Second Release of the Policy Engine

Editors: Slim Trabelsi, (SAP)

Reviewer: Tobias Pulls, (KAU)

Identifier: D5.3.2
Type: Deliverable
Class: Public
Date: September 30, 2010

Abstract

The document presents the implementation details of the second version of the PrimeLife policy engine (called PPL engine). This deliverable describes the specification of the PPL language, the architecture of the PPL engine and the testing demo provided with the engine.

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.

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

This deliverable has been jointly authored by multiple PrimeLife partner organisations. The following list presents the contributors for the individual parts of this deliverable.

Chapter	**Author(s)**
Introduction	SAP, EMIC
Chapter 1	SAP
Chapter 2	SAP
Chapter 3	SAP
Chapter 4	SAP
Chapter 5	SAP
Chapter 6	SAP
Chapter 7	SAP
Conclusion	SAP

Contents

Chapter 1

Introduction

This document presents the implementation and integration results obtained in the context of Work Package 5.3. The main goal of the deliverable is to develop the second release of the PPL (PrimeLife Policy Language) policy engine. The concept and the specifications of this language were already defined in the internal deliverable H5.3.2 [1]. Part of these specifications are in this document, but it is important to take a look to the H5.3.2 specification document in order to understand the concepts presented here.. Since the PPL language is specified as an extension of the XACML (eXtensible Access Control Markup Language) [2] language, the PPL engine is designed to run together with the HERAS-XACML engine [3] (that only handles XACML access control rules). The architecture chosen for the deployment of the PPL engine is symmetric because data owners (that we call data subjects) and data collectors (that we call data controllers) have similar requirements: deciding whether a given personal information (resp. collected data) can be shared with a data controller (resp. third party); handling obligations associated with data; storing data and associated preferences (resp. sticky policies). Using the same architecture everywhere to handle scenarios where one party can have multiple roles (e.g. collecting data and next disclosing it to third parties). The PPL engine executes multiple tasks in an automated way like: enforcing access control policies, generating and verifying cryptographic proofs related to credential requirements, matching between data handling preferences and data handling policies, generating and enforcing sticky policies, checking authorization, controlling the downstream usage of data, handling obligations …

Besides the documentation related to the implementation engine, we propose in chapter 6 a description of the web-based demo illustrating the scenario of a subscription to the PrimeLife social network Clique[1]. This demo is used as a working prototype of the PPL engine used to test the underlying concepts and principles defined in the specification part. This demo shows how a user that wants to create an account on a social network website will control the

[1] http://www.primelife.eu/results/opensource/40-clique

disclosure of her personal data as required by the server. At any point of time when executing the demo we can visualize how the data is treated.

Terminology

Access Control

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

Credentials

The 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. The authenticity of the attribute values can be verified using concrete mechanisms (cryptographic or other). 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 PII

Personally Identifiable Information or PII 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 [4].

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.

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 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 her data and to require the Controller to rectify without delay any inaccurate or incomplete personal data. The Data Subject has the right to require the Controller to erase data if the processing is unlawful.

Data Subject's privacy preferences

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

Sticky privacy policy

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

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

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

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

User Agent

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

Chapter 2

PPL Specification

2.1 Introduction

In this section we describe briefly the specification aspects behind the software deliverable. This section is a summary (with some updates) about what was proposed in the internal delivery H5.3.2 [1]. This summary is proposed to help the understanding of the new concepts and functionalities proposed in the in this deliverable.

2.2 Collaboration diagram

PPL (Privacy Policy Language), the new proposed policy language considers the scenario depicted in Figure 1, 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. The PPL 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.

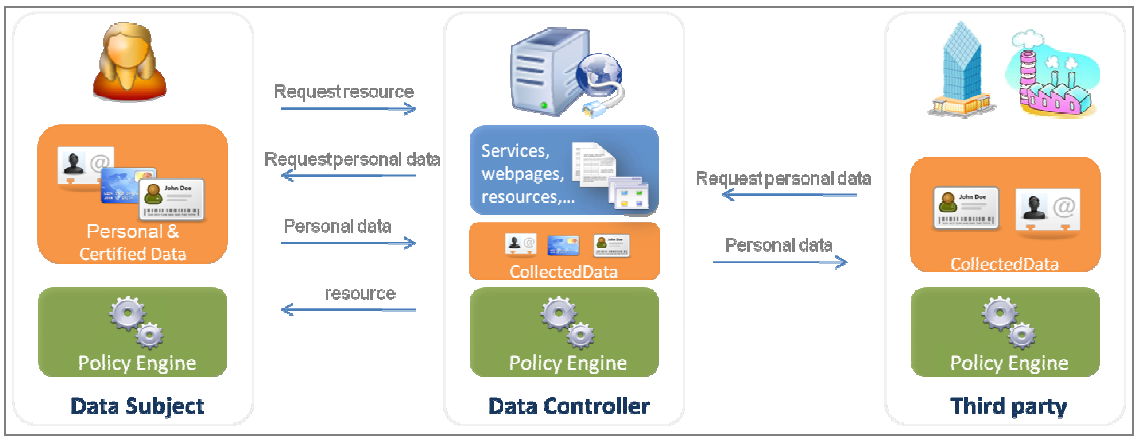


Figure 1: Collaboration diagram

The PPL 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.

2.3 Functional specification

2.3.1 PPL language specification

As described above in a simplified way, the new privacy policy language should define different new requirements to strengthen the users' privacy. The language handles access control and data usage at the same time. It provides a new obligation handling mechanism taking into account temporal constraints, pre-obligations, conditional obligations, and repeating obligations together with a down-stream usage authorization. The downstream usage authorization system definines the access control rules under which personal information collected by an entity can be forwarded to a third party. Moreover, the language should support privacy enhanced credential-based access control. The credential-based aspect of the language introduces credentials as a common abstraction for the various authentication mechanisms.

The privacy enhancements allow attaching two-sided data handling policies to privacy-sensitive resources, specified by means of a concrete set of obligations and set of authorizations.

Moreover, the typical interaction sequence is changed so that the Data Subject is informed about the applicable policies and if his preferences agree with the data handling policy by an automated matching mechanism, before transmitting his personal information.

2.3.1.1 Credential

The credential-based aspect of the new language introduces credentials as a common abstraction for the various authentication mechanisms. The privacy enhancements allow attaching 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.

The requirements language is oriented 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 should be designed to be technology-agnostic in the sense that it addresses general credential concepts without targeting one technology in particular.

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.

2.3.1.2 Obligation

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

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

We identify and define four main challenges related to obligations.

- Service providers must avoid committing to obligations that cannot be enforced. For instance, it is not simple to delete data when backup copies do exist. Tools to detect inconsistencies are necessary.

- 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.

- Services need a way to communicate acceptable obligations to users, to link obligations and personal data, and to enforce obligations.

- Finally, users need a way to evaluate the trustworthiness of service providers, i.e. know whether the obligation will indeed be enforced. This could be achieved by assuming that misbehavior impacts reputation, by audit and certification mechanisms, and/or by relying on trusted computing.

The first challenge is focused by providing a mechanism to enforce obligations, and the second and the third aspect by providing mechanisms to match obligations. The fourth challenge is addressed by assuming a simple trust model: audit and reputation mechanisms.

Chapter 3

Architecture

In this chapter we will present the design phase of the PPL engine. We present the architecture of the PPL system by defining a high level and a detailed architecture.

3.1 High level architecture

As presented previously in the collaboration diagram (Figure 1), the high level architecture presents an abstract overview of the PPL architecture and the interaction between the different entities; DS, DC and third party.

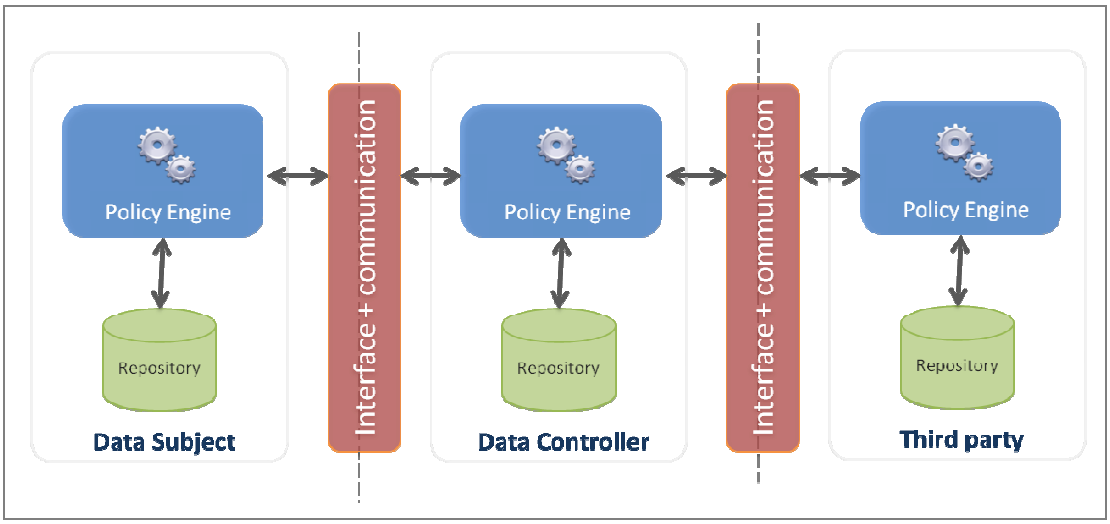


Figure 2: High level architecture

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

Repository: represent the PII and policy repositories. It is a database containing data owned by the Data Subject. This data could be composed of personal data, credentials, certificated, and other information that should be used during the interaction with the Data Controller application. 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.

3.1.2 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 her interaction with the Data Controller. This data represents PIIs, 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.

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.

3.1.3 Third party

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.

3.2 Detailed architecture

The entire architecture can be represented by three layers: the first one presents the user interface layer. The second, business layer, represents the core of the PPL Engine. The last layer represents the persistence layer that is in charge of data persistence.

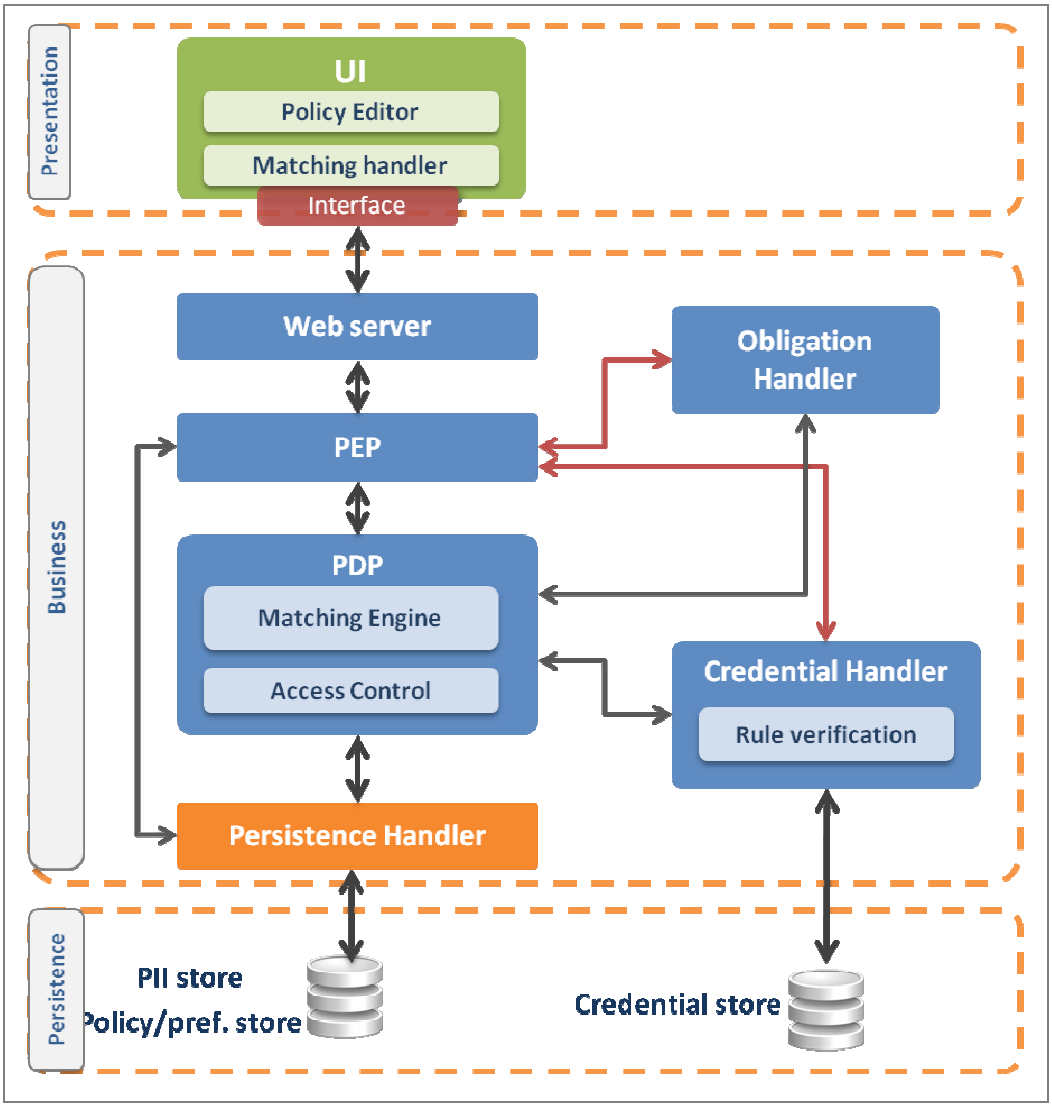


Figure 3: PPL architecture

3.2.1 Presentation layer

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

- The 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 or preference, the information involved during a transaction between the different entities.This component is not yet integrated to the current version of the demonstrator but should be part of the next release. Actually policies are written manually.

- The matching handler: displays to the user the result of the matching. In the case of a mismatch a set of tools are provided that allows the data subject to manage, or make an informed decision about, the mismatch.

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

3.2.2 Business layer

The business layer which represents the core of the PPL engine is composed of four main elements that implements the new concepts introduced within PrimeLife. These components are:

- **Policy Enforcement point (PEP):** this component formats and then dispatches the data to the corresponding component 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 one resource, this data or resource is collected, formatted and sent to the receiver through the PEP.

- **Policy Decision Point (PDP):** it is the core of the PPL engine where all decisions are made. It can be broken down into two subcomponents:
  - *Matching engine:* this functionality matches between the preferences of the Data Subject and the privacy policy of the Data Controller. The matching is done to verify if the intentions of the data controller in terms of PII usage are compliant with the data subject's preferences.
  - *Access control engine:* this component is in charge of the application of the access control rules related to the local resources. It analyses the resource query, checks the access control policy of the requested resource and decides whether or not the requester satisfies the rules.

- **Credential handler:** one of the new features introduced in PPL is the support of the credential based access control. This feature is implemented by the credential handler that manages the collection of credentials held by an entity, selects the appropriate credentials in order to generate a cryptographic proof and verifies the cryptographic proofs of the claims received from external entities. The credential handler component contains the subcomponent Rule Verification; the PPL policy contains a description of the credential requirements (for access control), the Rule Verification component evaluates whether the claim provided by a user that wants to access a resource satisfies the credential based access control rule.

- **Obligation handler:** it is responsible for handling the obligations that should be satisfied by the Data Controller/third party. This engine executes two main tasks; it setups the triggers related to the actions required by the privacy preferences of the Data Subject, and executes the actions specified by the data subject whenever it is required.

The other components of the architecture play a secondary role in the concept introduced by the PPL engine:

- **Web server:** an embedded web server that represents the entry point of the core of the PPL Engine. It can be seen as an interface to the PEP.

- **Persistence handler:** can be described as an interface between the business layer and persistence layer. It encapsulates access to the storage medium business objects. It makes transparent to the business layer location and storage model of the data it manipulates. In general, this layer is supported by a Persistence Framework. The defined objects in this layer are generally DAOs (Data Access Object). The persistence handler provide management functions to handle the DAOs known as CRUD (Create, Retrieve, Update, Delete) methods. The persistence handler provides the functions to manage the PIIs and the policies in the different databases.

3.2.3 Persistence layer

The persistence layer is represented by the:

- PII/Policy store: this database (or other way of persistence, such as LDAP, etc) contains all the information related to the PIIs and their related policies.

- Credential store: this database contains all the credentials and certified information held by an entity. Access to this store is exclusively dedicated to the credential handler component.

Chapter 4

The Language

4.1 Language Structure

The PPL language extends XACML 2.0 with a number of privacy-enhancing and credential based features. The PPL 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;
- by the Data Subject to specify how she wants this implicit information to be treated.

For that, we maintain the overall structure of the XACML language and we introduce a number of new elements to support the advanced features that our language aims to offer.

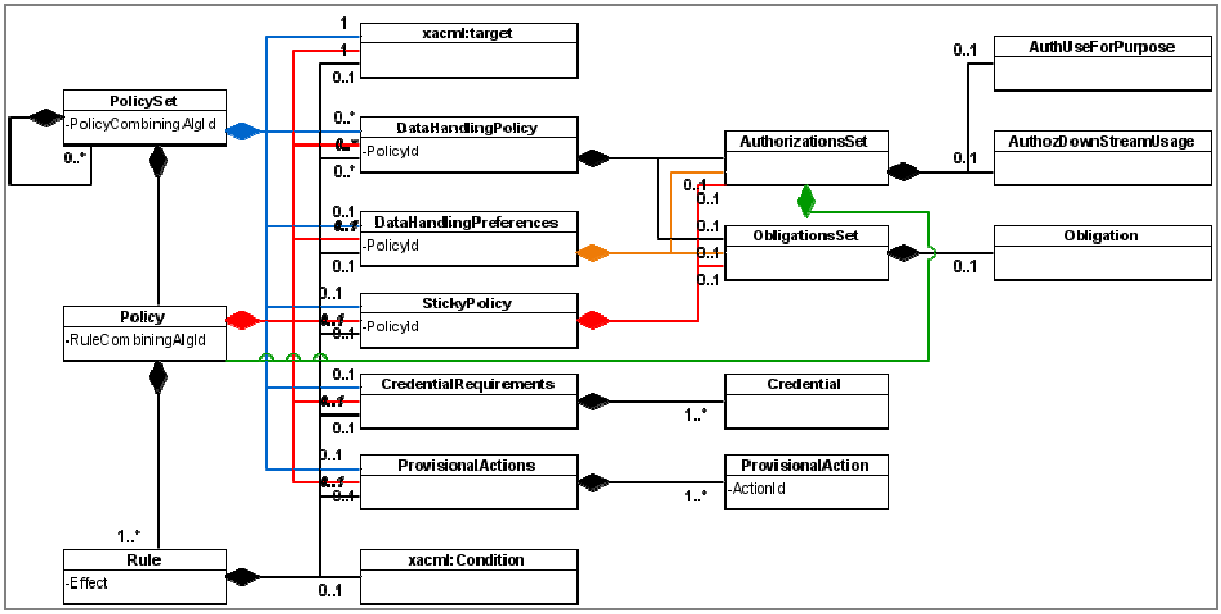


Figure 4: Domain class diagram

PolicySets, Policy and Rules

As in XACML, the main elements of our language are *PolicySet*, *Policy* and *Rules*.

Each *Rule* element 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 *Policy*. When a *Policy* is evaluated, the rule combining algorithm[2] of the policy (as stated in an XML attribute of the *Policy*) defines how the effects of the applicable rules are combined to determine the effect of the *Policy*. *Policies*, on their turn, are grouped together in *PolicySet*; the effect of a *PolicySet* is determined by the effects of the contained *Policies* and the stated policy combining algorithm. Finally, different *PolicySet* can be further grouped together in parent *PolicySet*.

The *PolicySet*, *Policy* and *Rule* elements are composed by different elements:

- a *Target* (plain *XACML:Target*), which describes the resource, the subject, and the environment variables for which this *PolicySet*, *Policy* or *Rule* are applicable;
- *CredentialRequirements*, describing the credentials that need to be presented in order to be granted access to the resource; this element was not defined in XACML
- *ProvisionalActions*, describing which actions (e.g., revealing attributes or signing statements) have to be performed by the requestor in order to be granted access; this element was not defined in XACML
- *XACML:Condition*, specifying further restrictions on the applicability of the rule beyond those specified in the target and the credential requirements;

[2] Rule combining algorithms provides the final authorization decision by combining the effects of all the rules in a policy, as: Deny-overrides: If one of the rules evaluates to Deny, then the final authorization decision is Deny. Permit-overrides: If any rule evaluates to Permit, then the final authorization decision is also Permit.

- *DataHandlingPolicy*, describing how the information that needs to be revealed to satisfy this rule will be treated afterwards; this element was not defined in XACML

- *DataHandlingPreferences*, describing how the information contained in the resource that is protected by this rule has to be treated; this element was not defined in XACML.

Credential Requirements

The policy language that we present is geared towards enabling technology-independent, user-centric and privacy-friendly access control on the basis of certified credentials. 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. As credentials are not directly supported in the traditional policy languages, we extended the XACML *Rule* element such that credentials are the basic unit for reasoning about access control.

Each *Rule* can contain a *CredentialRequirements* element to specify the credentials that have to be presented in order to satisfy the *Rule*. The *CredentialRequirements* element contains a separate *Credential* element for each credential that needs to be presented. Each *Credential* element contains a unique identifier *CredentialId* that is used to refer to the credential from elsewhere in the *Rule*.

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

Provisional Actions

The *ProvisionalActions* element is used to specify the provisional actions that a requestor must perform before being granted access to the resource. Currently supported actions include revealing of attributes (to the Data Controller or to a Third Party) optionally under handling policy and credential proof, signing a statement, and so-called “spending” of credentials, which allows to put restrictions on the number of times that the same credential is used to obtain access. Each action is described in a *ProvisionalAction* element; the language has to be extensible so that new types of *ProvisionalActions* can easily be added later on, and can refer to *DataHandlingPolicy* and *Credential* elements.

Data Handling Policies

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, for example, therefore contains an optional reference to the applicable *DataHandlingPolicy*.

Each *Rule*, *Policy*, or *PolicySet* element can contain a number of *DataHandlingPolicy*. A *DataHandlingPolicy* can be referred to from anywhere in the rule by its unique *PolicyId* identifier.

A *DataHandlingPolicy* consists of a set of *Authorizations*, that the Data Controller wants to obtain on the collected information, and a set of *Obligations*, that he promises to adhere to.

Before the Data Subject reveals her information, these *Authorization* and *Obligation* are matched against the Data Subject’s *DataHandlingPreference* to see whether a matching StickyPolicy can be agreed upon.

Data Handling Preferences

The data *DataHandlingPreference* how the information obtained from the resource protected by the Data Subject is to be treated after access is granted. The preferences are expressed by means of a set of *Authorizations* and *Obligations*, just like *DataHandlingPolicy*. When access to the resource is requested, the *DataHandlingPreference* element has to be matched against a proposed *DataHandlingPolicy* to derive the applicable *StickyPolicy* – if a match can be found.
An important difference between *DataHandlingPreference* and *DataHandlingPolicy* is the resource that they pertain to: *DataHandlingPreference* always describe how the resource protected by the Data Subject itself has to be treated after being collected.,While *DataHandlingPolicy* pertain to inform that a requester will have to reveal information in order to be granted access to the resource.
The main use of *DataHandlingPreference* is for a Data Subject to specify how she wants her PII 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* the Data Controller will have to adhere to.
Optionally, if the *DataHandlingPreference* contain a *AuthorizationDownstreamUsage*, this can be interpreted by, optionally, including a Policy specifying the downstream access control policy, i.e., the access control policy that has to be enforced on the downstream data controllers.

Sticky Policies
The *StickyPolicy* element is associated to a resource, meaning the agreed-upon sets of granted authorizations and promised obligations with respect to a resource. The *StickyPolicy* is usually the result of an automated matching procedure between the Data Subject's *DataHandlingPreference* and the Data Controller's *DataHandlingPolicy*.
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 *Authorization* or *Obligation* concerning her own PII, so her policy will not contain a *StickyPolicy* element. The Data Controller, on the other hand, will describe in the *StickyPolicy* the *Authorization* and Obligation that she, herself, has to adhere to, while the *DataHandlingPreferences* contain those that a Downstream Data Controller has to adhere to. Usually, the Downstream Data Controller (Third Party) will be subject to the same or stronger restrictions than the Data Controller herself, meaning that the policy specified in the *DataHandlingPreferences* will usually be at most as permissive as the policy specified in the *StickyPolicy*.

Obligations
The data handling policies, preferences, and sticky policies contain a set of *Obligations*. An *Obligation* is defined as: "A promise made by a Data Controller to a Data Subject in relation to the handling of her PII. The Data Controller is expected to fulfill the promise by executing and/or preventing a specific action after a particular event, e.g. time, and optionally under certain conditions".
An obligation is often defined as Event-Condition-Action:

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

For facilitating the comparison of *Obligations*, we consider *Triggers* as events filtered by conditions. In other words, we replace the notions of events and conditions by *Trigger*. The *Triggers* are events that are considered by an obligation and can be seen as the set of events

that result in actions. Additionally, in order to simplify obligations management, we specify a validity period for each obligation: Do *Action* when *Trigger* (from *Start* to *End*)
For example, we can define "Do {Notify User} when {User's personal data is read} (from .. to ..)". Obligations are thus defined as a set of *Triggers*, an *Action*.
Obligations in PPL language, consists of a set of *Obligation* elements. This latter defines a set of *Triggers* describing the events that trigger the obligation, and the related *Action* that has to be performed.
The reason that we didn't choose to use the standard XACML *Obligations* element to specify the obligations that we embed in the data handling policies or preferences, is that XACML Obligations can only be used to specify obligations that the PEP has to adhere to when an access request occurs for the resource that is protected by this rule. This cannot be used for our data handling policies, since the latter pertain to information that the requestor will have to reveal in order to obtain access, rather than to the resource being protected. It cannot be used for our data handling preferences either, since the latter specify obligations that the recipient of the resource has to adhere to, rather than the PEP that is protecting access to the resource. Meaning, by populating the *XACML:Obligations* element that protects her personal data, a Data Subject would impose obligations that she herself has to adhere to each time a Data Controller requests access to the personal data, rather than imposing obligations on the Data Controller.
The only use that we could have had for the *XACML:Obligations* element is to store and enforce those obligations, that the Data Controller committed to in an agreed-upon sticky policy that are triggered by access requests. Since obligations triggered by access requests are only a small subclass of the obligations that we consider here, we chose to leave the storage and enforcement of obligations entirely up to the obligation Engine, and let the PEP simply signal the obligation Engine each time an access request occurs.

Authorizations
DataHandlingPolicy, *DataHandlingPreference*, and *StickyPolicies* 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 (to forward the information to third parties), and we describe how these authorizations can be efficiently matched via a general strategy.

- **Authorization Purposes:** 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 ontologies.

- **Authorization for downstream usage:** The second concrete authorization type that we define is the authorization to forward the information to third parties, so-called downstream data controllers. Optionally, this authorization enables the data subject to specify the access control 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.

4.2 Defining matching rules

This section describes the matching rules, by means of how a Data Controller's proposed data handling policy (DHPolicy) is matched with a Data Subject's required data handling preferences (DHPreferences). 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 a 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 DHPolicy of the downstream Data Controller with committed DHPreferences attached to a piece of personal data.
Matching service's privacy policy 'PS' with user's privacy preference 'PU' is necessary to inform the user under which condition his PII should be used. 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:

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

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.
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 authorizations and the list of obligation are more restrictive. The 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.
The matching function for obligations is quite different:

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

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

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

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

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

The matching function for a list of triggers is:

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

In other words, for a given obligation, all triggers in the preferences must be in the policy, but the policy can specify other triggers.

Chapter 5

Use Case

To explain in a more easy way how the PPL engine works, we present a scenario that describes a subscription to an online shopping website. We present in Figure 5 the different information exchanged between the different parties using the entities format presented in the domain class diagram above.
Alice is a privacy-aware user who regularly shops online, but who is concerned about what happens to the data that she provides about herself. Before starting shopping online, she has to create an account at an online store store.example.com (step 1). In order to validate her subscription, the online store will need to collect some personal information, namely her (uncertified) e-mail address and her street address as stated on her identity card..

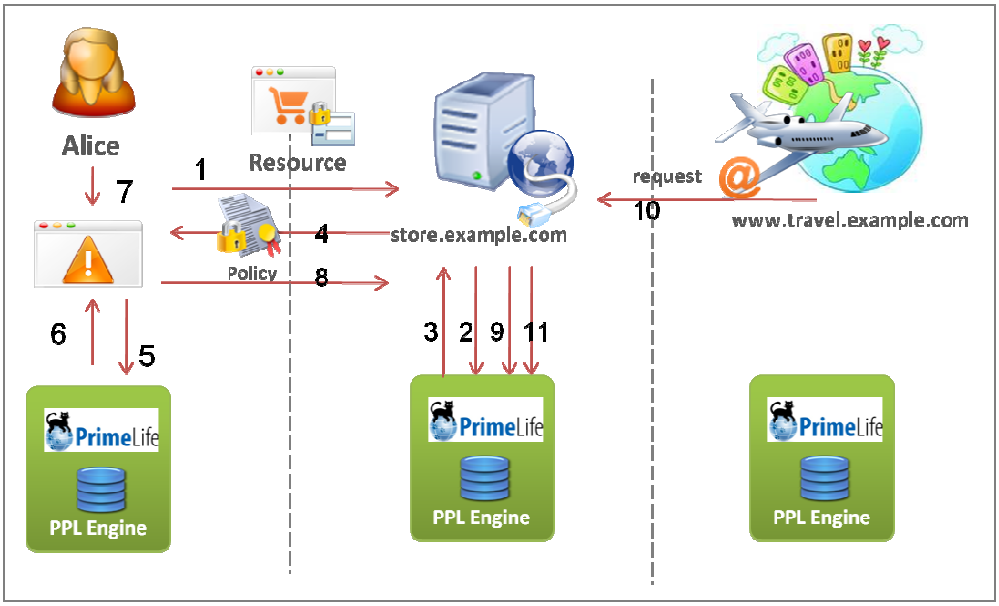


Figure 5: Domain sequence diagram

This information is contained in an access control policy (step 2 and 3). The online store will send this policy to Alice (sequence 4), together with the privacy policy related to the requested

personal data, and an assertion proving the authenticity of the web store and its privacy seals. The access control and data handling policy sent to Alice will contain these elements:

- Policy
  - Target (represents the assertions about the Data Controller):
    - Subject:- ID: store.example.com
      - EuroPriSe privacy seal: true
  - ProvisionalAction:
    - Reveal [e-mail] Under DataHandlingPolicy [DHPolicy1]
    - Reveal [id.address[3]] Under DataHandlingPolicy [DHPolicy1] and should be certified by a credential
    - Reveal [cc.number[4]] Under DataHandlingPolicy [DHPolicy2] and should be certified by a credential
    - Reveal [cc.exp] Under DataHandlingPolicy [DHPolicy2] and should be certified by a credential
  - DHPolicy1:
    - Use for purposes: Statistics, Administration, Marketing
    - Can pass the data to Downstream usage: yes
    - Obligation: Delete after 1 year
  - DHPolicy2:
    - Use for purpose: Payment
    - Obligation: Delete within 1 month
  - CredentialRequirements:
    - Credential id::NationalID by fgov.be
    - Credential cc::CreditCard by visa.com or amex.com
    - Condition: id.birthdate < today – 18Y and cc.exp > today and id.name = cc.name

After receiving this resource request, Alice's PPL Engine will process the policy by first checking if the required credentials can be provided (step 5 and 6). Suppose that her credential store contains one Belgian eID card, one Visa credit card and one American Express credit card, so that Alice has two possible ways of satisfying the policy: either by using her eID card and Visa card, or by using her eID card and American Express card. For both combinations, let's suppose that the proposed website privacy policy is matched with Alice's privacy preferences for the different requested attributes. Alice eID card (all attributes), and uncertified e-mail preferences contain these elements:

- Policy
  - Target (represent the access control):
    - Permit : any subject, any action

[3] Id.address : the address under the identity identification

[4] cc: means credit card. cc.number: means the credit card number

- DHPreference:
  - Use for purposes: Statistics, Administration, Marketing, Contact, Account
  - Can pass the data to Downstream usage: yes, with this restriction
    - Policy (Europrise privacy seal, any action)
    - Purpose: Contact, Marketing
    - Obligation: Delete after 3 months
  - Obligation: Delete after 1 year

Alice credit card preferences contain these elements:

- Policy
  - Target (represent the access control):
    - Permit : any subject, any action
  - DHPreference:
    - Use for purposes: Payment
    - Can pass the data to Downstream usage: no
    - Obligation: Delete within 7 days

Alice's matching engine generates two possible claims that she can reveal to obtain access, one using her Visa Card and one using her American Express card. She is notified however that the proposed retention time of 1 month for the credit card number is longer than the 7 days that she specified in her preferences. She chooses (step 7) to use her Visa card to authenticate and agrees to overrule her preferences regarding the retention time of the credit card number, resulting in the following claim to be transported to the online store:

- Policy
  - Target: Europrise privacy seal, any action
  - StickyPolicy1:
    - Purpose: Statistics, Administration, Marketing
    - Downstream usage: yes, with restriction - Policy (Europrise privacy seal, any action)
      - Purpose: Contact, Marketing
      - Obligation: Delete (3 months)
    - Obligation: Delete (1 year)
  - StickyPolicy2:
    - Purpose: Payment
    - Obligation: Delete within 1 month
  - Credentials:
    - Credential id::NationalID by fgov.be
    - Credential cc::CreditCard by visa.com or amex.com
    - Condition: id.birthdate < today – 18Y and cc.exp > today and id.name = cc.name
  - ProvisionalAction:
    - RevealUnderDHP (e-mail = "alice@example.com", StickyPolicy1)

- RevealUnderDHP (id.address = “Kerkstraat 1, 1000 Brussel”, StickyPolicy1)
- RevealUnderDHP (cc.number = “1234 005678 90”, StickyPolicy2)
- RevealUnderDHP (cc.expirationdate = “2011-02-28”, StickyPolicy2)

The online store receives Alice’s personal data and its sticky policy (step 8). First, the PPL Engine verifies by the credential handler Alice’s credentials (step 9), after that the data will be used to create Alice’s account. Then the PIIs are stored in a secure repository with a pointer to their respective sticky policy. Finally, the online store configures the actions and the triggers related to the obligations defined in the sticky policy.
The online travel agency (www.travel.example.com) decided to start an e-mail advertising campaign. In order to target a wide scope of persons, the www.travel.example.com admin asked his partner store.example.com (step 10) to provide him with valid e-mail addresses for marketing and statistics purposes. The request will contain a resource query for e-mail and a privacy policy:

- DHPolicy:
  - Purpose: Marketing, Statistics
  - Obligation: Delete within 2 months

The policy engine of store.example.com will match the privacy policy of travel.example.com with the sticky policy related to the e-mail of Alice (step 11), and will conclude that access must be refused, because the sticky policy does not allow to forward for the purpose of statistics (marketing alone would have been allowed though).
In general, the full matching occurs between travel.example.com’s policy and store.example.com’s preferences, then between travel.example.com policy and Alice sticky policy.

The shipping company shipping.example.com needs to get in touch with Alice to inform her of the tracking number of her package. The shipping company requests access to Alice’s email address, with the following privacy policy:

- DHPolicy:
  - Purpose: Contact
  - Obligation: Delete within 7 days

After the matching, a new sticky policy will be generated from the matching result:

- StickyPolicy:
  - Purpose: Contact
  - Obligation: Delete in 7 days

The shipping.example.com website receives the e-mail address of Alice with a sticky policy and configures the actions and the triggers related to the obligations, after storing the PII and the sticky policy in a secure manner.

Chapter 6

Demo

6.1 Getting Started With the Demo

In this section we explain how to install and execute the Demo proof of concept of the PLL engine. We decided to add a web interface in the scenario in order to make more user friendly the interaction with the engine. For this version of the engine, since the policy editor is not yet ready, the tester has to check manually the XML policies that are used and generated by the engine. The idea here is to deploy the PPL engine on the Data Controller and the Data Subject sides and execute a simple scenario of a registration to a website. The scenario is quite simple, but the idea here is to play with this tool by verifying the generated policies, sticky policies and claims. The user of this demo can modify the policies and the preferences and observe the behaviour of the engine. For some integration reasons the Credential Handler is not yet connected to the PPL engine and does not work for this Demo.

6.1.1 System Requirement

First of all copy the zip file , extract the archive on your local folder that we will call $root.

6.1.1.1 Database

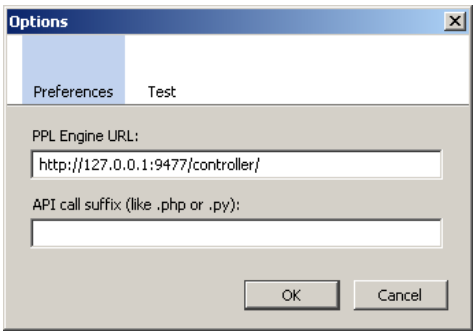
We used the MySQL 5.1.43 database (should be changed for the final deliverable). In order to manage this database we used the EasyPHP 5.3.2 package that includes the Apache HTTP web server, PHP, MySQL and phpMyAdmin. You can find the installation file for EasyPHP 5.3.2 in the $root */Install* folder of the deliverable or at this url: http://www.easyphp.org/download.php

6.1.1.2 .Net Framework 4

The obligation handler and the obligation matching engine are launched as Web Services running over the .Net framework. You can find the installation file for the .Net Framework 4 in the $root */Install* folder of the deliverable or at this url: http://www.microsoft.com/net/

6.1.1.3 Java VM

The executable for the PPL engine requires Java version 1.6.0_18. In order to install it you can use the JDK file in the $root */Install* folder called jdk-6u18-windows-i586.exe or go to this url: http://www.oracle.com/technetwork/java/javase/downloads/jdk6-jsp-136632.html



6.1.1.4 Mozilla Firefox 4 Beta 3

The demo is web-based and the user interface should be displayed in a browser. The user interface is a Firefox Plug-in that is compatible with the Firefox 4 Beta 3 version. For this reason this version of the browser must be installed and the setup file can be found in the $root */Install* folder or go to this url: http://www.mozilla.com/en-US/firefox/4.0b3/releasenotes/ . If an automatic update of the browser is asked, you have to refuse it.

6.1.1.5 PPL Firefox Plug-in

This plug-in is only available in the Install directory under the name PrimeLifePolicyUI.V4.2.xpi

In order to install the plug-in you just have to drag and drop this file into the Firefox browser. In order to configure it please do the following. This has to be done only once. Go to Firefox: Choose Tools >> PrimeLife Policy UI; in preference tab please set PPL Engine URL to http://127.0.0.1:9477/controller/. Verify thet the API call suffix box is empty

6.1.2 Installation of the Demo

6.1.2.1 Creating the persistence database

Launch your MySQLAdmin session from you're my SQL application (Figure 6 shows the admin interface of the EasyPHP application) . In your admin interface you have to import the SQL files $root //sql/*ppl.sql* and /sql/*ppl-dc.sql* if the admin interface does not have an import function, just edit the two files in Notepad, copy the contents and execute it as an SQL query in phpMyAdmin.

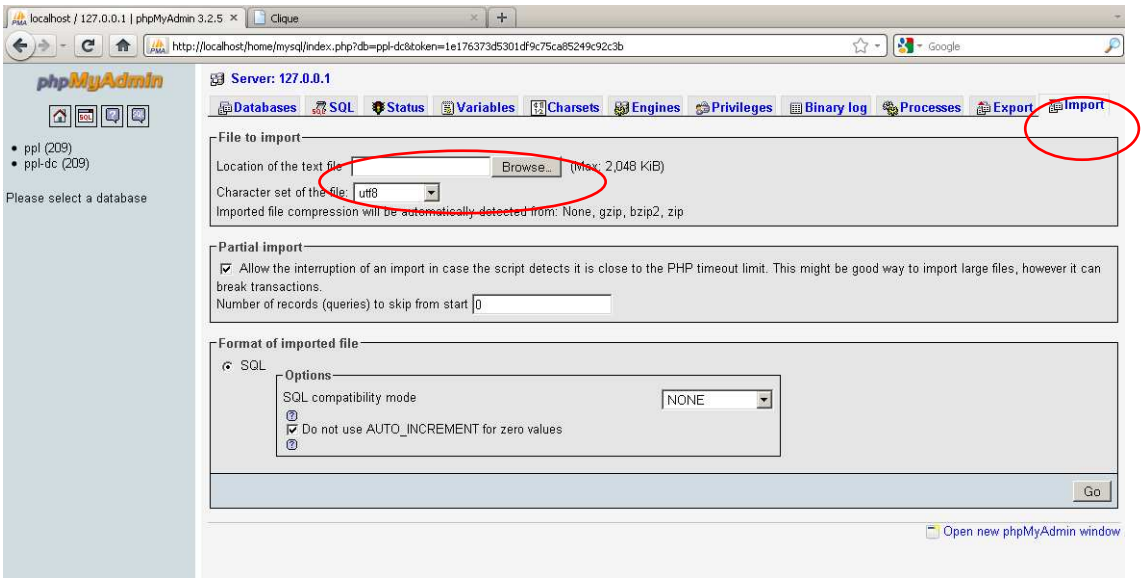


Figure 6: Creating the persistence database

Figure 6 shows how to create the databases using the Easyphp admin tool.

Figure 7 shows how to check if the persistence database is created and where to find the PII elements stored these.

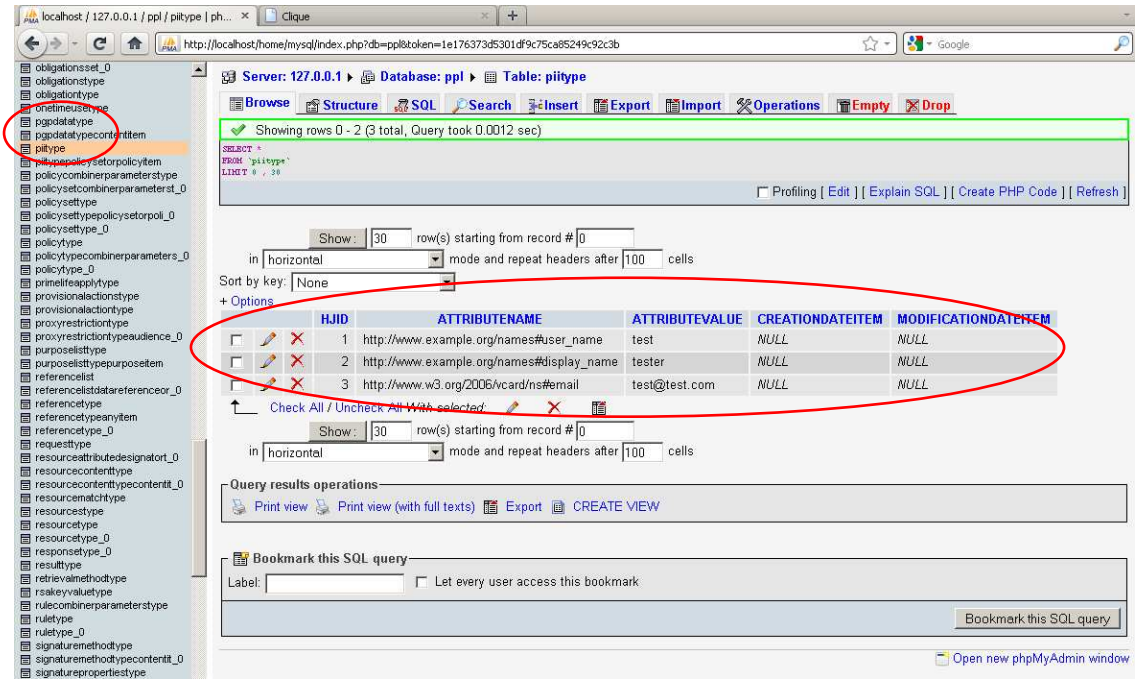


Figure 7: Checking the PIIs in the database

6.1.2.2 Starting the Demo

To run the demo, please launch:

1. Run the installer "/Obligation Matching Engine/PrimeLife Obligation Matching Engine.msi (this installer is only executable on Vista and Windows 7)
2. Check if the obligation engine is running on the application on your Start-> Programs menu of Windows
3. Lauch the Data Subject's Server ($root /ds/*ds.bat*)
4. Launch the Data Controller Server ($root /dc/*dc.bat*)

The Firefox on *http://localhost:8081/*

6.2 Scenario

The task for the demo is to register a new user. The process works as follows.

1. Please click on the *Register* Link in the Login box on the main page. (Figure 8)
2. The Register page (register.html) will open. After some time (at few seconds), the PrimeLife plugin overlay shows the result of the policy matching. (Figure 9). IMPORTANT: don not fill the form, everything will be done automatically.
3. In case there are mismatches accept the mismatch and click on send (Figure 10).
4. After some seconds, the success-page opens and you will find links to the sticky policies. (Figure 11). After displaying the result page, you can click on the links in order to see what are the PII collected by the DC and under which sticky policy they are handled.
5. To try again, you can use the link *Restart*.

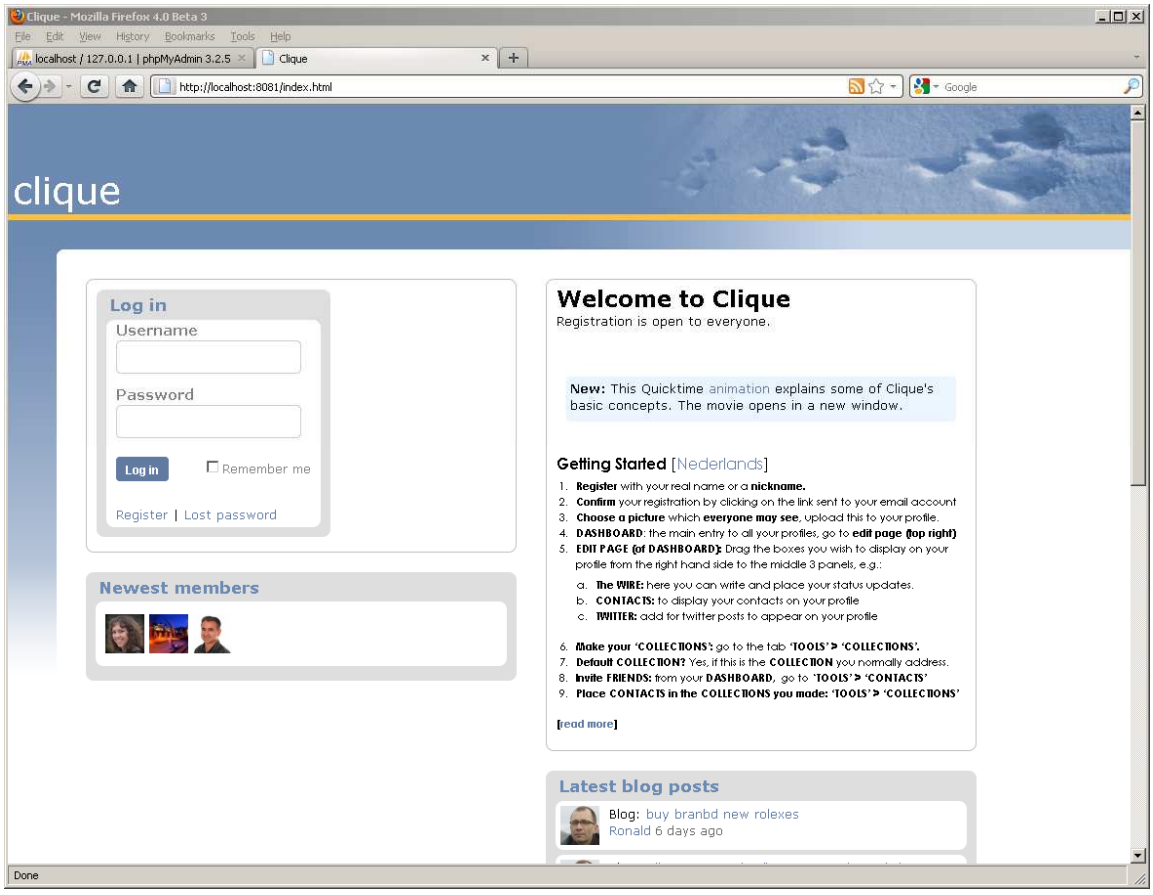


Figure 8: Clique page

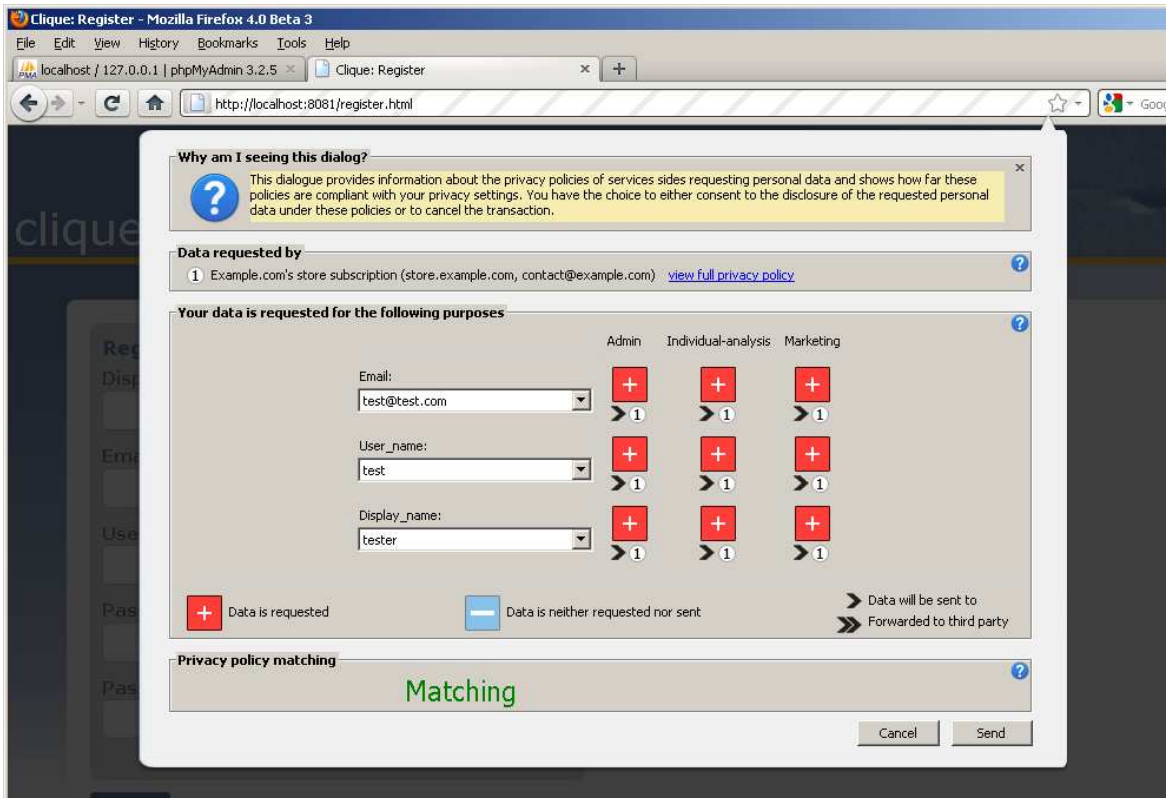


Figure 9: PrimeLife plug-in dialog

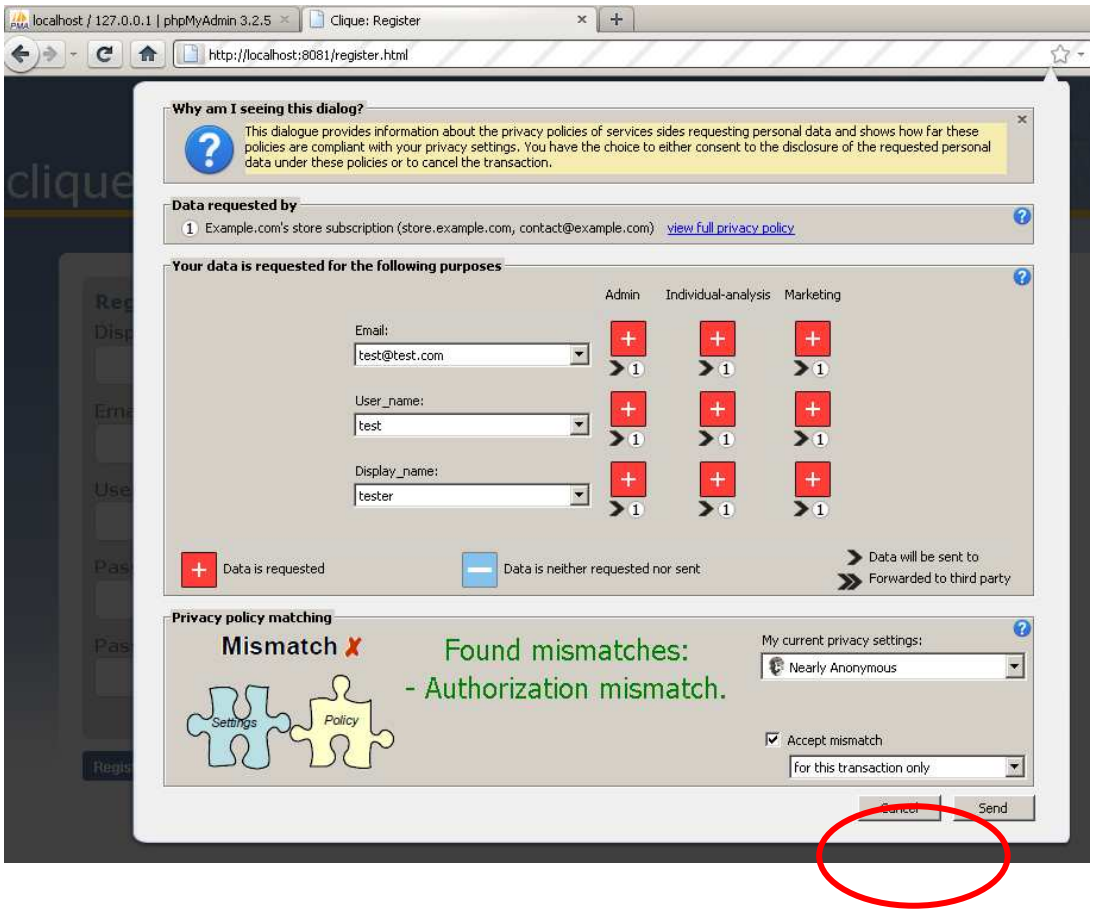


Figure 10: Mismatching example

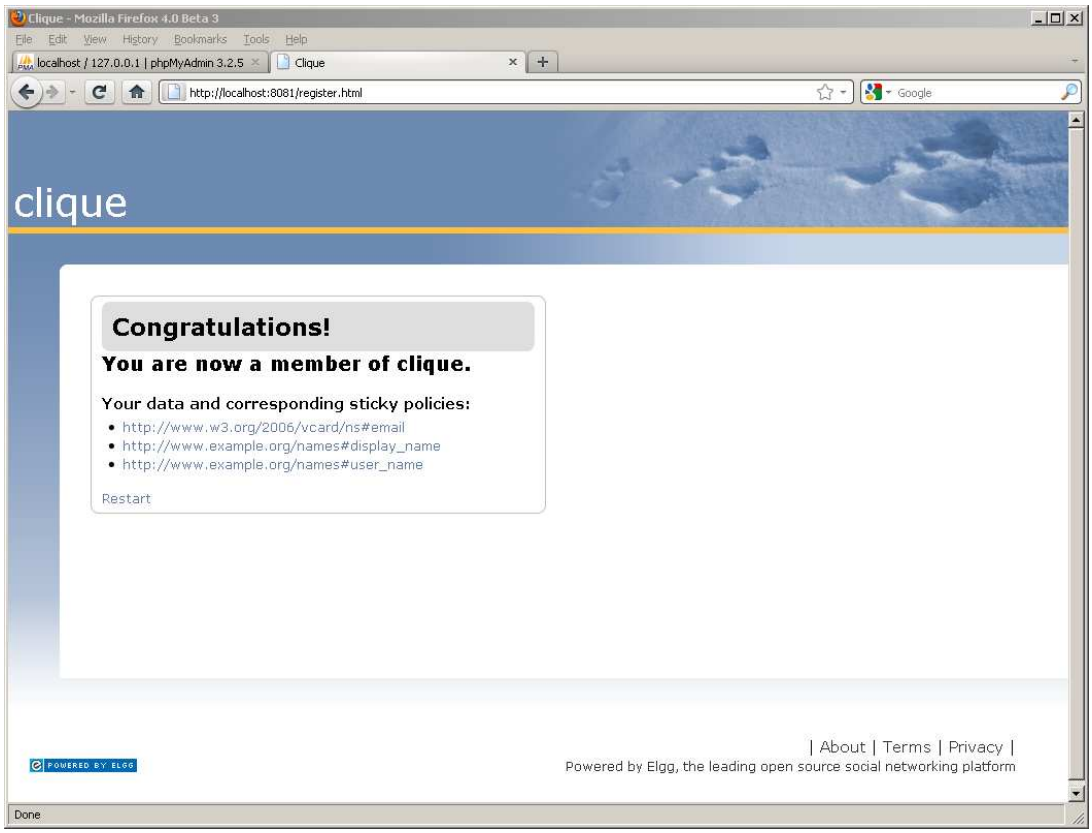


Figure 11: Display result for accessing resource

6.3 Playing and Changing the Configuration of the Demo

To change the policy preferences on the client side, you can edit the $root //pii-import/preferences/*DemoPii.xml* and the corresponding preference files. Initially we provide the files $root //pii-import/preferences/*AddressPreferences.xml* and $root //pii-import/preferences/*EmailPreferences.xml*. All these files can be found in the *resource* folder. These files must be imported to the client database (ppl) by running $root //pii-import/*pii-import.bat.*

> ***NOTE: To make the changes take effect, please re-launch the DS Server (/ds/ds.bat) after the import.***

Examples:

- Add a new PII to the DemoPii.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE piiscript [
    <!ELEMENT piis (pii*)>
    <!ELEMENT pii EMPTY>
    <!ATTLIST pii AttributeName CDATA #REQUIRED
        AttributeValue CDATA #REQUIRED
        Policy CDATA #REQUIRED>
]>
<piis>
    <pii AttributeName="http://www.example.org/names#user_name" AttributeValue="test" Policy="AddressPreferences.xml"></pii>
    <pii AttributeName="http://www.example.org/names#display_name" AttributeValue="tester" Policy="AddressPreferences.xml"></pii>
    <pii AttributeName="http://www.w3.org/2006/vcard/ns#email" AttributeValue="test@test.com" Policy="EmailPreferences.xml"></pii>

    <!-- New PII -->
    <pii AttributeName="http://www.w3.org/2006/vcard/ns:postalCode" AttributeValue="12345" Policy="DemoPreferences.xml"></pii>
</piis>
```

- Create file DemoPreferences.xml, using AddressPreferences.xml as a draft and modify the authorization preferences in DemoPreferences.xml:
    - allow DownstreamUsage or
    - change the purpose from individual-analysis to e.g. statistics

```
<ppl:DataHandlingPreferences>
    <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"/> <!-- change to true to allow DownstreamUsage -->
    </ppl:AuthorizationsSet>
```

- Change the $root //dc/policies/DCtoDSpolicy.xml (that is the DC privacy policy related to the Clique Website) to ask for the postal code by adding a provisional action. You may also add a new DataHandlingPolicy which matches with your privacy preferences in order to get a matching.

In order to see the content of the sticky policy that is generated after every matching go to the sub-folder $root //ds/sticky_policies

In order to see the claims sent by the DS to the DC go to the subfolder $root //dc/claims

Chapter 7

PPL Engine Data Model

In this section we define all the implementation details of the PPL engine. We are using class and package diagrams to describe in detail how the different components of the engine are implemented and how the communication between different entities (sometimes developed by different partners of the project) are done.

In order to facilitate the manipulation of the different elements of the XML policy we decided to map all these elements into Java classes. These classes are then stored into the persistence database and called as soon as we need to read, modify or generate a new policy. for example if we want to generate a sticky policy, after matching a privacy policy and a preference, we call all the classes related to the elements of this sticky policy and we generate an XML file. This method is less complex than the selection and assembling of pure XML elements. In this chapter all the PPL language elements are describes as Java classes.

7.1 Data model package diagram

The package diagram is used to represent the dependencies between the different packages.

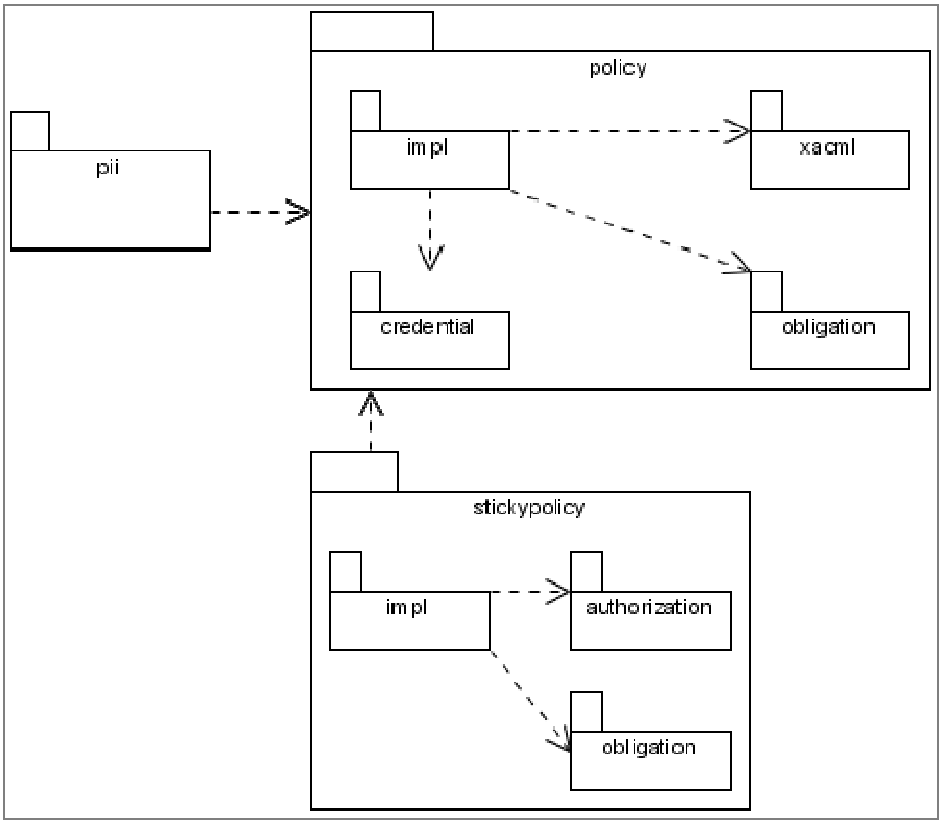


Figure 12: Data model package diagram

The Figure 12 represents the dependencies between the packages that we will describe in the next section.
For the next sections, we define the PPL prefix of one element to define that it as a PrimeLife element, and the XACML prefix for the XACML elements.

7.1.1 Package pii

The package *pii* contains the data class to represent the PII. It's constituted by the *PIIType* class. The *PPIType* is used to represent the Personally Identifiable Information (PII) in a simple way.

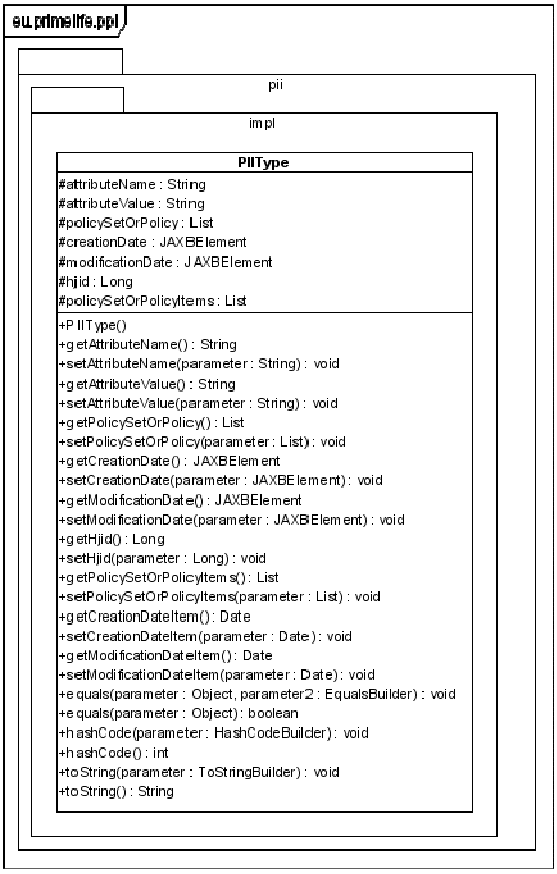


Figure 13: Simplified PII package

This class is composed by:

- The *AttributeName* element, describing the name identifier of the PII, for example, http://www.w3.org/2006/vcard/ns#email which indicates the email PII, or also http://www.fgov.be/eID/address to indicates the address information;

- The *AttributeValue* element, describing the value of the PII, for example if we consider the previous *AttributeName* examples, we can have mail@mail.com as a value corresponding to the http://www.w3.org/2006/vcard/ns#email *AttributeValue*;

- *CreationDate* and *DateModification*, describing some extra date information to the user.

7.1.2 Package policy.Impl

This package contains all the data classes to represent the skeleton of the language data structure.

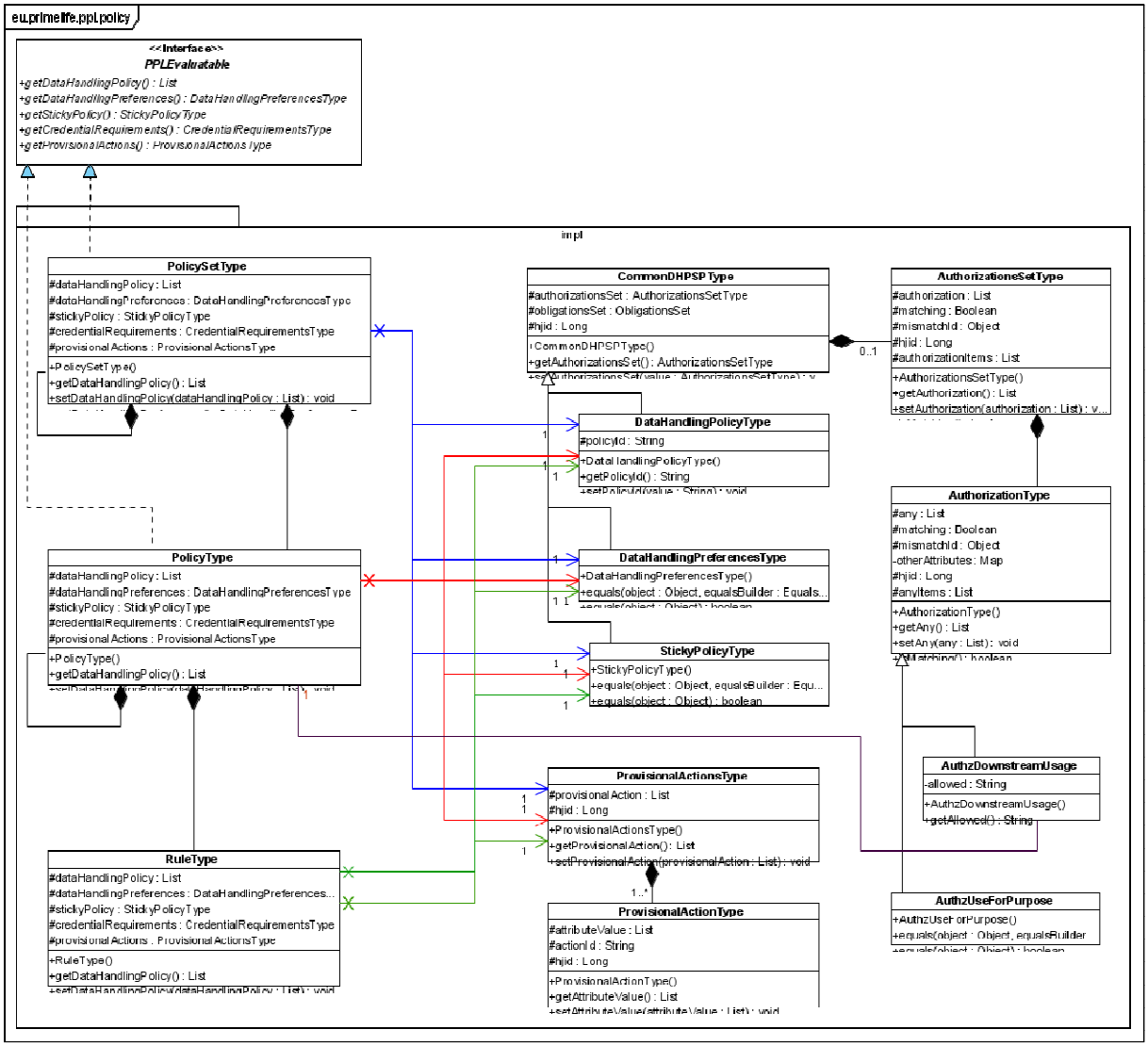


Figure 14: Simplified policy data model class diagram

At the high level, we have the PPL *PolicySetType* and PPL *PolicyType* elements that successively extend the XACML *PolicySet* and the XACML *Policy* classes. The latter are present in the XACML package, and the former implements the *PPLEvaluatable* interface.

The use of *PPLEvaluatable* interface is to provide a generic way to define a 'Policy, because a *Policy* can be defined by either the *PolicySetType* class or by the *PolicyType* class.

The *PolicySetType* class is a top level element. It is an aggregation of other *PolicySetsType's* and *PolicyType's*. And the *PolicyType* class is an aggregation of other *PolicyType's* and *RuleType's* elements.

The *PolicySetType*, *PolicyType* and *RuleType* classes are composed of a different class; *CommonDHPSPType*, which is a generic class for the *DataHandlingPolicy*, *DataHandlingPreferences* and *StickyPolicy* classes, *CredentialRequirements* class and *ProvisionalActions* class.

As the *DataHandlingPolicyType*, the *DataHandlingPreferencesType* and the *StickyPolicyType* classes represent the same data structure, and they are only different from the interpretation meaning, they extends the generic class, *CommonDHPSPType*.

The *CommonDHPSPType* consists of a set of authorizations, defined by the *AuthorizationsSetType* class, and a set of obligations, expressed in the *ObligationsSet* class (defined in the next section).
The *AuthorizationsSetType* class is composed of a set of authorizations defined by the abstract class *AuthorizationType*. This is due to that 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 the *AuthzUseForPurpose* class. The second predefined authorization type ,*AuthzDownstreamUsage*, contains a Boolean attribute indicating whether downstream usage is allowed or not in association with a *PolicyType* attribute that represents the policy preferences for the third party.
The *ProvisionalActionsType* class is composed by a set of *ProvisionalActionType*. This latter describes a single provisional action that needs to be fulfilled in order to satisfy a rule. The *ProvisionalActionType* class contains the *ActionId* attribute and a set of xacml *AttributeValue*. The *ActionId* attribute represent the identifier (URI) of the action to be performed. The set of the XACML *AttributeValue* represents arguments of the action, which may include other functions. The semantics of the argument depend on the particular action being performed. Some actions are defined:

- 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 credential identifier (CredentialID)of the credential that contains the attribute.

- 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. The third (optional) argument is a URI referring to the credential identifier of the credential that contains the attribute.

- 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 credential identifier of the credential that contains the attribute.

- 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. The fourth (optional) argument is a URI referring to the credential identifier of the credential that contains the attribute.

- 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. 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.

7.1.3 Package Credential

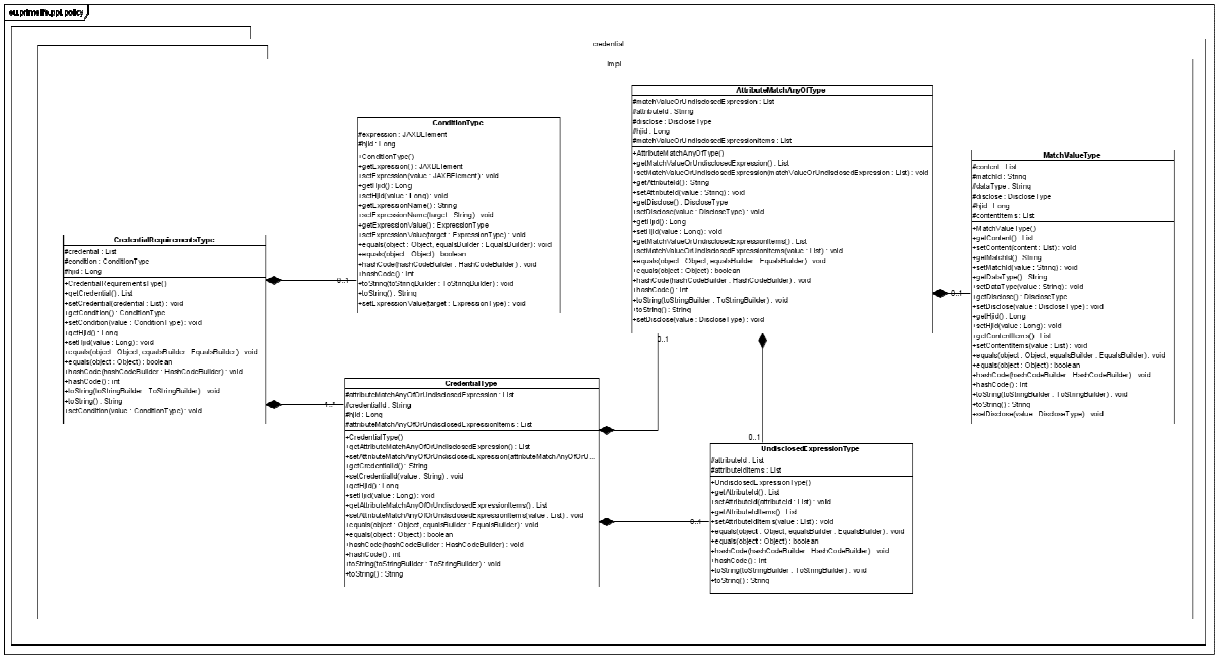


Figure 15: Simplified Credential data model class diagram

The *CredentialRequirementsType* class is composed of a set of credentials and conditions. Each individual credential is a condition within a credential.

The PPL *ConditionType* class contains an abstract attribute of type *XACML:Expression*. This latter class provides a way to define an obligation within a credential.

The *CredentialType* class is used to declare a credential that has to be held by an access requester. It contains a *CredentialId* attribute that identifies the credential element and is in relation of 1..0 with the *UndisclosedExpressionType* class. This class acts as a placeholder to indicate that a credential condition was omitted due to policy sanitization.

Also, the *CredentialType* class is related to the *AttributeMatchAnyOfType* class. This class can specify conditions directly within the *CredentialType* class.

The *AttributeMatchAnyOfType* class is used for matching a given attribute with a list of values, whereby for every list element an individual matching algorithm is used.

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.

The element *AttributeMatchAnyOfType* class contains the following attributes; *AttributeId* which determine the name of the attribute in this credential that is matched against the list of values, disclose attribute, 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”.
- 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 appendix policy sanitization.
The *MatchValueType* class defines 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 class contains the following attributes;

- *MatchId* that indicates the name of the matching algorithm that is used to match the attribute with the literal.
- *DataType* attribute of the literal value against which the attribute will be matched
- *Disclose* attribute that defines the type of policy disclosure used for this element when this policy is sent to the Data Subject. Possible values are “yes” and “no”.

7.1.4 Package Obligation

The obligation package contains all the data classes to define obligations. The main class in this package is *ObligationsSet* class which is composed of a set of *Obligation* objects. Each Obligation contains one *TriggerSet*, which in turn contains a set of *Trigger* objects describing the events that trigger the obligation, and one *Action* element defining the action that has to be performed.

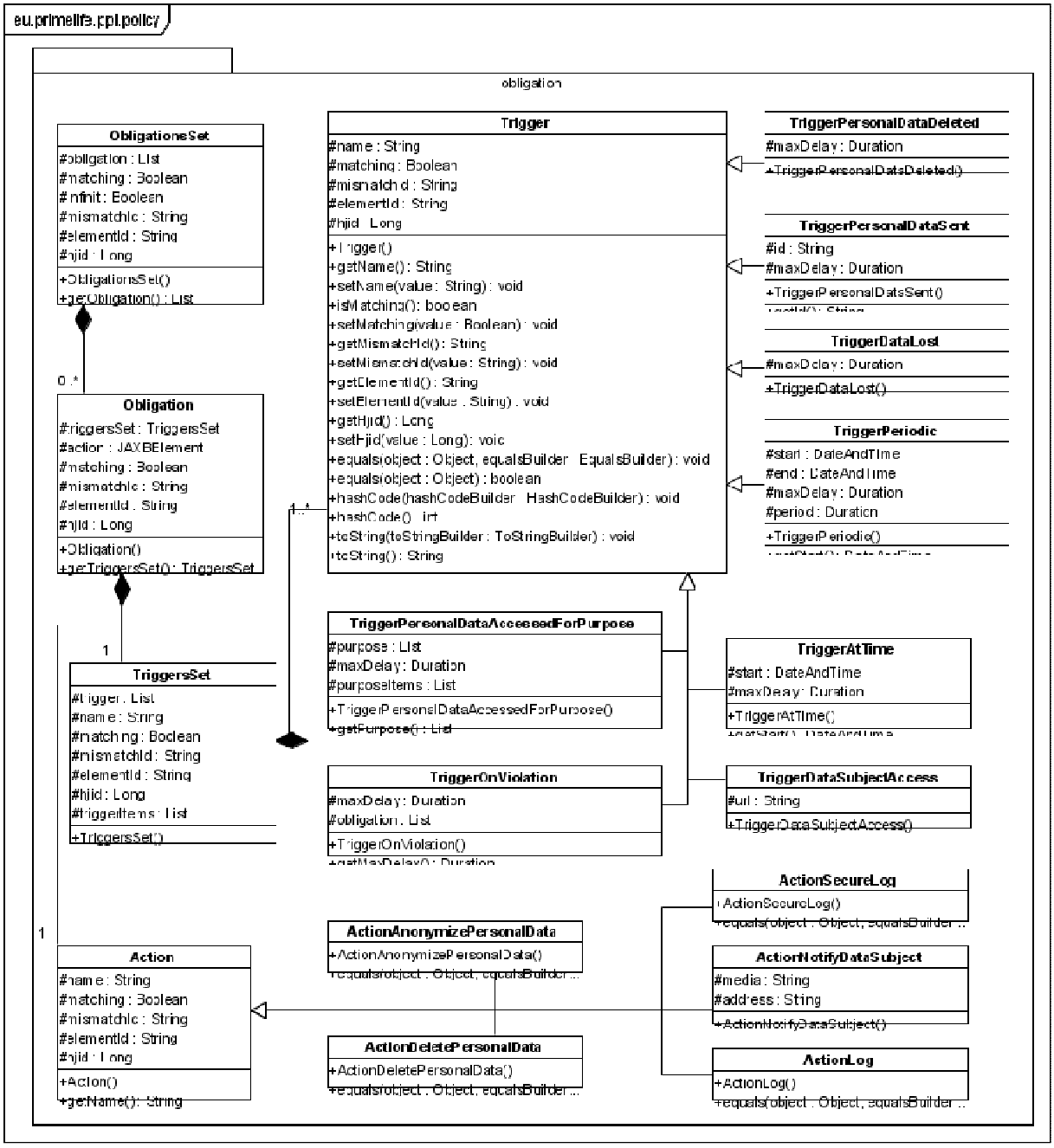


Figure 16: Simplified obligation data model class diagram

There are different types of triggers that extend the abstract class *Trigger*, and also, different types of actions that extend the abstract class *Action*.

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

- The Data Subject's privacy preferences specifies "required obligations", i.e. what the Data Subject requires in terms of obligations to provide a given piece of personal data to a given Data Controller.

- The Data Controller's privacy policy specifies "proposed obligations", i.e. what the Data Controller is willing (and able) to enforce in terms of obligations for a given collected data.

- The sticky policy specifies "committed obligations", i.e. the obligations the Data Subject and the Data Controller agreed upon and that must be enforced by the Data Controller.

Here is a briefly description of some of the common triggers and actions:

- Trigger at Time: A time-based trigger that occurs only once between start and start + maxDelay.
- Trigger Periodic : A time-based trigger that occurs multiple times on a periodic basis between start and end.
- Trigger Personal Data Accessed for Purpose: An event-based trigger that occurs each time the personal data associated with the obligation is accessed for one of the specified purposes.
- Trigger Personal Data Sent : An event-based trigger that occurs when the PII associated with the obligation is shared with a third party (downstream Data Controller).

- Action *DeletePersonalData* : This action deletes a specific piece of information, and is intended for handling data retention.
- Action *NotifyDataSubject*: This action notifies the Data Subject when triggered, i.e. send the trigger information to the Data Subject.
- Action *Log* : This action logs an event, e.g. write in a trace file the trigger information.

7.1.5 Package StickyPolicy

The *StickyPolicy* package contains all the data classes to define the result of the matching process. This package is composed of three sub-packages.

7.1.5.1 Sub-package impl

The sub-package impl of the sticky policy package contains the data structure skeleton of the matching process result.
The StickyPolicy class represents the main component. It contains matching attribute which indicate if the final matching is true (there's matching) or not (there's mismatch), and is composed by different AttributeType elements.
The AttributeType class represents the data type of a matching PII. So, if in the policy we have three revealing action, under some data handling policy, of three PII, we will have three AttributeType objects within the StickyPolicy object. This class contains, as for the *StickyPolicy* class:

- a *matching* attribute to indicate if the PII has a matching or not.
- an *attributeURI* element to represent the attribute name value of the PII, and composed by:

- *authorizationsSet* element, which represent the *authorizationSet stickyPolicy* of the matching
- *obligationsSet* element, which represent the *obligationsSet StickyPolicy* of the matching process, and a *mismatches* element in cast that we have a mismatch result of the matching.

- The *MismatchesType* class contains two types; *authorizationsMismatch* and *obligationsMismatch*. These elements are only present if it occurs a mismatch of the corresponding type. Each *MismatchType* is defined in a separate package, because the *PPL language is extensible*, and the definition of the mismatch depends of the *DataHandling* type.

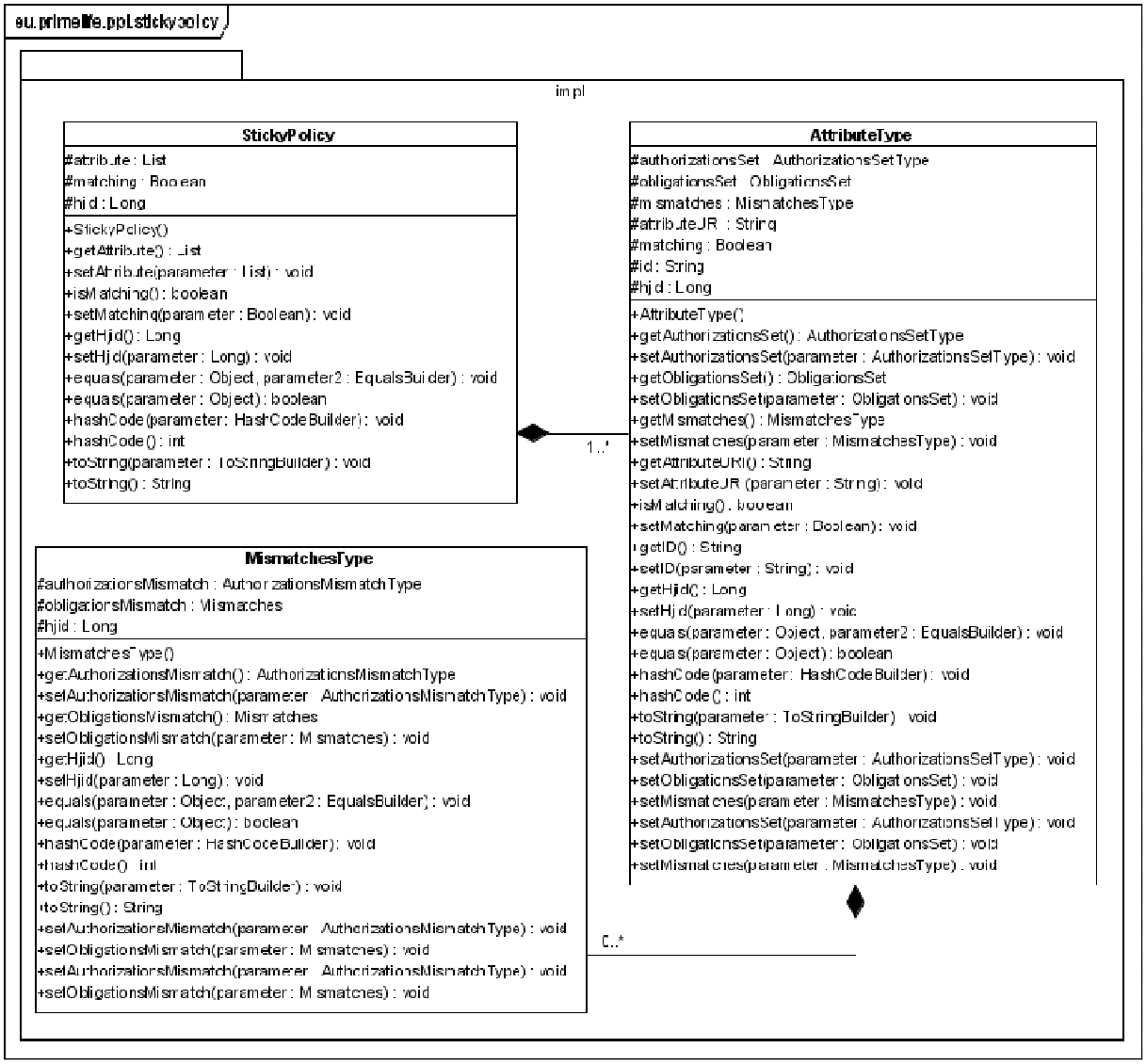


Figure 17: Simplified Sticky policy data model class diagram

7.1.5.2 Sub-package Authorizations mismatch

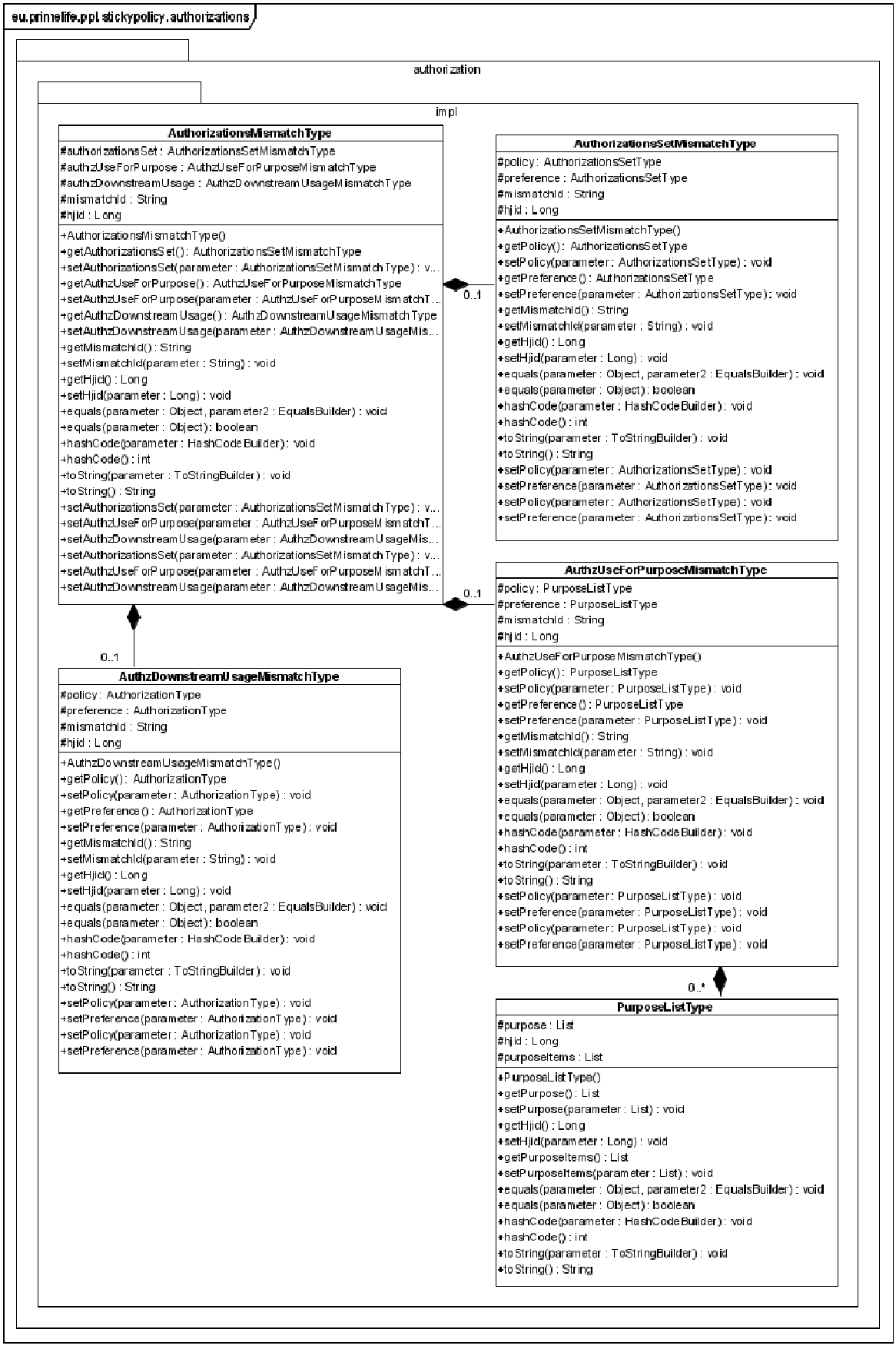


Figure 18: Simplified authorization mismatch data model class diagram

The sub-package *Authorization* represents the data classes of the authorizations mismatch.
The *AuthorizationsMismatchType* class represents the main class. It contains a *mismatchId* attribute that is used to be referred within the *AuthorizationsSet* element, and composed either by the *AuthorizationsSetMismatchType* or with the two *AuthzUseForPurposeType* and *AuthzDownStreamUsageType* together (or only one of the elements).
The *Mismatch* class is defined to help the engine and the UI to display the mismatching elements and permit to the user to make a decision without being obliged to come back to his preferences and compare it with the sticky policy.
We distinguish the authorization and the obligation mismatches using the two classes *AuthzUseForPurposeType* and *AuthzDownStreamUsageType* . In some cases, we can have either an authorization use for purpose mismatch either an authorization downstream usage mismatch, or both together. To notify and describe the mismatch, we use the same concept mentioned above, we indicates the policy values (which represent the data controller values) and the preferences values (which represent the data subject values).

7.1.5.3 Sub-package Obligations mismatch

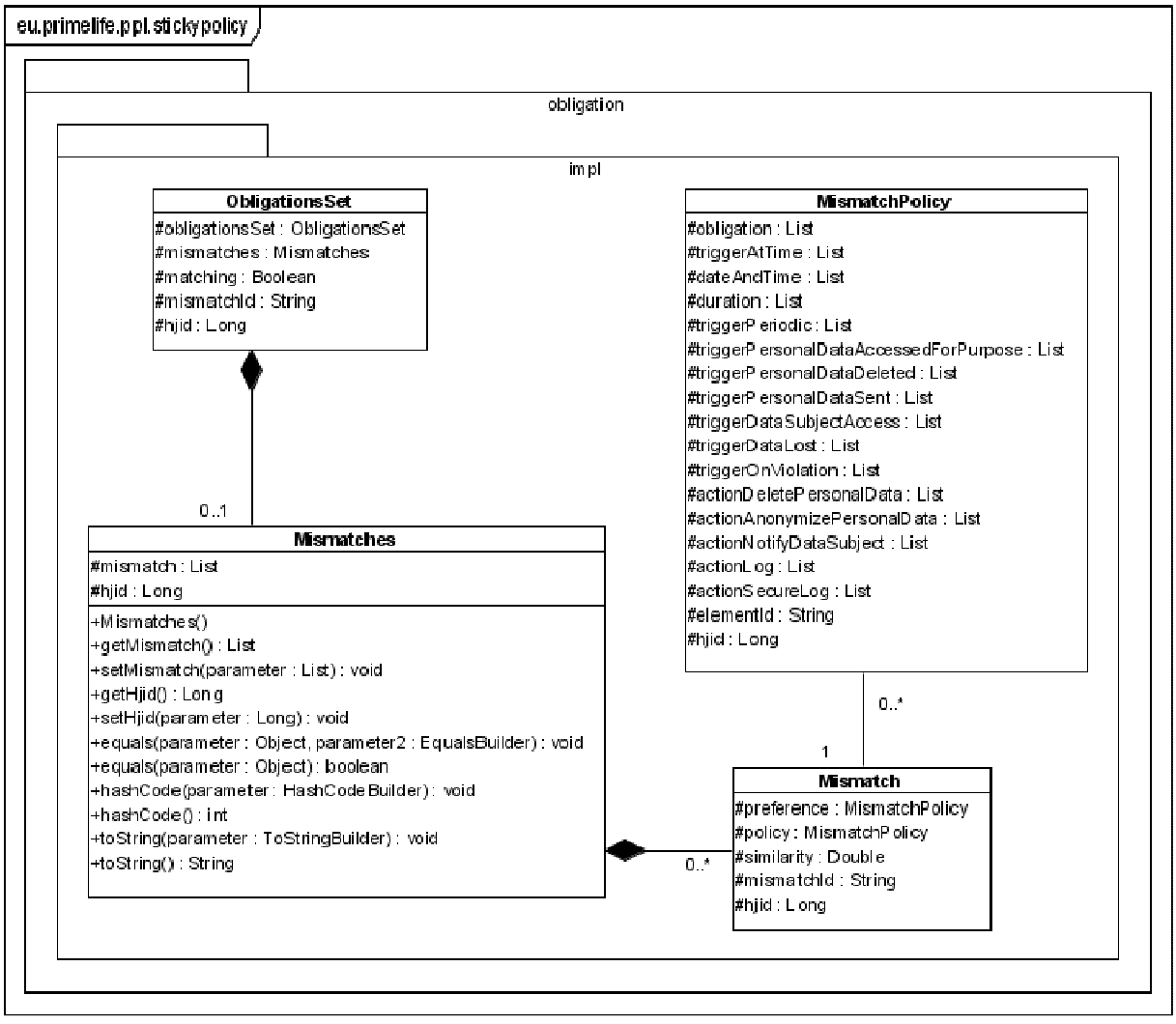


Figure 19: Simplified obligation mismatch data model class diagram

To notify and describe the mismatch, we use the same concept mentioned previously in the authorization mismatch, we use the matching attribute to indicate whether a matching occurs or not, and we indicates the policy values (which represent the data controller values) and the preferences values (which represent the data subject values), in case of matching.

7.2 Detailed Sequence diagrams

Let suppose that the Data Subject wants to access a resource that is held by a Data Controller, and makes a request. The controller will intercept this request, and notice that it's a restricted resource and needs some proof for that the access can be grant. the Data Controller asks the PPL Engine to provide him the privacy policy corresponding to this resource. The PEP asks the persistence handler for the resource privacy policy. The persistence handler makes a request to the database to have the appropriate policy of the resource. Then, this policy is sent to the DS.

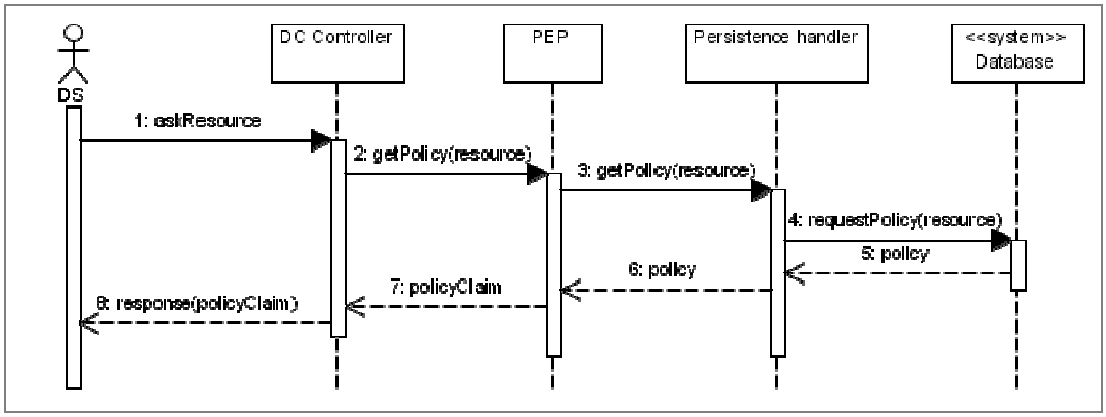


Figure 20: Resource request sequence

The Data Subject UI intercepts the Data Ccontroller response and notices that it is a PPL response and that contains a PPL privacy policy. This latter is then, send to the PEP bypassing the PPL web server. The PEP starts to process this policy which is in a special format (SAML [5] policy claim) to extract the different needed information. The way how SAML is used in the message formatting is explained in the specification document [1]. The policy is then sent to the Credential handler to process the credential part of the policy and returns a response containing different claims combinations[5].

After, the PEP formats the original policy claim, concatenates the combinations returned by the credential handler and sends this new request to the PDP that will generate a response containing a sticky policy.

The PDP will intercept the request . For each combination, the PDP generates a response decision evaluation that contains the:

- sticky policy result of the matching process

[5] Let's consider an example to better explain the meaning of the combinations. Let's suppose that we have two credit cards (Visa and Master Card with different numbers and expiration dates), and we suppose that the DC asks the DS to reveal the credit card information (card number, expiration date, name…) the credential handler will provide two combinations. One with each credit card (because the DS has two).

- list of the missing PII
- list of the PII that present a 'Deny' access
- list of the PII that present a mismatch
- list of the response result for each provisional action. This response will be used to create the final response decision of the PDP.

The PDP uses:

- The provisional action handler to process the different provisional actions;
- The attribute enforcement class to make the enforcement of the requested PII and to check whether the access can be granted or not.
- The matching handler to provide the sticky policy. The obligation handler is called at this point to process the obligation matching.

After that, the PDP returns the final decision. The result is sent to the PEP that formats the result into a claim policy and sends it to the web server that will redirect this response to the UI to notify the DS about the decision and the matching result. The Data Subject, handle this result either by accepting or denying the access.
if there is a mismatch, the user is informed about this mismatch and and has to possibility to evaluate it regarding to his privacy preference, then create an exception to this communication and sends the agreed sticky policy to the Data Controller.

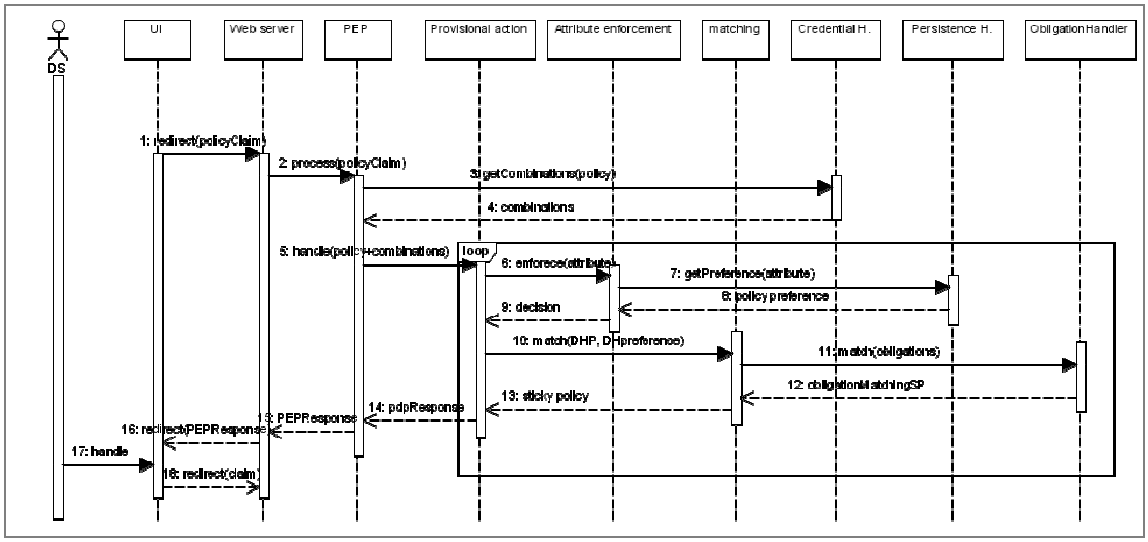


Figure 21: DS policy handling sequence diagram

The Data Controller intercepts a resource request and includes the cryptographic proof for the authentication.
Before the Data Controller grants access to the Data Subject , she has to verify the proof regarding the resource policy.. The Data Controller sends the proof to the PEP then contacts the credential handler to verify the authenticity of the requested attributes.
If all the verifications are successful, the obligation handler activates the different triggers.

7.3 Component diagram

Figure 22 presents the component diagram of the PPL Engine. The UI component represents the user interface. It provides the Data Subject (respectively the Data Controller) with tools to manage the PII -personal information-, (respectively the services, resources, web pages…) and

their corresponding preferences (respectively their policies). The UI notifies grafically the Data Subject with the matching result, .

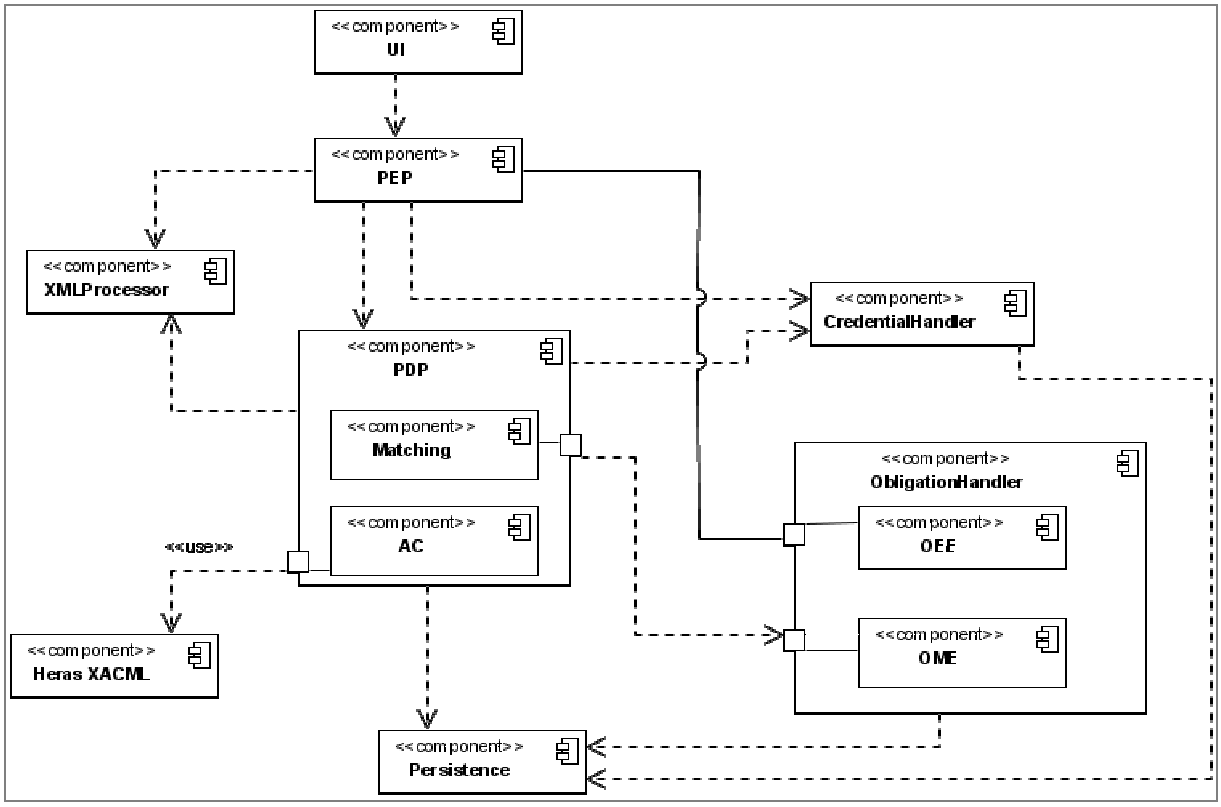


Figure 22: Component diagram

The UI component communicates with the PEP. It intercepts the request, and uses the XMLProcessor component to convert the data from the XML data structure to a Java object, so that the PPL Engine can process the enforcement. The PEP and the PDP components communicates with the credential handler.

The PDP is composed of two subcomponents; Matching and the AC (Access Control) components. The matching component contains the Java classes that perform the authorization matching, and uses the subcomponent OME (Obligation Matching Engine) of the *ObligationHandler* component for the obligation matching. The AC component performs the access control enforcement, this done by delegating this functionality to the Heras XACML[6] implementation component.

Apart from the OME subcomponent, the *ObligationHandler* component contains also the Obligation Enforcement Engine (OEE) subcomponent.

[6] Hears XACML is an open-source project that provides an XACML 2.0 implementation

Annex: XSD Policy Schema

In this annex we publish all the XSD schema used in the PPL engine.

XACML 2.0 schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:oasis:names:tc:xacml:2.0:context:schema:os"
xmlns:xacml="urn:oasis:names:tc:xacml:2.0:policy:schema:os" xmlns:xacml-
context="urn:oasis:names:tc:xacml:2.0:context:schema:os"
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:element name="Request" type="xacml-context:RequestType"/>
	<xs:complexType name="RequestType">
		<xs:sequence>
			<xs:element ref="xacml-context:Subject"
maxOccurs="unbounded"/>
			<xs:element ref="xacml-context:Resource"
maxOccurs="unbounded"/>
			<xs:element ref="xacml-context:Action"/>
			<xs:element ref="xacml-context:Environment"/>
		</xs:sequence>
	</xs:complexType>
	<!-- -->
	<xs:element name="Response" type="xacml-context:ResponseType"/>
	<xs:complexType name="ResponseType">
		<xs:sequence>
			<xs:element ref="xacml-context:Result"
maxOccurs="unbounded"/>
		</xs:sequence>
	</xs:complexType>
	<!-- -->
	<xs:element name="Subject" type="xacml-context:SubjectType"/>
	<xs:complexType name="SubjectType">
```

```
            <xs:sequence>
                  <xs:element ref="xacml-context:Attribute" minOccurs="0"
maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="SubjectCategory" type="xs:anyURI"
default="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"/>
      </xs:complexType>
      <!-- -->
      <xs:element name="Resource" type="xacml-context:ResourceType"/>
      <xs:complexType name="ResourceType">
            <xs:sequence>
                  <xs:element ref="xacml-context:ResourceContent"
minOccurs="0"/>
                  <xs:element ref="xacml-context:Attribute" minOccurs="0"
maxOccurs="unbounded"/>
            </xs:sequence>
      </xs:complexType>
      <!-- -->
      <xs:element name="ResourceContent" type="xacml-
context:ResourceContentType"/>
      <xs:complexType name="ResourceContentType" mixed="true">
            <xs:sequence>
                  <xs:any namespace="##any" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:complexType>
      <!-- -->
      <xs:element name="Action" type="xacml-context:ActionType"/>
      <xs:complexType name="ActionType">
            <xs:sequence>
                  <xs:element ref="xacml-context:Attribute" minOccurs="0"
maxOccurs="unbounded"/>
            </xs:sequence>
      </xs:complexType>
      <!-- -->
      <xs:element name="Environment" type="xacml-context:EnvironmentType"/>
      <xs:complexType name="EnvironmentType">
            <xs:sequence>
                  <xs:element ref="xacml-context:Attribute" minOccurs="0"
maxOccurs="unbounded"/>
            </xs:sequence>
      </xs:complexType>
      <!-- -->
      <xs:element name="Attribute" type="xacml-context:AttributeType"/>
      <xs:complexType name="AttributeType">
```

```
            <xs:sequence>
                  <xs:element ref="xacml-context:AttributeValue"
maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="AttributeId" type="xs:anyURI"
use="required"/>
            <xs:attribute name="DataType" type="xs:anyURI" use="required"/>
            <xs:attribute name="Issuer" type="xs:string" use="optional"/>
      </xs:complexType>
      <!-- -->
      <xs:element name="AttributeValue" type="xacml-
context:AttributeValueType"/>
      <xs:complexType name="AttributeValueType" mixed="true">
            <xs:sequence>
                  <xs:any namespace="##any" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:complexType>
      <!-- -->
      <xs:element name="Result" type="xacml-context:ResultType"/>
      <xs:complexType name="ResultType">
            <xs:sequence>
                  <xs:element ref="xacml-context:Decision"/>
                  <xs:element ref="xacml-context:Status" minOccurs="0"/>
                  <xs:element ref="xacml:Obligations" minOccurs="0"/>
            </xs:sequence>
            <xs:attribute name="ResourceId" type="xs:string"
use="optional"/>
      </xs:complexType>
      <!-- -->
      <xs:element name="Decision" type="xacml-context:DecisionType"/>
      <xs:simpleType name="DecisionType">
            <xs:restriction base="xs:string">
                  <xs:enumeration value="Permit"/>
                  <xs:enumeration value="Deny"/>
                  <xs:enumeration value="Indeterminate"/>
                  <xs:enumeration value="NotApplicable"/>
            </xs:restriction>
      </xs:simpleType>
      <!-- -->
      <xs:element name="Status" type="xacml-context:StatusType"/>
      <xs:complexType name="StatusType">
            <xs:sequence>
                  <xs:element ref="xacml-context:StatusCode"/>
```

```
                    <xs:element ref="xacml-context:StatusMessage"
minOccurs="0"/>
                    <xs:element ref="xacml-context:StatusDetail"
minOccurs="0"/>
             </xs:sequence>
      </xs:complexType>
      <!-- -->
      <xs:element name="StatusCode" type="xacml-context:StatusCodeType"/>
      <xs:complexType name="StatusCodeType">
             <xs:sequence>
                    <xs:element ref="xacml-context:StatusCode" minOccurs="0"/>
             </xs:sequence>
             <xs:attribute name="Value" type="xs:anyURI" use="required"/>
      </xs:complexType>
      <!-- -->
      <xs:element name="StatusMessage" type="xs:string"/>
      <!-- -->
      <xs:element name="StatusDetail" type="xacml-context:StatusDetailType"/>
      <xs:complexType name="StatusDetailType">
             <xs:sequence>
                    <xs:any namespace="##any" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
             </xs:sequence>
      </xs:complexType>
      <!-- -->
      <xs:element name="MissingAttributeDetail" type="xacml-
context:MissingAttributeDetailType"/>
      <xs:complexType name="MissingAttributeDetailType">
             <xs:sequence>
                    <xs:element ref="xacml-context:AttributeValue"
minOccurs="0" maxOccurs="unbounded"/>
             </xs:sequence>
             <xs:attribute name="AttributeId" type="xs:anyURI"
use="required"/>
             <xs:attribute name="DataType" type="xs:anyURI" use="required"/>
             <xs:attribute name="Issuer" type="xs:string" use="optional"/>
      </xs:complexType>
      <!-- -->
</xs:schema>
```

PPL main 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>
```

PPL authorization 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>
```

PPL Obligation Schema

```
<?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>
```

PPL Credential Schema

```
<?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>
```

PPL Sticky Policy 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>
```

References

[1] H5.3.2 Draft 2nd Design for Policy Languages and Protocols. PrimeLife internal deliverable. http://www.primelife.eu/images/stories/deliverables/h5.3.2-seconddesign.pdf

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

[3] Heras – XACML – engine http://www.herasaf.org/heras-af-xacml.html

[4] Article 10 of Regulation (EC) 45/2001). *(From the European Directive on the protection of personal data, Regulation (EC) 45/2001, article 2*

[5] P. Mishra et al., Differences between OASIS Security Assertion Markup Language (SAML) V1.1 and V1.0. OASIS Draft, May 2003