv11/delivery</ppl:Purpose>
```

```
                                <ob:MaxDelay>
                                  <ob:Duration>
                                        P0Y0M0DT0H15M0S
                                  </ob:Duration>
                                </ob:MaxDelay>
                                  </ob:TriggerPersonalDataAccessedForPurpose>
                                  <ob:TriggerPersonalDataAccessedForPurpose
xmlns="http://www.primelife.eu/PPL/obligation">
        <ppl:Purpose>http://www.w3.org/2002/01/P3Pv1/pseudo-analysis</ppl:Purpose>
                                <ob:MaxDelay>
                                  <ob:Duration>
                                        P0Y0M0DT0H0M30S
                                  </ob:Duration>
                                </ob:MaxDelay>
                                  </ob:TriggerPersonalDataAccessedForPurpose>
                                </ob:TriggersSet>
                                <ob:ActionLog/>
                        </ob:Obligation>


<!-- Obligation to delete collected PII within 7 days -->
                        <ob:Obligation>
                                <ob:TriggersSet>
                                  <ob:TriggerAtTime>
                                        <ob:Start>
                                          <ob:StartNow/>
                                        </ob:Start>
                                        <ob:MaxDelay>
                                          <ob:Duration>
                                                P0Y0M7DT0H0M0S
                                          </ob:Duration>
                                        </ob:MaxDelay>
                                  </ob:TriggerAtTime>
                                </ob:TriggersSet>
                                <ob:ActionDeletePersonalData/>
                        </ob:Obligation>
                </ob:ObligationsSet>

        </ppl:DataHandlingPolicy>

                        <ppl:ProvisionalActions>
                                <ppl:ProvisionalAction
ActionId="http://www.primelife.eu/ppl/RevealUnderDHP">
                                        <xacml:AttributeValue
DataType="xs:anyURI">http://www.w3.org/2006/vcard/ns#email</xacml:AttributeValue>
                                        <xacml:AttributeValue
DataType="xs:anyURI">#DHP3</xacml:AttributeValue>
                                </ppl:ProvisionalAction>

                                <ppl:ProvisionalAction
ActionId="http://www.primelife.eu/ppl/RevealUnderDHP">
                                        <xacml:AttributeValue
DataType="xs:anyURI">http://www.example.org/names#display_name</xacml:AttributeValue>
                                        <xacml:AttributeValue
DataType="xs:anyURI">#DHP1</xacml:AttributeValue>
                                </ppl:ProvisionalAction>

                                <ppl:ProvisionalAction
ActionId="http://www.primelife.eu/ppl/RevealUnderDHP">
                                        <xacml:AttributeValue
DataType="xs:anyURI">http://www.example.org/names#user_name</xacml:AttributeValue>
                                        <xacml:AttributeValue
DataType="xs:anyURI">#DHP1</xacml:AttributeValue>
                                </ppl:ProvisionalAction>
                        </ppl:ProvisionalActions>

                </ppl:Policy>
        </samla:Statement>
</samla:Assertion>
```

8.2.3 Obligations

This section presents the policies, preferences, and sticky policies of each example presented in Sect. 2.4.2.5. It is important to notes that only the obligation part of PPL policies are presented here as consumed and generated by the OME and OEE.

Sticky policies presented in this section are “annotated” sticky policies with additional data used by the user interface. Those annotations do not leave the user’s trust domain.

8.2.3.1 Example 1 (matching)

Use of PII for Purpose X, Y will log within 5 Minutes + Delete PII within 7 days

Files of this example, namely “preference.xml”, “policy.xml” and “matchresults.xml”, are installed to “%PL_OBL_Samples%\DocExample 1 - Matching Obligations”.

8.2.3.1.1 Preference-side obligations set

```
<?xml version="1.0" encoding="utf-8" ?>

<!-- This is the obligation part of a PPL pref -->

<ob:ObligationsSet xmlns:ob="http://www.primelife.eu/ppl/obligation"
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xmlns:ppl="http://www.primelife.eu/ppl"
                   xsi:schemaLocation="http://www.primelife.eu/PPL/obligation
file:./PrimeLifeObligation.xsd">

  <!-- Obligation to log (within 5 minutes) use of PII for purpose "contact" and "pseudo-
analysis" -->
  <ob:Obligation>
    <ob:TriggersSet>
      <ob:TriggerPersonalDataAccessedForPurpose
        xmlns="http://www.primelife.eu/PPL/obligation">
        <ppl:Purpose>http://www.w3.org/2002/01/P3Pv1/contact</ppl:Purpose>
        <ob:MaxDelay>
          <ob:Duration>
            P0Y0M0DT0H5M0S
          </ob:Duration>
        </ob:MaxDelay>
      </ob:TriggerPersonalDataAccessedForPurpose>
      <ob:TriggerPersonalDataAccessedForPurpose
        xmlns="http://www.primelife.eu/PPL/obligation">
        <ppl:Purpose>http://www.w3.org/2002/01/P3Pv1/pseudo-analysis</ppl:Purpose>
        <ob:MaxDelay>
          <ob:Duration>
            P0Y0M0DT0H5M0S
          </ob:Duration>
        </ob:MaxDelay>
      </ob:TriggerPersonalDataAccessedForPurpose>
    </ob:TriggersSet>
    <ob:ActionLog/>
  </ob:Obligation>

  <!-- Obligation to delete collected PII within 7 days -->
  <ob:Obligation>
    <ob:TriggersSet>
      <ob:TriggerAtTime>
        <ob:Start>
          <ob:StartNow/>
        </ob:Start>
        <ob:MaxDelay elementId="myIdA">
          <ob:Duration>
            P0Y0M7DT0H0M0S
          </ob:Duration>
        </ob:MaxDelay>
      </ob:TriggerAtTime>
```

```
    </ob:TriggersSet>
    <ob:ActionDeletePersonalData/>
  </ob:Obligation>

</ob:ObligationsSet>
```

8.2.3.1.2 Policy-side obligations set

```
<?xml version="1.0" encoding="utf-8" ?>

<!-- This is the obligation part of a PPL policy -->

<ob:ObligationsSet xmlns:ob="http://www.primelife.eu/ppl/obligation"
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xmlns:ppl="http://www.primelife.eu/ppl"
                   xsi:schemaLocation="http://www.primelife.eu/PPL/obligation
file:./PrimeLifeObligation.xsd">

  <!-- Obligation to log within 5 minutes  use of PII for purpose "contact"
                         within 15 minutes use of PII for purpose "delivery"
                         within 30 seconds use of PII for purpose "pseudo-analysis" -->
  <ob:Obligation>
    <ob:TriggersSet>
      <ob:TriggerPersonalDataAccessedForPurpose
        xmlns="http://www.primelife.eu/PPL/obligation">
        <ppl:Purpose>http://www.w3.org/2002/01/P3Pv1/contact</ppl:Purpose>
        <ob:MaxDelay>
          <ob:Duration>
            P0Y0M0DT0H5M0S
          </ob:Duration>
        </ob:MaxDelay>
      </ob:TriggerPersonalDataAccessedForPurpose>
      <ob:TriggerPersonalDataAccessedForPurpose
        xmlns="http://www.primelife.eu/PPL/obligation">
        <ppl:Purpose>http://www.w3.org/2006/01/P3Pv11/delivery</ppl:Purpose>
        <ob:MaxDelay>
          <ob:Duration>
            P0Y0M0DT0H15M0S
          </ob:Duration>
        </ob:MaxDelay>
      </ob:TriggerPersonalDataAccessedForPurpose>
      <ob:TriggerPersonalDataAccessedForPurpose
        xmlns="http://www.primelife.eu/PPL/obligation">
        <ppl:Purpose>http://www.w3.org/2002/01/P3Pv1/pseudo-analysis</ppl:Purpose>
        <ob:MaxDelay>
          <ob:Duration>
            P0Y0M0DT0H0M30S
          </ob:Duration>
        </ob:MaxDelay>
      </ob:TriggerPersonalDataAccessedForPurpose>
    </ob:TriggersSet>
    <ob:ActionLog/>
  </ob:Obligation>

  <!-- Obligation to delete collected PII within 5 days -->
  <ob:Obligation>
    <ob:TriggersSet>
      <ob:TriggerAtTime>
        <ob:Start>
          <ob:StartNow/>
        </ob:Start>
        <ob:MaxDelay elementId="myId1">
          <ob:Duration>
            P0Y0M5DT0H0M0S
          </ob:Duration>
        </ob:MaxDelay>
      </ob:TriggerAtTime>
    </ob:TriggersSet>
    <ob:ActionDeletePersonalData/>
  </ob:Obligation>

</ob:ObligationsSet>
```

8.2.3.1.3 Result (Sticky Policy)

```
<?xml version="1.0"?>
<ObligationsSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.primelife.eu/ppl/obligation/mismatch">
  <ObligationsSet xmlns="http://www.primelife.eu/ppl/obligation">
    <Obligation>
      <TriggersSet>
        <TriggerPersonalDataAccessedForPurpose>
          <Purpose xmlns="http://www.primelife.eu/ppl">
             http://www.w3.org/2002/01/P3Pv1/contact</Purpose>
          <MaxDelay>
            <Duration>P0Y0M0DT0H5M0S</Duration>
          </MaxDelay>
        </TriggerPersonalDataAccessedForPurpose>
      </TriggersSet>
      <ActionLog />
    </Obligation>
    <Obligation>
      <TriggersSet>
        <TriggerPersonalDataAccessedForPurpose>
          <Purpose xmlns="http://www.primelife.eu/ppl">
             http://www.w3.org/2002/01/P3Pv1/pseudo-analysis</Purpose>
          <MaxDelay>
            <Duration>P0Y0M0DT0H0M30S</Duration>
          </MaxDelay>
        </TriggerPersonalDataAccessedForPurpose>
      </TriggersSet>
      <ActionLog />
    </Obligation>
    <Obligation>
      <TriggersSet>
        <TriggerAtTime>
          <Start>
            <DateAndTime>2010-08-19T11:53:00.8824003+02:00</DateAndTime>
          </Start>
          <MaxDelay>
            <Duration>P0Y0M5DT0H0M0S</Duration>
          </MaxDelay>
        </TriggerAtTime>
      </TriggersSet>
      <ActionDeletePersonalData />
    </Obligation>
  </ObligationsSet>
</ObligationsSet>
```

8.2.3.2 Example 2 (mismatching)

Files of this example, namely “preference.xml”, “policy.xml” and “matchresults.xml”, are installed to “%PL_OBL_Samples%\DocExample 2 - Mismatching Obligations”

8.2.3.2.1 Preference-side obligations set

Same as in 7.1.1.1..

8.2.3.2.2 Policy-side obligations set

First obligation: same as in 7.1.1.2.
Second obligation:

```
  <!-- Obligation to delete collected PII within 10 days -->
  <ob:Obligation>
    <ob:TriggersSet>
      <ob:TriggerAtTime>
        <ob:Start>
          <ob:StartNow/>
```

```
        </ob:Start>
        <ob:MaxDelay elementId="myId1">
          <ob:Duration>
            P0Y0M10DT0H0M0S
          </ob:Duration>
        </ob:MaxDelay>
      </ob:TriggerAtTime>
    </ob:TriggersSet>
    <ob:ActionDeletePersonalData/>
  </ob:Obligation>
```

8.2.3.2.3 Result (Sticky Policy)

```
<?xml version="1.0"?>
<ObligationsSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" matching="false"
xmlns="http://www.primelife.eu/ppl/obligation/mismatch">
  <ObligationsSet xmlns="http://www.primelife.eu/ppl/obligation">
    <Obligation>
      <TriggersSet>
        <TriggerPersonalDataAccessedForPurpose>
          <Purpose xmlns="http://www.primelife.eu/ppl">
             http://www.w3.org/2002/01/P3Pv1/contact</Purpose>
          <MaxDelay>
            <Duration>P0Y0M0DT0H5M0S</Duration>
          </MaxDelay>
        </TriggerPersonalDataAccessedForPurpose>
      </TriggersSet>
      <ActionLog />
    </Obligation>
    <Obligation>
      <TriggersSet>
        <TriggerPersonalDataAccessedForPurpose>
          <Purpose xmlns="http://www.primelife.eu/ppl">
             http://www.w3.org/2002/01/P3Pv1/pseudo-analysis</Purpose>
          <MaxDelay>
            <Duration>P0Y0M0DT0H0M30S</Duration>
          </MaxDelay>
        </TriggerPersonalDataAccessedForPurpose>
      </TriggersSet>
      <ActionLog />
    </Obligation>
    <Obligation matching="false">
      <TriggersSet matching="false">
        <TriggerAtTime matching="false">
          <Start>
            <DateAndTime>2010-08-19T11:59:25.1098192+02:00</DateAndTime>
          </Start>
          <MaxDelay mismatchId="mismatchId_4" matching="false">
            <Duration>P0Y0M10DT0H0M0S</Duration>
          </MaxDelay>
        </TriggerAtTime>
      </TriggersSet>
      <ActionDeletePersonalData />
    </Obligation>
  </ObligationsSet>
  <Mismatches>
    <Mismatch mismatchId="mismatchId_4">
      <Similarity>0.5714285714285714</Similarity>
      <Preference elementId="myIdA">
        <Duration xmlns="http://www.primelife.eu/ppl/obligation">
          <Duration>P0Y0M7DT0H0M0S</Duration>
        </Duration>
      </Preference>
      <Policy elementId="myId1">
        <Duration xmlns="http://www.primelife.eu/ppl/obligation">
          <Duration>P0Y0M10DT0H0M0S</Duration>
        </Duration>
      </Policy>
    </Mismatch>
  </Mismatches>
</ObligationsSet>
```

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).

[ACP11] C. Ardagna, S. De Capitani di Vimercati, S. Paraboschi, E. Pedrini, P. Samarati, and M. Verdicchio, "Expressive and Deployable Access Control in Open Web Service Applications," in IEEE Transactions on Service Computing (TSC), 2011

[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/jhustice_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), Washington, 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

[D4.3.2] UI Prototypes: Policy Administration and Presentation - Version 2 http://www.primelife.eu/images/stories/deliverables/d4.3.2-policy_administration_and_presentation_ui_prototypes_v2-public.pdf

[D5.2.3] D5.2.3 Final research report on research on next generation policies

[D5.3.2] Second Release of the PrimeLife Policy Engine http://www.primelife.eu/images/stories/deliverables/d5.3.2-second_release_of_the_policy_engine-public.pdf

[D5.3.3] D5.3.3 Final release of the policy engine

[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.

[HER] Holistic Enterprise-Ready Application Security http://www.herasaf.org/heras-af-xacml.html

[HIB] Hibernate. http://www.hibernate.org/.

[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

[HYP] HyperJAXB. http://java.net/projects/hyperjaxb3

[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.

[JAXB] JAXB. http://jaxb.java.net/.

[JSON] http://www.json.org/

[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/

[TOP] TopLink. http://www.oracle.com/technology/products/ias/toplink/.

[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.