Infrastructure for Privacy for Life

Editors:	Ulrich Pinsdorf (EMIC)
Reviewers:	Dieter Sommer (IBM)
	Karel Wouters (K.U.Leuven)

Identifier:	D6.3.2
Type:	Deliverable
Version:	1.0
Class:	Public
Date:	January 31, 2011

Abstract

This report presents results from our research on privacy for Service Oriented Architectures. It consists of two parts that mutually influenced each other in course of the project. The first part gives a generalized overview of privacy-friendly data handling in cross-domain service compositions. We sketch a generic framework that describes the general processing steps to achieve privacy compliance and proper data handling. The framework abstracts from concrete technologies and policy languages and is thus intended to support implementation based on arbitrary technologies for service-oriented architectures. The second part shows the exemplary implementation of a job application scenario. We used this scenario both to generate and test ideas for the abstract privacy framework. Moreover, the demonstrator is the showcase for real integration and test of the PrimeLife Policy Language (PPL). The scenario has been jointly implemented by EMIC, SAP, and GD. We give technical insight in architecture and design of this demonstrator.

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

Contributions from several PrimeLife partners are contained in this document. The following list presents the contributors for the chapters of this deliverable.

Chapter	**Author(s)**
Executive Summary	Ulrich Pinsdorf (EMIC)
Chapter 1	Ulrich Pinsdorf (EMIC)
Chapter 2	Ulrich Pinsdorf (EMIC)
Chapter 3	Laurent Bussard (EMIC), Ulrich Pinsdorf (EMIC)
Chapter 4	Laurent Bussard (EMIC), Ulrich Pinsdorf (EMIC)
Chapter 5	Laurent Bussard (EMIC), Ulrich Pinsdorf (EMIC)
Chapter 6	Laurent Bussard (EMIC)
Chapter 7	Ulrich Pinsdorf (EMIC), Stuart Short (SAP)
Chapter 8	Fatih Gey (EMIC), Ulrich Pinsdorf (EMIC), Stuart Short (SAP)
Chapter 9	Fatih Gey (EMIC)
Chapter 10	Stuart Short (SAP)
Chapter 11	Ulrich Pinsdorf (EMIC)

Chapter	Author(s)
Appendix A	Laurent Bussard (EMIC). We would like to thank Claudio Agostino Ardagna (Uni Milano), Gregory Neven (IBM), Franz-Stefan Preiss (IBM), Slim Trabelsi (SAP), Mario Verdicchio (Uni Bergamo), and Rigo Wenning (W3C), who kindly accepted to review the evaluation with their expertise on respective systems.
Appendix B	Fatih Gey (EMIC), Ulrich Pinsdorf (EMIC)

Executive Summary

This report presents results from our research on privacy for Service Oriented Architectures. It consists of two parts that mutually influenced each other in course of the project.

The first part gives a generalized overview of privacy-friendly data handling in cross-domain service compositions. We sketch a generic privacy policy framework that describes the general processing steps to achieve privacy compliance and proper data handling. The framework abstracts from concrete technologies and policy languages and is thus intended to support implementation based on arbitrary technologies for service-oriented architectures.

The second part shows the exemplary implementation of a job application scenario. We used this scenario both to generate and test ideas for the abstract privacy framework. Moreover, the demonstrator is the showcase for real integration and test of the PrimeLife Policy Language (PPL). The scenario has been jointly implemented by EMIC, SAP, and GD. We give technical insight in architecture and design of this demonstrator.

This report offers three main takeaways:

1. First, it gives very precise, yet technology agnostic hints how to build a privacy friendly service-oriented architecture. These findings are very generic and can be applied on a wide range of concrete implementations. We describe the abstract privacy policy framework in Chapters 3 to 5.

2. The second takeaway is the detailed analysis of existing privacy-enhancing technologies against the abstract privacy policy framework. We compare to what extend today's solution fulfill or not fulfill the technical demands of a privacy-friendly SOA. We present a summary in Chapter 6 and an extended evaluation in Appendix A.

3. Thirdly, this report gives details about design and architecture of a privacy-friendly SOA application. This *eCV* (electronic curriculum vitae) is also a showcase for the PrimeLife Policy Language (PPL). Chapters 7 to 10 and Appendix B give details.

Although we present the two parts in the given order, both the theoretical aspect and the practical implementation have continuously cross-fertilized each other. The implementation took advantage from the abstract privacy policy framework, while it provided interesting use cases.

Contents

List of Figures

List of Tables

Chapter 1

Aims of this document

Service-oriented architectures (SOA) is a technology-independent architecture concept adhering to the principle of service-orientation. It aims at enabling the development and usage of applications that are built by combining autonomous, interoperable, discoverable, and potentially reusable services. Meissner et al. [MS09] collected in a PrimeLife report a list of 39 requirements for privacy-friendly data handling in SOA. This report tries to come up with a technical suggestion how SOA should be built, so that they leverage the privacy of the end-users.

This document describes our research results from two different points of view. Part I presents the condensed lessons learned on privacy policies in service oriented architectures. We describe these results in a technology agnostic way, so that they can be applied on a large variety of technologies. In Part II we summarize our implementation of a privacy-friendly service-oriented architecture. The Appendix contains more details about both views on the topic.

The detailed structure of the report is as follows. Chapter 2 gives some background on service-oriented architectures and introduces the privacy issues these systems imply. Next, we give in Chapter 3 an overview of our abstract framework for privacy policies in distributed systems. The concepts introduced there will be described in more detail in Chapters 4 and 5. Chapter 4 focuses in the data provider's while Chapter 5 concentrates the data consumer. A comparison of the abstract framework with existing technologies in Chapter 6 concludes Part I. This comparison shall validate our abstract approach and show that the abstraction is sound enough to cover different implementations. Chapter 7 opens Part II and introduced the eCV scenario and the ideas why this setting was chosen. Next, we describe the architecture of the whole eCV demonstrator in Chapter 8. The two following Chapters highlight details of the two main parts of the demonstrator. Chapter 9 features the "upstream" part of the demo, implemented by SAP, while Chapter 10 emphasizes the "downstream" part of the demo, implemented by EMIC. Chapter 11 concludes the whole reports and summarizes the key-takeaways. Moreover, the report features two Appendicies. Appendix A gives a very detailed explanation on the evaluation of different technologies against the abstract framework for privacy policies. Here goes special thanks to a number of external experts and reviewers who helped us to

come up with this detailed reasoning. Finally, Appendix B offers XML schemas, example messages, and screenshots from the eCV application.

The structure of this report offers different ways to read it. The two parts are self-contained, i.e. the reader can understand each part without reading the other. We suggest various aspects of the document for the different target audiences:

- Readers interested in a generalized approach for privacy in SOA should concentrate on Chapters 3 to 6.
- Software Developers looking for an example showcase how to make an existing SOA privacy-friendly should read Chapter 3, Chapters 7 to 10, and Appendix B.
- Researchers with interest to compare existing privacy-enhancing technologies should refer to our comparison in Chapter 6 and Appendix A.
- Software Architects and Researchers interested in an exemplary integration of the PrimeLife Policy Language into an existing application should refer to Chapter 9 and the PPL-related part in Appendix A.
- Related research projects which want to take up the idea of the eCV scenario will find details on that in Chapters Chapters 7 to 10 and Appendix B.

This report does not contain an evaluation of the abstract framework for privacy policies against the requirements for privacy-friendly data handling in SOA [MS09]. This is because PrimeLife will produce a dedicated report on that. This document focuses more on establishing a link between the abstract architectural considerations and the experience we got from building a privacy-friendly SOA demonstrator application.

Chapter 2

Privacy in SOA

This chapter explains the term Service Oriented Architecture, the type of infrastructure we want to apply privacy measures at.

2.1 Service Oriented Architectures

SOA is a technology-independent architecture concept adhering to the principle of service-orientation. It aims at enabling the development and usage of applications that are built by combining autonomous, interoperable, discoverable, and potentially reusable services. These services jointly fulfill a higher-level operation through communication. They fall in the class of distributed systems [CDK05].

One core principle of SOA is the so-called loose coupling of partial services: Single services are not permanently bound to each others, but their binding happens only at run-time enabling a dynamic composition of services [CK05]. Moreover, it is even feasible to dynamically bind services hosted in different security domains and by different legal entities. We refer to this as "cross-domain service composition" [BNP09]. One prominent example for this are services rendered via so-called "service chains" that comprise of several partial services offered by different organizations. To facilitate the use of such services, usually one legal entity might serve as single point of contact for (potential) customers. In times of the Internet, places of business of organizations providing partial services for one high-level service can be widely distributed around the globe.

In many cases, a SOA might involve the processing of personal data and thus pose risks for the privacy of data subjects concerned. Two specific risks can be identified with regard to cross-domain service composition. The first concerns transparency of processing of personal data: The involvement of different legal entities may lead to the situation that data subjects are no longer aware of what data relating to them are handled by what entity for what purpose. This is particularly true if services are bound dynamically at run-time. In case a high-level service delivery involves different organizations, but is exposed by only one of them, customers even might not be aware of the involvement of further legal entities at all. The second risk concerns the issue

of linkability of data: The use of standardized formats and interfaces within an SOA facilitates linkage of systems and data sets. Without the implementation of appropriate technical and organizational measures, organizations could be able to link different sets of personal data and generate profiles on data subjects.

However, the implementation of an SOA also provides some options to achieve a high level of privacy for the data subjects concerned. First, one can realize that each single service that forms part of an SOA usually serves a specific purpose (e.g., authentication, payment). In combination with privacy-compliant logging techniques, this circumstance can be used to implement an automated review of adherence to the privacy principle of purpose limitation. Second, tailoring of single services to specific purposes simplifies determination of personal data that are really needed for the implementation of the respective service. This circumstance facilitates adherence to the privacy principles of collection, use, and disclosure limitation as well as obeisance to the principle of data minimization. Third, SOA provides some possibilities for the implementation of an automated data protection management. This results from the fact that technical integration of a SOA nowadays typically is taking place on the basis of web services and XML. As the same holds true for existing and emerging standards for an automated data protection management, these standards could easily be applied within an SOA.

This document aims at generalizing privacy-friendly data handling in multi-domain service compositions. We sketch a generic framework that describes the general processing steps to achieve privacy compliance and proper data handling. The framework is designed in a way that it addresses multi-step data sharing by repeated application of the same principle. Hence, we use the SOA design principle idea to chain services by chaining our protocol to achieve proper data handling in a multi-domain service composition. The framework abstracts from concrete technologies and policy languages and is thus intended to support implementation based on arbitrary technologies.

2.2 Design Goals

The abstract framework for privacy handling in SOA we are describing in this document shall follow certain design principles.

Generalization. The framework shall be generic in a way that the reader can apply the principles and protocols to any service-oriented scenario regardless of the technology that is actually being used. Hence, rather than giving advise on specific technologies (policy languages, communication protocols, etc.), we will stay quite abstract. However, Chapter 6 gives an idea how the principles would be adopted with different technologies.

Abstraction from Scenario. The framework's principles have to be general enough that they can be applied to any service-oriented scenario. That means we cover variations in the producer-consumer relations in a service chain, allow for arbitrary communication patterns between the services, and do not assume or impose certain trust relationships[1].

[1]In fact, the only trust assumption we make is that any communication partner adheres to agreements it establishes with other communication partners. That means we do not *enforce* privacy by technical measures, but motivate proper data handling for communication partners that would suffer

The principles can be applied to applications where each service belongs to the same trust domain as well as applications with services from different trust domains. We support static and dynamic binding of subsequent services, which has an impact on the privacy policies as well. Finally, we support variations in the business relationships between two services, i.e. which service can or cannot impose execution conditions on the other, i.e. which party selects the service level[2].

Modular and iterative pattern. Service-oriented architectures take great strength from the design principles of modularity and iteration of the same pattern. It makes sense to follow the same approach here. The framework should be modular and self-contained enough such that it involves only two parties, data provider and data consumer. At the same time the principle shall be applicable in an iterative way. That means a former data consumer may become the data provider in a next communication step of the SOA. In that way our protocol may be used end-to-end throughout the service chain.

Non-invasive protocol. Building on iterative patterns, the framework should be complementing the existing service communication but not impose that any service in the service chain *must* use it. With that requirement our protocol can be used only in parts of the service chain. It is up to any two communicating services if they follow the protocol or not.

Late adoption. We support late adoption of the protocol. That is, any existing SOA application shall be able to be enhanced even after deployments. Certainly, that may cause invasive correction in the current implementation of the services, deployment of new components, and mutual agreement of data provider and data consumer to support this protocol. However, the protocol could be adopted only for parts of the service chain as pointed out above.

Extensible with other privacy-enhancing technology. The protocol we suggest in this document shall be complementing other privacy-enhancing technologies, such as data encryption schemes, anonymous credentials, communication anonymization, trustworthy user interfaces etc.

An explicit non-goal of our approach is to cover privacy-enhancing technologies like anonymous credentials, privacy-aware queries, user interfaces, or provide concrete implementations of the protocol. The reader may refer to Chapter 6 for comments on how this protocol would be implemented with state-of-the-art technologies, but for sake of general applicability we do not intend to go any further than that.

These goals set the guideline for an abstract framework addressing the lifecycle of privacy policies in distributed systems. Meissner et al. [MS09] collected in a PrimeLife report a list of 39 requirements for privacy-friendly data handling in SOA. It builds on

from a privacy incident at their company. In fact, the protocol we are suggesting creates evidence at different parties' sides about whom data was handled to, for which purpose, and under which data handling constraints. In case of a privacy incident this information would make it easy to point out the misbehaving party.

[2] cf. *lazy binding* in [BNP10]

earlier research done in [BGS+07]. This list is more scenario-oriented and addresses both technical and legal aspects. The list complements the design goals we give above.

Part I

Abstract Privacy Policy Framework

Chapter 3

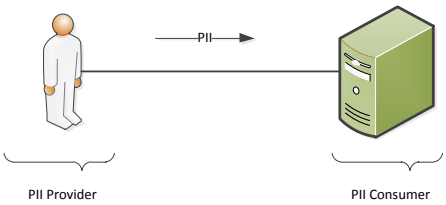
Overview

This section gives a generalized overview of privacy-friendly data handling in cross-domain service compositions. We sketch a generic framework that describes the general processing steps to achieve privacy compliance and proper data handling. The framework is designed in a way that it addresses multi-step data sharing by repeated application of the same principle. Hence, we use the SOA design principle idea to chain services by chaining our protocol to achieve proper data handling in a multi-domain service composition. The framework abstracts from concrete technologies and policy languages and is thus intended to support implementation based on arbitrary technologies for service-oriented architectures. This approach allows for both an abstract consideration of privacy implication and a late adoption. Late adoption means that any existing SOA application shall be able to be enhanced even after deployment. Certainly, that may cause invasive correction in the current implementation of the services, deployment of new components, and mutual agreement of data provider and data consumer. Finally, this framework focuses on the lifecycle of privacy policies and can be complemented by other privacy-enhancing technologies that are described throughout this document, such as data encryption, anonymous credentials, anonymous communication, trustworthy user interfaces etc.

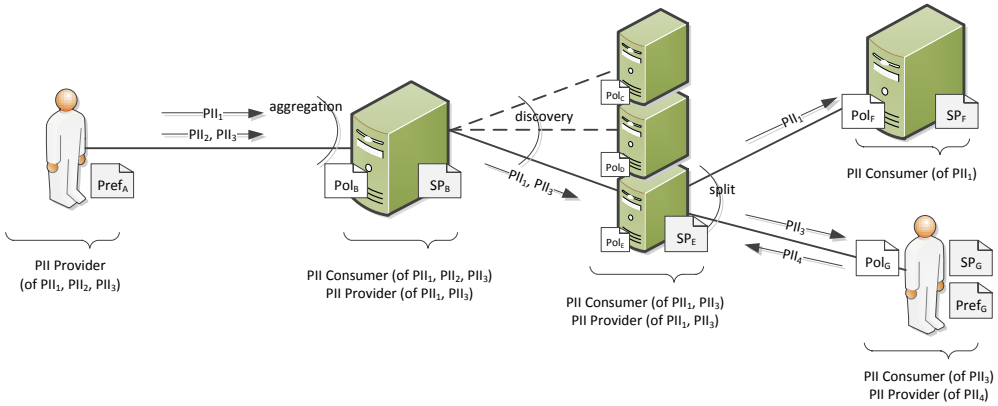
We introduce the idea for an abstract policy framework with a description of the simplest possible scenario, a client-server interaction. Client-server technology can be seen as the ancestor of service-oriented architectures, yet it is still the nucleus of each service-oriented architecture pattern, since even the most complicated service interaction can be broken down to individual interactions between two entities, which represent a client and a server.

Figure 1(a) shows the usual scenario for privacy policies, which is also applicable to client/server systems. The service exposes a privacy policy (e.g., P3P [W3C06]) expressing how it will handle collected data. The user has privacy preferences (e.g., APPEL [W3C02]) that reflects her expectations in terms of privacy. By comparing privacy preferences and privacy policies, the user (or user agent) determines whether it is suitable to share data.

Service-oriented architectures can be seen as applying client/server communication



(a) PII exchange in Client-Server scenario



(b) PII exchange in Service Oriented Architecture

Figure 1: Comparison of privacy policies in client/server architectures and in service-oriented architectures.

in a recursive way. The user invokes a single service. The service does not perform the full operation itself, but invokes one or many other services to perform parts of the task. These invoked services act likewise and in turn invoke other services. The result is a tree[1] of service invocation where each node represents a service. We can apply the same recursive pattern to privacy policies that are communicated between the individual services in a SOA.

Data is collected by a service (data controller) that may share it with third parties. When third parties act on behalf of the data controller, they are referred as *data proces-*

[1]In theory the service invocation would even form a directed graph containing loops. But for sake of simplicity and based on the common practice we assume the invocation graph is a tree.

sors. When third parties are in a different trust domain, they are referred as *downstream data controllers*. In the latter case, the policy of the third party is taken into account when deciding whether data can be shared.

3.1 Requirements

Figure 2 shows the complexity of privacy in cross-domain service compositions. We observe the following privacy-related aspects:

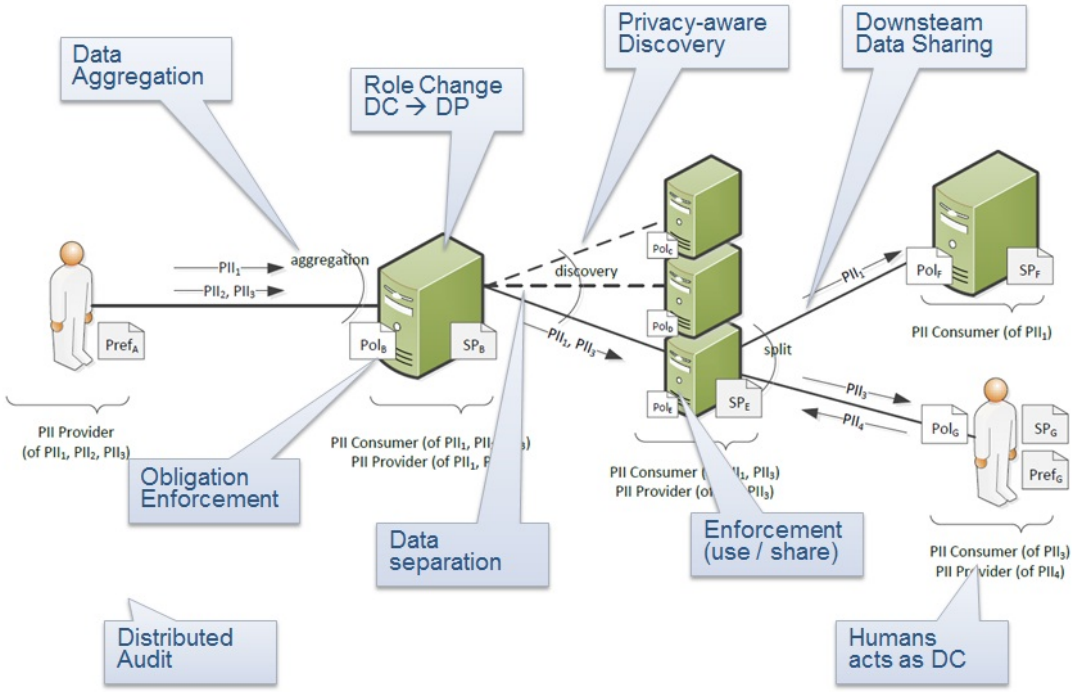


Figure 2: Privacy-related aspects in Service Oriented Architectures

Downstream data are collected by services (data controller) that may share it with third parties. When third parties act on behalf of data controllers, they are referred to as *data processors*. When third parties are in a different trust domain, they are referred as *downstream data controllers* [BNP10]. In the latter case, the policy of third parties is taken into account by users (data subjects) when deciding whether data can be shared.

Provider and Consumer stakeholders can have both PII Provider and PII Consumer roles. For instance a service collecting customers' e-mail addresses acts as a PII Consumer but when this same service shares collected data with another (downstream) service, it acts as PII Provider.

Downstream accessibility A data subject can indirectly (i.e. downstream) interact with a third party and subsequently have a direct interaction with this one.

Users can be PII Consumer in basic scenarios, users only provide data. However, a human being can also collect personal data and act as a PII Consumer. In this case the user must have a privacy policy expressing how he/she handles collected data and must accept sticky policies (e.g. license in Enterprise Right Management [Mic09]) attached to data (e.g. documents).

Aggregation collected data can be aggregated. Mechanisms to compute the privacy constraints on the aggregation are necessary.

Split collected data can be split. Mechanisms to compute the privacy constraints on each piece of data are necessary.

Privacy-aware service discovery privacy impact may be taken into account when discovering services. Services may be ranked or filtered out based on privacy constraints.

Targeted disclosure PII selection (including Identity Selection) may require generating claims for a given party, or disclosing personal data to a group.

Distributed enforcement the enforcement of data handling (including access control when sharing data) is done by each party getting access to a piece of data.

Distributed audit traces may be generated by all parties getting access to data. Mechanisms to federate the analysis of the audit traces are necessary.

3.2 Outline

We will now describe an abstract protocol that takes into account the complexity issues for privacy that are arising from SOA. The protocol is technology agnostic, which has the advantage that may be implemented in various ways. It is, hence, a blueprint how to build privacy-enhancing SOA applications or – in case of late adoption – a blueprint how to make an SOA application more privacy-preserving.

Figure 3 shows the abstract framework from a high-level perspective. It shows a PII provider on the left and a PII consumer on the right side. This interaction can be applied recursively on all segments in a service chain of a SOA application. A PII consumer will then take the role of a PII provider as explained earlier, when it invokes other services.

The PII provider side consists of three building blocks: the PII provider behavior, the PII store and the preferences store. The structure of the PII consumer side matches and extends the structure of the PII provider, which is an enabler for switching the roles from PII consumer to PII provider. The consumer part consists of the PII consumer protocol, the PII store, the policy store (which is comparable to the preference store), and an additional sticky policy store.

PII Store is the database for storing personal data. Depending on scenarios, this can be a local database (e.g., on client machine), a service in the same trust domain (SQL server at data collector side), or a service offered by a trusted third party (e.g., cloud storage with appropriate trust model), a local credential store (certified PII), a remote credential store (e.g., Security Token Service), or a combination of them. *Preferences Store* is storing all privacy constraints. The Preferences Store keeps track of constraints

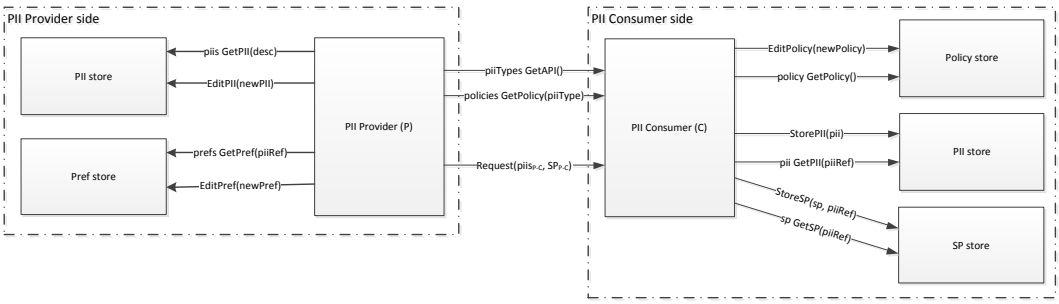


Figure 3: Overview of the generic privacy lifecycle in SOA applications

that are locally defined (and can be overwritten) and committed (and must be enforced). *Policy Store* is the information necessary to describe how data will be handled when sent to this PII consumer. A policy can be statically defined or derived from the business process. A policy can be generic or specific to a user (i.e. depending on authentication). Moreover, a policy can be local or can depend on external policies (e.g., the policies of downstream services). *SP Store* (sticky policies store) stores the policies that were agreed between PII provider and consumer for a specific piece of information. The sticky policy is a result from matching the PII provider's preferences with the PII consumer's policy.

PII Provider and PII consumer communicate via a simple three-step protocol.

1. **Requested PII types** First, the PII provider asks the consumer side for the PII types that are needed for the service invocation. This request may have different technical incarnations. It could be a specification in a web form, it could be a service description such as WSDL[2], but it could also be a dedicated method call to request this meta-data about the service that the PII provider intends to access. In any case the PII provider learns about the types of information that are requested to perform this service.

2. **Policy request** In the second step the PII provider asks for the policy that shall be applicable to the provided information. In other words, it requests a description (in a formal language) of the PII consumer's commitment on how collected data are going to be handled.

3. **Service invocation** In the third step, the PII provider submits the requested PII together with a sticky policy for each data item.

Internally this three-step protocol is backed up by many sub-procedures and decisions. We will look into this individually for PII provider and PII consumer when discussing Figure 4 below. From this high-level perspective it is important to understand that the PII store provides the personal data that is shared in the third protocol step. The preference store allows to retrieve and edit stored information. The preference

[2]WSDL stands for web service description language.

store participates in the creation of a sticky policy for the submitted data items. The privacy policy provided by the PII provider side is compared with the preferences for this specific piece of information. The result of this matching process is a minimal policy that considers both the PII provider's preference and the PII consumer's policy.

The internals on the PII provider's side look very similar. The policy store allows retrieving and editing policies, which are sent in the second protocol step. The PII store is needed to store the received PII from the PII provider. While the PII is kept in the PII store, the sticky policy is stored in the sticky policy store. However, a linking mechanism, e.g., a dedicated link table in a database, keeps a relation between the piece of PII data and the sticky policy.

This high-level protocol could be embedded in an existing application interaction. The last protocol step is usually the service invocation of any given SOA application. All we do is adding an interaction step that requests the required information before the actual invocation. Service discovery mechanisms provide this information anyway, but usually only on a data type level. For instance, a WSDL description clearly states that a function call `calculatePension(date)` requires the parameter to be a data type encoding a date. Today the service description is usually on a syntactical level. That means the service description does not state that the submitted data has to be the user's date of birth. However, semantic web technologies even do specify this. Mechanisms for requesting policies are widely used for Service Level Agreements (SLA). Hence, both protocol steps 1 and 2 are special ways of meta-data lookups. The additional step when adopting our scheme is that the service does this privacy-specific meta-data lookup before the real service invocation.

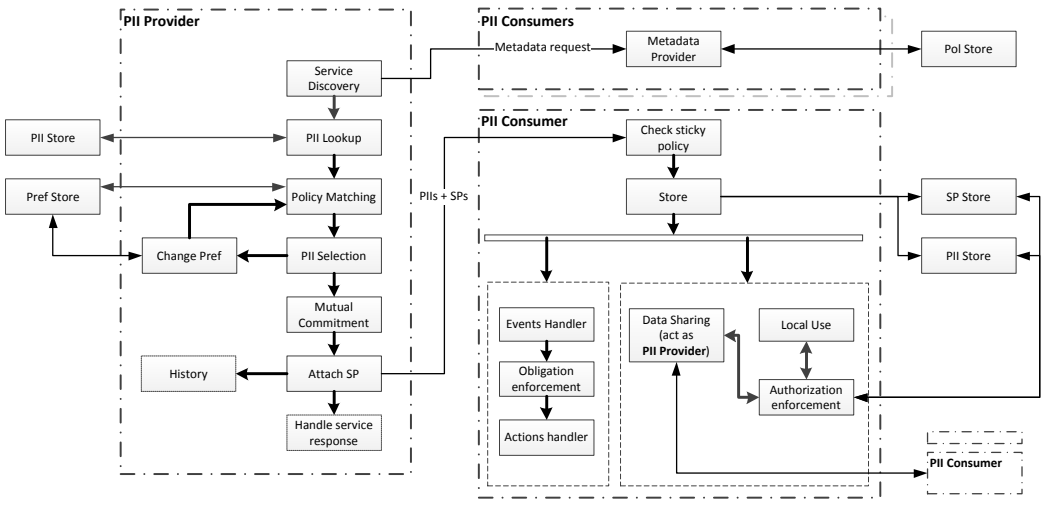


Figure 4: Generic privacy lifecycle in SOA applications in detail

Figure 4 shows the Abstract Privacy Framework in more detail. The figure contains again the top-level components for the PII Provider and the PII Consumer (dashed boxes). Furthermore it shows the iterative approach by chaining from the PII Consumer

to another PII Consumer. Each top-level component contains a best-practice workflow. This workflow is an ideal, scenario-independent view on data handling in a composed service. Moreover, the workflows introduce more components that should be part of each role's technical representation. The next sections describe all top-level components from Figure 4. Chapters 4 and 5 go into more details for the PII Provider and resp. the PII Consumer.

3.3 PII Provider

PII Provider P is the role of entities sharing personal data with PII Consumers. In most scenarios, the user (or data subject) is acting as PII provider. Sharing personal data with another party is generally restricted by privacy constraints (access control and/or expected data handling). Those privacy constraints can be locally specified (e.g. a data subject can specify privacy constraints on her data), can be external i.e. provided by another party (e.g. a data controller sharing collected data with a third party has to enforce constraints imposed by the data subject), or can be a combination of local and external constraints. The PII Provider's role is essentially about deciding whether it is worth sharing pieces of personal data in order to get services from PII consumers.

3.4 PII Consumer

PII Consumer C is the role of entities collecting personal data provided by PII Providers. In most scenarios, a service (or data controller) is acting as PII Consumer. PII consumers are in charge of enforcing agreed data handling on collected data. Data handling is imposed by the PII Provider and can be refined by the PII Consumer. The PII Consumer's role is essentially about 1) checking whether actions are authorized before acting on collected personal data and 2) enforcing obligations regarding those data.

3.5 PII Store

PII Store PII_P is the database containing all personal data $pii \in PII_P$ of PII Provider P. This database can be hosted by the PII Provider (e.g. part of the user agent) or kept remotely (e.g. cloud storage). When personal data are signed credentials, the PII Store can be a local credential store or a remote credential issuer (e.g. Security Token Service).

3.6 Preferences Store

Preferences Store $Prefs_P$ contains all preferences of PII Provider P regarding any of her personal data $pii \in PII_P$. Preferences define how personal data has to be handled by other parties. The PII Provider has preferences for each personal data she is ready to share: $\forall pii \in PII_P \cdot Prefs_P[pii] \neq \emptyset$. No preference would mean that no rights are provided and all possible obligations are expected. Where $Prefs[pii]$ is the subset of preferences in *Prefs* that applies to *pii*. Preference Store can be local (e.g. part of the

user agent) or remote (e.g. provided by a trusted third party or by a group). At anytime, a PII Provider can create or modify her preferences.

3.7 Policy Store

Policy Store $Pols_C$ contains privacy policies of data controller C regarding collected data. Policies define how collected data are handled. The data controller has a policy for each parameter *param* of each interface *api* collecting personal data: $\forall api \in API_C \cdot \forall param \in api \cdot Pols_C[param] \neq \emptyset$. Policies can be statically defined or derived from business process. They may depend on the PII Consumer, e.g. when this one is authenticated. Moreover, policies can be local or can depend on external policies (e.g. the policies of downstream data controllers).

3.8 Sticky Policy Store

Sticky Policy Store SP_C is very similar to *Preferences Store* (see section 3.6). It contains sticky policies of each collected data pii_C stored in PII_C, i.e. each instantiation of a parameter *param* with a personal data $pii \in PII_P$. Sticky policies are defined for each collected data $\forall pii_C \in PII_C \cdot Prefs_C[pii_C] \neq \emptyset$.

Chapter 4

A Closer Look at *PII Provider* Role

This section provides more insight on components necessary to fulfill the "PII Provider" role in Figure 4.

This sections illustrates the general steps a PII provider has to undertake internally in order to support a privacy preserving protocol we described in the last section. The figure is intended to be a block diagram, so it is neither just a workflow nor a collection of software components, but rather a mixture of both.

The *Service Discovery* step refers to a mechanism to find and select one or more Data Controllers to be used and get their meta-data, such as functionality, credentials, SLA. Bringing together PII Provider and PII Consumer is necessary before any interaction. We call this phase service discovery even if, in some scenarios, it can be initiated by the PII Consumer. In our picture, requesting required PII and requesting privacy policies are covered in two separate steps, but they could technically be merged into a single meta-data request.

PII Lookup is a mechanism to determine whether the PII Requirements can be met by the personal data in the PII store. It aims at finding all combinations of personal data that may satisfy the request of the PII Consumer. The PII Provider can have different personal data that match one element of the request and can even load or create new personal data (enter a date in a form, attach a picture) with no a priori attributes. The PII Consumer may accept different types of personal data (e.g., name and age), may specify different attributes of PII (e.g., signed by a given third party), and may accept special combinations of personal data (e.g., credit card number and expiry date must come from the same credit card credential).

The *Policy Matching* mechanism decides whether personal data can be shared according to its privacy constraints and the preference of the PII consumer. The selected PII must not only match the requested PII in type and semantic, but also the privacy preferences associated with the individual PII data items. Privacy preferences settings are always specific to a given personal data. This can be the combination of different privacy preferences. For instance, an e-mail address can be subject to preferences related to any address, to any e-mail address, and to this specific e-mail address. Moreover a given piece of data can be subject to constraints (preferences and sticky policies) from

different parties e.g., data issuer (for a credential), data subject, or data controller (local preferences). The preferences of the selected PII must match with the privacy policy of the service. The PII selection process may be very complicated since the selected PII and its associated preference mutually influence each other and must comply with the service policy. For instance, the user could have multiple credit cards with different privacy preferences. Hence, the user can influence the policy matching process by selecting the specific piece of PII and by changing the policy preferences sticking to this PII. The block diagram reflects this with a loop around PII selection, policy matching, and PII selection.

PII Selection is a process which allows the user to pick the appropriate PII from the PII store. PII selection can be seen as an extension to Identity selection (e.g. Identity Selector such as CardSpace[1]) where not only minimal disclosure is taken into account but also privacy policies.

The selection could be done automatically or in a manual process by the data controller. It is already challenging finding a suitable solution in the search space across multiple user credentials, but it may be even more challenging to visualize this complexity to the user. In case of more than one suitable solution the user must also be empowered to understand what the best solution is. This needs proper user interface support. Depending on the actual case the user may want to pick the solution that enforces the most restrictive privacy preference, the solution that shares the least amount of data, or the solution which does not use a specific set of credentials.

As interesting as finding a valid combination of credentials with associated preferences is thinking about the non-matching case. When there is no suitable option, the user needs to understand the reason why there is no match and propose different options to proceed. The user could simply stop the transaction and not invoke the service at all. Probably he/she would repeat the service discovery step and pick another service that provides a similar functionality. The user could also continue the processing and violate his/her own preferences, respectively adapt the preference in the preferences store so that the PII selection yields at least one match. In principle, the service could adapt its privacy policy as well, e.g., if the user picks a more expensive service level ("premium service" vs. "free service").

When the user decides to amend his/her privacy preference in order to achieve a match with the PII consumer's policy, we cover this in the step *change preferences*. Note that different types of updates can be envisioned ranging from adding a preference for this specific case to changing the preferences that covers a larger group of PII Consumers and/or personal data.

Finally, when the PII selection was made that matches the policy, we create a *mutual commitment* between the PII provider and the PII consumer. We call this step commitment rather than agreement, since we see a policy negotiation as an optional step. In other words, we assume for most use cases the privacy policy of the PII consumer is fixed and any adaptation will be made on the preferences of the data provider. Hence, this leads to a mutual commitment, but not necessarily to a negotiation.

The agreement itself can be a sticky policy or just a Boolean response indicating an acceptance of the PII Consumer's policy. A more sophisticated technical implementation

[1]More details on CardSpace at `http://msdn.microsoft.com/en-us/library/aa480189.aspx`

could even foresee statement that is signed by both parties or witnessed by a trusted party.

In case the agreement is expressed with a sticky policy, which we assume here without loss of generality since it is the most expressive form of agreement, the sticky policy may specify recursively usage control that must be enforced by the PII consumers (including downstream). Indeed, usage control may specify access control towards downstream services including downstream usage control.

The action to *attach a sticky policy* involves the communication of the requested PII from the data provider to the data consumer together with the sticky policy. In other words this step also involves the *service invocation*. When this step has been performed, the PII consumer possesses the requested PII and the sticky policy. Technically the communication of PII and sticky policy can be performed in a single service call or in two separate calls. Moreover, one sticky policy may apply to multiple pieces of data. In any case, it is necessary to keep the link between data and the related sticky policy. The PII provider may keep track of this interaction in a history store. This is helpful e.g., when a trusted third party audits the PII consumer at a later point in time, when the user wants to base a PII selection decision on earlier decisions, or when the user wants to verify to whom a certain piece of data was disclosed under which conditions.

4.1 Service Discovery

Service Discovery is the process of listing potential PII Consumers. Service Discovery takes into account functional properties as well as non-functional properties such as QoS and privacy. This returns a set of discovered interfaces API_{disc} with privacy policies defined for each parameter $param \in api \in API_{disc}$. Note that optional parameters also need a policy. For instance, a PII Consumer asking for $param_1 \vee param_2$ where $param_1$ = *birth date* and $param_2$ = *proof of majority*, has to provide privacy policy for both parameters even if the PII Provider will only assign one of them.

4.2 PII Lookup

PII Lookup aims at finding all combinations of personal data that satisfy PII Consumers, i.e all discovered interface API_{disc}. A PII Consumer may accept different types of personal data (e.g. confirmation by e-mail or by SMS), may specify different attributes on personal data (e.g. claim signed by a given third party), and may accept different combinations of personal data. The result of the lookup is a set of possible personal data PII_{param} for each *param* in each $api \in API_{disc}$. Preferences must exist for each personal data. When personal data is created on the fly (e.g. when filling HTML Forms), generic preferences are used or new preferences are created.

4.3 Policy Matching

Policy Matching aims at deciding whether privacy expectations $Prefs_P[pii]$ regarding personal data *pii* are satisfied by privacy promises $Pols_C[param]$ regarding a parameter *param* before assignment $param \leftarrow pii$.

We say that privacy constraint p_2 is a valid enforcement of p_1 when p_1 is more permissive than p_2, denoted as $p_1 \trianglerighteq p_2$, i.e. p_1 specifies more rights and/or less obligations than p_2. We use this operator to implement matching as $Prefs_P[pii] \trianglerighteq Pols_C[param]$. Matching is mainly checking that all privacy constraints defined in preferences and policies can be satisfied.

4.4 PII Selection

PII Selection aims at selecting (or creating) a suitable piece of personal data pii_{sel} for each parameter $param_{sel}$ that has to be instantiated. $\forall(param_{sel} \leftarrow pii_{sel}) \cdot Prefs_P[pii_{sel}] \trianglerighteq Pols_C[param_{sel}]$. When the PII Provider is the data subject, she can decide to override a mismatch and modify her preferences (see Sect. 4.5).

PII Selection is a complex task that may combine service selection, minimal disclosure (selection and combination of individual pieces of data), identity selection (when personal data are claims), mismatches solving (based on metrics to compare mismatches), impact of released data [ADF+10], and history of previously released data.

4.5 Change Preferences

In case of mismatch, i.e. $\exists(param_{sel} \leftarrow pii_{sel}) \cdot Prefs_P[pii_{sel}] \not\trianglerighteq Pols_C[param_{sel}]$, the PII Provider may decide to modify her preferences. *Change Preferences* aims at replacing preferences $Prefs_P$ by $Prefs'_P$ so that $\forall(param_{sel} \leftarrow pii_{sel}) \cdot Prefs'_P[pii_{sel}] \trianglerighteq Pols_C[param_{sel}]$.

A usual example is to extend generic preferences (e.g. by default any bookseller can use my e-mail address to confirm order) with a specific exception (e.g. a specific bookseller can also use my e-mail address for advertisement).

4.6 Sticky Policy: Mutual Commitment

The *Sticky Policy sp* expresses the agreement between the PII Provider and PII Consumer. The enforcement of the sticky policy has to be an acceptable enforcement of the PII Provider's preferences, i.e. $Prefs_P \trianglerighteq sp$. Behavior of the PII Consumer has to be a valid enforcement of the sticky policy, i.e. $sp \trianglerighteq Pols_C$. More precisely: $\forall(param \leftarrow pii) \cdot Prefs_S[pii] \trianglerighteq sp_{param \leftarrow pii} \trianglerighteq Pols_C[param]$. Generating a sticky policy is about finding an instance that satisfies all privacy constraints defined in preferences and policies.

Depending on the use case, the sticky policy may have to be signed by one or both parties to ensure integrity, authentication of origin, and non-repudiation.

4.7 Attach Sticky Policy

The link between the sticky policy and the data it applies to has to be preserved when communicated, when stored in databases, and when the data is shared further (i.e. downstream). Depending on the trust model, different mechanisms can be used. For

instance, Enterprise Right Management [Mic09], could bind sticky policies (licenses) to personal data (document).

4.8 Domain Specific Languages

Multiple representation of a policy language can be envisioned: XML representation, English assertions, predefined options (check boxes), or graphical. Those representations have to be translated to the underlying language. Retrieved policies and results from reasoning (e.g. sticky policy, mismatching information) need a valid translation to the representation chosen by the PII Provider [Rah10].

Chapter 5

A Closer Look at *PII Consumer* Role

This section provides more insight on components necessary to fulfill the "PII Consumer" role in Figure 4.

This section illustrates the general steps a PII consumer has to undertake internally in order to support a privacy-preserving protocol we described in Section 3. Again, this figure is intended to be a block diagram, so it is neither just a workflow nor a collection of software components, but rather a mixture of both.

The Metadata Provider is the matching part to the meta-data request on the PII provider side. It provides the requester with a functional description of the service and optionally with the SLA. In this context it is important that meta-data states information about certification, the PII requirements, and the privacy policy. The information is gathered from the service implementation itself, e.g., could be part of a WSDL document, and from the policy store. The meta-data can be static, which means each caller gets the same information, or the meta-data can be dynamic, in which case (part of) the information is specific to the context of the request. For instance, different callers (PII providers) may get different privacy policies because the PII consumer shares individual legal agreements with the various PII providers. Another example is that the policy depends on the geographical region of the caller.

Now, the PII provider digests this information, selects the right PII data, matches preferences and policies. Finally, the PII provider invokes the PII consumer's service and thus submits the requested PII and a sticky policy. The PII provider must verify that the sticky policy indeed matches its policy. This check is necessary to avoid service errors or even legal implications through a wrong sticky policy. The sticky policy could, for instance, disregard the PII consumer's policy and define arbitrary obligations for the PII consumer.

Since we consider sending the PII and the sticky policy as a single step, the PII consumer's answer is the result of the service invocation.

During the execution of the service or thereafter the PII Consumer may store the data and likewise the sticky policy. Moreover the PII consumer needs to establish a link

between both data items. That is, the service provider needs to remember which PII is associated with which sticky policy and vice versa. The PII is stored in the PII store, the sticky policy is kept in the SP store. The link between both goes either as a reference to both stores to allow a bidirectional mapping or is kept in a dedicated data structure that holds references to the respective entries in PII store and SP store.

Even after the service invocation, the PII consumer may want to take advantage of the collected PII and use this data for an allowed purpose. We distinguish two ways to use these data: it might be used for local purposes or it might be passed on to a third party. Local use means that the data is used inside the trust domain of the PII consumer, e.g., by a second service that operates on the same PII store. In this case the PII consumer has to verify if the data is allowed to be used for the given purpose. A different case is passing on the data to a third party, i.e. a PII consumer in a different trust domain. This is the moment when the PII consumer switches its role to become a PII provider. In both cases the verification access rights to stored PII is performed by an access control engine, which enforces the authorization. The sticky policy associated with each piece of PII helps the authorization engine to decide whether the user agreed to this purpose or not.

But besides guarding collected data whenever it is accessed, the PII consumer has to follow-up on obligations that were agreed with the PII provider. It has to react to events (scheduled or relevant) and execute appropriated actions. We foresee two types of infrastructure component in a PII provider-side obligation enforcement engine. First, there are action handlers. The action handlers are the link to the legacy systems in the PII provider's IT infrastructure, e.g., a database or mail server. Their duty is to execute actions on legacy applications such as sending notification, logging, or deleting data. The second component type is the event handler. It is responsible for handling events from legacy systems that are relevant from a privacy point of view. Typical events are a time-based events and data access events from the legacy system.

An interesting extension is the logging of all obligation enforcement actions and access control decisions. A formalized logging would allow that a trusted third party, such as a accredited auditor, compares these logging data with the obligations in the sticky policy. An automated matching process would enable the auditor to see which obligations were kept by the PII provider and certify the PII consumer accordingly.

5.1 Provide Metadata

Each PII Consumer C must provide metadata about it's service. This paper only focuses on data collected as parameters $param$ of an interface api_C and associated policy $Pols_C[param]$. The PII Consumer has to enforce its policy, i.e. $Pols_C \trianglerighteq Behavior_C$. In other words, the PII Consumer has to 1) check that its policy is enforceable, e.g. not committing to delete data within one day when some specific execution may require one week, and 2) enforce (sticky) policies.

5.2 Check Sticky Policy

When personal data pii is assigned to parameter $param$ with sticky policy $sp_{param \leftarrow pii}$, the PII Consumer C has to check that this is a valid sticky policy, i.e. $sp_{param \leftarrow pii} \trianglerighteq Pols_C[param]$. Otherwise, a malicious PII Provider may provide sticky policies with insufficient rights or with too strict obligations. This check can be part of a mutual commitment protocol.

5.3 Authorization Decision

Checking authorization before using collected personal data is often necessary. This step can be skipped in static settings where policies do not evolve and service execution cannot violate the policy. Authorization decision regarding action a on data pii results in checking $sp_{pii} \trianglerighteq Behavior(a, pii)$. There are mainly two types of actions: 1) using locally (within PII Consumer's trust domain) collected data and 2) sharing collected data with a third party.

5.4 Local Use

Local Use refers to the use of personal data within the trust domain of the PII Consumer for a given purpose. This also covers data controller sharing data with a data processor under its control.

5.5 Data Sharing

Data Sharing is the action of sharing collected data with a third party (downstream PII Consumer). In this case the data controller C (formerly acting as PII Consumer) acts as a PII Provider and the third party acts as a PII Consumer C'. The main difference between P providing pii to C and C sharing pii with C' is that P can override her preferences $Prefs_P[pii]$ to have a match with $Pol_C[param]$ while C cannot override $SP_{param \leftarrow pii}$.

5.6 Composing Sticky Policies

Personal data can be merged and extracted. Computing the policy of resulting personal data is not straightforward and is out of the scope of this paper.

When combining data pii_1 with sticky policy sp_1 and data pii_2 with sticky policy sp_2, we have $pii_{1,2} = f(pii_1, pii_2)$ and corresponding sticky policy $sp_{1,2}$ where $sp_1 \trianglerighteq sp_{1,2}$ and $sp_2 \trianglerighteq sp_{1,2}$. Note that this does not apply when the resulting data $pii_{1,2}$ is structured and makes it possible to refer to initial data (e.g. pii_1) and different sticky policies can thus be applied to different part of $pii_{1,2}$.

When data pii_b (e.g. street name) is extracted from pii_a (e.g. address) with sticky policy sp_a, the sticky policy sp_b to apply to $pii_b = g(pii_a)$ must satisfy $sp_a \trianglerighteq sp_b$. In other words, composing data cannot increase permissiveness.

5.7 Obligation Enforcement

A set of well-defined obligations has to be available to let PII Provider and PII Consumer agree on the PII Consumer's obligations. We define an obligation as a pair (a, T), where a is an action and $T = (t_1, t_2, \ldots, t_n)$ is a set of triggers, meaning "*Do action a when triggers* $\in (t_1, t_2, \ldots, t_n)$". The element "action" defines the action to execute in order to fulfill the obligation and elements "triggers" specify the events and conditions requiring the execution of this action. For instance data retention of one year could be expressed as $(Delete(thisPII), (AtTime(now, now + 365d)))$.

5.8 Action Hander

Action Handler is a mechanism to implement the enforcement of actions to execute in order to fulfill obligations. Different actions can result from an obligation: logging, deleting data, notifying data subject, etc. Since it is not possible to define an exhaustive list of actions, domain-specific extensions should be possible.

For instance, in PPL [Pri10a], the following actions are predefined:

- ActionSecureLog(integrityLevel, confidentialityLevel, nonRepudiationLevel, timeStampingLevela, availabilityLevel): the action of logging specific events related to personal data.
- ActionDeletePersonalData: the action of deleting personal data.
- ActionAnonymizePersonalData: the action of removing identifiers from personal data
- ActionNotifyDataSubject(media, address): the action of notifying data subject about specific events related to personal data.

5.9 Event Handler

Event Handler is a mechanism to trigger actions in order to fulfill obligations. Different events can trigger an obligation: scheduler, access decision, action on data, sharing data, request from data subject, violation of an obligation, etc. Since it is not possible to define an exhaustive list of triggers, domain-specific extensions should be possible.

For instance, in PPL [Pri10a], the following triggers are predefined:

- *TriggerAtTime(start, maxDelay)*: executes once the associated action at some time between start and start + maxDelay.
- *TriggerPeriodic(start, end, maxDelay, period)*: executes once per period the associated action.
- *TriggerPersonalDataAccessedForPurpose(purpose, maxDelay)*: executes the associated action each time the personal data is used for specified purposes.

- *TriggerPersonalDataDeleted(maxDelay)*: executes the associated action when the personal data is deleted.
- *TriggerPersonalDataSent(thirdParty, maxDelay)*: executes the associated action when the personal data is shared with a third party.
- *TriggerDataLost(maxDelay)*: executes the associated action in case of major issue leading to data theft.
- *TriggerOnViolation(obligation, maxDelay)*: executes the associated action in case of violation of the referenced obligation.

5.10 Log and Audit

When each privacy-relevant action is logged, an internal or external auditor can verify that the behavior observable in trace $Behavior_{log}$ is compliant to the policies. In other words $\forall pii \in PII_C \cdot \forall b_{pii} \in Behavior_{log}[pii] \cdot SP_C[pii] \trianglerighteq b_{pii}$ where b_{pii} is a behavior related to data *pii*. When data are shared with third parties, it is necessary to take their policy into account or to perform a distributed audit.

5.11 Trust Model

The abstract framework presented in this paper requires that PII Consumer enforces (sticky) policies. This trust model can be implemented with different mechanisms: 1) PII Provider may know that PII Consumer cannot afford decreasing its reputation, 2) PII Consumer may be audited and certified, and 3) PII Consumer may prove that it is relying on trustworthy hardware (e.g. TPM) and software.

Chapter 6

Comparison of Privacy Policy Technologies in SOA

The *Abstract Privacy Policy Framework* (see Chapter 3) defines an ideal setting to enforce privacy policies in *Service Oriented Architectures* (SOA). In this chapter, the abstract framework is instantiated with concrete technologies in order to compare them. More precisely, criteria emerging from the abstract framework are used to compare existing privacy policy technologies and to evaluate their relevance to implement privacy policies in SOA.

6.1 Scope of the Evaluation

Several parts of the abstract framework are covered by existing privacy-enhancing technologies. Table 2 summarizes the evaluation of five privacy policy technologies (APPEL + P3P, PrimeLife Policy Language, SecPAL for Privacy, remote configuration of access control, and PRIME data handling policy) with around fifty criteria derived from components of the abstract framework. We have chosen criteria corresponding to key features of the eCV scenarios, possible extensions, and its generalization as abstract privacy policy framework. The list of criteria covers major issues when applying privacy policies to service oriented architecture but could be extended with other aspects of privacy or more detailed criteria. Each cell of this table is briefly described in Appendix A. This appendix provides for all evaluation criteria: 1) a short description of the criterion, 2) a description of each defined value for the evaluation of the implementation, 3) a description of each defined value for the evaluation of possible extensions, 4) an evaluation of each technology according to those criteria.

6.1.1 Evaluated Technologies

This section briefly describes each privacy policy technology that is (or may be) evaluated, i.e. each column of Table 2.

The first instantiation of the abstract framework is based on a combination of two well-known standards: privacy preferences are expressed with *APPEL* (A P3P Preference Exchange Language) [W3C02] and privacy policies are expressed with *P3P* (the Platform for Privacy Preferences Project) [W3C06]. Enforcement may rely on other technology such as *EPAL* (Enterprise Privacy Authorization Language) [AHK+03].

The second instantiation of the abstract framework is based on *PrimeLife Policy Language* (PPL) [Pri09a, Pri10a] an extension of XACML [Ris10] with support for data handling. This technology is well aligned with the evaluation criteria since a large part of them were informally taken into account during its design. PPL is also the language used in the prototype demoing privacy in service oriented architecture (see Chapter 8).

The third instantiation of the abstract framework is based on *SecPAL for Privacy* (S4P) [BMB10] an extension of logic-based authorization language SecPAL [BFG09]. Logic foundations makes it possible to reason on the causes of mismatches and furthermore to propose modification of preferences and/or policies thanks to abduction queries [BMD09].

The fourth instantiation refers to *remote management of access control policies* (AC) by the Data Subject at the Data Controller. In this setting, the data subject uploads her data to a data controller and configures the access control policy that must be enforced by this data controller. The evaluation assumes an expressive access control language, i.e. XACML [Ris10]. This approach can be considered as "inadequate" [KA10] to enforce privacy but is largely used. For instance *OAuth* [HL10] and *User-Managed Access* (UMA) [Kan] offer remote management of access control policies.

The fifth instantiation is based on the *PRIME Data Handling Policy* (PDH or PRIME-DHP) [ACDCdVS08]. This language is focusing on privacy policies but lacks important features to enable multi-hop data handling.

In this evaluation, we decided not to address technologies related to *Usage Control* and *Right Expression* such as *eXtensible rights Markup Language* (XrML) [Con02], *Obligation Specification Language* (OSL) [PSSW08], MPEG-21 REL [Wan04], or *Open Digital Rights Language* (ODRL) [ODR02]. Even if those technologies could be used to express and enforce privacy constraints on personal data, the way constraints are agreed upon is fundamentally different than what is required to implement privacy in service oriented architectures. Indeed, in usage control and rights management, constraints are imposed by the author (i.e. the data subject) without preliminary protocol with the party receiving the data. As a result key features such a preferences, policies, and matching algorithm are out of scope.

6.1.2 Evaluation Criteria

This section describes all criteria used to evaluate relevence of privacy policy technologies in SOA, i.e. all rows of Table 2.

PII Provider's Preferences (Sect. 3.3)

- *Simple Syntax*: Privacy preferences are expressed in a human-readable language. Syntax and semantics are well defined and can be processed by machines. See details in Appendix A.1.1.

Abstract features / Concrete technologies	A+P		PPL		S4P		AC	PDH	
PII Provider's Preferences (Sect. 3.3)									
- Simple Syntax	●	●	◐	◐	●	●	○	○	◐
- Can Express Access Control	◐	◐	●	●	●	●	○	○	◐
- Can Express Expected Data Handling	◐	◐	●	●	●	●	○	○	◐
- Can Express Expected Downstream Access Control	◐	◐	●	●	●	●	○	○	◐
- Can Express Expected Downstream Data Handling	◐	◐	●	●	●	●	○	○	○
- Can Take Downstream Path into Account	◐	◐	●	●	◐	◐	○	○	○
- Can Retrieve Applicable Preferences (Sect. 3.6)	○	◐	◐	●	○	●	○	○	◐
PII Consumer's Policy (Sect. 3.4)									
- Simple Syntax	◐	●	◐	◐	●	●	○	◐	◐
- Can Express Claims (Credentials)	◐	◐	●	●	●	●	○	●	●
- Can Express Data Handling	◐	◐	●	●	●	●	○	●	●
- Can Express Downstream Claims (Credentials)	◐	◐	◐	●	●	●	○	●	●
- Can Express Downstream Data Handling	◐	◐	○	●	●	●	○	◐	◐
- Can Retrieve Applicable Policy (Sect. 3.7)	◐	◐	◐	●	○	●	○	●	●
PII Store (Sect. 3.5)	○	◐	◐	◐	○	●	●	○	●
Privacy-Aware Service Discovery (Sect. 4.1)	○	◐	○	●	○	◐	○	○	○
PII Lookup (Sect. 4.2)	○	◐	◐	●	○	◐	◐	○	◐
Policy Matching (Sect. 4.3)									
- Has Logic Foundations	○	◐	○	◐	●	●	○	○	◐
- Takes Data Handling into Account	◐	◐	●	●	●	●	○	○	◐
- Takes Obligations into Account	◐	◐	●	●	◐	●	○	○	◐
- Takes Downstream Properties into Account (One Hop)	◐	○	◐	●	●	●	○	○	○
- Supports Recursive Downstream	○	○	◐	●	●	●	○	○	○
PII Selection (Sect. 4.4)	○	○	●	●	○	●	○	◐	◐
Change Preferences (Sect. 4.5)									
- Can Show Mismatches	○	◐	◐	◐	○	●	○	○	○
- Can Suggest Modifications	○	◐	◐	◐	○	◐	○	○	○
Sticky Policy (Sect. 4.6)									
- Optional Sticky Policy	●	●	○	●	●	●	○	◐	◐
- Can be Expressive	○	○	●	●	○	●	●	●	●
- Supports Signature or Commitment	○	○	○	●	○	●	◐	○	●
- Can Change Sticky Policy	○	○	◐	●	○	◐	●	○	○
- Can Store and Retrieve Sticky Policy (Sect. 3.8)	○	○	◐	●	○	●	●	◐	●
Attach Sticky Policy (Sect. 4.7)	○	○	◐	●	○	●	◐	◐	●
High-Level Policy Language (Sect. 4.8)									
- Same Language for Preferences and Policies	○	○	●	●	◐	◐	○	○	○
- Language Expressiveness	○	◐	●	●	●	●	●	●	●
- Clear Separation of Obligations and Rights	○	○	●	●	◐	◐	●	●	●
Check Sticky Policy (Sect. 5.2)	○	○	○	●	○	●	●	●	●
Authorization Decision (Sect. 5.3)									
- Enforces Local Use, e.g. Purpose (Sect. 5.4)	○	●	●	●	○	●	◐	●	●
- Enforces Access Control when Sharing (Sect. 5.5)	○	◐	●	●	○	●	●	●	●
- Checks Downstream Data Handling when Sharing	○	○	●	●	○	●	○	○	○
- Attach (New) Sticky Policy when Sharing	○	○	●	●	○	●	○	◐	◐
Composing Sticky Policies (Sect. 5.6)	○	◐	○	◐	○	◐	◐	○	○
Obligations (Sect. 5.7)									
- Supports Enforcement of Obligations	○	◐	●	●	●	●	◐	◐	●
- Checks Rights of Enforcing Obligations	○	○	○	◐	●	●	○	◐	◐
- Specifies Action Handler (Sect. 5.8)	○	◐	●	●	○	●	●	◐	◐
- Specifies Event Handler (Sect. 5.9)	○	◐	●	●	○	●	◐	◐	◐
Log and Audit (Sect. 5.10)	○	◐	◐	◐	○	●	◐	◐	◐
Trust Model (Sect. 5.11)	○	○	○	◐	◐	●	◐	○	◐
Protocol independent (HTTP, WS)	○	●	●	●	◐	●	●	◐	●
Policy for Implicit PII (e.g. IP address)	●	●	◐	●	○	●	○	○	●

Table 2: Instantiations of the abstract framework. Features are rated as completely implemented ●, partially implemented ◐, or not implemented ○. The second column refers to features that could be implemented without breaking changes ●, that could be partially implemented ◐, or that would require important changes ○.

- *Can Express Access Control*: The language used to express privacy preferences supports access control, i.e. the data subject can specify which (or what kind of) data controllers can get a given type of personal data. See details in Appendix A.1.2.

- *Can Express Expected Data Handling*: The language used to express privacy preferences lets the PII Provider specify how collected data must be handled by the PII Consumer. See details in Appendix A.1.3.

- *Can Express Expected Downstream Access Control*: The preferences can express access control constraints on third parties. In other words, the preferences specify with what kind of third parties the data controller is authorized to share collected data. See details in Appendix A.1.4.

- *Can Express Expected Downstream Data Handling*: The preferences can express data handling constraints on third parties. In other words the preferences specify how third parties are expected to handle data they would get from data controllers. See details in Appendix A.1.5.

- *Can Take Downstream Path into Account*: Privacy constraints that apply to data controllers downstream depend on the path. In other words, it is possible to have different privacy constraints for personal data d at service S when S is a data controller directly collecting d, when S acts as downstream data controller and gets d from data controller S_1, or from data controller S_2. See details in Appendix A.1.6.

- *Can Retrieve Applicable Preferences (Sect. 3.6)*: This technology provides a mechanism to get the privacy preferences that apply to a piece of personal data. This mechanism supports different types of personal data: retrieved from a PII Store (e.g. a database), dynamically created by the user (e.g. free text in a HTML Form), or certified (e.g. attributes of credentials). See details in Appendix A.1.7.

PII Consumer's Policy (Sect. 3.4)

- *Simple Syntax*: Privacy policies are expressed in a human-readable language. Syntax and semantics are well defined and can be processed by machines. See details in Appendix A.2.1.

- *Can Express Claims (Credentials)*: The policy language can describe trust level and certification of PII Consumers. For instance, it is possible to link Public Key Infrastructure to the policy. See details in Appendix A.2.2.

- *Can Express Data Handling*: Privacy policies can express proposed data handling in terms of purpose, obligations, etc. In other words, PII Consumers express how collected data will be handled. See details in Appendix A.2.3.

- *Can Express Downstream Claims (Credentials)*: The policies can express credentials of third parties. In other words the policy specifies with what kind of third parties the data controller may share collected data. See details in Appendix A.2.4.

- *Can Express Downstream Data Handling*: The policies can express proposed data handling of third parties. In other words the policy specifies how third parties would handle data they may get from the data controllers. See details in Appendix A.2.5.

- *Can Retrieve Applicable Policy (Sect. 3.7)*: There is a mechanism to get or generate the policy applicable to a given parameter, e.g. one "label" of an HTML Form, one parameter of a Web Service, or one claim of a requested credential. See details in Appendix A.2.6.

PII Store (Sect. 3.5)

- Personal data are stored in a database and can be queried according to attributes such as the type of data (e.g. e-mail address) or its certification (e.g. name in identity card). See details in Appendix A.3.

Privacy-Aware Service Discovery (Sect. 4.1)

- This technology provides mechanisms to discover services based on functional properties and on non-functional properties such as privacy. See details in Appendix A.4.

PII Lookup (Sect. 4.2)

- This technology offers mechanisms to gather pieces of personal data that are required by a given interface of the PII Consumer. See details in Appendix A.5.

Policy Matching (Sect. 4.3)

- *Has Logic Foundations*: The evaluation whether privacy policies do fulfill privacy preferences has logic foundations. See details in Appendix A.6.1.

- *Takes Data Handling into Account*: Expected data handling is expressed by the PII Provider and proposed data handling is expressed by the PII Consumer. Both aspects are taken into account during match. See details in Appendix A.6.2.

- *Takes Obligations into Account*: Expected obligations are expressed by the PII Provider and proposed obligations are expressed by the PII Consumer. Both aspects are taken into account during match. See details in Appendix A.6.3.

- *Takes Downstream Properties into Account (One Hop)*: Not only the privacy policy of the data controller is taken into account during match but also the policy of third parties, which may get subsequently access to the personal data. See details in Appendix A.6.4.

- *Supports Recursive Downstream*: Complex chains of downstream data sharing can be expressed in privacy policies and preferences and can be taken into account during match. See details in Appendix A.6.5.

PII Selection (Sect. 4.4)

- Privacy-aware identity selection is supported by the protocol (i.e. privacy policies are specified for all expected claims and privacy preferences are associated with all issued claims) and by the user interface (i.e. the selection of claims takes privacy into account). See details in Appendix A.7.

Change Preferences (Sect. 4.5)

- *Can Show Mismatches*: In case of mismatch, the root causes of the mismatch can be identified and highlighted. See details in Appendix A.8.1.

- *Can Suggest Modifications*: Privacy preferences can be automatically updated to get a match next time a similar case occurs. Previous changes, and similarity of preferences can be taken into account. See details in Appendix A.8.2.

Sticky Policy (Sect. 4.6)

- *Optional Sticky Policy*: Instead of creating a sticky policy describing agreed privacy constraints on personal data, a Boolean response can be used to state that the privacy policy is acceptable and must be enforced. The Boolean response can be implicit, e.g. agree by sending personal data. See details in Appendix A.9.1.

- *Can be Expressive*: The sticky policy can express complex constraints with conditions. See details in Appendix A.9.2.

- *Supports Signature or Commitment*: The sticky policy can be signed by one or more parties to ensure non-repudiation of agreed privacy constraints. See details in Appendix A.9.3.

- *Can Change Sticky Policy*: There is a mechanism to let data subjects modify sticky policies associated with their own personal data when such an action is authorized. See details in Appendix A.9.4.

- *Can Store and Retrieve Sticky Policy (Sect. 3.8)*: There is a mechanism to store sticky policies and to query the sticky policy associated with a given piece of personal data. See details in Appendix A.9.5.

Attach Sticky Policy (Sect. 4.7)

- Mechanism to attach the sticky policy to data on the wire and in databases. Mechanisms such as Enterprise Rights Management (e.g. [Mic09]) would be an example where personal data cannot be decrypted without acknowledging the sticky policy (i.e. licenses). See details in Appendix A.10.

High-Level Policy Language (Sect. 4.8)

- *Same Language for Preferences and Policies*: Privacy preferences, policies, and sticky policies are expressed in a common language that avoid semantics mismatches. See details in Appendix A.11.1.

- *Language Expressiveness*: The common language is expressive and allows the specification of conditions, nested or recursive policies, and variables. See details in Appendix A.11.2.

- *Clear Separation of Obligations and Rights*: Obligations and rights are clearly expressed to handle, for instance, the right to store personal data 3 months, the obligation of storing data 3 months, and the obligation of deleting data within 3 months. See details in Appendix A.11.3.

Check Sticky Policy (Sect. 5.2)

- When a sticky policy is pushed to a PII Consumer, this one can check whether the sticky policy is acceptable, i.e. more permissive than the related policy. See details in Appendix A.12.

Authorization Decision (Sect. 5.3)

- *Enforces Local Use, e.g. Purpose (Sect. 5.4)*: Before using collected data, the PII Consumer can verify that actions are authorized according to sticky policies. See details in Appendix A.13.1.

- *Enforces Access Control when Sharing (Sect. 5.5)*: Authorization of sharing data with a third party takes into account the sticky policy and attributes (e.g. certificates) of the third party. See details in Appendix A.13.2.

- *Checks Downstream Data Handling when Sharing*: Authorization of sharing data with a third party takes into account the sticky policy of the personal data and the privacy policy of the third party. See details in Appendix A.13.3.

- *Attach (New) Sticky Policy when Sharing*: A new sticky policy is created when the personal data is shared with a third party. The rights and obligations of a third party may be different than the rights and authorizations of the initial data controller. See details in Appendix A.13.4.

Composing Sticky Policies (Sect. 5.6)

- Possibility of computing the resulting sticky policy $sp_{1,2}$ of personal data $pii_{1,2}$ resulting from the combination of multiple personal data. In other words, defining each $sp_{1,2} = F(sp_1, sp_2)$ for each way of combining $pii_{1,2} = f(pii_1, pii_2)$. See details in Appendix A.14.

Obligations (Sect. 5.7)

- *Supports Enforcement of Obligations*: There are mechanisms to automatically enforce obligations that can be specified in (sticky) policies. See details in Appendix A.15.1.

- *Checks Rights of Enforcing Obligations*: Mechanisms to define lower bound and upper bound of behavior. See details in Appendix A.15.2.

- *Specifies Action Handler (Sect. 5.8)*: There are mechanisms to parse and execute actions associated with obligations. It is possible to extend the set of actions that are handled. See details in Appendix A.15.3.

- *Specifies Event Handler (Sect. 5.9)*: There are mechanisms to parse triggers and react to specific events (time, event) leading to the execution of an action. It is possible to extend the set of triggers that are handled. See details in Appendix A.15.4.

Log and Audit (Sect. 5.10)

- There are mechanisms to log privacy-relevant events such as: use of personal data, authorization decisions, obligation enforcement, etc. Audit can be based on those traces. See details in Appendix A.16.

Trust Model (Sect. 5.11)

- Support for different trust models such as certification, audit, reputation, and/or trusted hardware. This makes the link between the committed behavior and the actual behavior. See details in Appendix A.17.

Protocol independent (HTTP, WS)

- It is possible to use the evaluated language and associated mechanisms with different communication protocols (Web Services, HTTP, etc.) and to define separately protocol-specific aspects (e.g. cookies). See details in Appendix A.18.

Policy for Implicit PII (e.g. IP address)

- It is possible to specify how PII Consumer handles personal data that are implicitly collected (e.g. IP address). See details in Appendix A.19.

6.2 Results and Conclusion

Results of the evaluation are summarized in Table 2 and detailed in Appendix A.

It appears that using P3P [W3C06], APPEL [W3C02], and EPAL [AHK+03] together is not suitable to tackle complex scenarios. First those technologies do not support multi-hop data handling, which is quite common in SOA. This is mainly due to the fact that those technologies were mainly targeting Web 1.0 scenarios. Second the use of three different languages for expressing privacy preferences, privacy policies, and enforcement leads to semantics mismatches and difficulty to use them recursively.

Letting data subjects specify access control on their data (e.g. OAuth [HL10], UMA [Kan]) is not sufficient even when obligations can be specified (e.g. XACML [Ris10]). The main limitation is due to the fact that remote setting of access control only covers a small subset of data handling. One advantage of this approach is to limit the number of copies of personal data and to centralize their management.

PRIME-DHP [ACDCdVS08] provides more features than P3P but does not address the preference side and complex downstream cases.

S4P [BMB10] offers promising features but only provides the core functionality (evaluation of queries) and lacks implementation of tools for creating sticky policies, for enforcing policies, and for auditing execution traces.

Finally PPL [Pri09a] supports a large part of the abstract privacy policy framework. This is not surprising since notions of the abstract privacy policy framework were already taken into account when designing PPL (PrimeLife Work Package 6.3 on SOA and Activity 5 on Policies are working together from the beginning). PPL mainly lacks homogeneity and logic foundations to enable reasoning on the policies.

Part II

eCV Demonstrator

Chapter 7

eCV Scenario

PrimeLife experimented with a demonstrator prototype in order to evaluate the challenges in turning a common SOA application into a privacy-preserving SOA application. Moreover, the demonstrator scenario was shaped in a way that it gives room to showcase many privacy-enhancing extensions, especially w.r.t. the PPL policy engine from Activity 5 [Pri09a]. The electronic CV scenario was already presented it in PrimeLife reports H6.3.1 [MS09] and D6.1.1 [Pri09b]. At the point of writing the latter report we could only present concepts and early technical designs. This document gives much more insight into the whole technical implementation and details such as protocols and applied technologies.

This chapter describes the "eCV scenario" we selected for this kind of demonstration and explains the privacy enhancements from a conceptual point of view. The next chapters will go into more technical detail on both aspects.

7.1 Roles and Workflows

Inga Vainstein is 46 years old and is currently working as journalist.[1] As a part of her job she is traveling to various countries. Inga makes heavy use of online applications for new job opportunities. She uses a platform called "eCV" to get job offers and apply for new positions in a convenient and easy way. In fact, this is one of her main motivations to collect all certificates and testimonials (also) in electronic form.

She uses this platform to collect all her digital certificates and documents. She attends professional trainings on a regular basis. For each completed course she gets a certification. Moreover she collects testimonials from former employers. Last year she won an award for her outstanding press story on identity theft.

The eCV platform allows to create many profiles based on these claims, e.g. one profile with an emphasis on her academic achievements and another profile with journalistic achievements. For each job offer she can decide which profile to send to the employer.

[1] The persona of Inga is taken from the list of PrimeLife personas [KWWD08]. Personas are fictional characters created to represent the different user types.

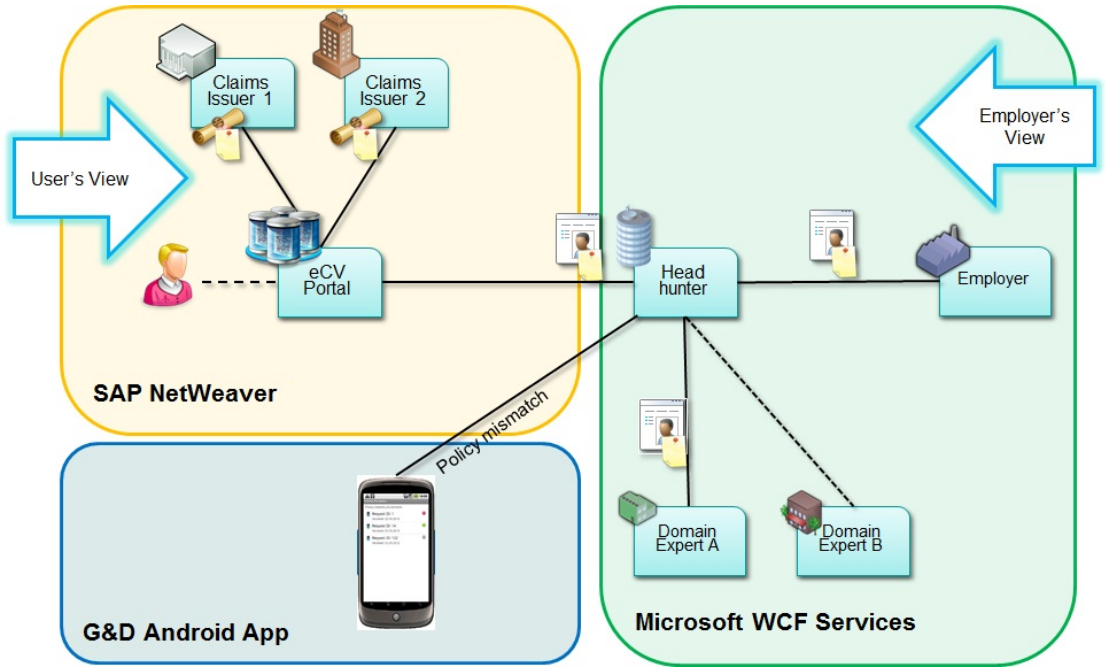


Figure 5: Roles and communication paths in the eCV scenario

Altogether the eCV scenario features five roles, which we will describe in more detail now. Please notice that only the roles of User and Employer are needed to showcase the scenario. The remaining roles act more or less autonomously, as we will see in Chapter 8.

Claims Issuer. The claims issuer certifies an attribute to the user. This attribute comes in the form of a digital claim to the user and is accompanied by a privacy policy. This policy is crafted by the claims issuer and defines how long the claim is valid and what rights and obligations are associated with it. The claim policy is expressed in the PrimeLife Policy Language (PPL). For instance, a former professor might write a recommendation letter (claim), but express the obligation that the user may not use it for an application outside this university. A former employer could state for another letter of recommendation that this latter may not be used for an application at the employer's main competitor. The claims come in the form of a "sticky policy" attached to the claims. This attaching is done by means of an API or schema that features two slots, one for the claim and one for the policy.

User. Main goal for the user is to enjoy the benefits of a privacy-friendly online job application portal. For this purpose she collects claims from the claims issuers. She stores these in the portal and combines them to profiles. One profile consists of a collection of claims, additional information, and a privacy policy. The eCV portal composes this policy from all individual polices attached to the claims. Moreover the user can scope

this automatically generated policy further down. She can add more obligations or grant fewer rights.

Employer. The employer generates a job offer. The intention of the employer is to hire somebody for an open position. Thus, the employer generates a description of the open position. The employer communicates through the headhunter with the eCV portal. That means, the employer gives the open position to the headhunter and leaves it to him to find a suitable candidate. The headhunter in turn uses the eCV portal (potentially as one of many means) to find this candidate. This setting allows us to simulate and experiment with a longer chain of downstream data controllers. Besides a description of the open position, the job offer document features a policy that the employer promises to adhere to in course of this hiring transaction. It is thus a service provider policy. Unlike the general PPL scenario this policy is not submitted via a kind of web service meta-data request, but is attached to the job offer document and communicated upstream through the headhunter to the eCV portal.

Headhunter. The headhunter is the central turning point in this scenario. The reason to introduce this extra layer of communication in the scenario is to make the scenario richer in terms of downstream data usage. Moreover, the user is not aware of this instance, which creates some interesting use cases for the reasoning on policies. The headhunter receives a job offer (with an attached sticky policy) from the employer and forwards it to the eCV portal. The portal in turn sends a job application to the headhunter. The headhunter is now responsible to evaluate the capabilities of the applicant on behalf of the employer. We assume that this evaluation needs domain-specific knowledge. Hence, the headhunter dynamically looks up a suitable domain expert service and hands over the job application for evaluation. Of course, the headhunter needs to make sure that the domain expert complies with the policy attached to the job application.

eCV Portal. The eCV portal is the interface to the user. It allows for requesting claims, administrating issued claims, creating of profiles, and definition of user's privacy policy. Furthermore it utilizes policy composition and policy matching.

Domain Expert. The domain expert receives a job application from the headhunter. Its job is to evaluate the skills of an applicant according the skill set demanded by the employer. It stores the data as maximally as long as the obligation allows it to. Primary objective is to showcase the automatic execution of obligations.

7.2 End-to-End Workflow

The general workflow in the eCV scenario looks as follows:

1. Policy Composition Phase
   (a) User requests claim from claim issuer
   (b) Claim issuer generates claim, defines policy and attaches it to the claim

(c) Claim issuer sends claim with attached sticky policy back to the user

(d) User stores claim and policy

(e) User creates a profile and adds a subset of her claims to it.

(f) eCV portal calculates the most permissive privacy policy from all claims policies. The user can scope this generated policy further down

2. Matching Phase

(a) Employer creates a job offer and attaches privacy policy of employer

(b) Employer sends both job offer and privacy policy to headhunter

(c) Headhunter forwards both job offer and employer's privacy policy to the eCV portal

(d) eCV portal matches periodically all job offer against the users' profiles. It performs a profile matching (requested skills vs. claimed skills) and a policy matching. The policy matching compares the sticky policy of the job offer (created by the employer) with the sticky policy of the profile (created by the user)

(e) eCV portal signals a successful match between offer and profile back to the user. The user decides to send out a job application

3. Downstream Phase

(a) eCV portal generates a job application from the profile and the job offer. It attaches the sticky policy that results from the comparison between job offer policy and user profile policy. Two samples of such a policy are shown at listings B.3 and B.4 of section B.2.

(b) eCV portal sends the job application with the attached sticky policy as shown at listing B.2 (section B.2) to the headhunter.

(c) The headhunter needs to find a domain expert to evaluate the job application. It looks up the privacy policies of all available domain experts.

(d) If the headhunter finds a domain expert whose privacy policy complies with the job application's sticky policy, it sends the job application to the domain expert. If it does not find such a domain expert, it initiates a policy conflict resolution on the user's interface (e.g., mobile phone). If both is not possible, the execution of the workflow stops here.

(e) Domain expert evaluates the proposal and stores the data. It triggers the obligation enforcement engine that will follow-up on any obligation the domain expert agreed to

(f) Domain expert sends an evaluation back to the headhunter

(g) In case this evaluation is positive, the headhunter communicates the user's job application on to the employer

(h) The employer visualizes the result, esp. the policy

7.3 Showcases

The eCV demonstrator was shaped in a way that it features various privacy showcases. We describe now the general privacy scenarios, c.f. Figure 6.

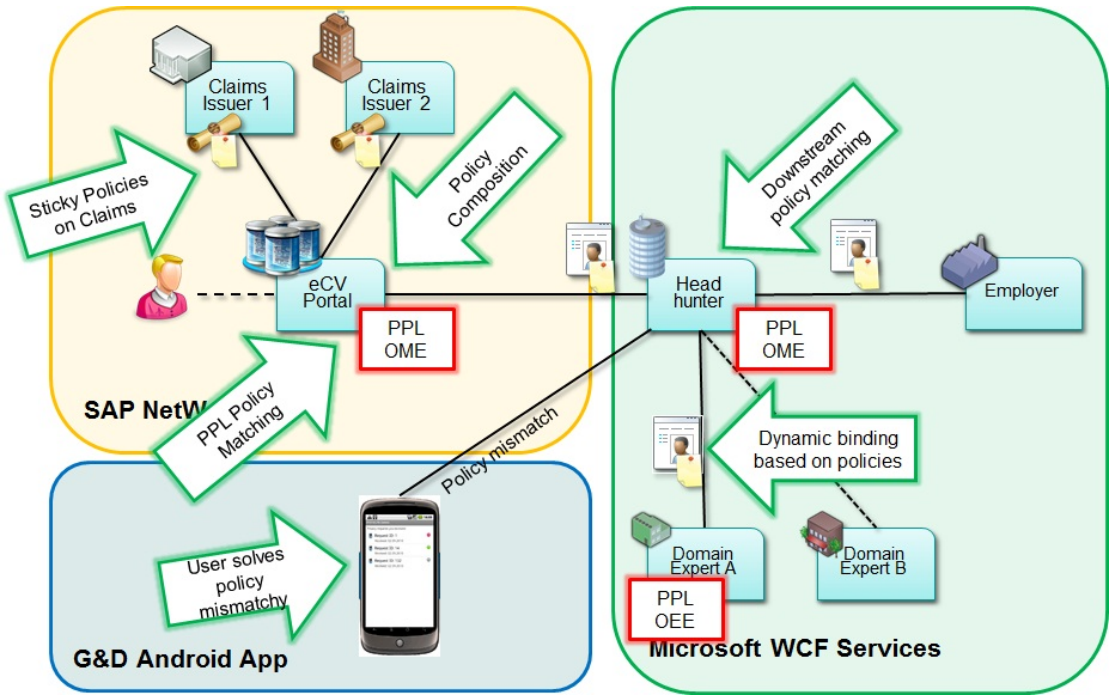


Figure 6: Showcases of the eCV Scenario

Sticky policies on claims. At various places throughout the scenario we communicate the actual data together with the policy. Most prominently, when the claim issuer submits claims to the user with policies attached. But also when the job application is communicated downstream together with a sticky policy reflecting the user's preferences, the claim issuers' policies, and the service policy of the employer.

Policy composition. In the data subject part of the demo we combine various documents (claims) into a single document (profile resp. job application). The policies sticking at the claims will be condensed into one single policy for the profile resp. job application. This composed policy can be further scope down by the privacy preferences of the user.

Policy matching with PPL. The demonstrator shows PPL policy matching at two places. Matching of rights is used inside the eCV portal. It compares the resulting sticky policy of a applicant's profile with the service policy of the employer. The focus is on the matching of rights. The obligation part of the PPL policy engine is used in the headhunter. Here, the eCV portal compares the sticky policy of the job application with

the service policies of the domain experts. The focus is on the obligation part, i.e. if the domain expert is willing to execute the obligations demanded by the user.

Downstream policy matching. Downstream matching shows the fact that an entity that has just acted as data consumer has to fulfill the due of a data provider from the very moment that it receives PII and a policy from a data provider.[2]

Dynamic policy-based binding. This part of the demo mimics the fact that data might be shared with entities that are not know a priori. We employ a dynamic service lookup and service binding of the domain expert service instances. The actual binding is based on policy meta-data which is exposed by the domain expert services.

Obligation Matching with PPL. The headhunter needs to find a matching policy service. For this reason the headhunter needs to compare the sticky PPL policy against the service policies of multiple domain experts. We focus here on the obligation part of the PPL language.

Obligation Enforcement with PPL. The domain expert stores the job applications for a certain time and accepts to fulfill any obligations expressed in the sticky policy, as long as it is compliant with its service policy. We implemented an obligation enforcement engine, that keeps track of such promises and executes them. That means, that data will be automatically deleted from the database after a given time.

Scenario Entity	**Privacy-specific Role**
Claims Issuer	Data Subject
User	Data Subject and Data Controller for the Claims Issuer
Headhunter	Data Controller
Domain Expert	Downstream Data Controller, ad-hoc bound
Employer	Downstream Data Controller, a priori known

Table 3: Generic roles in the scenario

Another way to look at the scenario is by considering the roles the various stakeholders take w.r.t. privacy. Table 3 summarizes them. Claims issuer and user are acting as data subject. Thus, they could be seen as one entity in the demo. In general, the user is the entity that shares data and whose privacy rights need to be supported. The downstream part starts with the headhunter and all subsequent services. The scenario is shaped in a way, that it features not only covers the disclosing of PII to a third party (User $\rightarrow$ Headhunter), but even a multi-hop data disclosure from headhunter to employer and from headhunter to domain expert. The data sharing via the headhunter to the employer is somewhat expected from the user's perspective. The user gets to see the service policy that was crafted by the employer and communicated upstream via the headhunter. However, the second data disclosure from headhunter to domain expert

[2]Refer to Chapter 3 for a detailed definition of data provider and data consumer in this context.

is unexpected and maybe even unintended by the user. It is very likely that the user's policy does not allow to share the job application with another entity besides headhunter and employer, or that the domain expert has a policy that does not fit at all with the user's sticky policy on the job application document.

7.4 Joint Implementation

The implementation of the eCV scenario was done jointly by PrimeLife partners European Microsoft Innovation Center, SAP Research and Giesecke & Devrient. SAP was responsible for the data subject part of the scenario. That includes claims issuing, the user's experience of the portal, collection of claims, and the generation of a common policy (policy composition). Microsoft took over responsibility for the downstream data handling part, i.e. investigated the data controller part. This includes dynamic binding of the domain experts, the overall workflow, evaluation of policy conflicts, policy editor and policy visualization, user experience at the employer, and connection to the mobile device. Giesecke & Devrient built a mobile eCV application that allows the user to override her own policy in case of a policy mismatch (Data subject policy mismatch interaction) This work is described in greater detail in D6.3.1 [Pri11]. We will shed some light on the communication between Headhunter and the mobile device in Chapter 8.

Chapter 8

Architecture

In this chapter we will describe the eCV scenario, which was introduced in Chapter 7, in a technical sense and will focus on its architecture. That is, how the structure of the scenario can be described, which participants are communicating with whom, and what their dependencies are. We will also move a bit deeper in details and show how every single participant acts at the particular phases of that scenario. We will discuss the implementation details in subsequent chapters.

8.1 Conceptual Properties

Before explaining the eCV scenarios architecture in detail, we will first introduce some conceptual properties that are related to all participants.

8.1.1 Development Environments

As outlined in Chapter 3, the development of the eCV demonstrator was split into two main parts to accommodate the parallel work of the particpating partners, namely SAP and EMIC. The following explains the technologies used by the aforementioned parties and how this fits into the overall scenario.

For the data subject side of the scenario, SAP decided to build the applications using SAP Netweaver Compostion Environment(CE) 7.1 in order to learn from applying European project concepts to industrial software. The SAP Netweaver CE platform allows for the building and running of applications based on Service-Oriented (SOA) principles and enables the development, modelling and design of web services (Java EE), user interfaces(Web Dynpro) and business process management (Process Composer).

On the data controller side, EMIC used standalone applications based on web service (WS-*) protocols and using Microsoft .NET WCF technology to build the web services endpoints. This standalone-applications-approach is meant for demonstration purposes. For scalability reasons EMIC recommandeds migrating this work to an application environment like Microsoft Biztalk.

8.1.2 Application Structure of Participants

SAP used Web Dynpro which is the SAP NetWeaver programming model for user interfaces and provides support when developing the Web representation of business applications. The Web Dynpro model is based on the Model View Controller paradigm [SUN], and has the following features that build on the standard dynpro model: clear separation of business logic and display logic, uniform metamodel for all types of user interfaces, execution on a number of client platforms, extensive platform independence of interfaces.

EMIC created the participants with a rapid development approach by building them as console applications at first. Subsequently, these console applications were enhanced by an additional graphical user interface. The applications' logic was extracted from console applications to libraries and used them to build console and graphical user interface applications. They kept the console applications still maintained as it facilitated debugging the applications.

By extracting the common logic to libraries, they ensured that the ongoing development covers both (all) types of applications that run the eCV scenario. Furthermore, it allows for an easier generation of additional application types, like server applications, e. g. using Windows Service, or even add-ins into an application server like Microsoft BizTalk.

8.2 Structure and Behavior of the eCV Scenario

Figure 7 shows the architecture of the eCV scenario. The participants eCV portal, headhunter, domain expert, employer, the secure mobile device, as well as the policy engines PME and PEE, that already have been introduced in the previous chapter, are presented in its structural details in the following sections.

8.2.1 Headhunter Tool

In this section, we go into details of the *headhunter's tool (HHT)*, explaining the communication structure and features first, and describing its internal behaviour next.

Talking to almost every other participant, the HHT has a central position in our schema. It provides a web service interface to receive either a job offer from the *employer's tool (EPT)* or a job application from the *eCV portal's tool (ECVPT)*. It also does web service calls to forward the job offer to the ECVPT, to forward application to a *domain expert's tool (DET)* or to send assessed applications to the EPT.

Multiple domain experts are considered by the HHT: Depending on the sticky policy attached to the application, HHT will automatically detect and choose a DET that is compliant to that policy and is therefore authorized to process the application's PII. We call this detection process *dynamic binding*, as it technically discovers and binds available services according to a given policy.

In order to be able to reach the data subject in case of user-assistance-needed mismatch via a secure mobile device (SMD), HHT provides additional web service interfaces over *SOAP*[W3C07] and *REST* [Fie00], that can be used from the SMD to get information about particular mismatches and react on these. The REST interface is additionally

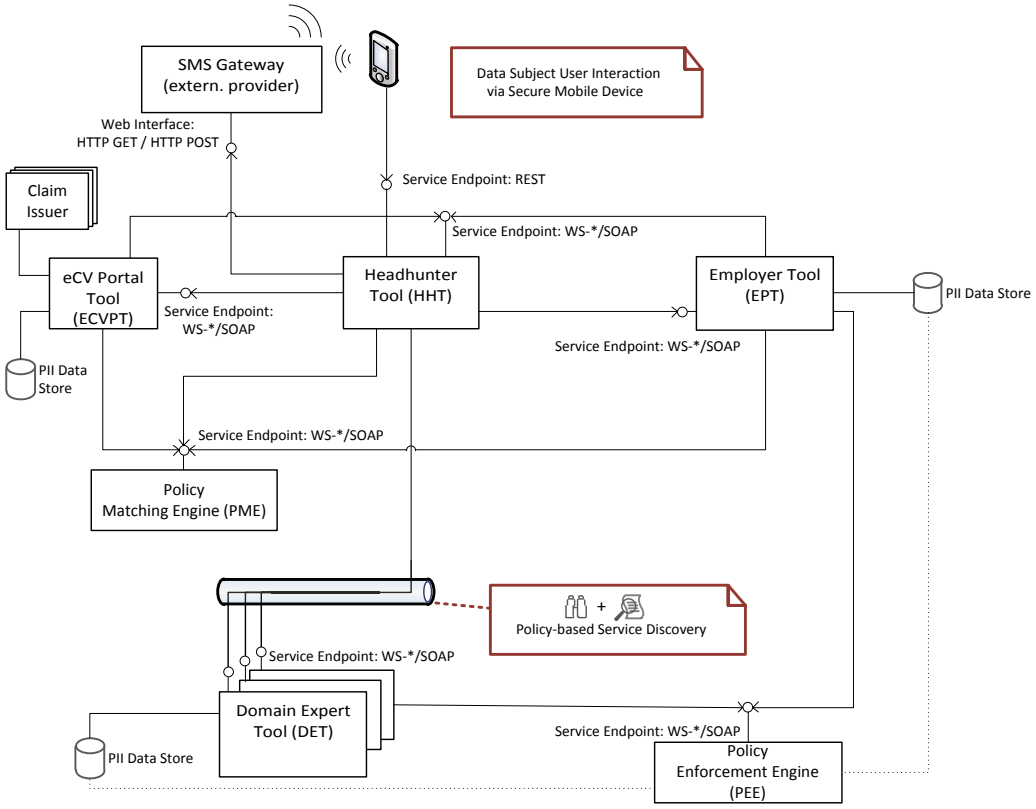


Figure 7: Architecture of the eCV scenario

provided to simplify the access for mobile devices to our eCV scenario. A SMD cannot be expected to be online and polling permanently for mismatches due to limited energy resources. To trigger a SMD, the headhunter tool therefore needs to send SMS messages via an external SMS gateway provider.

Looking at its internal behaviour, the HHT can be described best in a state machine as shown in figure 9.

At initialization, which is described as state *Pre-Init* in the figure, the HHT will trigger the dynamic binding facility to discover all available DET service instances. Figure 8 shows such a service discovery process. As this process requires waiting for a time-out, it is considered as fairly expensive and is therefore executed automatically only once, at the initial phase. But it still can be triggered manually while processing an application, e.g. in case when DETs are expected to join or leave after the HHT has started. However, the actual binding to a DET is processed in a separate step at the time when an application is being processed and cannot be done beforehand as it depends on the particular application's sticky policy.

Having done the initialization once, the HHT will switch to the *Idle* state and will wait for incoming job offers to resume its work. Job offers are received on a thread,

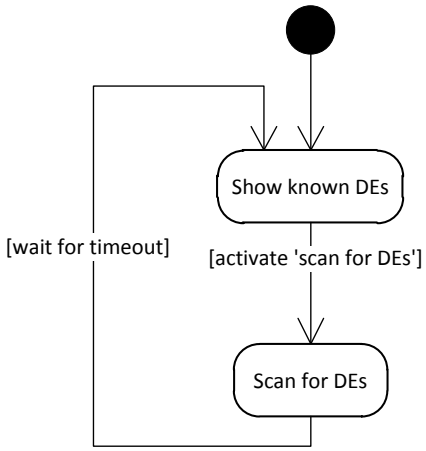


Figure 8: Service Discovery of Domain Experts at eCV scenario's headhunter tool

different from the main thread on which the HHT runs. This enables the feature that job offers can be received at any time and are being queued until the headhunter state machine is ready to process one after the other.

Having been received at least one job offer, the HHT will switch to its next state *Job offers available*.

Clicking then on the button no 1, which says *Send job offer to eCV portal*, the HHT will send the oldest job offer of its job offer queue to the ECVPT and changes to the *Waiting for application* state, waiting for applications from the ECVPT.

After having received suitable applications from the ECVPT, the HHT will do the actual dynamic binding before it is able to get the applications assessed by a DET. That is, it will ask every known DET service for its privacy policy and will match this policy against the job applications' sticky policy. For doing the policy matching, the PME is consulted as a service. In case of matching, the PME will respond positively and in case of a mismatch, it will provide a quite expressive mismatch description included in its negative response. If no DET service is available, which means that the service discovery of DETs from the initialization failed to find at least one DET, then HHT will jump to the *Stop* state. Also if the sticky policy of the application does not allow any downstream usage of the PII, further investigation of policies is no more necessary and the HHT will jump to the Stop state as well. There are two more cases to consider in this dynamic binding process: Either at least one DET fulfills the privacy policy of the application, in that case HHT will just pick one of them randomly, or no DET can be found to do so. In case of the latter, we will trigger applicant's secure mobile device (SMD) by sending a SMS message and will present the mismatch of its privacy preferences with the first domain expert that HHT has asked for its policy. Please note that it would certainly be

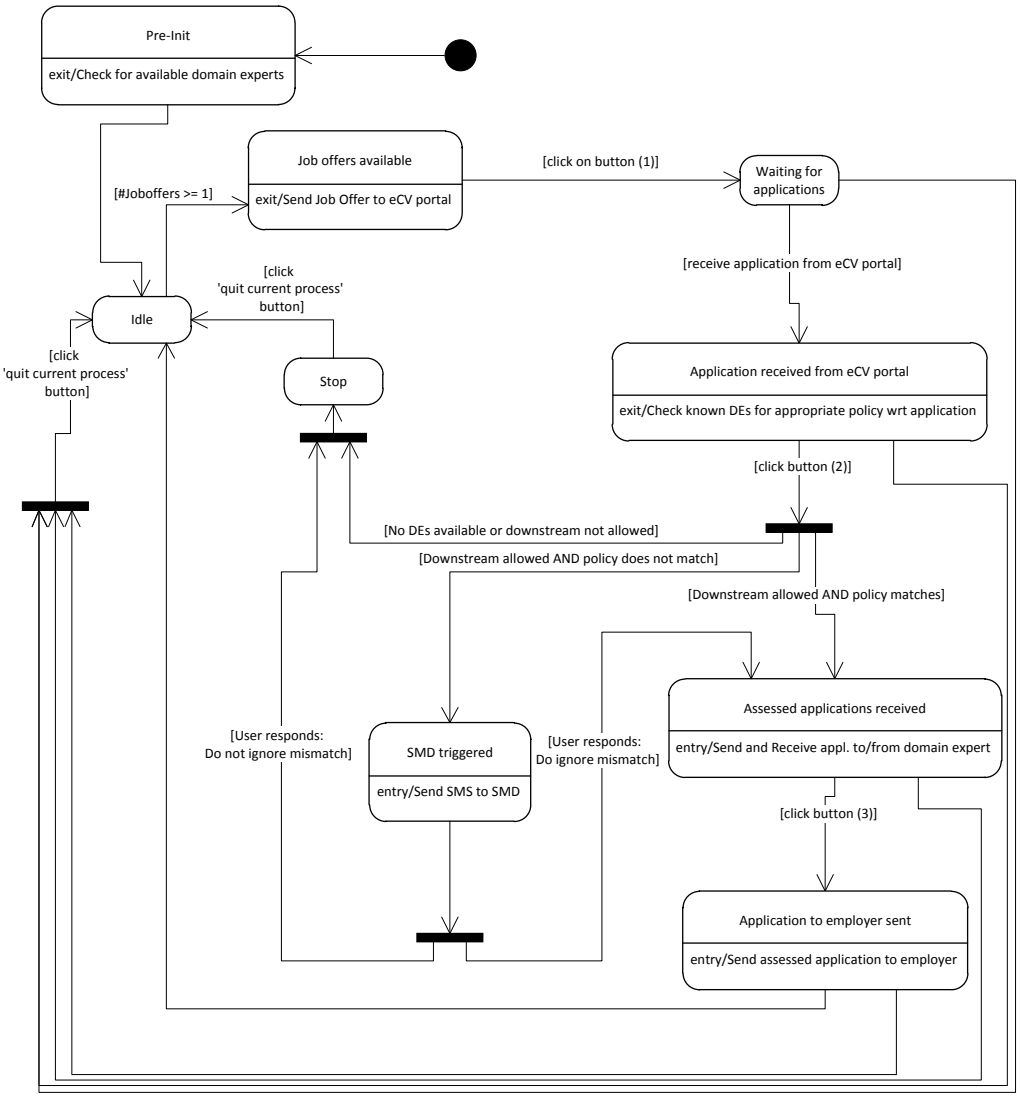


Figure 9: eCV scenario's headhunter tool as state machine

better to choose the DET with the closest-to-match policy but as such a metric is not available in PPL yet, we simply take the first mismatching DET. The user can now allow the usage of that DET anyway by accepting the mismatch, meaning: click on accept at the SMD application, or disallow it by ignoring the action request or actively rejecting the mismatch, that is, clicking on reject at the SMD application. The SMD will then do a REST web service call on the HHT to submit the user's response.

If the user gave the permission to transfer its PII to that DET, the HHT will proceed to its next state, otherwise it will jump to the Stop state. Being at the next processing

state, HHT will send the applications to the dynamically bound DET, who will instantly – HHT doing a synchronous call to DET – assess the applications and return a set of approved applications. Afterwards HHT will send the approved applications to the EPT and return to the Idle or Job offers available state, being ready for processing upcoming job offers.

8.2.2 Employer Tool

Next we would like to introduce technical details of the *employer's tool (EPT)*. The EPT sends its job offers to the HHT using web service client calls on the one hand, and receives assessed job applications from the HHT via its own web service endpoint on the other hand. As the latter implies that applicants' PIIs are transmitted to the EPT and are stored in a local database for further usage, the EPT therefore also requires policy handling.

By that stage, when the EPT receives an application, we assume that the applicant's privacy preferences are compliant with the EPT's policy. In fact, the sticky policy of the application is the agreed policy between the two parties. In order to get the agreed privacy policy enforced, the EPT connects to the Policy Enforcement Engine Service (PEE) immediately when it receives the PII. It then registers the sticky policy at the PEE and stores the PII in its local database. The PEE is in charge of acting on the privacy policy that has been registered and is able to trigger actions on the related PII directly.

8.2.3 Domain Expert Tool

The *domain expert tool (DET)* is similar to the EPT from a technical privacy point of view, as it also receives the application's PII with a sticky policy that should be compliant to its own policies or at least the applicant agreed on DET's policy using the SMD user interactions. Therefore DET only needs to take care of the enforcement by using the PEE service. Figure 7 shows the connection from DET to the same PEE as was used by the EPT.

Please note that this illustrates the architectural view on the eCV scenario and does not inherently mean to deploy only one instance of PEE for all components. As sensitive data is transferred between the PME and PEE services and their consumers, every trust domain should have its own instance deployed. Please see Figure 10 for deployment recommendation.

8.2.4 eCV Portal Tool

The *eCV portal tool (ECVPT)* is build on Java and runs on SAP NetWeaver Application Server. In Figure 7 the ECVPT is drawn as a single entity. Actually, the ECVPT consists of multiple parts as described in section 7.1. But as the Figure 7 has the intention to show the eCV scenario from a privacy architecture perspective, the ECVPT is simply represented as a data subject entity.

The ECVPT creates users and application profiles of users at its initialization. After starting up, the ECVPT is waiting for incoming job offers to process. These offers are received by ECVPT through its web service end point from the HHT. Afterwards a

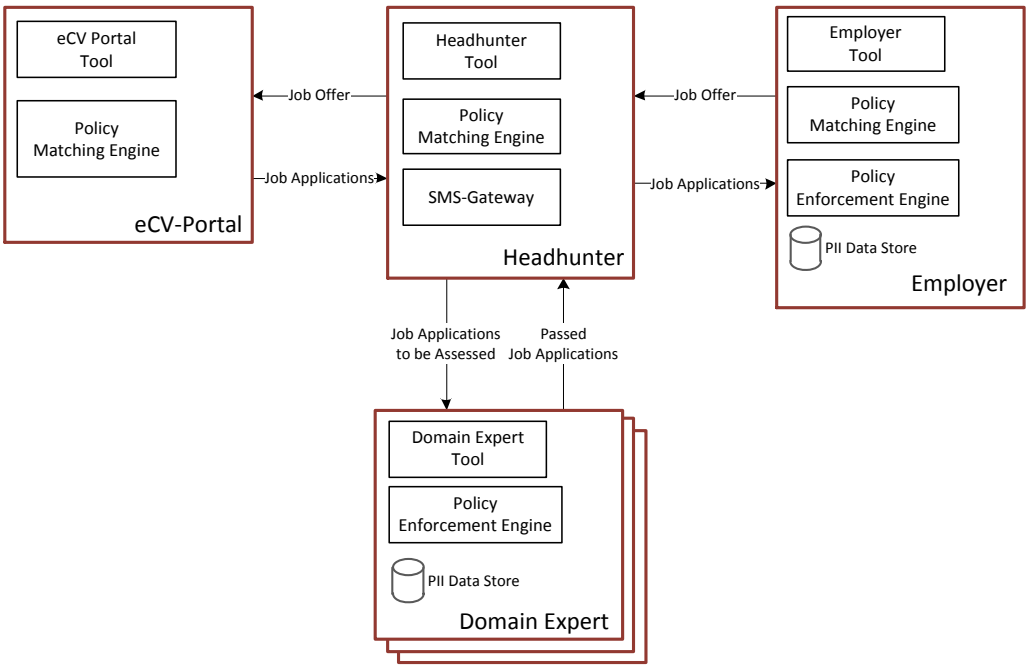


Figure 10: Sample Deployment of the eCV scenario

matching application profile is looked up at the ECVPT's user database and also checked for a privacy preference that is compliant to the policy that is given by the HHT. To perform the policy matching the PME service is used. Please note again, that only one instance of PME is shown in our architectural schema which does not imply that the HHT, the EPT and the ECVPT should share a single instance of PME. Please refer to Figure 10 for a sample deployment setup. Finally, the ECVPT will call the HHT using a web service to transmit the matching and policy-compliant applications.

8.3 Mapping eCV Scenario and Abstract Privacy Framework

In part II of this deliverable, we will generalize findings on privacy in service oriented architectures and describe a technology-agnostic architecture to design privacy-enhancing SOA applications. In this section we anticipate this knowledge and show how the eCV architecture maps to the abstract privacy framework (cf. Chapter 3).

This mapping may happen w.r.t. various technical views. In our case, the eCV

scenario maps (at least) w.r.t. the components (cf. deployment diagram, Figure 10) and w.r.t. the application behavior (cf. state diagram, Figure 9).

Figure 11 shows the state diagram and the deployment diagram side-by-side with the abstract privacy framework schema. The red arrow indicates exemplary places where the eCV architecture maps to the abstract framework. We look at the abstract architecture from the perspective of the headhunter during the evaluation phase. That is, the headhunter is the data provider, while the domain expert is the data consumer. One specialty of the eCV implementation is that the headhunter has no dedicated PII store nor Pref store, because it just passes through PII it receives from the eCV Portal. In other words, the process memory is the PII store and Pref store. The preferences are specified by the sticky policy attached to the job application document.

We are following now roughly the workflow of the PII provider. Service discovery is visualized in the state diagram. As described earlier, the headhunter uses a dynamic binding mechanism to discover and bind the domain expert service. Next, also expressed in the headhunter's state diagram, it retrieves the policy of each domain expert. The policy matching engine used by the headhunter is used for the policy matching step; we visualize it in the deployment diagram. Finally, the headhunter sends the job application together with the attached sticky policy. Due to the fact that the headhunter and the eCV portal proxy the user, there is no dedicated PII selection step in this scenario. Or to express it in other words: the PII, which is the job application, is automatically selected since there is no choice. The applicant's data is then stored in the PII store of the domain expert. We store PII and sticky policy in a joint data store, which is again visualized in the deployment diagram.

This short example illustrates what the abstract privacy framework could be used for. Our reasoning from the abstract privacy framework was partly deduced from observations and technical discussions we had in course of implementing the eCV scenario. So it is fair to say that the eCV scenario and the abstract privacy framework mutually fertilized each other. We will go into the details in Chapter 3.

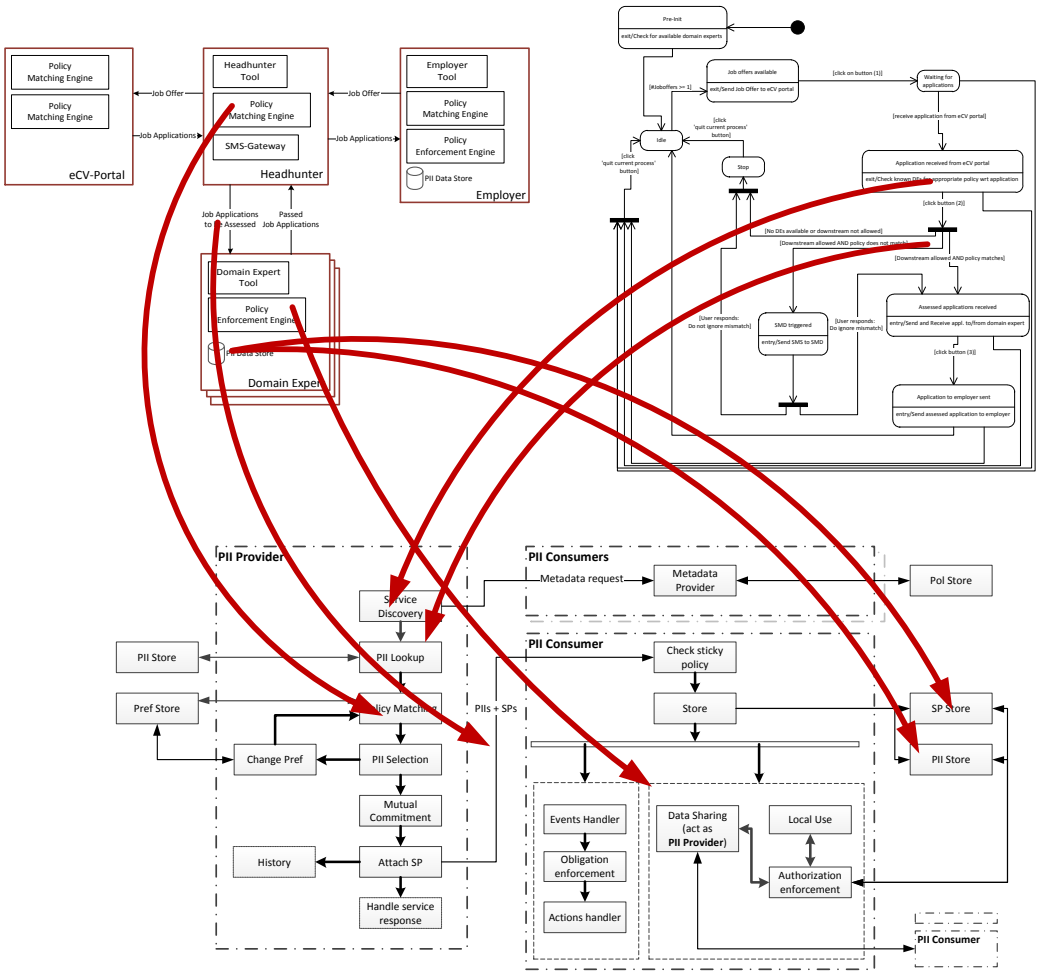


Figure 11: Mapping between the eCV Scenario and our generalization for privacy in SOA

Chapter 9

Implementation Details on Data Controller Part

In this chapter we will present implementation details at the data controller part of the eCV scenario. More specifically, this will cover the HHT, EPT, and DET.

9.1 Implemented Parts of eCV Scenario

In chapter 8 we have described the eCV scenario at its full implementation level. But, demonstrating a proof of concept, our actual implementation focuses on the privacy-relevant functional parts of the demonstrator. This section will describe limitations that result from skipping less privacy-relevant parts for the implementation.

9.1.1 Limited Scaleablilty and Concurrent Processing

The eCV scenario is working on *job finding tasks (JFTs)* initiated and triggered by a job offer. As the scenario allows to have multiple job offers and JFTs simultaneously, and it deals with multiple participants, all the participants should be able to handle multiple JFTs in parallel. That is, keeping track of current states of different JFTs, picking and proceeding the correct one, when being requested to do so.

As this feature is mainly required in real-life scenarios and is not necessary to showcase the privacy enhancements in SOA environments, we reduced the ability of participants to handle multiple JFTs to a minimum.

In HHT, this means, that although multiple job offers received from the employer are queued, only a single job offer can be processed at a time. The HHT can therefore only accept applications from the ECVPT in a particular state where it has sent the job offer to ECVPT right before. Received applications are then, of course, associated with the current job offer that is being processed. The communication between HHT and the DET is not affected by this limitation as the HHT runs a synchronous call on the DET. That is, HHT initiates the action from DET and waits until DET has finished and returns its response. This limitation also makes no difference at the EPT, as the

EPT is at the end of the processing-chain and will not process received job applications any further.

While this limitation has no impact on our showcases, it will prevent the eCV scenario from scaling up, because it can effectively process one JFT at a time. For real-life usage of that scenario, the feature of processing JFTs concurrently is necessary and can be implemented using an appropriate scaling infrastructure like application servers, for example Microsoft BizTalk Server.

9.1.2 Simplified Employer Tool and Domain Expert Tool

We simplified the EPT and the DET. The EPT does not store job offers in a history. Once a job offer has been sent and a new one has been created, the former is lost. The EPT does also not use the PME and the PEE as we show their integration using HHT and DET

The DET has been simplified in the way it assesses the applications. As we did not want to implement a complex algorithm to assess applications, we decided to have approve-all or approve-none mechanism in place. At the graphical UI, the assess-modes can be switched.

9.1.3 Controller Parts Considers Policy Obligations Only

One key goal of the eCV implementation is to evaluate the PrimeLife Policy Language (PPL) in a complex distributed setting. The current implementation is based on the "*PPL Engine*", a set of tools and services supporting evaluation of PPL (e.g. matching, generation of sticky policies) and enforcement of PPL (e.g. authorization decision, obligation enforcement). More information on this engine can be found in [Pri09a] and [Pri10a].

To showcase particular features of PPL, we split PPL usage among the scenario. While the data subject part of the eCV scenario focuses on authorizations, the controller part of the eCV scenario implementation considers the obligations part of PPL only. That is, in data controller part, two PPL policies are compliant and will match if they comply on their obligations. Thus we use the *Obligations Matching Engine (OME)* as the Policy Matching Engine (PME) and the *Obligations Enforcement Engine (OEE)* as the PEE.

9.1.4 Local PII Storages

The architecture of the eCV scenario defines local PII data storages at the EPT and the DET. Implementing this architecture, we took an *in-memory database (IMDB)* as a data storage. This fact has also impact on the access of PEE or OEE to that data. Working on an IMDB as database, a PEE/OEE has to contact the hosts of these databases in order to manipulate IMDB's data. That is, whenever the OEE would like to modify or delete the data that is stored at the DET's IMDB, the OEE will have to call the DET to do the job.

9.1.5 Integration of Enforcement Engine

As introduced in the sections above, we used the Obligation Enforcement Engine (OEE) as PEE and integrated it in the eCV scenario at the DET. Every time the domain expert receives applications (PII data), it registers its privacy policy in relation to the received data. The OEE will then in return call the DET, when it needs to modify the IMDB where the PII data has been stored. Because we have no additional tool to produce events that are relevant as obligation triggers, the `TriggerAtTime`[1] trigger is the only obligation trigger that makes sense to use in a showcase scenario. On the communication back channel, that is, when OEE calls the DET, there is only the `ActionDeletePersonalData`[1] action of PPL obligations supported.

To sum it up, if the eCV scenario implementation with OEE integration is to be to showcased, please make sure you are using the `TriggerAtTime` trigger and the `ActionDeletePersonalData` action at the DETs policy.

9.2 Service Interfaces

As the eCV scenario is built in a service oriented architecture environment, its components offer separate services. The service interfaces (APIs) are discussed in this section.

9.2.1 Headhunter Tool

Function	Short description
`void submitOpenPosition (OpenPosition` *`op`*`)`	Job offers are pushed from EPT to HHT using this method
`void submitJobApplicationsPPL (JobApplicationSetPPL` *`ja`*`)`	Applications are pushed from ECVPT to HHT with that method
`string GetMismatch(string` *`mId`*`)`	This method will give details on a particular mismatch
`void ProceedMismatchStickyPolicy (string` *`mId`*`, PPL.PolicySet` *`stickyPol`*`)`	This method can be called to override a sticky policy in order to solve a policy mismatch
`void ProceedMismatchBoolean (string` *`mId`*`, bool` *`doAccept`*`)`	The same as above, but mismatch can be accepted without overriding the sticky policy

Table 4: Web Service Interface of HHT

[1]The triggers `TriggerAtTime` and `TriggerPersonalDataAccessedForPurpose` as well as the actions `ActionLog` and `ActionDeletePersonalData` are part of the PrimeLife Privacy Language [Pri09a]

Table 4 show the methods that are exposed from HHT in order to be called as a web service. The first two functions are available via SOAP interface only. `submitOpenPosition` is called by the EPT with its job offer as parameter. This way, HHT receives a job offer and can start processing it. The function `submitJobApplicationsPPL` is to be called from the other end of the processing chain, the ECVPT. It will add a parameter of type `JobApplicationSetPPL` which is a list of job applications with a PPL policy, the sticky policy, each.

The last three functions are also available via REST interface. The first of them, `GetMismatch`, requires the mismatch ID as parameter and will return an `EcvMismatchData`[2] object as a serialized string. To use the REST interface to access this function, a HTTP-GET call is required to `http://`*`base-url`*`/Mismatch/`*`mId`*. The next function, `ProceedMismatchStickyPolicy`, is meant to be an response to a mismatch, if the user is smart/feature-rich enough to modify the sticky policy to fix the conflict, and resend that version back to HHT. The REST interface call would be a HTTP-POST call to `http://`*`base-url`*`/Mismatch/`*`mId`*, sending the sticky policy in the request body. The last function, `ProceedMismatchBoolean`, is semantically very similar to the one before but can be used without being able to parse the mismatch object. Sending `true` as boolean parameter will let HHT continue its process, not caring about the mismatch. To call this function via REST, an HTTP-POST has to be performed at the former URL, sending a boolean value as request body.

9.2.2 Domain Expert Tool

Function	Short description
`PPL.PolicySet getPolicy()`	This is called by the HHT as part of the dynamic binding
`AssessmentResult assessEcv (JobApplicationPPL` *`app`*`)`	Main function to be called synchronously by HHT to get applications assessed
`void RemoveApplication (string` *`appId`*`)`	Database function to remove stored application. Most likely to be called by OEE

Table 5: Web Service Interface of DET

The DET provides three web service functions (see Table 5). The first one is required by the HHT to know the privacy policies of all available DEs in order to be able to do the dynamic binding. It takes no argument and will return a PPL policy. The next function, `assessEcv`, will take an application as parameter and will return a positive or negative approval on it. Please note that the return value of this function is also

[2]Data type (ID: `http://www.primelife.eu/ecv/EcvMismatchData`, see Listing B.1 in Appendix B) that consists of an obligation mismatch (ID: `http://www.primelife.eu/ppl/obligation/mismatch`) and additional information strings about the mismatch

its response, which means that this function needs to be called synchronously and will take some (blocking) time to finish. The function `RemoveApplication` is to be called by the OEE. It will remove the application from DET's local storage that is identified by function's argument *`appId`* if there is such an application.

9.2.3 Employer Tool

Function	Short description
`void submitJobApplications (JobApplicationSetPPL apps)()`	Job offers are pushed from EPT to HHT using this method

Table 6: Web Service Interface of EPT

From a SOA point of view, the EPT is the most simple one. It only exposes one function, that is `submitJobApplications`. This function is used to push applications from HHT to EPT in the very last phase of a job offer processing.

9.3 Complex Features Explained

After having introduced the architecture and the limits of its implementation, in this section, we discuss some complex features of the main components in detail.

9.3.1 Privacy Policy Visualization: The PPL Policy Editor

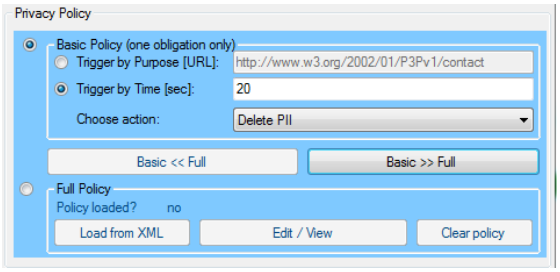


Figure 12: eCV UI, PPL Editor

The PrimeLife PPL language is fairly complex and therefore it is challenging to create an easy-to-use UI that can be used to create, edit or view PPL policies. As in the data controller part we consider the obligation part of PPL only, we limited the UI scope on that and built a PPL editor component, which should be able create, modify, or display a PPL policy. This component is used in EPT and DET but not in HHT as the HHT is

about a forwarding and triggering state machine without actively creating or changing policies.

Figure 12 is a screenshot of that PPL editor. It has basically two editing modes. The *basic-editing* mode, that can be selected by clicking at the radio button at the top of the upper grouping box, can be used to build or modify a very simple policy with one obligation having one obligation trigger and one obligation action. The choice of triggers are limited to `TriggerPersonalDataAccessedByPurpose`[1] and `TriggerAtTime`[1] and its arguments can be set freely. The action can be chosen to be `ActionDeletePersonalData`[1] and `ActionLog`[1]. The basic-editing mode should be sufficient to showcase our different scenario cases which were described at Chapter 7.

To add more enhanced elements to the PPL policy, you need to switch to the *full-editing* mode, which is active when the lower group box is selected. In this mode, you can load the full PPL policy from a file and edit it freely in a text field.

The PPL editor also provides mechanisms to convert a PPL policy from basic-editing mode to the full-editing mode and vice versa. For example, you can start off by creating a PPL policy in the basic-editing mode and can click on the *Basic » Full* button to convert your current policy into a full-editing-mode-accessible policy and then continue adding further properties there. The conversion using the opposite direction is somewhat more dangerous as the full-editing mode is expected to be more expressive than the basic one. In case the policy of the full-editing mode is not more expressive than the basic-editing mode can display, the conversion can be processed without objections. After a successful conversion, the policy is cleared and is not longer available at the full-editing part of the editor. Please note, that there is a text saying *Policy loaded?* right at the top the full-editing mode's group box, which indicates whether a policy is available in this editing mode. If you are in the other case at that type of conversion, that is, a more expressive policy has to be converted to a less expressive one, then the policy in the full-editing mode will not be cleared after successful conversion and you are still able to go back to the full-editing mode by reselecting the lower box without clicking at the conversion buttons in the center. Please note, that this behavior of the PPL editor implies, that there can be two different policies loaded at the same time. To prevent ambiguity, the hosting application of that editor will always take the policy in the current activated editing mode.

Beside the two editing modes, the PPL editor component can also be used in a read-only *view* mode. That is the case at the EPT in the section where applications are selected to get their details displayed. In view mode, PPL editor gets its input policy from the hosting application and tries to display it in basic mode. However, the full-viewing mode is always available to provide the full PPL policy as an XML structure.

9.3.2 Headhunter and User Interaction via Secure Mobile Device

In Chapter 8 we described the semantics about the interaction between the headhunter's tool (HHT) and the secure mobile device (SMD). Now, the technical details of that interaction will be explained.

The interaction starts off with a policy obligation mismatch between the downstream part of the application's sticky policy and the domain expert's policy. This mismatch is noticed by and also stored from the HHT. It then will send a SMS message to the

applicant's SMD using an external SMS provider that is contacted using a HTTP-GET call. The content of the SMS is `<PrimeLife>`*mID*`</PrimeLife>` where *mID* is a 6-digit, random identifier generated by HHT.

At the SMD, the SMS message is received and will be readable by the PrimeLife application only which then will trigger a privacy mismatch notification. The user can react on that notification by starting the PrimeLife application on that device and selecting the particular mismatch case. In order to show some details on the mismatch, the PrimeLife application on the SMD will then poll for the mismatch data at HHT using the GetMismatch method via REST interface that was discussed in the former sections. The mismatch details will be stored on the SMD for the purpose of showing a privacy mismatch history at a later point in time. At the bottom of that mismatch details view, there are two buttons allowing the user to choose either to ignore the mismatch and allow the HHT to use its data anyway (accept button) or to stick to its privacy preferences (reject button). The user decision is then send to HHT's ProceedMismatchBoolean method using the REST interface.

9.3.3 Dynamic Binding using WCF Service Discovery

The privacy-aware dynamic binding feature of the eCV scenario has been introduced at the architecture description of the HHT. We have defined this feature to have two phases. The first phase, which is service discovery of DETs, is based on WCF and will be discussed next.

The Windows Communication Foundation (WCF) brings a service discovery facility. There are several types of discovery. The one we make use of is based on the service interface. That is, when our web service consumer (e.g. HHT) knows the interface of its desired service in advance and is using broadcast network messages to signal a search of that particular type of service. The web service provider needs to prepare several properties in order to respond and support the binding to that kind of discovery.

As this feature only makes sense in our scenario when having multiple instances running at the same time, the service provider (DET) first needs to find a proper network address and port to listen to dynamically. Planning to have all participants running on the same machine, we decided to let DETs search for available ports but stay on the same IP address. We also use the port number as an identifier of that DET instance. Next, the DETs need to expose meta data about their service, this is, providing a WSDL file over a regular HTTP-GET call. The last step needed is to switch the service discovery feature on at the DETs. This will enable them to respond on the active service discoveries.

The service consumer has a configurable discovery timeout in which all consumer instances (DETs) need to respond. After that service discovery timeout, all instances are available as endpoint-proxy objects at the consumer side and therefore are ready to get called.

HHT will do the service discovery phase once per start up automatically, as this is considered expensive due to time(-out) consuming. Afterwards it will keep all known DETs and ask them on every single JFT to hand-out their policy in order to process the second phase of the privacy-aware dynamic binding.

9.3.4 Integration of OEE

The domain expert's tool (DET) is connected to the OEE such that it uses the OEE web service to register its policy in relation to received applications on the one hand. On the other hand, as DET uses an in-memory database, all actions of OEE on the PII data result in a call of DETs' web service from OEE.

The first type of use, in which the DET calls the web service of OEE, is the basic usage of OEE and does not need further discussion. The communication in the opposite direction is less trivial; this required additional integration work to get those two components connected and it also covers a tricky technique.

We will give a short overview on the structure of OEE first. OEE registers PPL policies including a PII-ID and will then listen for relevant events to trigger actions that are listed in the registered PPL policies. For executing an action, OEE provides a modular architecture to plug-in so called *action handlers*. If an action is to be triggered, then all known action handlers are called to do the action. Every action handler should then decide by itself if the current action and PII is relevant and whether it should effectively execute some action or not.

This is where the integration with the eCV scenario comes into play. We have implemented a separate OEE action handler (DET-AH) to communicate to DETs. Every time the DET-AH is hit to execute an action, it invokes a service discovery process, which we described in the section above, in order to distribute the action call to all DETs, telling them the ID of the PII data that should be affected. This way of calling DETs 'back' seems very inefficient at first sight. But taking into account that OEE is a generic obligation enforcement engine that was not designed to store some kind of sender information when registering policies, one would realize that there is no other way than detecting the sender at the moment when it is needed ('lazy'). But there is no extractable information neither in the PPL policy nor in the PII-ID that could reveal the sender DET and could be used for this purpose. Thus all DETs have to be contacted and requested to do the certain action on a specific PII.

As the PII will always be stored at one DET only in our eCV scenario, all other DETs need to check if the given PII-ID is known by them and proceed the request if this is the case and ignore it otherwise.

Chapter 10

Implementation Details on Data Subject Part

In this chapter we will outline details concerning the data subject part of the eCV scenario. It includes the interaction between the eCV portal, the headhunter and claims issuer (e.g. a proof that the applicant has attended a course) and the creation of job applicant profiles.

10.1 Introduction

From the data subject perspective, we consider the eCV portal, a claim issuer and interaction with the data controller. Integral to all these parts of the scenario are the concepts of data handling policies, user preferences and sticky policies. The portal is where job applicants create an electronic curriculum vitae, receive justifications that they have a certain experience or qualification, specify how their data will be consumed, receive job offerings, match jobs against their profile (verifying if there is or is not a mismatch against their preferences/policies) and then apply for a job. The receiving of available jobs and sending of an application, along with the corresponding policies in both instances, are the communication points with the data controller.

In the following sections there is a step-by-step explanation of the eCV scenario, including sequence diagrams with an emphasis put on the data subject side, along with an explanation of the web services used. Finally we will describe the technologies used during development.

10.2 eCV Scenario

1. Receiving available jobs

- Whenever any open job position is created, an employer creates the job vacancy, fills the details, attaches a policy and sends to the Headhunter. Details can consist

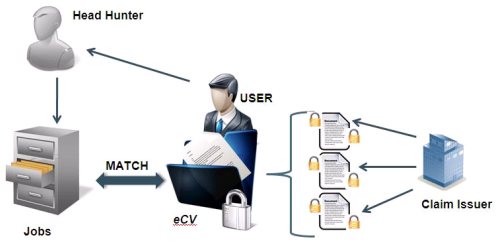


Figure 13: eCV Scenario Overview

of Job Description, Function Area, Job Position, Salary, Company, Location, Start Date, End Date. Policies regulate the usage of data and in this instance we look at Purpose, Recipients and Retention. Purpose refers to using data to perform a specific task. The data that is collected by a service to do a certain task (e.g. RESEARCH) must be used only for that purpose. Recipient specification allows the data provider and the data consumer to be aware of the users or geographic domains that have access to the data (e.g. EU, USA). An external service that is consuming data is obliged to accept the data retention period which means that the external service can't access the data after this period (e.g. 90 days).

- The Headhunter receives the job application and forwards it to the eCV Portal. A headhunter is a job recruiter who specializes in matching skilled professionals with corporate clients. A headhunter might be an independent contractor or work through an agency of headhunters where each agent specializes in particular areas of employment and possibly also in geographic areas.

- The eCV portal collects, stores, manages and publishes the eCVs. It also has the ability to store job searches and offers from the Headhunter. Using this information, the eCV portal is able to perform internal searches on the candidates and find a possible matching. The eCV portal is the actual service provider entity in the architecture. Users have to register to the eCV portal in order to be able to participate.

2. Claim Issuer

- User logs in to the eCV portal and requests the Claim Issuer for the claim which can be either educational or work-related. Users can request for several claims at a time.

The claim issuer is one that provides evidence, on request, for all or some of the claimed experience and knowledge the user lists in his eCV. Issuers of the claim/document are always an organization that is in a position to assert user's statements, e.g. employers, universities, training center, government, and other community members.

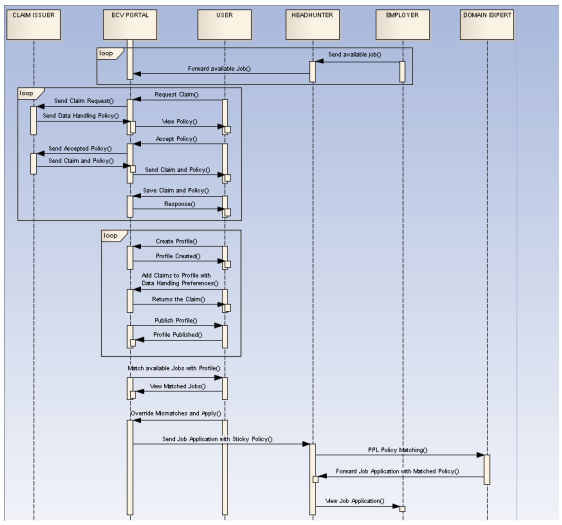


Figure 14: Sequence Diagram eCV Scenario with focus on Data Subject

The user would typically verify the claim upon receipt and may then add a privacy preference to it (which has to be more restrictive than the policy issued with the claim), to further restrict access to the claim before publishing it. Both the user and the claim issuer are concerned about how the information expressed in the claim will be used by other entities in downstream and secondary usage.

3. Profile creation

- User creates the job profile on eCV portal, attaches the claims and then publishes the profile. Users can further refine the policies at this stage either before or after publishing the profile.

- Now the user can search for the jobs matching to his profile and apply for matching jobs on the eCV portal. There can be a mismatch when policies of the available job are not compliant with user's privacy policy. If the user wants to apply for a job that shows a mismatch, then he can go back to his profile and change his privacy policy according to the Job requirement policy, republish his profile and then apply again. Overriding a privacy policy may occur if an element in the available job is of interest to the applicant, for example, if the salary or location is of interest. If this action is done we can say that the user foregoes his original privacy preference.

- The final step is for the job applicant to send the CV to the headhunter along with the policies and preferences.

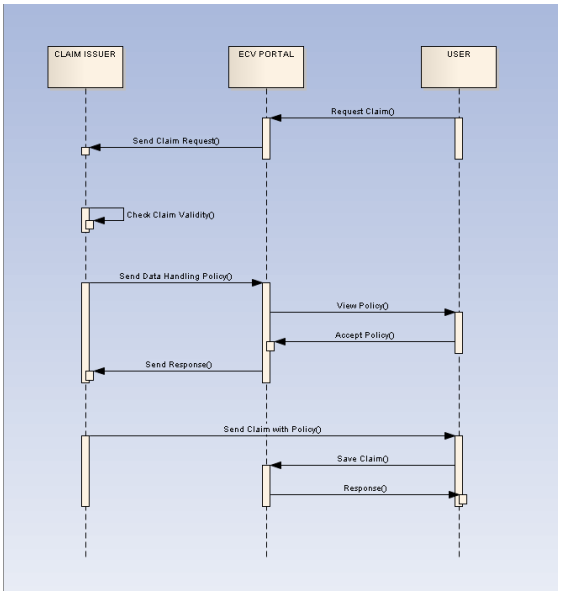


Figure 15: Claim Issuance

10.3 Web Services descriptions

10.3.1 Claim Issuer

The claim issuer web service contains methods that receive the claim request from the eCV portal, check the validity of the requested claim and finally send a response back to the eCV portal.

It contains the following methods: a) Receive Claim Request b) Check Claim Validity c) Send Policy d) Send Claim e) Display Request

Receive Claim Request:

This method receives a claim requested by the user through the eCV portal. The request contains the claim requested and the claim type (education or work experience), the user's name, institution from where this claim is being requested and the country of this institution along with a transaction ID. This information is stored in the database in the table "CLAIMREQ". It then returns the transaction ID.

Check Claim Validity :

This method receives the transaction ID and then retrieves the claim from the CLAIMREQ table in the database using the transaction ID. It then checks the validity of the claim by comparing against the claims tables in the database (WORKCLAIMS and EDUCLAIMS). It then updates the CLAIMREQ table with the ClaimID and Claim Type, if the claim is valid. Finally, if the claim is valid, it returns true or else it returns false along with the transaction ID.

Send Policy:

This method takes the transaction ID as an input and retrieves the claim associated with that request. It then retrieves the policy of the claim and sends it to the eCV portal

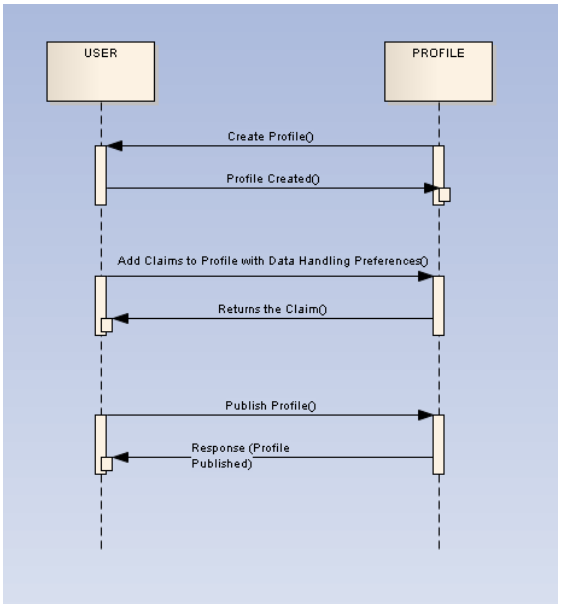


Figure 16: eCV Profile

for the user's approval.

Send Response: This method receives the transaction ID and the approval status (either a true or false). If it is true, i.e. if the policy has been approved, it retrieves the claim (using the ClaimID and ClaimType from the CLAIMREQ table) and sends the claim as a response along with the PPL policy attached to the claim. For this process it retrieves the values from tables WORKCLAIMS, EDUCLAIMS and CLAIMPOLICY. If the user does not agree to the policy, it returns null.

Display Request:

This method is used for the Claim Issuer UI, it retrieves the claim request from the CLAIMREQ table and displays the status as recieved, invalid/valid and Issued.

10.3.2 ECV Claim Handler

The ECV Claim Handler web service contains methods that are used to send the claim request from the eCV portal to the claim issuer, receive the policy of the claim, and if the policy is accepted then claim is received and finally saved in the database of the eCV portal.

It contains the following methods: Send Claim Request; Receive Claim Policy; Receive Claim; Save Claim; and Display Claim.

Send Claim Request:

This method sends a claim request by the user through the eCV portal. The request contains the claim requested and the claim type (education or work experience), the user's name, institution from where this claim is being requested and the country of this institution along with a transaction ID that is automatically generated. The transaction

ID is the key that is exchanged between the eCV portal and the claim issuer. This information is stored in the database in the table "ECVCLAIMREQ".

Receive Claim Policy :

This method receives the Policy of the Claim sent by the claim Issuer. The policy sent is a PPL Policy. The transaction ID is also sent by the claim issuer. The user's response is awaited and if the user agrees (or disagrees) to the policy, the response is sent back to the claim Issuer. The transaction ID is used by the eCV portal to retrieve the requested claim data from the ECVCLAIMREQ table and display the policy for that request. This method does not alter the tables of the database.

Receive Claim:

This method receives the requested claim along with the agreed policy and also the transaction ID. This information is then displayed (Display Claim) to the user. The user can then choose to save the claim (Save Claim) in his profile.

Save Claim:

This method receives the claim along with the policy. The claim is stored in the claims table (ECVCLAIM) while the policy is stored in the ECVCLAIMPOLICY table. This method parses through the PPL policy and retrieves the relevant attributes (recipients, retention time, purpose etc) and saves them in the table (ECVCLAIMPOLICY) table.

Display Claim:

This method is used for the Claim Handler UI, it displays the claim it receives from the claim issuer along with the policy.

10.3.3 Job Handler

The Job Handler web service contains methods that are used to match the available jobs to the user's profiles.

It contains the following methods: Match Jobs on Profile; Match Jobs on Policy and Send Notifications.

Match Jobs on Profile:

This method takes the userid of the user as an input and retrieves the profiles of the users from the ECVPROFILE table. These attributes in the profiles are matched against the relevant jobs using a Java Persistance API Query. This query finds the jobs that meet the specifications of the users profile (salary, functional area, etc).

For each job, the job id and the matched profile id are retrieved and paired together. These pairs (jobid and profileid) are then returned by this method.

Match Jobs on Policy : This method takes the pairs of profile id and job id. A new set is now formed, which consists of ProfileID, JobID and the MatchStatus. The match status is a boolean. For each pair, the profile policy is retrieved and the job policy is retrieved and compared. Only if the policies of both match or if the profile policy is a superset of the job policy, it is considered as a match and the value of MatchStatus is set to true. If there is any mismatch between the policies the MatchStatus is set to false. After processing all the pairs and generating the corresponding sets, this method returns the sets.

Send Notifications: This method receives the sets which contain the profileid, jobid and the matchstatus. It then adds this information in a database table, MATCHJOBS,

and the column "MATCH" is set to the value contained in the field "matchstatus" of the set.

10.3.4 Profile Handler

The Profile Handler web service contains methods that are used to manage the profiles of the user

It contains the following methods: a) List Profiles b) List Profile Claims c) List Claims d) Create Profile e) Delete Profile f) Publish Profile g) Add Claim to Profile h) Remove Claim from Profile i) Display Claim Details

List Profiles:

This method retrieves the profiles created by the user and displays them. It takes as an input the userid of the user and searches the table ECVPROFILE for all the profiles of that user. It then displays the profile name, profile id and the status of the profile.

List Profile Claims:

This method retrieves the claims that have been added to the profile and displays them. It takes as an input a profile ID and searches the table ECVPROFILECLAIM and finds all the claims that are added to this profile. It displays the claim name and claim ID.

List Claims:

This method retrieves all the claims that have not been added to the profile and displays them to the user. It takes as an input the profile ID. It performs a search on the database using a query that joins the ECVPROFILE and ECVPROFILECLAIM tables and retrieves the claims that have not yet been added to the ECVPROFILECLAIM table for the selected profile. It displays the Claim name and the Claim ID.

Create Profile:

This method takes the details of the profile that have been filled by the user using the form provided in the UI of the profile handler and adds this record in the ECVPROFILE table. It returns a boolean value indicating the success or failure of this operation.

Delete Profile:

This method takes the profile ID and deletes all occurences of it in the database tables. It clears the ECVPROFILE, ECVPROFILECLAIM and the ECVUCLAIM tables of any records containing this profileID. It returns a boolean value indicating the success or failure of the operation.

Publish Profile:

This method takes the profile ID as an input and publishes the profile. Publishing the profile indicates that the profile has been complete and can be used for matching against available jobs. This method generates a policy considering all the claim policies of the claims that are added to the profile. The policy satisfies the most restrictive policy in the collection of the policies considered. In other words, it has the most restrictive policy among the claims that are added to the profile. It then adds this policy in the ECVPROFILEPOLICY table. Also it updates the ECVPROFILE table, marking the profile as published. It returns a boolean value indicating the the success or failure of this operation.

Add Claim to Profile:

This method takes the profile id and the claim id as an input. This information is then added to the ECVPROFILECLAIM table. It returns a true or false, indicating the success or failure respectively.

Remove Claim From Profile:

This method takes the profile id and the claim id as an input. This information is then used to delete the record from the ECVPROFILECLAIM table. It returns a true or false, indicating the success or failure respectively.

Display Claim Details: This method takes the claim id as an input and displays the claim details. It uses the claimid to retrieve the details of the claim from the ECV-CLAIMS table and also from the ECVCLAIMPOLICY table and displays it to the user.

10.3.5 Application Handler

The Application Handler web service contains methods that are used to manage the applications of the user. It contains the following methods: View Jobs; User Action and Send Applications.

View Jobs:

This method takes input as the user ID and displays the jobs that have been matched to the user's profiles. The userid is used to retrieve all the profiles of the user from the ECVPROFILE table and for each profile that is retrieved, the MATCHJOBS table is searched if it contains a record that contains that profile id. These records contain both the matches and mismatched jobs (on policy). These jobs are then displayed to the user.

User Action:

This method takes the MATCHJOBID and the userchoice as an input. Userchoice is an integer value that indicates the choice of the user.

-1 indicates that the user has deleted the job.

0 indicates no change.

1 indicates that the user has applied for the job, if it is a matched job.

2 indicates that the user has applied for the job, by overriding the mismatch that occured.

Finally, it updates the MATCHJOB table with the userAction in the column of the same name.

Send Applications:

This takes as an input the user ID and sends applications to the headhunter. The userid is used to retrieve all the profiles of the user from the ECVPROFILE table and for each profile that is retrieved, the MATCHJOBS table is searched if it contains a record that contains that profile id and containing the useraction as 1 or 2. It then invokes the client of the headhunter and sets the data accordingly to the application format of the headhunter. Subsequently it dispatches the contents to the headhunter. Finally, this method updates the MATCHJOB table by setting the headhunterAction set to 1, indicating that the application has been sent.

10.4 Technology Used

10.4.1 Introduction

The ECV portal is designed using SAP Netweaver BPM to design the business processes and Web Dynpro for developing the User Interfaces. Web services are used extensively to provide functionality to the User Interfaces and to use them in a business process.

The SAP NetWeaver Composite Environment (CE) 7.1 version provides tools for developing, running and managing composite applications based on SOA principles. SAP NetWeaver CE uses proven technologies and integrates them to provide greater functionality.

The NetWeaver CE suite is divided into three parts:

- The process composer in the NetWeaver Developer Studio (NWDS), an Eclipse-based environment with a special perspective for creating and deploying processes and applications.
- The process server running on the NetWeaver Composition Environment (CE), a Java application server.
- The process desk running on the NetWeaver portal with the universal work list and task execution.

The application is built using SAP Netweaver 7.1 package. There are many different perspectives available in the NWDS, however, the five major perspectives that have been used in the development of the prototype are shown below:

- Development Infrastructure: Used for creating Development Components (ejb and ear).
- Java EE: Used for creating web services, web service clients, java bean skeletons and entities.
- Web Dynpro: Used for creating User Interfaces.
- Process Composer: Used for creating the BPM (Business Process Management) process.
- Dictionary: Used for creating Tables.

The database used to store and retrieve data is MaxDB. We connect to the database using the dictionary component in the NWDS. We use the JPA API to read and write data to the database.

Chapter 11

Conclusions

This report describes our research results from two different points of view. Part I describes our architectural consideration for a privacy-enhanced SOA. We noted them in a technology agnostic way, so that they can be applied on a large variety of technologies. In Part II we summarized our practical experiences with making a service-oriented architecture privacy-friendly.

This report offers three main takeaways:

1. First, it gives very precise, yet technology agnostic hints how to build a privacy friendly service-oriented architecture. These findings are very generic and can be applied on a wide range of concrete implementations. We describe the abstract privacy policy framework in Chapters 3 to 6.

2. The second takeaway is the detailed analysis of existing privacy-enhancing technologies against the abstract privacy policy framework. We compare to what extend today's solution fulfill or not fulfill the technical demands of a privacy-friendly SOA. We present a summary in Chapter 6 and an extended evaluation in Appendix A.

3. Thirdly, this report gives details about design and architecture of a privacy-friendly SOA application. This *eCV* (electronic curriculum vitae) is also a showcase for the PrimeLife Policy Language (PPL). Chapters 7 to 10 and Appendix B give details.

We showed that more privacy-friendly SOA are achievable with today's technology. The use privacy policies exposed as service meta-data, a decent privacy policy language, and the notion of sticky policies traveling with personal data would allow users to keep control over already disclosed data.

Chapter A

Detailed Comparison of Privacy Policy Technologies in SOA

This appendix describes how table 2 of Chapter 6 has been filled. The remaining of this appendix provides for each evaluation criteria: 1) description of the criteria, 2) description of possible values for the evaluation of the implementation, 3) description of possible values for the evaluation of possible extensions, 4) the evaluation of each technology according to those criteria.

A.1 PII Provider's Preferences (Sect. 3.3)

A.1.1 Simple Syntax

Privacy preferences are expressed in a human-readable language. Syntax and semantics are well defined and can be processed by machines.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: The language to express privacy preferences has a short syntax, which is close to natural language, e.g. "*this PII must be deleted within 6 months*". The language semantics is not ambiguous.

- Partial implementation (◐) is defined as: The language to express privacy preferences is human-readable (e.g. XML) but not intuitive. For instance "*Obligation(ActionDelete(...), TriggerAtTime(Now, P0Y6M0DT0H0M0S))*".

- No implementation (○) is defined as: Privacy preferences are expressed in a binary format. Or privacy preferences are out of scope: the current implementation does not provide a language to express privacy preferences.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: A language close to natural language can be implemented on top of the existing language. The implementation is straightforward, has been prototyped, or is specified.

- Possible partial support (◐) is defined as: A human-readable language can be defined for the specific technology. The implementation is straightforward, has been prototyped, or specified.

- Cannot be supported (○) is defined as: Defining privacy preferences is out of the scope of this specific technology. Defining such a language would require major extensions.

APPEL + P3P (A+P):

- The implementation is evaluated as ●. The syntax of APPEL is easy to read. However, note that this is mainly due to the low expressiveness of APPEL and P3P.

- Possible extensions are evaluated as ●. The syntax is already simple. Further simplification (e.g. list, graphical representation) can be integrated in web browsers.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ◐. PPL expresses privacy preferences in a complex XML dialect extending XACML. This language can be used by IT specialists but not by end users.

- Possible extensions are evaluated as ◐. A prototype translating a high-level domain specific language to PPL preferences has been implemented (see [Rah10]). However, this work only covers a subset of the language.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ●. S4P defines privacy preferences as "will" queries and "may" assertions in an easy to read language. This language is directly translated into Datalog with constraints and can be serialized as XML.

- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ○. Privacy preferences are out of the scope of this approach. Specifying preferences to automate the remote configuration of access control would require new mechanisms and languages.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ○. Privacy preferences are not supported by PRIME-DHP. However, the choice of the PII Provider is simplified by "templates": the PII Consumer provides a template of its privacy policies, and the PII Provider can accept and customize it or stop the communication.

- Possible extensions are evaluated as ◐. Even if privacy preferences are out of the scope of PRIME-DHP, different prototypes using PRIME-DHP did implement ad-hoc preference languages. This could be reused.

A.1.2 Can Express Access Control

The language used to express privacy preferences supports access control, i.e. the data subject can specify which (or what kind of) data controllers can get a given type of personal data.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: The evaluated technology supports claim-based access control: privacy preferences express rules taking into account certified attributes of data controllers and the context.

- Partial implementation (◐) is defined as: The evaluated technology supports role-based access control (RBAC): privacy preferences express the list of data controller types (i.e. roles or identities) that can get personal data.

- No implementation (○) is defined as: The evaluated technology only supports identity-based access control or does no support access control at all.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: Claim-based access control could be combined with existing preference language.

- Possible partial support (◐) is defined as: Role-based access control could be combined with existing preference language.

- Cannot be supported (○) is defined as: Support for access control is out of scope and would require major extensions.

APPEL + P3P (A+P):

- The implementation is evaluated as ◐. APPEL can handle attributes: X.509 certificate (issuer, attributes) of the PII Consumer can be taken into account.

- Possible extensions are evaluated as ◐. Further support for access control would be difficult to add. No extension is foreseen.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ●. The full expressiveness of XACML can be used to specify access control in PPL privacy preferences.

- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ●. The full expressiveness of SecPAL can be used to specify access control in S4P privacy preferences.

- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ○. Access control in privacy preferences is out of scope since this approach does not address privacy preferences.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ○. Access control in privacy preferences is out of scope since PRIME-DHP does not address privacy preferences.

- Possible extensions are evaluated as ◐. PRIME-DHP could be combined with a legacy access control technology at data subject side. Moreover, basic support for access control exists in prototypes that specified ad-hoc preferences language.

A.1.3 Can Express Expected Data Handling

The language used to express privacy preferences lets the PII Provider specify how collected data must be handled by the PII Consumer.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: Privacy preferences express complex data handling expected by the PII Provider including rights (e.g. use for purpose) and obligations (e.g. log usage).

- Partial implementation (◐) is defined as: Privacy preferences express basic data handling, e.g. a flat list of purposes and data retention time.

- No implementation (○) is defined as: Privacy preferences cannot express how the PII Provider expect her personal data to be handled.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: The privacy preferences language provides extension points to define complex data handling.
- Possible partial support (◐) is defined as: The privacy preferences language could be extended with basic data handling.
- Cannot be supported (○) is defined as: Expressing data handling is out of scope and would require major extensions.

APPEL + P3P (A+P):

- The implementation is evaluated as ◐. APPEL lets define primary and secondary purpose as well as data retention.
- Possible extensions are evaluated as ◐. Further support for data handling would be difficult to add. No extension is foreseen.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ●. PPL defines data handling in terms of rights and obligations. Usual rights and obligations are part of the language and moreover both rights and obligations can be extended.
- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ●. S4P defines data handling in privacy preferences in terms of authorizations ("may" assertions) and obligations ("will" queries). New types of rights and obligations can be defined.
- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ○. Data handling in privacy preferences is out of scope since this approach does not address privacy preferences.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ○. Data Handling in privacy preferences is out of scope since PRIME-DHP does not address privacy preferences.
- Possible extensions are evaluated as ◐. Basic support for data handling has been implemented in prototypes that define ad-hoc preferences language.

A.1.4 Can Express Expected Downstream Access Control

The preferences can express access control constraints on third parties. In other words, the preferences specify with what kind of third parties the data controller is authorized to share collected data.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: Downstream access control takes into account claims of third parties: privacy preferences express rules on certified attributes of third parties.
- Partial implementation (◐) is defined as: Role-based or identity-based access control: privacy preferences express the list of third parties (e.g. roles or identities) with which the data controller may share collected data.
- No implementation (○) is defined as: Privacy preferences cannot express access control on third parties.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: Downstream claim-based access control could be combined with the existing privacy preferences language but is not yet implemented.
- Possible partial support (◐) is defined as: Downstream role-based or identity-based access control could be combined with existing privacy preferences language but is not yet implemented.
- Cannot be supported (○) is defined as: Support for downstream access control is out of scope and would require major extensions.

APPEL + P3P (A+P):

- The implementation is evaluated as ◐. APPEL can take attributes of third parties into account.
- Possible extensions are evaluated as ◐. Further support for downstream access control would be difficult to add. No extension is foreseen.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ●. The current implementation is based on "lazy matching", which lets define the access control (XACML) to be enforced by the PII Consumer.
- Possible extensions are evaluated as ●. This is already supported and does not require additional extension. Note that "Pro-active" matching change the creation of the sticky policy but not the preferences.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ●. Conditions and constraints associated with "may send" assertions can be used to specify arbitrary properties of third parties.

- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ○. Downstream access control in privacy preferences is out of scope since this approach does not address privacy preferences.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ○. Downstream access control in privacy preferences is out of scope since PRIME-DHP does not address privacy preferences.

- Possible extensions are evaluated as ◐. Basic support for data handling has been implemented in prototypes that define ad-hoc preferences language. Note that downstream access control is fully supported by PRIME-DHP policies and sticky policies.

A.1.5 Can Express Expected Downstream Data Handling

The preferences can express data handling constraints on third parties. In other words the preferences specify how third parties are expected to handle data they would get from data controllers.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: The privacy preferences can specify complex downstream data handling including rights (e.g. use for purpose) and obligations (e.g. log usage).

- Partial implementation (◐) is defined as: The privacy preferences can specify basic downstream data handling, e.g. flat list of purposes and data retention.

- No implementation (○) is defined as: The privacy preferences cannot specify expected data handling on third parties.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: The language could be extended to specify complex data handling that is expected downstream but this is not yet implemented.

- Possible partial support (◐) is defined as: The language could be extended to express minimum downstream data handling but this is not yet implemented.

- Cannot be supported (○) is defined as: Expressing expected downstream data handling is out of scope and would require major extensions.

APPEL + P3P (A+P):

- The implementation is evaluated as ◐. It is possible to specify different data handling for third parties. However this is complex and requires multiple preferences.

- Possible extensions are evaluated as ◐. Further support for downstream data handling would be difficult to add. No extension is foreseen.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ●. The current implementation is based on "lazy matching", which lets define how third parties must handle collected data.

- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ●. S4P defines data handling in privacy preferences for any PII Consumer including downstream.

- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ○. Downstream data handling in privacy preferences is out of scope since this approach does not address privacy preferences.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ○. Downstream data handling in privacy preferences is out of scope since PRIME-DHP does not address privacy preferences.

- Possible extensions are evaluated as ○. Downstream data handling is out of scope.

A.1.6 Can Take Downstream Path into Account

Privacy constraints that apply to data controllers downstream depend on the path. In other words, it is possible to have different privacy constraints for personal data d at service S when S is a data controller directly collecting d, when S acts as downstream data controller and gets d from data controller S_1, or from data controller S_2.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: Data handling for a given piece of personal data and for a given third party can be specified differently depending on which PII Consumer provided the data.
- Partial implementation (◐) is defined as: Some aspects of Data handling can be different depending on the path.
- No implementation (○) is defined as: The path does not impact the data handling.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: The language could be extended to take the path into account but this is not yet implemented.
- Possible partial support (◐) is defined as: Mechanisms could be built on top of the existing language to take some aspects of the path into account.
- Cannot be supported (○) is defined as: Taking path into account is out of scope and would require major extensions.

APPEL + P3P (A+P):

- The implementation is evaluated as ◐. Data handling depends on the path since a third party has equivalent or stricter constraints than the data controller (i.e. PII Consumer).
- Possible extensions are evaluated as ◐. Full support for downstream path would be difficult to add. No extension is foreseen.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ●. The current implementation is based on "lazy matching", which lets specify a chain of nested data handling preferences.
- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ◐. By default, preferences regarding a how a given party must handle a given piece of personal data is expressed one. However, path could be taken into account with additional conditions on "may send" assertions.
- Possible extensions are evaluated as ◐. Full support for downstream paths would be difficult to add. No extension is foreseen.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ○. Specifying downstream path in privacy preferences is out of scope since this approach does not address privacy preferences.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ○. Specifying downstream path in privacy preferences is out of scope since PRIME-DHP does not address privacy preferences.
- Possible extensions are evaluated as ○. Downstream paths are out of scope.

A.1.7 Can Retrieve Applicable Preferences (Sect. 3.6)

This technology provides a mechanism to get the privacy preferences that apply to a piece of personal data. This mechanism supports different types of personal data: retrieved from a PII Store (e.g. a database), dynamically created by the user (e.g. free text in a HTML Form), or certified (e.g. attributes of credentials).

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: It is possible to automatically query (and combine) privacy preferences related to a type of personal data or a given piece of data.
- Partial implementation (◐) is defined as: Preferences can be specified per personal data but there is no grouping mechanism (e.g. no way to refer to "all e-mail addresses").
- No implementation (○) is defined as: No mechanism to query preferences.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: It would be possible to build complex preference queries but this is not available in current implementation.
- Possible partial support (◐) is defined as: It would be possible to build basic preference queries but this is not available in current implementation.
- Cannot be supported (○) is defined as: Querying privacy preferences is out of scope and would require major extensions.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. This is out of scope. APPEL lets the client technology (e.g. Web Browser) make the link between personal data and preferences.
- Possible extensions are evaluated as ◐. Ad hoc mechanisms exist to query preferences.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ◐. Basic mechanism has been implemented to get the preference that applies to personal data including credentials.
- Possible extensions are evaluated as ●. The language let's define complex applicability for preferences that can be used to query preferences.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ○. Out of the scope of the current implementation.
- Possible extensions are evaluated as ●. A prototype preference language associates pieces of preferences (mainly "may" assertions and "will" queries) with description of PII. This could be used to select relevant assertions and queries.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ○. Querying preferences is out of scope since this approach does not address privacy preferences.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ○. Querying preferences is out of scope since PRIME-DHP does not address privacy preferences.
- Possible extensions are evaluated as ◐. There are prototypes associating ad-hoc preferences with personal data in a databases.

A.2 PII Consumer's Policy (Sect. 3.4)

A.2.1 Simple Syntax

Privacy policies are expressed in a human-readable language. Syntax and semantics are well defined and can be processed by machines.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: The language to express privacy policies has a short syntax, which is close to natural language, e.g. "*Commit to delete this collected PII within 6 months*". The language semantics is not ambiguous.
- Partial implementation (◐) is defined as: The language to express privacy policies is human-readable (e.g. XML) but not intuitive. For instance "*Obligation(ActionDelete(...), TriggerAtTime(Now, P0Y6M0DT0H0M0S))*".
- No implementation (○) is defined as: privacy policies are expressed in a binary format or are out of scope and cannot be expressed.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: A language close to natural language could be defined on top of the existing language. The implementation is straightforward, has been prototyped, or specified.
- Possible partial support (◐) is defined as: A human-readable language can be defined for the specific technology. The implementation is straightforward, has been prototyped, or specified.
- Cannot be supported (○) is defined as: Defining privacy policy is out of the scope of this specific technology. Defining such a language would require major extensions.

APPEL + P3P (A+P):

- The implementation is evaluated as ◐. P3P policies are readable. However, this is mainly due to low expressiveness.
- Possible extensions are evaluated as ●. A prototype [Rag10] using a subset of P3P feature serialized in JSON proposes an easy to read privacy policy.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ◐. PPL expresses privacy policies in a complex XML dialect extending XACML.
- Possible extensions are evaluated as ◐. Prototype translating a high-level domain specific language to PPL policies exists (see [Rah10]). However, this work only covers a subset of the language.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ●. S4P defines privacy policies as "may" queries and "will" assertions in an easy to read language.
- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ○. Privacy policies are out of the scope of this approach. Specifying policies to restrict the remote configuration of access control would require new mechanisms and languages.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ◐. PRIME-DHP provides an XML-based privacy policy language.
- Possible extensions are evaluated as ◐. No higher-level language does exist.

A.2.2 Can Express Claims (Credentials)

The policy language can describe trust level and certification of PII Consumers. For instance, it is possible to link Public Key Infrastructure to the policy.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: The language lets link policies with complex claims. Different trust models (e.g. reputation, web of trust, PKI) and different mechanisms (e.g. X.509, anonymous credential) are supported.
- Partial implementation (◐) is defined as: The language lets link policies with usual mechanisms (e.g. Public Key Infrastructure) but other mechanisms are not supported.
- No implementation (○) is defined as: The language does not handle credentials.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: Support for different trust models and mechanisms is possible but is not implemented.
- Possible partial support (◐) is defined as: Support for basic certification is possible but is not yet be implemented.
- Cannot be supported (○) is defined as: Support for credentials is out of scope or would require major modifications

APPEL + P3P (A+P):

- The implementation is evaluated as ◐. Full support for PKI and X.509.
- Possible extensions are evaluated as ◐. Adding support for other trust models and other types of credentials would require major changes.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ●. X.509 as well as anonymous credentials can be specified in the policy.
- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ●. Credentials are translated into assertions that are taken into account when evaluating the policy.
- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ○. Expressing attributes of parties in policies is out of scope.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ●. DHP can express conditions over data in credentials (i.e., credential/declaration predicates).

- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

A.2.3 Can Express Data Handling

Privacy policies can express proposed data handling in terms of purpose, obligations, etc. In other words, PII Consumers express how collected data will be handled.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: Privacy policies express complex data handling that are enforced by the PII Consumer. This includes rights (e.g. use for purpose) and obligations (e.g. log usage).

- Partial implementation (◐) is defined as: Privacy policies express basic data handling, e.g. a flat list of purposes and data retention time.

- No implementation (○) is defined as: Cannot specify proposed data handling.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: This privacy policy language provides extension points to define complex data handling.

- Possible partial support (◐) is defined as: This privacy policy language could be extended with basic data handling.

- Cannot be supported (○) is defined as: Expressing data handling is out of scope and would require major modifications.

APPEL + P3P (A+P):

- The implementation is evaluated as ◐. P3P lets define primary and secondary purposes as well as data retention.

- Possible extensions are evaluated as ◐. Full support for data handling would be difficult to add. No extension is foreseen.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ●. PPL defines data handling in privacy policies in terms of rights and obligations. Both rights and obligations can be extended.

- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ●. S4P defines data handling in privacy policies in terms of rights ("may" queries) and obligations ("will" assertions). New types of rights and obligations can be defined.

- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ○. Privacy policies are out of scope.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ●. The language defines complex data handling including conditions, obligations, and data sharing.

- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

A.2.4 Can Express Downstream Claims (Credentials)

The policies can express credentials of third parties. In other words the policy specifies with what kind of third parties the data controller may share collected data.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: Policies express complex certification of third parties: different claims, different trust models (e.g. reputation, web of trust, PKI) and different mechanisms (e.g. X.509, anonymous credential).

- Partial implementation (◐) is defined as: Policies express basic certification of third parties: only identity-based or only usual mechanisms (e.g. Public Key Infrastructure).

- No implementation (○) is defined as: No support for certification of third parties.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: Reference to complex certification of third parties could be combined with existing policy language but is not yet implemented.
- Possible partial support (◐) is defined as: Reference to basic certification of third parties could be combined with existing policy language but is not yet implemented.
- Cannot be supported (○) is defined as: Support for downstream certification is out of scope and would require major extensions.

APPEL + P3P (A+P):

- The implementation is evaluated as ◐. P3P can specify that personal data may be shared with specific types of third parties. The actual policy of the third party is taken into account by the data controller.
- Possible extensions are evaluated as ◐. Support for other technologies would require major changes. No extension is foreseen.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ◐. The current implementation is based on "lazy matching". The downstream access control is not part of the policy. However, the credentials of the third party are taken into account when sharing downstream.
- Possible extensions are evaluated as ●. Using "proactive" matching algorithm [BNP10] makes it possible to define nested and recursive policies to define downstream access control.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ●. Downstream access control is handled by specifying to which third party the data may be forwarded and by importing (or referencing) the policy of this third party including its attributes.
- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ○. Privacy policies are out of scope.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ●. PRIME-DHP lets define a complex access control policy as part of the privacy policy.
- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

A.2.5 Can Express Downstream Data Handling

The policies can express proposed data handling of third parties. In other words the policy specifies how third parties would handle data they may get from the data controllers.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: Description of complex downstream data handling including rights (e.g. use for purpose) and obligations (e.g. log usage).

- Partial implementation (◐) is defined as: Description of minimum downstream data handling (purposes and data retention).

- No implementation (○) is defined as: This technology cannot specify data handling of third parties.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: The language could be extended to express complex data handling that is proposed by third parties. This may not yet be implemented.

- Possible partial support (◐) is defined as: The language could be extended to express minimum data handling

- Cannot be supported (○) is defined as: Expressing data handling proposed by third parties is out of scope and would require major changes.

APPEL + P3P (A+P):

- The implementation is evaluated as ◐. Different data handling for third parties can be expressed but this is complex. This would require multiple policies.

- Possible extensions are evaluated as ◐. No extension is foreseen.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ○. The current implementation is based on "lazy matching". The downstream data handling is not part of the policy.

- Possible extensions are evaluated as ●. Using "proactive" matching algorithm [BNP10] makes it possible to define nested and recursive policies to define downstream data handling.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ●. Downstream data handling is handled by specifying to which third party the data may be forwarded and by including (or referencing) the policy of this third party.

- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ○. Privacy policies are out of scope.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ◐. The sticky policy is forwarded with personal data downstream. This can be used to have similar privacy constraint downstream.

- Possible extensions are evaluated as ◐. Full support would require major changes. No extension is foreseen.

A.2.6 Can Retrieve Applicable Policy (Sect. 3.7)

There is a mechanism to get or generate the policy applicable to a given parameter, e.g. one "label" of an HTML Form, one parameter of a Web Service, or one claim of a requested credential.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: Complex generation of privacy policy taking into account user authentication, choices in interface, etc.

- Partial implementation (◐) is defined as: Static privacy policy describing all parameters of the PII controller interface.

- No implementation (○) is defined as: No support for parameters.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: More complex mechanisms could be built on top of existing building blocks but are not available.

- Possible partial support (◐) is defined as: Simple mechanisms could be built on top of existing building blocks but are not available.

- Cannot be supported (○) is defined as: Issuing Privacy Policy for collected data is out of scope and would require major changes.

APPEL + P3P (A+P):

- The implementation is evaluated as ◐. P3P is mainly used to create static policies.
- Possible extensions are evaluated as ◐. No extension is foreseen..

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ◐. The current implementation assumes static policies. Basic mechanisms to retrieve applicable policy are provided.
- Possible extensions are evaluated as ●. PPL could be extended with more dynamic mechanisms.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ○. Out of the scope of the core implementation.
- Possible extensions are evaluated as ●. A prototype preferences language does associate pieces of policies (mainly "may" queries and "will" assertions) with description of PII. This could be used to select relevant assertions and queries.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ○. Privacy policies are out of scope.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ●. Query mechanisms exist to retrieve policy from a databases.
- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

A.3 PII Store (Sect. 3.5)

Personal data are stored in a database and can be queried according to attributes such as the type of data (e.g. e-mail address) or its certification (e.g. name in identity card).

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: Unified PII Store to query different type of personal data.
- Partial implementation (◐) is defined as: Basic PII store listing existing personal data.
- No implementation (○) is defined as: No support for querying personal data.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: Unified PII Store to query different types of personal data could be built on top of existing building blocks but may not yet be implemented.
- Possible partial support (◐) is defined as: Basic PII Store to query different types of personal data could be built on top of existing building blocks but may not yet be implemented.
- Cannot be supported (○) is defined as: Querying personal data is out of scope and would require major changes.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. PII storage and query is out of the scope of P3P and APPEL.
- Possible extensions are evaluated as ◐. Different mechanisms have been proposed, e.g. W3C Device APIs WG [W3C10] proposes mechanisms to get contact information.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ◐. PPL's Credential Handler [Pri09a] provides an API to retrieve or generate PIIs.
- Possible extensions are evaluated as ◐. No extension is foreseen.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ○. Out of the scope of the core implementation.
- Possible extensions are evaluated as ●. Using a legacy databases or building a new PII store with S4P would be straightforward.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ●. This solution proposes remote storage of (personal) data and remote management of related authorizations. It must provide some way to query data.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ○. This is out of the scope of PRIME-DHP.
- Possible extensions are evaluated as ●. Prototypes using a database to store personal data have been developed with PRIME-DHP and could be reused and generalized.

A.4 Privacy-Aware Service Discovery (Sect. 4.1)

This technology provides mechanisms to discover services based on functional properties and on non-functional properties such as privacy.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: This technology can be used to discover services (PII Consumers) and to sort results based on matches between privacy policies and PII Provider's privacy preferences. Some metrics to order results exist.

- Partial implementation (◐) is defined as: This technology can be used to discover services (PII Consumers) fulfilling PII Provider's privacy preferences.

- No implementation (○) is defined as: No support for privacy-aware discovery.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: Complex privacy-aware service discovery can be built on top of existing building blocks.

- Possible partial support (◐) is defined as: Basic privacy-aware service discovery can be built on top of existing building blocks.

- Cannot be supported (○) is defined as: Privacy-aware discovery is out of scope and would require major changes.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. Service discovery is out of scope.

- Possible extensions are evaluated as ◐. Using P3P with Web Services (e.g. [W3C03]) is a step towards privacy-aware service discovery.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ○. In the current implementation, privacy-aware service discovery is out of scope.

- Possible extensions are evaluated as ●. A mechanism to "measure" mismatches have been specified [Pri10a]. This mechanism could be used to sort results of individual matches.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ○. Privacy-aware service discovery is out of the scope of the core implementation.
- Possible extensions are evaluated as ◐. Individual matches could be used to get a list of PII Consumers that satisfy PII Provider's privacy preferences.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ○. This is out of scope.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ○. This is out of the scope of PRIME-DHP.
- Possible extensions are evaluated as ○. Without specification of preferences and automatic match, privacy-aware service discovery would require additional work.

A.5 PII Lookup (Sect. 4.2)

This technology offers mechanisms to gather pieces of personal data that are required by a given interface of the PII Consumer.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: Complex PII Lookup: Possibility to handle optional parameters, disjunction (e.g. prove "age of majority" ∨ "birth date").
- Partial implementation (◐) is defined as: Basic PII Lookup: Possibility to match types of parameters (e.g. e-mail address) with types of personal data (e.g. Alice@Contoso.com).
- No implementation (○) is defined as: No support for specifying and gathering personal data.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: Complex PII Lookup could be built on top of existing building blocks.
- Possible partial support (◐) is defined as: Basic PII Lookup could be built on top of existing building blocks.
- Cannot be supported (○) is defined as: PII lookup is out of scope and would require major changes.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. PII lookup is out of scope.

- Possible extensions are evaluated as ◐. Different mechanisms have been proposed, e.g. W3C Device APIs WG [W3C10] proposes mechanisms to get contact information.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ◐. PPL language and Credential Handler [Pri09a] make it possible to require a proof that the data subject is, for instance, older than 18. The result may be a specific proof or an X.509 certificate with additional attributes.

- Possible extensions are evaluated as ●. More complex mechanisms are foreseen but not yet implemented. Mechanism to compare matches can help.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ○. Out of the scope of the core implementation.

- Possible extensions are evaluated as ◐. Analyzing PII Consumer's API and evaluating the impact of different options would be possible on top of S4P.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ◐. This is out of scope.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ○. This is out of the scope of PRIME-DHP.

- Possible extensions are evaluated as ◐. Analyzing PII Consumer's API and evaluating the impact of different options would be possible on top of PRIME-DHP.

A.6 Policy Matching (Sect. 4.3)

A.6.1 Has Logic Foundations

The evaluation whether privacy policies do fulfill privacy preferences has logic foundations.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: It is possible to prove that PII Controllers enforcing their privacy (sticky) policy cannot violate PII Providers' privacy preferences.
- Partial implementation (◐) is defined as: Policies and preferences are expressed in a logic-based language but there is no proof on matching and behavior.
- No implementation (○) is defined as: Matching has no logic-based foundations.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: Proof could be added.
- Possible partial support (◐) is defined as: Pieces of policy could be translated into a logic-based language.
- Cannot be supported (○) is defined as: Out of scope and would require major changes.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. No logic foundations. Semantics mismatches between APPEL and P3P have been documented [YLA04].
- Possible extensions are evaluated as ◐. Some works proposed logic-based representation of APPEL and P3P, e.g. [YLA04].

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ○. PPL does not have logic foundations.
- Possible extensions are evaluated as ◐. Parts of PPL have been expressed with *FORmal Modeling Using Logic Programming and Analysis* (FORMULA) [JSS08].

SecPAL for Privacy (S4P):

- The implementation is evaluated as ●. S4P is translated into DataLog before evaluation of queries. There are proofs that the translation provides expected properties.
- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ○. There is no matching algorithm in those solutions. Moreover, access control is generally not logic-based.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ○. No logic foundations.

- Possible extensions are evaluated as ◐. A specification using Semantic Web triples (subject-predicate-object) exists.

A.6.2 Takes Data Handling into Account

Expected data handling is expressed by the PII Provider and proposed data handling is expressed by the PII Consumer. Both aspects are taken into account during match.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: Description of complex local data handling including rights (e.g. use for purpose) and obligations (e.g. log usage) is defined on both sides and used during match.

- Partial implementation (◐) is defined as: Description of basic data handling (purposes and data retention) is defined on both sides and used during match.

- No implementation (○) is defined as: This technology cannot take data handling into account during match.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: The current implementation could be extended to take complex local data handling into account when matching policy and preferences.

- Possible partial support (◐) is defined as: The current implementation could be extended to take basic local data handling into account when matching policy and preferences.

- Cannot be supported (○) is defined as: Taking data handling into account during match is out of scope and would require major changes.

APPEL + P3P (A+P):

- The implementation is evaluated as ◐. Basic data handling (purpose, retention) are taken into account.

- Possible extensions are evaluated as ◐. No extension is foreseen.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ●. When matching privacy policy and preferences, PPL compares rights (e.g. use for purposes) and obligations (e.g. retention or notifications).

- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ●. When matching privacy policy and preferences, S4P compares rights (by evaluating "may" queries) and obligations (by evaluating "will" queries).

- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ○. This approach does not address matching algorithms.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ○. PRIME-DHP does not address matching algorithms.

- Possible extensions are evaluated as ◐. Ad-hoc matching has been defined for ad-hoc preferences in prototypes.

A.6.3 Takes Obligations into Account

Expected obligations are expressed by the PII Provider and proposed obligations are expressed by the PII Consumer. Both aspects are taken into account during match.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: Complex obligations including log, access to collected data, retention, notification, etc. are taken into account during match.

- Partial implementation (◐) is defined as: Basic obligations (data retention) are taken into account during match.

- No implementation (○) is defined as: This technology cannot take obligations into account during match.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: The current implementation could be extended to support matching complex obligations.
- Possible partial support (◐) is defined as: The current implementation could be extended to support matching basic obligations.
- Cannot be supported (○) is defined as: Matching obligations is out of scope and would require major changes.

APPEL + P3P (A+P):

- The implementation is evaluated as ◐. P3P and APPEL mainly support data retention with a small number of parameters.
- Possible extensions are evaluated as ◐. No extension is foreseen.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ●. A set of obligations have been defined for usual scenarios (delete, log, notify, etc.). New domain specific obligations can be introduced.
- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ◐. Few sample obligations has been defined. Creating new obligations is a feature of the language.
- Possible extensions are evaluated as ●. The language lets easily define new obligations. A prototype using obligations similar to the one defined in PPL has been implemented.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ○. This approach does not address matching algorithms.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ○. Out of scope. A set of obligations has been defined but they are not "matched".
- Possible extensions are evaluated as ◐. A set of obligations has been specified and are taken into account when matching the preference and the policy.

A.6.4 Takes Downstream Properties into Account (One Hop)

Not only the privacy policy of the data controller is taken into account during match but also the policy of third parties, which may get subsequently access to the personal data.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: Preferences express complex description (authorizations, obligations, credentials) of third parties and full privacy policies of third parties are taken into account during match.

- Partial implementation (◐) is defined as: Preferences express basic description of third parties and simplified aspects of privacy policy of third parties are taken into account during match.

- No implementation (○) is defined as: Properties of third parties are not taken into account.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: The current implementation could be extended to support matching of complex privacy policy of third parties.

- Possible partial support (◐) is defined as: The current implementation could be extended to support matching simple privacy policy of third parties.

- Cannot be supported (○) is defined as: Matching third parties privacy properties is out of scope and would require major changes.

APPEL + P3P (A+P):

- The implementation is evaluated as ◐. The policy of a third party can be taken into account.

- Possible extensions are evaluated as ○. No extension is foreseen.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ◐. The current implementation supports "lazy matching" that results in doing the downstream match when the data is forwarded.

- Possible extensions are evaluated as ●. Another matching algorithm ("proactive matching") has been specified and allow matching nested and recursive preferences and policies.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ●. All relevant PII consumers are taken into account when evaluating queries.

- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ○. This approach does not address matching algorithms.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ○. Out of Scope. Since the privacy policy defines a full access control policy, matching preferences and policies with downstream would end in comparing access control policies, which is a difficult problem.

- Possible extensions are evaluated as ○. No extension is foreseen.

A.6.5 Supports Recursive Downstream

Complex chains of downstream data sharing can be expressed in privacy policies and preferences and can be taken into account during match.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: Chains of downstream data sharing can be specified with delegation mechanisms and/or nested policies.

- Partial implementation (◐) is defined as: Specification of preferences is defined in a flat way but can be reused recursively.

- No implementation (○) is defined as: Matching algorithm cannot handle complex data sharing chains.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: The current implementation could be extended to define complex chains of data sharing.

- Possible partial support (◐) is defined as: The current implementation could be extended to define reusable preferences.

- Cannot be supported (○) is defined as: Recursive match is out of scope and would require major changes.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. Out of scope.
- Possible extensions are evaluated as ○. No extension is foreseen.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ◐. The current implementation supports "lazy matching" that results in doing the downstream match when the data is forwarded. This works with recursive preferences.
- Possible extensions are evaluated as ●. Another matching algorithm ("proactive matching") has been specified and allow matching nested and recursive preferences and policies.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ●. Privacy preferences and policies related to specific PII Consumer are connected by "may send" and evaluated together.
- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ○. This approach does not address matching algorithms.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ○. PRIME-DHP does not address matching algorithms.
- Possible extensions are evaluated as ○. No extension is foreseen.

A.7 PII Selection (Sect. 4.4)

Privacy-aware identity selection is supported by the protocol (i.e. privacy policies are specified for all expected claims and privacy preferences are associated with all issued claims) and by the user interface (i.e. the selection of claims takes privacy into account).

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: It is possible to select different types of personal data (claim, value, ...) and privacy policies are taken into account. A user interface is available to handle selection and mismatches.

- Partial implementation (◐) is defined as: Basic selection of different types of personal data (claim, value, ...) or privacy policies taken into account.
- No implementation (○) is defined as: Selection of personal data is out of scope or not implemented.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: Existing language and mechanisms could be used to build identity selection. A dedicated user interface or an extension of an existing identity selection would be possible.
- Possible partial support (◐) is defined as: Existing language and mechanism could be used to build simple identity selection
- Cannot be supported (○) is defined as: Privacy-aware identity selection is not supported.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. Out of scope.
- Possible extensions are evaluated as ○. No extension is foreseen.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ●. PPL provides an interface to combine identity selection (with potentially multiple claims from different credentials) and (mis)match information. See [Pri10b] for more details.
- Possible extensions are evaluated as ●. This user interface may be extended with measure of disclosure impact [ADF+10].

SecPAL for Privacy (S4P):

- The implementation is evaluated as ○. Not implemented (out of scope of the core language and query evaluation).
- Possible extensions are evaluated as ●. Integrating identity selection and S4P is possible but would require some development effort.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ○. Out of scope.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ◐. An identity selector have been extended with support for privacy policies.
- Possible extensions are evaluated as ◐. No extension is foreseen.

A.8 Change Preferences (Sect. 4.5)

A.8.1 Can Show Mismatches

In case of mismatch, the root causes of the mismatch can be identified and highlighted.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: All major root cause of mismatching privacy preferences and policies are available and can be sorted with different criteria.

- Partial implementation (◐) is defined as: One major root cause of mismatching privacy preferences and policies is identified.

- No implementation (○) is defined as: Root cause of mismatching privacy preferences and policies is not available.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: Tools to identify and enumerate all root causes of mismatching preferences and policies could be built on top of existing language and mechanisms.

- Possible partial support (◐) is defined as: Tools to identify one root cause of mismatching preferences and policies could be built on top of existing language and mechanisms

- Cannot be supported (○) is defined as: Analyzing the cause of mismatches is out of scope and would require major changes.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. Out of scope.

- Possible extensions are evaluated as ◐. Privacy Bird [AC] provides an analysis of mismatches.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ◐. PPL can provide the most relevant root cause of a given mismatch. Only subset of the policy language is supported.

- Possible extensions are evaluated as ◐. No extension is foreseen. Enumerating causes would be possible.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ○. Not implemented (out of scope of the core language and query evaluation).
- Possible extensions are evaluated as ●. Query evaluation results in a proof graph that can be analyzed to identify root causes of a mismatch.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ○. This feature is out of scope because privacy preferences are not defined in this model.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ○. This is out of scope because PRIME-DHP does not specify preferences.
- Possible extensions are evaluated as ○. No extension is foreseen.

A.8.2 Can Suggest Modifications

Privacy preferences can be automatically updated to get a match next time a similar case occurs. Previous changes, and similarity of preferences can be taken into account.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: The modification of preferences can have different results (e.g. adding an exception or generalizing a rule) that depend on the PII Provider. This may be achieved by user interactions or by specifying "meta-preferences".
- Partial implementation (◐) is defined as: The modification of the preferences is straightforward. The first privacy-friendly modification is proposed.
- No implementation (○) is defined as: No support for automatic modifications of preferences.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: Tools to specify how the user expect his preferences to evolve can be built on top of existing language and mechanisms.
- Possible partial support (◐) is defined as: Tools to propose a straightforward modification (e.g. exception) of a mismatching preference in order to have a match can be implemented on top of existing language and mechanisms.
- Cannot be supported (○) is defined as: Automatic modifications of the preferences are out of scope and would require major work.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. Out of scope.
- Possible extensions are evaluated as ◐. Privacy Bird [AC] can suggest modifications of preferences.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ◐. PPL proposes the most relevant modification of preferences. This only works for a subset of the language.
- Possible extensions are evaluated as ◐. No extension is foreseen.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ○. Not implemented (out of scope of the core language and query evaluation).
- Possible extensions are evaluated as ◐. SecPAL abduction queries [BMD09] can propose missing assertions. More work would be required to cover the full language.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ○. This feature is out of scope because privacy preferences are not defined in this model.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ○. This is out of scope because PRIME-DHP does not specify preferences.
- Possible extensions are evaluated as ○. No extension is foreseen.

A.9 Sticky Policy (Sect. 4.6)

A.9.1 Optional Sticky Policy

Instead of creating a sticky policy describing agreed privacy constraints on personal data, a Boolean response can be used to state that the privacy policy is acceptable and must be enforced. The Boolean response can be implicit, e.g. agree by sending personal data.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: The policy can be enforced directly.
- Partial implementation (◐) is defined as: The sticky policy can be a copy of the policy.
- No implementation (○) is defined as: It is mandatory to match preferences and policies in order to create a sticky policy.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: Enforceable policy may not be implemented but could be done on top of existing mechanisms.
- Possible partial support (◐) is defined as: Enabling the use of a policy as sticky policy may not be implemented but could be done on top of existing mechanisms.
- Cannot be supported (○) is defined as: Enforcing policies is out of scope and would require major changes.

APPEL + P3P (A+P):

- The implementation is evaluated as ●. APPEL and P3P do not support sticky policies and always result in a Boolean response and the enforcement of the privacy policy.
- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ○. Out of scope. The model imposes sticky policies.
- Possible extensions are evaluated as ●. The languages to express policies and sticky policies are almost identical and enforcing policies would be possible.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ●. The response can be Boolean. Sticky policies are supported but their generation has not been implemented.
- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ○. Impossible in this model where the sticky policy (i.e. policy configured remotely) is the only policy.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ◐. The sticky policy is a "filled" version of the policy where variables are instantiated.
- Possible extensions are evaluated as ◐. No extension is foreseen.

A.9.2 Can be Expressive

The sticky policy can express complex constraints with conditions.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: The sticky policy can specify obligations, rights, access control for downstream, etc.
- Partial implementation (◐) is defined as: The sticky policy can specify basic data handling.
- No implementation (○) is defined as: Sticky policies are not supported

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: Expressive sticky policies can be built on top of existing mechanisms.
- Possible partial support (◐) is defined as: Basic sticky policies can be built on top of existing mechanisms.
- Cannot be supported (○) is defined as: Sticky policies are out of scope and would require major changes.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. Out of scope: no support for sticky policies
- Possible extensions are evaluated as ○. This is out of scope.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ●. Sticky policies define data handling (obligations and rights) and access control.
- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ○. Not implemented (sticky policies are out of scope of the core language and query evaluation).
- Possible extensions are evaluated as ●. Sticky policies can be expressed in S4P. However, mechanisms to create them from preferences and policies are not implemented.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ●. The policy that is remotely configured (e.g. in XACML) is the sticky policy.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ●. Sticky policies define data handling (obligations and rights) and access control.
- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

A.9.3 Supports Signature or Commitment

The sticky policy can be signed by one or more parties to ensure non-repudiation of agreed privacy constraints.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: A protocol exists to make sure that no data is communicated before agreeing on a sticky policy. Non-repudiation of all parties is ensured.
- Partial implementation (◐) is defined as: The sticky policy can be signed.
- No implementation (○) is defined as: Signing sticky policy is not supported.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: A protocol to commit to sticky policy could be built on top of existing building blocks.
- Possible partial support (◐) is defined as: A mechanism to sign the sticky policies could be implemented.
- Cannot be supported (○) is defined as: Signing sticky policies is out of scope or would require major changes.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. This is out of scope.
- Possible extensions are evaluated as ○. XML Signature can be used to sign the P3P policy in order to ensure PII Consumer side non-repudiation.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ○. Not implemented.

- Possible extensions are evaluated as ●. Adding such a protocol would be easy to add to the current implementation

SecPAL for Privacy (S4P):

- The implementation is evaluated as ○. Not implemented (out of scope of the core language and query evaluation).

- Possible extensions are evaluated as ●. S4P can be serialized in XML and a commitment protocol could be added.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ◐. The PII Provider could sign the policy she is issuing.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ○. Not supported in current implementation.

- Possible extensions are evaluated as ●. Adding a commitment protocol would be possible.

A.9.4 Can Change Sticky Policy

There is a mechanism to let data subjects modify sticky policies associated with their own personal data when such an action is authorized.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: The PII Provider, if authorized to do so, can subsequently update sticky policies associated with her data at PII Consumer.

- Partial implementation (◐) is defined as: The sticky policy can be updated but authorization to do so is handled with an ad-hoc mechanism.

- No implementation (○) is defined as: Out of scope or sticky cannot be updated.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: It is possible to build on top of the existing language and mechanisms full support for modification of sticky policies.
- Possible partial support (◐) is defined as: It is possible to build on top of existing language and mechanisms basic support for sticky policy modification in an ad-hoc way.
- Cannot be supported (○) is defined as: Modification of sticky policies is out of scope and would require major work.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. This is out of scope.
- Possible extensions are evaluated as ○. Modifying the policy (in a less permissive way) is possible. If the policy must become more permissive, data has to be collected again.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ◐. Supported by the language specification but not implemented.
- Possible extensions are evaluated as ●. The obligation to let modify data and/or sticky policies has been defined.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ○. Not implemented (out of scope of the core language and query evaluation).
- Possible extensions are evaluated as ◐. Possible but not straightforward. Would require new verb "edit Policy" and hook to authorization.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ●. Supported by default. No difference between setting policy and editing this one.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ○. Out of scope.
- Possible extensions are evaluated as ○. Not documented. Out of scope.

A.9.5 Can Store and Retrieve Sticky Policy (Sect. 3.8)

There is a mechanism to store sticky policies and to query the sticky policy associated with a given piece of personal data.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: Mechanisms to store and retrieve sticky policies in an efficient way are available.
- Partial implementation (◐) is defined as: Sticky policies are stored in a databases and can be queried with the identifier of the personal data.
- No implementation (○) is defined as: No mechanism to handle sticky policies is provided.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: Mechanisms to store and retrieve sticky policies in an efficient way could be built on top of existing building blocks.
- Possible partial support (◐) is defined as: Mechanisms to store sticky policies and query them with a unique identifier could be built on top of existing building blocks.
- Cannot be supported (○) is defined as: Storing sticky policies is out of scope or would require important work.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. This is out of scope.
- Possible extensions are evaluated as ○. This is out of scope.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ◐. Possibility of querying sticky policy knowing the unique identifier of the personal data.
- Possible extensions are evaluated as ●. Optimizing sticky policy storage and complex query could be built on top of existing building blocks.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ○. Not implemented (out of scope of the core language and query evaluation).
- Possible extensions are evaluated as ●. Optimizing sticky policy storage and complex query could be built on top of existing building blocks.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ●. Full support from access control infrastructure, e.g. XACML.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ◐. Sticky policies are stored in a database and can be queried by identifier.
- Possible extensions are evaluated as ●. Optimizing sticky policy storage and complex query could be built on top of existing building blocks.

A.10 Attach Sticky Policy (Sect. 4.7)

Mechanism to attach the sticky policy to data on the wire and in databases. Mechanisms such as Enterprise Rights Management (e.g. [Mic09]) would be an example where personal data cannot be decrypted without acknowledging the sticky policy (i.e. licenses).

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: Sticky policies are strongly linked to personal data. Mechanisms like in Enterprise Rights Management (e.g. [Mic09]) make it mandatory to handle the sticky policy before reading the data.
- Partial implementation (◐) is defined as: A basic mechanism to bind personal data and sticky policy on the wire and in databases exists.
- No implementation (○) is defined as: No mechanism to link data and sticky policy is provided.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: Strong mechanism to link data and sticky policies could be added independently of the language.
- Possible partial support (◐) is defined as: Basic mechanism to link data and sticky policies could be added independently of the language.
- Cannot be supported (○) is defined as: Attaching sticky policies to data is out of scope or would require important work.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. This is out of scope since there is no support for sticky policies.
- Possible extensions are evaluated as ○. This is out of scope.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ◐. Unique identifier is used to keep the link between data and sticky policies.
- Possible extensions are evaluated as ●. It would be possible to use a PPL sticky policy as a "license" attached to personal data (e.g. a document). This usage is however not specified.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ○. Not implemented (out of scope of the core language and query evaluation).
- Possible extensions are evaluated as ●. It would be possible to use a S4P sticky policy as a "license" attached to personal data (e.g. a document). This usage is however not specified.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ◐. Out of scope

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ◐. Unique identifier is used to keep the link between data and sticky policies.
- Possible extensions are evaluated as ●. It would be possible to use a PRIME-DHP sticky policy as a "license" attached to personal data (e.g. a document). This usage is however not specified.

A.11 High-Level Policy Language (Sect. 4.8)

A.11.1 Same Language for Preferences and Policies

Privacy preferences, policies, and sticky policies are expressed in a common language that avoid semantics mismatches.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: The languages are identical apart small details, e.g. preferences specify deletion time relatively to data exchange instant while sticky policies specify absolute times.
- Partial implementation (◐) is defined as: The language are different dialects with common syntax and semantics.
- No implementation (○) is defined as: Different languages.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: It is possible to specify a unified language on top of existing building blocks.

- Possible partial support (◐) is defined as: It is possible to specify similar dialects on top of existing building blocks.

- Cannot be supported (○) is defined as: A common language is out of scope and would require major work.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. APPEL and P3P have different syntaxes. Semantics issues are highlighted in [YLA04].

- Possible extensions are evaluated as ○. No extension is foreseen.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ●. Same language for a large part of policies, preferences, and sticky policies.

- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ◐. Common language for preferences and policies. However, "will" queries are specific to preferences and "may" queries are specific to policies.

- Possible extensions are evaluated as ◐. Expressing part of the policy and preferences with symmetric assertions and queries is a key feature of the language.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ○. Out of scope: only one language for sticky policies.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ○. Same language for policy and sticky policy. However there is no language for preferences.

- Possible extensions are evaluated as ○. No extension is foreseen.

A.11.2 Language Expressiveness

The common language is expressive and allows the specification of conditions, nested or recursive policies, and variables.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: The language supports conditions, nested or recursive policies, and variables.
- Partial implementation (◐) is defined as: The language supports conditions and variables.
- No implementation (○) is defined as: No support for conditions or variables.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: Conditions and nested policies could be added on top of the existing language.
- Possible partial support (◐) is defined as: Conditions could be added on top of the existing language.
- Cannot be supported (○) is defined as: Conditions and nested policies are out of scope and would require major changes.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. Basic language without conditions or variables.
- Possible extensions are evaluated as ◐. Combining P3P with semantic web (RDF) and ontology (OWL) makes it more expressive.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ●. Support complex conditions (XACML) in the access control part but not in the data handling part. Support for nested and recursive policies.
- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ●. Support for constraints, conditions, and delegation. Policies can be chained.
- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ●. Full expressiveness of access control language such as XACML.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ●. This language supports variables and complex conditions.
- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

A.11.3 Clear Separation of Obligations and Rights

Obligations and rights are clearly expressed to handle, for instance, the right to store personal data 3 months, the obligation of storing data 3 months, and the obligation of deleting data within 3 months.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: Rights and obligations are explicitly separated and do not overlap.
- Partial implementation (◐) is defined as: Rights and obligations are specified differently but overlaps exist, e.g. to specify rights of executing obligations.
- No implementation (○) is defined as: No clear separation between rights and obligations.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: The language could be used in a way that clearly separates rights and obligations and avoids overlaps.
- Possible partial support (◐) is defined as: The language could be used in a way that separates rights and obligations with minimum overlaps.
- Cannot be supported (○) is defined as: The language does not explicitly support concepts such as rights or obligations. Support would require major work.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. Even if some rights and obligations are specified, there are no explicit concepts of rights and obligations.
- Possible extensions are evaluated as ○. No extension is foreseen.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ●. Rights are defined to authorize local use of data or to forward data. Obligations are defined for data retention, log, notifications, etc.

- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ◐. Modal verbs "will" and "may" specify obligations and rights respectively. However, the same action can be associated with both notions (e.g. right to notify and obligation of notifying).

- Possible extensions are evaluated as ◐. No extension is foreseen.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ●. Even if such approaches mainly focus on rights (access control), some locally defined obligations may be supported (e.g. XACML let define some obligations). In this case, there is a clear distinction between rights and obligations.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ●. Rights and obligations are defined without overlaps.

- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

A.12 Check Sticky Policy (Sect. 5.2)

When a sticky policy is pushed to a PII Consumer, this one can check whether the sticky policy is acceptable, i.e. more permissive than the related policy.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: Both PII Provider and PII Consumer can check that the sticky policy is acceptable.

- Partial implementation (◐) is defined as: The PII Consumer can check that the sticky policy is acceptable. In other words, a malicious PII Provider cannot inject unexpected privacy constraints.

- No implementation (○) is defined as: Verifying sticky policies is out of scope or not implemented.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: Mutual verification could be implemented with existing mechanisms.
- Possible partial support (◐) is defined as: PII Consumer-side verification of sticky policies could be implemented with existing building blocks.
- Cannot be supported (○) is defined as: Verification of sticky policies is out of scope or would require major changes.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. This is out of scope. The P3P policy is enforced and thus does not require verification.
- Possible extensions are evaluated as ○. This is out of scope.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ○. Specified but not yet implemented.
- Possible extensions are evaluated as ●. Straightforward to implement: another usage of matching algorithm.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ○. Not implemented (out of scope of the core language and query evaluation).
- Possible extensions are evaluated as ●. When the sticky policy is evaluated, the policy is taken into account as well. As a result, a non-acceptable sticky policy would result in unsuccessful query evaluations.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ●. The language (and tools) to remotely specify the policy that applies to data can add constraints.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ●. The sticky policy is a "filled" version of the policy where variables are instantiated. As a result, checking the sticky policy is straightforward.
- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

A.13 Authorization Decision (Sect. 5.3)

A.13.1 Enforces Local Use, e.g. Purpose (Sect. 5.4)

Before using collected data, the PII Consumer can verify that actions are authorized according to sticky policies.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: A dynamic authorization decision based on the sticky policies is performed before using collected data for a specific purpose. Authorization decision can trigger obligations.
- Partial implementation (◐) is defined as: Static authorization decision. The PII Consumer and its policy are designed in a way that makes it impossible to violate the (sticky) policy.
- No implementation (○) is defined as: Authorization decision is not implemented.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: Dynamic authorization decision for local use could be implemented on top of existing building blocks.
- Possible partial support (◐) is defined as: Static Authorization decision for local use could be implemented on top of existing building blocks.
- Cannot be supported (○) is defined as: Authorization verification for local use is out of scope and would require major changes.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. Enforcement is out of the scope of P3P and APPEL.
- Possible extensions are evaluated as ●. EPAL [AHK+03] can be used to enforce the (sticky) policy locally.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ●. An extension of XACML PDP point is used to make purpose-aware decision based on the sticky policy.
- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ○. Not implemented (out of scope of the core language and query evaluation).

- Possible extensions are evaluated as ●. Straightforward to implement. A query specifying the usage has to be evaluated with all assertions of the policy and sticky policy.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ◐. The language (e.g. XACML) and services (e.g. PEP and PDP) are used to grant or deny access. Support for purpose is generally lacking.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ●. Tools exist to decide whether a piece of personal data can be used for a given purpose.

- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

A.13.2 Enforces Access Control when Sharing (Sect. 5.5)

Authorization of sharing data with a third party takes into account the sticky policy and attributes (e.g. certificates) of the third party.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: A dynamic authorization decision based on the sticky policies is performed before sharing collected data with a third party. Sharing data can trigger obligations.

- Partial implementation (◐) is defined as: Static authorization decision before sharing. The PII Consumer and its policy are designed in a way that makes it impossible to share data with an unauthorized third party.

- No implementation (○) is defined as: No authorization decision based on third party properties is implemented.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: Dynamic authorization decision before data sharing could be implemented on top of existing building blocks.

- Possible partial support (◐) is defined as: Static Authorization decision before data sharing could be implemented on top of existing building blocks.

- Cannot be supported (○) is defined as: Authorization verification before data sharing is out of scope and would require major changes.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. Enforcement is out of the scope of P3P and APPEL.

- Possible extensions are evaluated as ◐. EPAL [AHK+03] can be used to enforce the (sticky) policy and decide to share the data with a third party.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ●. XACML part of the sticky policy is used to decide whether data can be shared with a given third party.

- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ○. Not implemented (out of scope of the core language and query evaluation).

- Possible extensions are evaluated as ●. Straightforward to implement. A query specifying the usage has to be evaluated with all assertions of the policy and sticky policy.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ●. The language (e.g. XACML) and services (e.g. PEP and PDP) are used to grant or deny access to third parties.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ●. PRIME-DHP specifies and enforce access control when third parties access collected personal data.

- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

A.13.3 Checks Downstream Data Handling when Sharing

Authorization of sharing data with a third party takes into account the sticky policy of the personal data and the privacy policy of the third party.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: Downstream data sharing reuses policy language and enforcement mechanisms in a recursive way and enables multi-hops enforcement.
- Partial implementation (◐) is defined as: Downstream data sharing takes into account the privacy policy of the third party.
- No implementation (○) is defined as: Third party's policy is not taken into account.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: The current language and mechanisms could be reused recursively.
- Possible partial support (◐) is defined as: The current language could be extended with basic support to take privacy policy of third parties into account.
- Cannot be supported (○) is defined as: Matching privacy policy of third parties is out of scope or would require major work.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. Enforcement is out of the scope of P3P and APPEL. It is however possible to make sure that data is only shared under stricter privacy constraints.
- Possible extensions are evaluated as ○. EPAL [AHK+03] is mainly targeting single trust domain.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ●. PPL sticky policy specifies expected data handling which is matched with the data handling specified in third party's policies.
- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ○. Not implemented (out of scope of the core language and query evaluation).
- Possible extensions are evaluated as ●. Straightforward to implement. Queries from the sticky policy ("will") and from the policy ("may") are evaluated with all assertions.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ○. This is out of scope.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ○. This is out of scope. Downstream sharing is evaluated as a pure access control decision.

- Possible extensions are evaluated as ○. Major changes would be required.

A.13.4 Attach (New) Sticky Policy when Sharing

A new sticky policy is created when the personal data is shared with a third party. The rights and obligations of a third party may be different than the rights and authorizations of the initial data controller.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: A new sticky policy targeting the exchange between the PII Consumer and a given third party is created after matching.

- Partial implementation (◐) is defined as: The sticky policy can be "forwarded" with personal data when they are shared with third parties.

- No implementation (○) is defined as: No support for sticky policies downstream.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: The current language and mechanisms could be reused to create a new sticky policy.

- Possible partial support (◐) is defined as: Existing mechanisms could be reused to keep the original sticky policy attached with data when forwarding them to third parties.

- Cannot be supported (○) is defined as: No support for sticky policies downstream. This is out of scope or would require major work.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. Enforcement is out of the scope of P3P and APPEL.

- Possible extensions are evaluated as ○. This is out of scope.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ●. Matching sticky policy and policy of third party results in a new sticky policy.

- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ○. Not implemented (out of scope of the core language and query evaluation).

- Possible extensions are evaluated as ●. Mechanism to create a sticky policy from preferences of the PII Provider and the policy of PII Consumer could be reused to create a "downstream sticky policy" from the original sticky policy and the policy of the third party.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ○. This is out of scope.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ◐. The original sticky policy is forwarded downstream.

- Possible extensions are evaluated as ◐. Further support would require major changes.

A.14 Composing Sticky Policies (Sect. 5.6)

Possibility of computing the resulting sticky policy $sp_{1,2}$ of personal data $pii_{1,2}$ resulting from the combination of multiple personal data. In other words, defining each $sp_{1,2} = F(sp_1, sp_2)$ for each way of combining $pii_{1,2} = f(pii_1, pii_2)$.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: Two sticky policies can be merged when data are combined.

- Partial implementation (◐) is defined as: The language makes it possible to associate multiple sticky policy with one piece of data and to interpret it without violating individual policies.

- No implementation (○) is defined as: No support for composing sticky policies.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: It is possible to merge policies expressed in the current language.
- Possible partial support (◐) is defined as: Existing language and enforcement could be used to support multiple sticky policies associated with one piece of data.
- Cannot be supported (○) is defined as: No support for composing sticky policies. This is out of scope or would require major work.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. This is out of scope.
- Possible extensions are evaluated as ◐. This is out of scope.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ○. This is not implemented.
- Possible extensions are evaluated as ◐. Part of the language (especially data handling) could easily be composed. Other parts (e.g. access control) would be complex and require more work.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ○. Not implemented (out of scope of the core language and query evaluation).
- Possible extensions are evaluated as ◐. Associating different sticky policies with one piece of data would require extra logic to handle it properly.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ◐. This is out of scope.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ○. This is out of scope.
- Possible extensions are evaluated as ○. This is out of scope.

A.15 Obligations (Sect. 5.7)

A.15.1 Supports Enforcement of Obligations

There are mechanisms to automatically enforce obligations that can be specified in (sticky) policies.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: Mechanisms to enforce obligations have been implemented and can easily be extended to support new obligations and new legacy systems.
- Partial implementation (◐) is defined as: Basic enforcement mechanisms exist but cannot be easily extended to other systems.
- No implementation (○) is defined as: Enforcement of obligations is not supported.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: Existing language and mechanisms could be used to enforce obligations. Support for new obligations and legacy systems would be possible.
- Possible partial support (◐) is defined as: Existing language and mechanisms could be used for basic enforcement of obligations.
- Cannot be supported (○) is defined as: Enforcement of obligations is out of scope and would require major work.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. Enforcement is out of the scope of P3P and APPEL.
- Possible extensions are evaluated as ◐. EPAL [AHK+03] can be used to enforce the sticky policy. However, the obligations would be restricted to what P3P and APPEL can express even if EPAL is more expressive.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ●. The "Obligation Enforcement Engine" is in charge of enforcing obligations. Plug-ins can be implemented to handle new obligations or enforcement targeting new systems.
- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ●. Not implemented (out of scope of the core language and query evaluation).
- Possible extensions are evaluated as ●. A prototype enforcing S4P obligations has been implemented. [BMB10] specifies how sticky policies and policies must be enforced.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ◐. XACML offers a placeholder for simple obligations triggered by access control decisions. Enforcement is not specified but implementations do exist.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ◐. PRIME DHP defines a placeholder for arbitrary obligations. Only a simple subset has been implemented.

- Possible extensions are evaluated as ●. Enforcement of more complex obligations could be added.

A.15.2 Checks Rights of Enforcing Obligations

Mechanisms to define lower bound and upper bound of behavior.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: The language lets define expected and committed privacy-relevant behavior in terms of upper bound and lower bound.

- Partial implementation (◐) is defined as: Obligation and rights are specified as ranges. For instance, it is possible to say that "user must be notified each x" where $x \in$ (week, month).

- No implementation (○) is defined as: Enforcement of obligations is implicitly granted.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: Upper an lower bound of behavior could be expressed on top of the existing language.

- Possible partial support (◐) is defined as: Parameters in rights and obligations could be ranges.

- Cannot be supported (○) is defined as: Explicitly granting rights of enforcing obligations is out of scope and would require major work.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. This is out of scope.

- Possible extensions are evaluated as ○. This is out of scope.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ○. Not implemented. The right of enforcing obligations is implicitly granted.

- Possible extensions are evaluated as ◐. An extension of the language to support ranges is proposed in [BNS10] but would require modifications of the language and engine.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ●. To enforce obligations (e.g. "will delete"), it is mandatory to be authorized to do so (e.g. "may delete").

- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ○. Out of scope

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ◐. PRIME DHP lets define constraints on variables to be instantiated as sticky policy.

- Possible extensions are evaluated as ◐. No extension is foreseen.

A.15.3 Specifies Action Handler (Sect. 5.8)

There are mechanisms to parse and execute actions associated with obligations. It is possible to extend the set of actions that are handled.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: Usual actions (delete, notify, log, etc.) are provided and can be extended with new actions. Actions can be interpreted by matching algorithm and can be enforced.

- Partial implementation (◐) is defined as: Usual actions are provided or a place holder to define actions is provided.

- No implementation (○) is defined as: Actions are out of the scope of this implementation.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: Any action can be built on top of the existing framework.
- Possible partial support (◐) is defined as: Existing tools and language could be extended to support a set of actions.
- Cannot be supported (○) is defined as: Specifying actions is out of scope and would require major work.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. This is out of the scope of P3P and APPEL.
- Possible extensions are evaluated as ◐. EPAL [AHK+03] can be to define actions resulting from obligations but this cannot be specified in P3P.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ●. PPL provides a set of actions and a mechanisms to define new actions.
- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ○. This is not implemented (out of scope of the core language and query evaluation).
- Possible extensions are evaluated as ●. Adding new actions only requires specifying new verbs. Enforcement has to be implemented.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ●. Some authorization language, e.g. XACML, support the definition of arbitrary actions in obligations. The semantics and enforcement are locally defined.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ◐. PRIME -DHP provides a policy section where obligations can be defined. Within PRIME architecture, obligations are handled by a specific component that is able to manage events and trigger actions.
- Possible extensions are evaluated as ◐. Specifying a full set of actions and taking them into account during match and enforcement would require additional work.

A.15.4 Specifies Event Handler (Sect. 5.9)

There are mechanisms to parse triggers and react to specific events (time, event) leading to the execution of an action. It is possible to extend the set of triggers that are handled.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: Usual triggers (scheduled time, data used for purpose, data deleted, data shared, etc.) are provided and can be extended with new triggers. Triggers can be interpreted by matching algorithm and can be enforced.

- Partial implementation (◐) is defined as: Usual triggers are provided or a place holder to define triggers is provided.

- No implementation (○) is defined as: Triggers are out of the scope of this implementation.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: Any trigger can be built on top of the existing framework.

- Possible partial support (◐) is defined as: Existing tools and language could be extended to support a set of triggers.

- Cannot be supported (○) is defined as: Specifying triggers is out of scope and would require major work.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. This is out of the scope of P3P and APPEL.

- Possible extensions are evaluated as ◐. EPAL [AHK+03] can be used to react to specific events but this cannot be specified in P3P.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ●. PLL provides a set of triggers and mechanisms to define new ones.

- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ○. Not implemented (out of scope of the core language and query evaluation).

- Possible extensions are evaluated as ●. Adding new actions only requires specifying new verbs. Enforcement has to be implemented.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ◐. Some authorization language, e.g. XACML, support specific triggers for obligations. Generally, only triggers related to access control decision are supported.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ◐. PRIME -DHP provides a policy section where obligations can be defined. Within PRIME architecture, obligations are handled by a specific component that is able to manage events and trigger actions.

- Possible extensions are evaluated as ◐. Specifying a full set of triggers and taking them into account during match and enforcement would require additional work

A.16 Log and Audit (Sect. 5.10)

There are mechanisms to log privacy-relevant events such as: use of personal data, authorization decisions, obligation enforcement, etc. Audit can be based on those traces.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: The semantics of traces and policy is clear and it is possible to verify that traces fulfill a policy. Tools to log privacy-relevant events exist.

- Partial implementation (◐) is defined as: Tools to log privacy-relevant events exist.

- No implementation (○) is defined as: Tools to log privacy-relevant events are out of scope or not implemented.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: The semantics of traces and policy is clear and it is possible to verify that a set of traces fulfill a policy, i.e. traces can be audited. Tools to log privacy-relevant events could be implemented on top of existing language and mechanisms.

- Possible partial support (◐) is defined as: Tools to log privacy-relevant events could be implemented on top of existing language and mechanisms.

- Cannot be supported (○) is defined as: Tools to log privacy-relevant events could be implemented on top of existing language and mechanisms.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. This is out of scope.

- Possible extensions are evaluated as ◐. Not part of the policy but could be enforced with EPAL.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ◐. PPL specifies the obligation to log specific events and enforcement mechanisms.

- Possible extensions are evaluated as ◐. No extension is foreseen.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ○. Not implemented (out of scope of the core language and query evaluation).

- Possible extensions are evaluated as ●. The semantics of traces and of the language are formally specified. A prototype generating traces and validating them has been implemented.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ◐. Generating traces is generally restricted to access control decision. Audit of traces is not specified.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ◐. PRIME-DHP can specify which events has to be logged and provides enforcement mechanisms.

- Possible extensions are evaluated as ◐. No extension is foreseen.

A.17 Trust Model (Sect. 5.11)

Support for different trust models such as certification, audit, reputation, and/or trusted hardware. This makes the link between the committed behavior and the actual behavior.

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: Preferences and policies define the trust model that is expected and implemented respectively. For instance, properties of auditors, reputation mechanisms, or certified hardware are taken into account.
- Partial implementation (◐) is defined as: Preferences and policies define classes of trust model.
- No implementation (○) is defined as: Trust model is out of scope or handled by external mechanisms.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: Existing mechanisms and language could be used to take into account properties of different trust models.
- Possible partial support (◐) is defined as: Existing mechanisms and language could be used to take into account classes of trust models.
- Cannot be supported (○) is defined as: Tackling trust model is out of scope and would require major changes.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. This is out of scope.
- Possible extensions are evaluated as ○. This is out of scope.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ○. Not implemented. Certification and reputation is assumed.
- Possible extensions are evaluated as ◐. Minor modifications could be implemented to take such properties into account.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ◐. Complex properties of PII Consumer is supported in preferences and policies. This can be taken into account.
- Possible extensions are evaluated as ●. Additional work would be required to prove the claims in policies regarding reputation, trusted hardware and so on.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ◐. Mechanisms to specify the trust model can be added.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ○. This is not implemented. Certification and reputation is assumed.

- Possible extensions are evaluated as ◐. Minor modifications could be implemented to take such properties into account.

A.18 Protocol independent (HTTP, WS)

It is possible to use the evaluated language and associated mechanisms with different communication protocols (Web Services, HTTP, etc.) and to define separately protocol-specific aspects (e.g. cookies).

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: The technology is independent of the protocol (HTTP, Web Services, RPC, etc.). Different prototypes use different protocols.

- Partial implementation (◐) is defined as: The technology has protocol specific aspects but can be reused with other protocols. Or, the language is independent from protocols but was only integrated with one protocol.

- No implementation (○) is defined as: The language is targeting one specific protocol and cannot be used in another setting.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: The language could be used as it is with other protocols.

- Possible partial support (◐) is defined as: Support for different protocols could be implemented with existing language and mechanisms.

- Cannot be supported (○) is defined as: The language is targeting one specific protocol and cannot be used in another setting. Extension would require major work.

APPEL + P3P (A+P):

- The implementation is evaluated as ○. Protocol specific aspects are part of the language, e.g. cookies.

- Possible extensions are evaluated as ●. P3P can be used with other protocols e.g. Web Services [W3C03].

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ●. PPL is already used with HTTP (see [Pri10a]) and Web Services (see Sect. 8).

- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ◐. Not implemented (out of scope of the core language and query evaluation). The language XML serialization is independent of the protocol.

- Possible extensions are evaluated as ●. S4P is independent of the protocol. Using it with Web Services or HTTP is straightforward.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ●. The protocol used to configure the access control policy does not have any impact.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ◐. Out of scope. Only HTTP has been used in prototypes.

- Possible extensions are evaluated as ●. PRIME-DHP is independent of the protocol. Using it with Web Services and HTTP is straightforward.

A.19 Policy for Implicit PII (e.g. IP address)

It is possible to specify how PII Consumer handles personal data that are implicitly collected (e.g. IP address).

Classification criteria on exiting features. Here is how the existing implementation of a given technology is evaluated:

- Complete implementation (●) is defined as: The language can handle personal data implicitly and explicitly collected in a unified way.

- Partial implementation (◐) is defined as: The language handles both implicit and explicit data but in different ways.

- No implementation (○) is defined as: The language only targets explicit data collection.

Classification criteria on possible extensions. Here is how the possible extensions of a given technology are evaluated based on specifications and prototypes:

- Possible complete support (●) is defined as: The language could be used to target implicit data collection as it is.
- Possible partial support (◐) is defined as: The language could be combined with other mechanisms to address implicit data collection.
- Cannot be supported (○) is defined as: Implicit data collection is out of scope and would require major work.

APPEL + P3P (A+P):

- The implementation is evaluated as ●. P3P supports implicit data collection (e.g. IP address).
- Possible extensions are evaluated as ●. This is already supported and does not require additional extension.

PrimeLife Policy Language (PPL):

- The implementation is evaluated as ◐. PPL offers a specific language (subset of P3P) to target implicit data collection.
- Possible extensions are evaluated as ●. PPL could be used for implicit data collection as well.

SecPAL for Privacy (S4P):

- The implementation is evaluated as ○. Not implemented (out of scope of the core language and query evaluation).
- Possible extensions are evaluated as ●. S4P mainly targets explicit data collection but could be used for implicit data collection.

User-Specified Access Control (AC):

- Possible extensions are evaluated as ○. Out of scope. All available data are explicitly provided in this setting.

PRIME Data Handling Policy (PDH):

- The implementation is evaluated as ○. This is not implemented.
- Possible extensions are evaluated as ●. PRIME-DHP mainly targets explicit data collection but could be used for implicit data collection.

Chapter B

Demonstrator

B.1 Policy Mismatching Schema

Listing B.1: Schema for a Policy Mismatch

```
ï»¿<?xml version="1.0" encoding="utf-8"?>
<xs:schema targetNamespace="http://www.primelife.eu/ecv"
    elementFormDefault="qualified"
    xmlns="http://www.primelife.eu/ecv"
    xmlns:ecvmm="http://www.primelife.eu/ecv"
    xmlns:mstns="http://tempuri.org/XMLSchema.xsd"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:obmm="http://www.primelife.eu/ppl/obligation/mismatch"
>
  <xs:import namespace="http://www.primelife.eu/ppl/obligation/
      mismatch" schemaLocation="./PrimeLifeObligationMismatch.xsd" />

  <xs:element name="EcvMismatchData" type="ecvmm:EcvMismatchData"/>
  <xs:complexType name="EcvMismatchData">
    <xs:sequence>
      <xs:element ref="ecvmm:ContextData" minOccurs="1" maxOccurs="1"
          />
      <xs:element ref="obmm:ObligationsSet" minOccurs="1" maxOccurs="
          1" />
    </xs:sequence>
  </xs:complexType>


  <xs:element name="ContextData" type="ecvmm:ContextData"/>
  <xs:complexType name="ContextData">
    <xs:sequence>
      <xs:element name="JobId" type="xs:string" minOccurs="1"
          maxOccurs="1" />
```

```
25        <xs:element name="ProfileId" type="xs:string" minOccurs="1"
              maxOccurs="1" />
26        <xs:element name="ProfileName" type="xs:string" minOccurs="1"
              maxOccurs="1" />
27        <xs:element name="TimeOfOffer" type="xs:dateTime" minOccurs="1"
               maxOccurs="1" />
28        <xs:element name="TimeOfMismatchOccurance" type="xs:dateTime"
              minOccurs="1" maxOccurs="1" />
29        <xs:element name="DCRole" type="xs:string" minOccurs="1"
              maxOccurs="1" />
30      </xs:sequence>
31    </xs:complexType>
32
33
34  </xs:schema>
```

B.2 Example Messages of eCV Demonstrator

Listing B.2: Example of a JobApplication message.

```
1  <?xml version="1.0"?>
2  <JobApplicationSetPPL xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
3    <JobApplication xmlns="http://www.primelife.eu/ecv">
4      <JobApplicationPPL>
5        <ProfileId>c1cb18bb-5676-4465-8200-c637deaf18b4</ProfileId>
6        <JobId>728538059</JobId>
7        <Profile>
8          <ProfileName>Alice Wonderworks - Academic</ProfileName>
9          <Skill>
10           <Skill>
11             <Description>C#</Description>
12             <Experience>3</Experience>
13           </Skill>
14           <Skill>
15             <Description>Java</Description>
16             <Experience>4</Experience>
17           </Skill>
18           <Skill>
19             <Description>Perl</Description>
20             <Experience>1</Experience>
21           </Skill>
22         </Skill>
23       </Profile>
24       <ClaimSet>
25         <Claim>
26           <ClaimName>University Degree</ClaimName>
27           <Subject>
28             <FirstName>Alice</FirstName>
29             <LastName>Worderworks</LastName>
```

```
30              <Nationality>US</Nationality>
31              <DateOfBirth>0001-01-01T00:00:00</DateOfBirth>
32            </Subject>
33            <Issuer>
34              <Institution>University of Floria Keys</Institution>
35              <Address>Holiday Blvd., Florida Keys, Florida, US</
                    Address>
36              <Contact>Vincent Vacation, Managing Director</Contact>
37            </Issuer>
38          </Claim>
39        </ClaimSet>
40        <Policy>
41          <StickyPolicy xmlns="http://www.primelife.eu/ppl">
42            <AuthorizationsSet>
43              <AuthzDownstreamUsage allowed="true">
44                <Policy>
45                  <Target xmlns="urn:oasis:names:tc:xacml:2.0
                        :policy:schema:os">
46                    <Subjects>
47                      <Subject>
48                        <SubjectMatch>
49                          <AttributeValue DataType="http://www.w3.org
                                /2001/XMLSchema#anyURI">http://www.
                                primelife.eu/ecv/participants/roles/
                                domainexpert</AttributeValue>
50                          <SubjectAttributeDesignator AttributeId="
                                http://www.primelife.eu/ppl/
                                DataControllerRole" DataType="http://www.
                                w3.org/2001/XMLSchema#anyURI" />
51                        </SubjectMatch>
52                      </Subject>
53                    </Subjects>
54                  </Target>
55                  <DataHandlingPreferences>
56                    <AuthorizationsSet />
57                    <ObligationsSet xmlns="http://www.primelife.eu/ppl/
                          obligation">
58                      <Obligation>
59                        <TriggersSet>
60                          <TriggerAtTime>
61                            <Start>
62                              <StartNow />
63                            </Start>
64                            <MaxDelay>
65                              <Duration>PT30S</Duration>
66                            </MaxDelay>
67                          </TriggerAtTime>
68                        </TriggersSet>
69                        <ActionDeletePersonalData />
70                      </Obligation>
71                    </ObligationsSet>
72                  </DataHandlingPreferences>
```

```
73                  <ProvisionalActions />
74                </Policy>
75              </AuthzDownstreamUsage>
76            </AuthorizationsSet>
77            <ObligationsSet xmlns="http://www.primelife.eu/ppl/
                  obligation">
78              <Obligation>
79                <TriggersSet>
80                  <TriggerAtTime>
81                    <Start>
82                      <StartNow />
83                    </Start>
84                    <MaxDelay>
85                      <Duration>PT1M</Duration>
86                    </MaxDelay>
87                  </TriggerAtTime>
88                </TriggersSet>
89                <ActionDeletePersonalData />
90              </Obligation>
91            </ObligationsSet>
92          </StickyPolicy>
93          <ProvisionalActions xmlns="http://www.primelife.eu/ppl" />
94        </Policy>
95      </JobApplicationPPL>
96    </JobApplication>
97  </JobApplicationSetPPL>
```

Listing B.3: Example of user preferences; the user has configured only obligations no access control settings.

```
1  <?xml version="1.0"?>
2  <PolicySet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://www.
       primelife.eu/ppl">
3    <StickyPolicy>
4      <AuthorizationsSet />
5      <ObligationsSet xmlns="http://www.primelife.eu/ppl/obligation">
6        <Obligation>
7          <TriggersSet>
8            <TriggerAtTime>
9              <Start>
10               <StartNow />
11             </Start>
12             <MaxDelay>
13               <Duration>PT1M</Duration>
14             </MaxDelay>
15           </TriggerAtTime>
16         </TriggersSet>
17         <ActionDeletePersonalData />
18       </Obligation>
19     </ObligationsSet>
```

```
20    </StickyPolicy>
21    <ProvisionalActions />
22  </PolicySet>
```

Listing B.4: Example of user preferences allowing for downstream data sharing; the user has configured only obligations no access control settings.

```
1  <?xml version="1.0"?>
2  <PolicySet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://www.
       primelife.eu/ppl">
3    <DataHandlingPreferences>
4    <AuthorizationsSet>
5      <AuthzDownstreamUsage allowed="true">
6        <Policy>
7        <Target xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os">
8          <Subjects>
9          <Subject>
10           <SubjectMatch>
11           <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#
                 anyURI">http://www.primelife.eu/ecv/participants/roles/
                 domainexpert</AttributeValue>
12           <SubjectAttributeDesignator AttributeId="http://www.
                 primelife.eu/ppl/DataControllerRole" DataType="http://
                 www.w3.org/2001/XMLSchema#anyURI" />
13           </SubjectMatch>
14         </Subject>
15         </Subjects>
16       </Target>
17       <DataHandlingPreferences>
18         <AuthorizationsSet />
19         <ObligationsSet xmlns="http://www.primelife.eu/ppl/obligation
               ">
20         <Obligation>
21           <TriggersSet>
22           <TriggerAtTime>
23             <Start>
24             <StartNow />
25             </Start>
26             <MaxDelay>
27             <Duration>PT30S</Duration>
28             </MaxDelay>
29           </TriggerAtTime>
30           </TriggersSet>
31           <ActionDeletePersonalData />
32         </Obligation>
33         </ObligationsSet>
34       </DataHandlingPreferences>
35       <ProvisionalActions />
36       </Policy>
37     </AuthzDownstreamUsage>
```

```
38    </AuthorizationsSet>
39      <ObligationsSet xmlns="http://www.primelife.eu/ppl/obligation">
40        <Obligation>
41          <TriggersSet>
42            <TriggerAtTime>
43              <Start>
44                <StartNow />
45              </Start>
46              <MaxDelay>
47                <Duration>PT30S</Duration>
48              </MaxDelay>
49            </TriggerAtTime>
50          </TriggersSet>
51          <ActionDeletePersonalData />
52        </Obligation>
53      </ObligationsSet>
54    </DataHandlingPreferences>
55    <ProvisionalActions />
56  </PolicySet>
```

B.3 Screenshots of eCV Demonstrator

In this section, we will give a brief overview on the user interface of the eCV demo by describing screenshots that show the processing of an overall privacy policy compliant case.

Figure 17 shows a job offer being created by the employer tool and that will be send to headhunter tool next.

Looking at the screenshot in figure 18, it will show the headhunter tool right before a job offer is being received.

After the headhunter has received a job offer, it will send it to the eCV portal. In the following three screenshots, we would like to show an emulator for SAP's eCV portal we used for demonstrating so far. Being a small and simple placeholder, the eCV portal emulator is not doing the actual matching but let the user edit its result directly. That said, figure 19 shows the eCV portal tool emulator in the application view, where PII data like name, skills, etc. can be edited. The emulator also allows to edit the applications sticky policy. Figure 20 shows the sticky policy without a permission for the data controller to use the data for downstream, whereas figure 21 adds this permission.

After the eCV portal tool has sent its applications to the headhunter tool, the headhunter tools state will look like as in figure 22. Being in verbose mode, the headhunter tool is also showing the received application's sticky policy in its logging area.

To proceed to the next stage successfully, the application's sticky policy needs to permit downstream usage and its privacy preferences for that downstream usage should be compliant to the privacy policy of the domain expert. In that case the headhunter tool will successfully send the applications to the domain expert and will get the response quickly. Figure 23 show the state of the headhunter tool after it has received the assessed applications from the domain expert.

The user interface of the domain expert can be seen at figure 24

Back in the eCV scenario walk-trough, the headhunter tool will send the positive

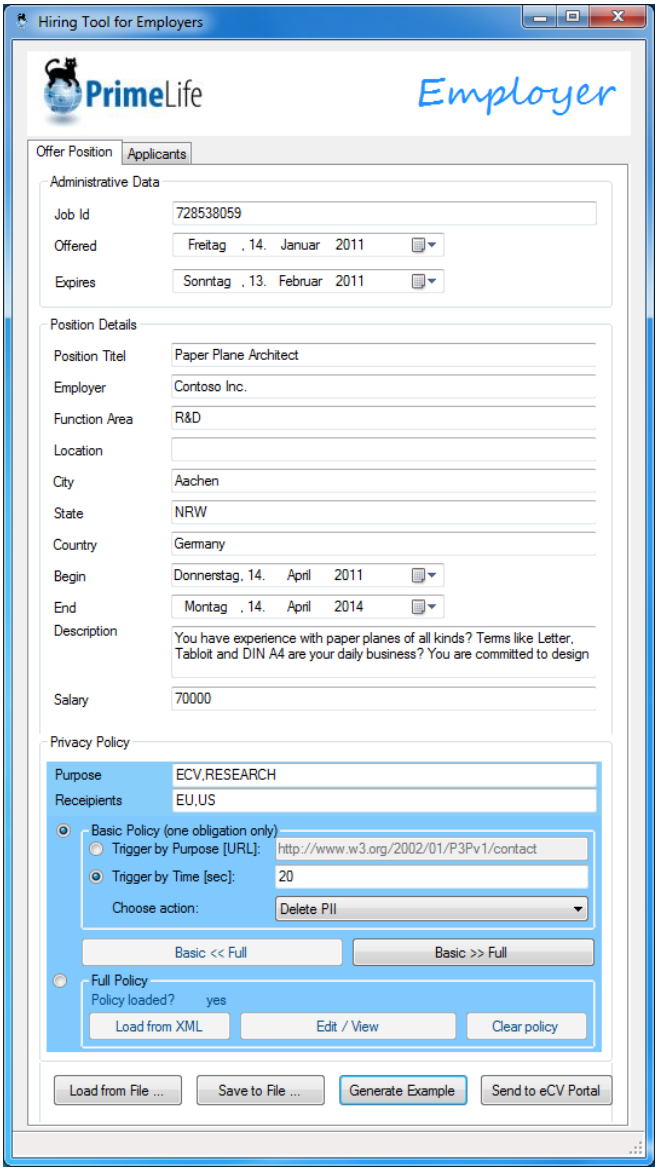


Figure 17: Employer tool showing a job offer

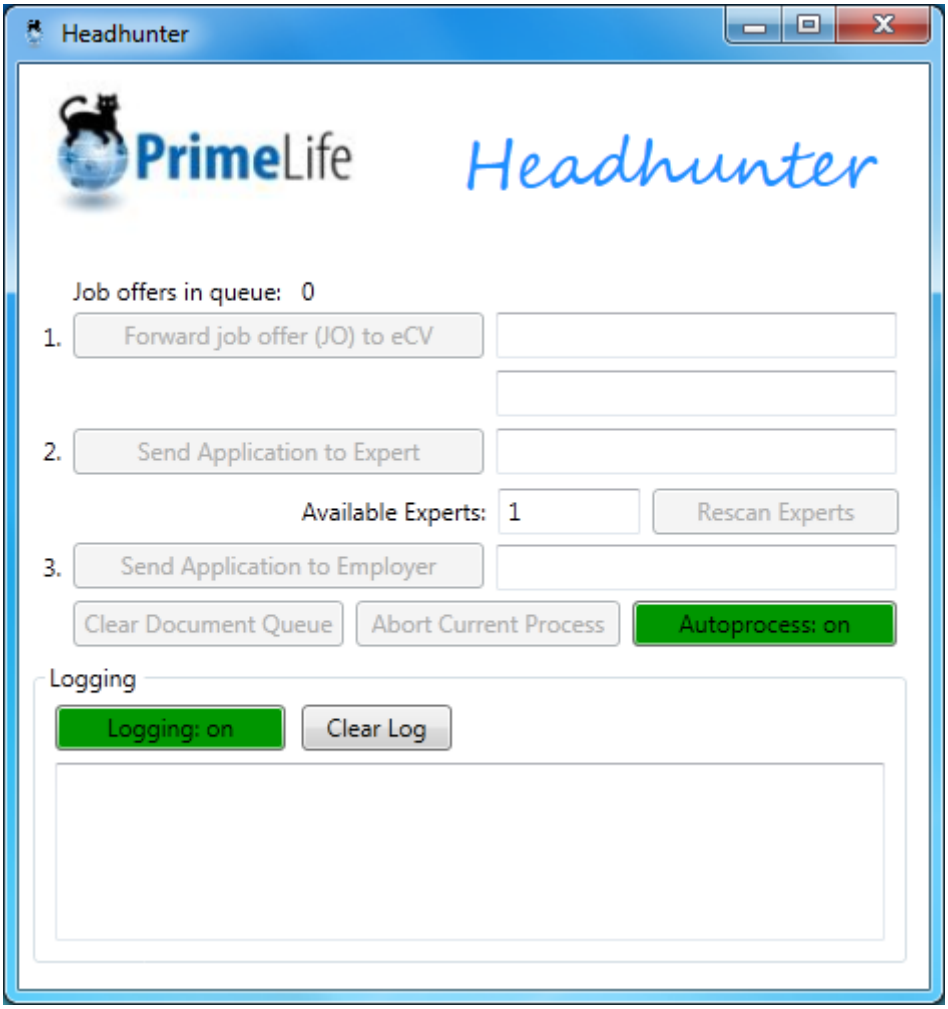


Figure 18: User interface of the headhunter tool

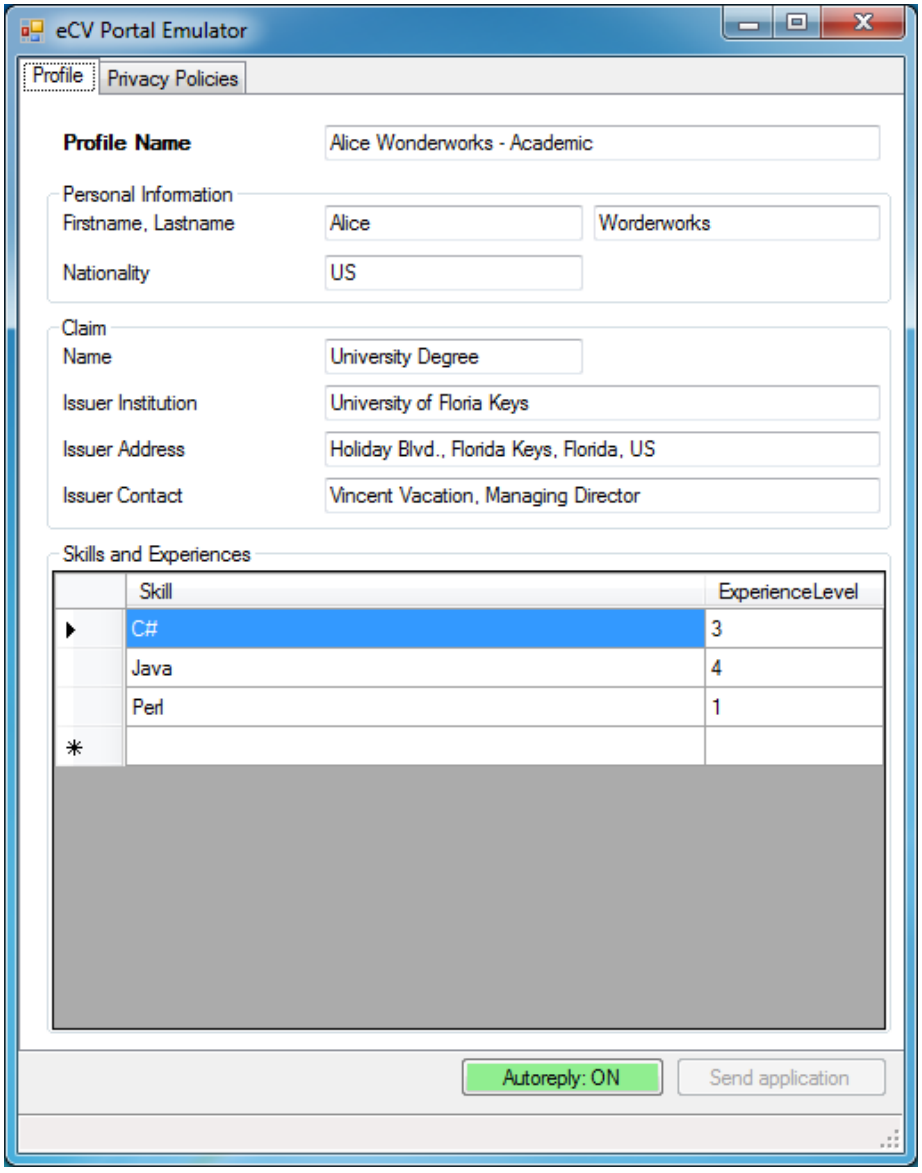


Figure 19: Emulator for SAP's eCV portal tool

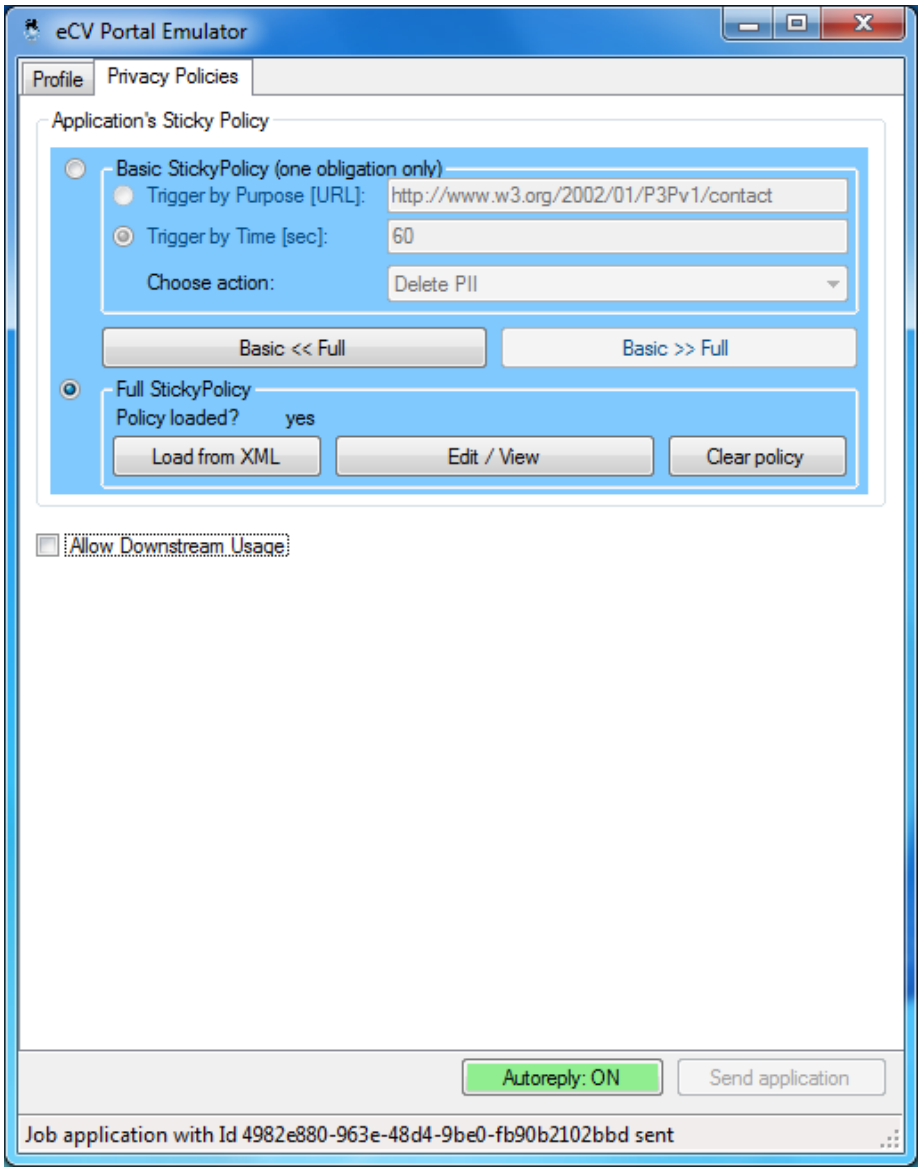


Figure 20: eCV portal tool emulator showing user's privacy preferences as sticky policy. The user does not allow downstream usage control.

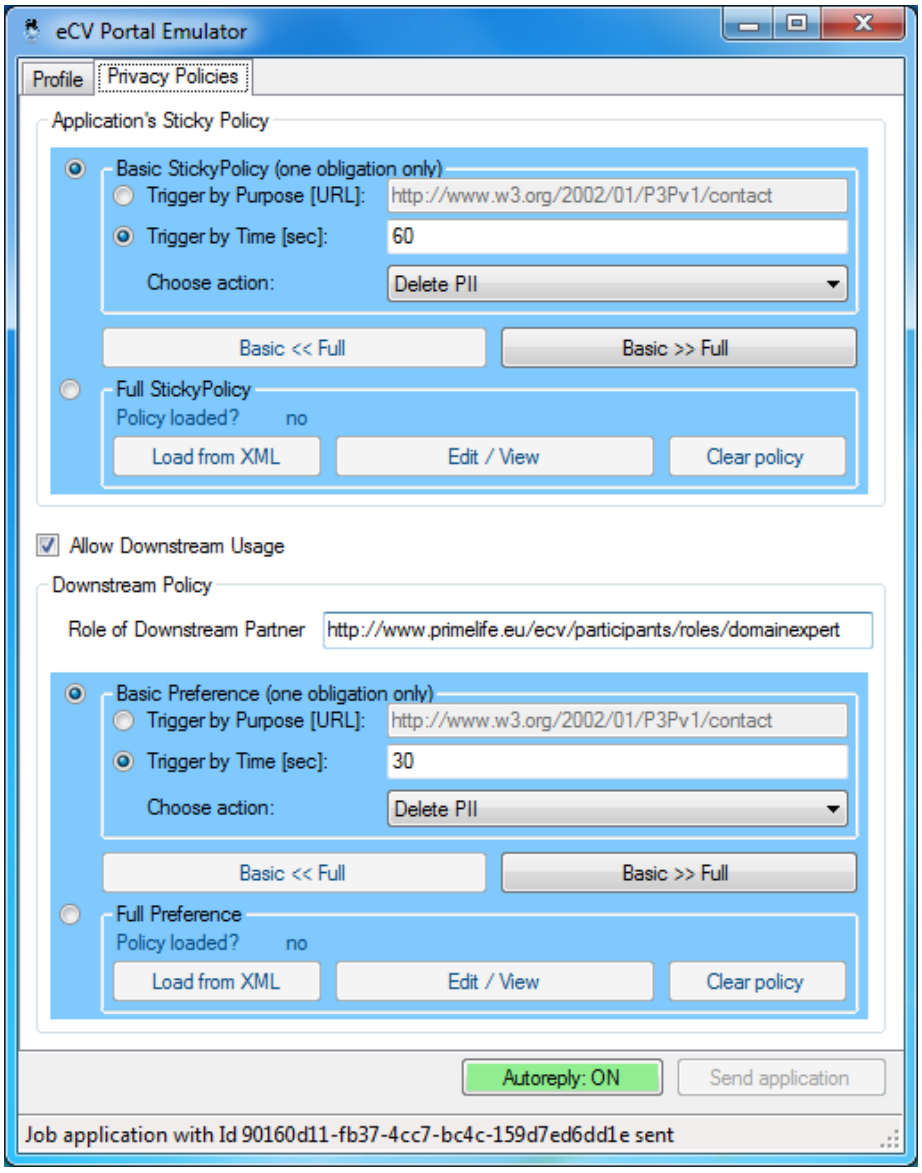


Figure 21: eCV portal tool emulator showing user's privacy preferences as sticky policy. The user allows downstream usage control.

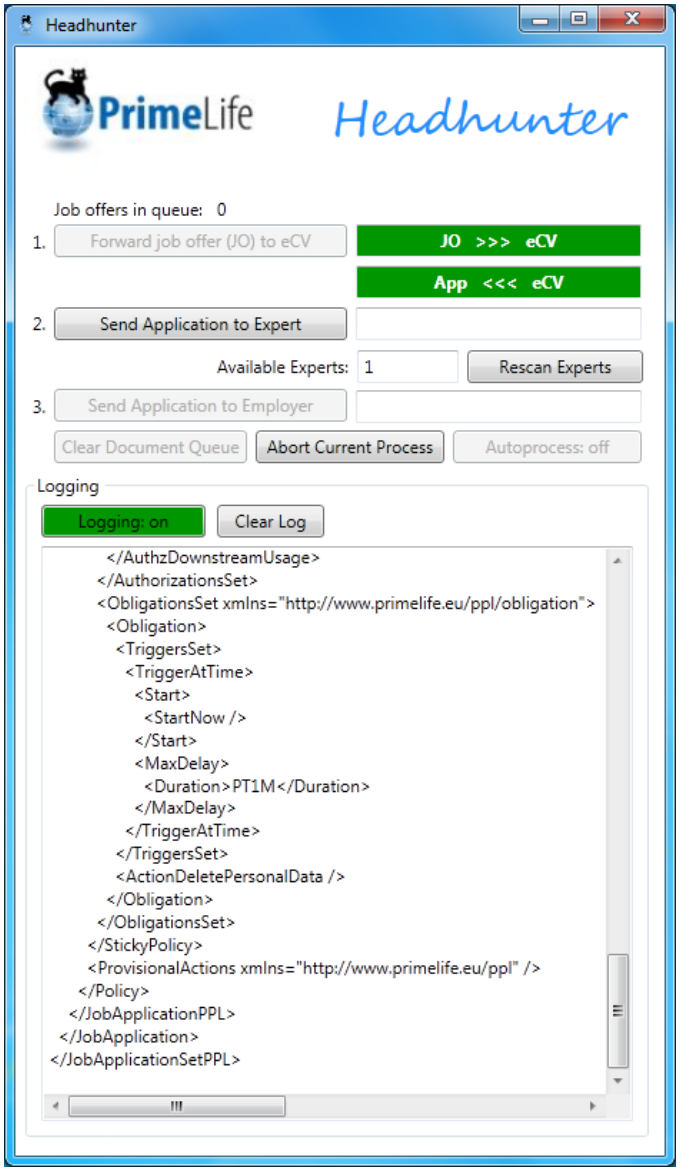


Figure 22: Headhunter tool UI showing the communicated sticky policy

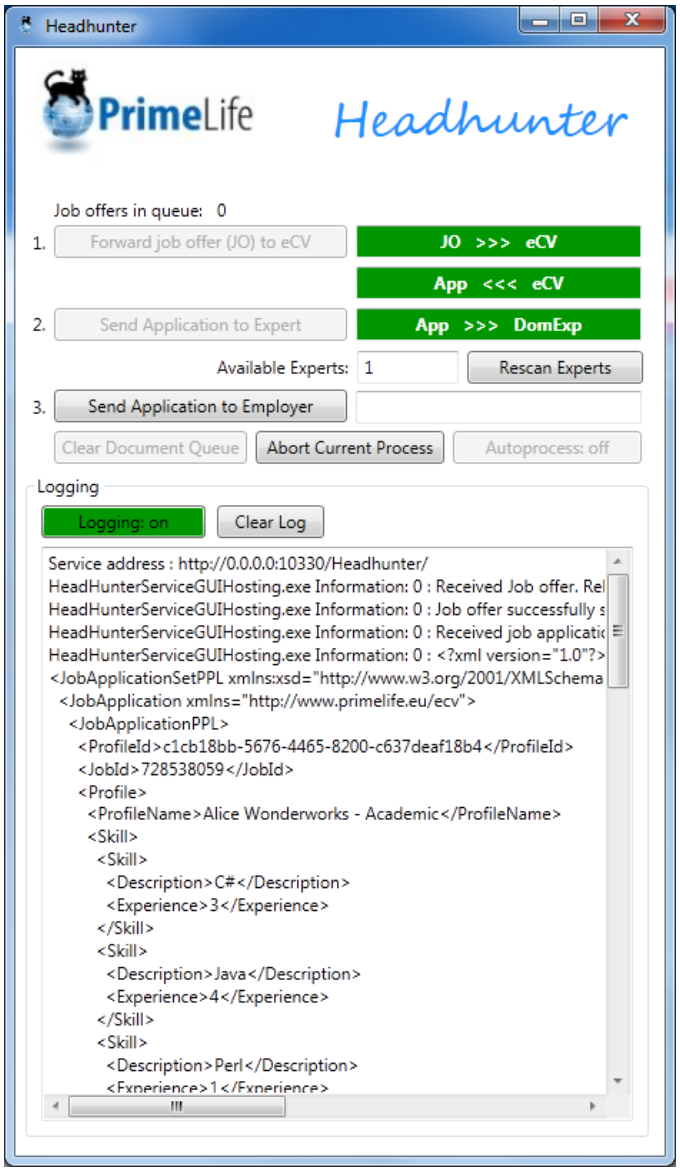


Figure 23: Headhunter tool after having received assessed applications from the domain expert

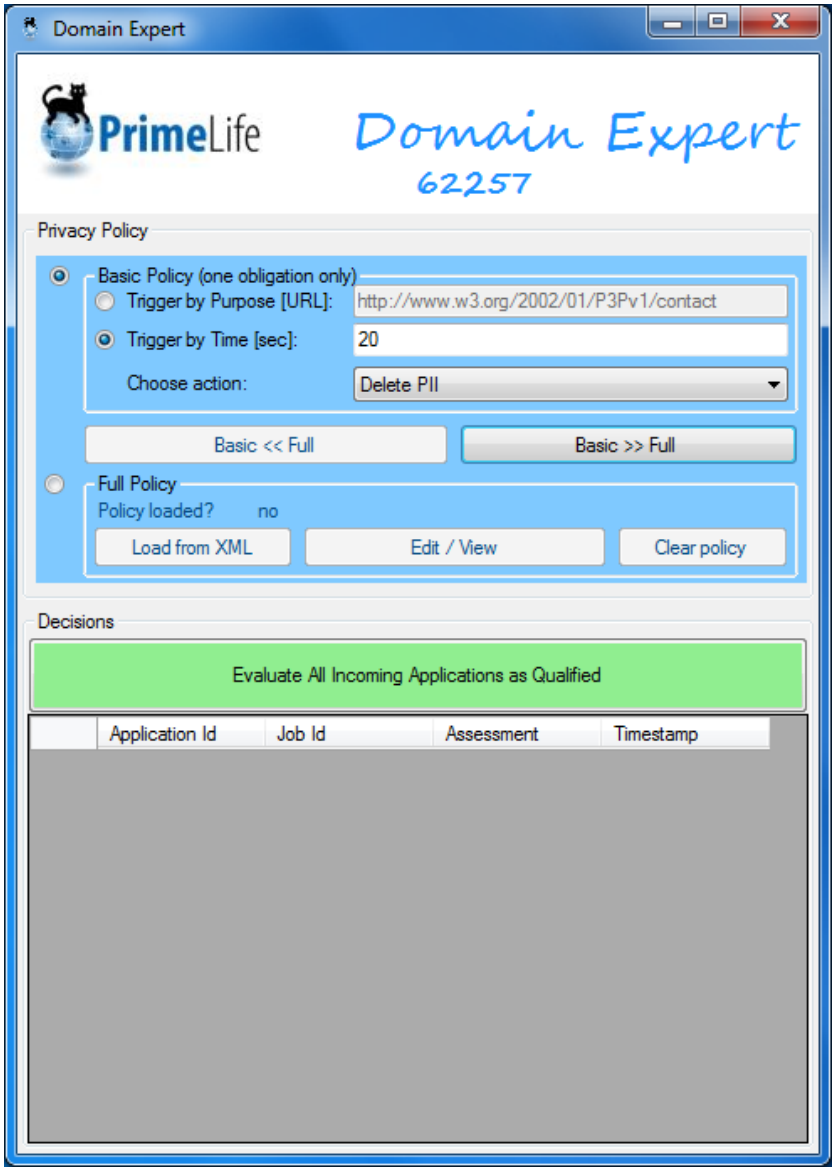


Figure 24: Domain expert tool's user interface

assessed applications the the employer. Figure 25 will show the employer tool at the very last step of the scenario, having received and displaying job applications.

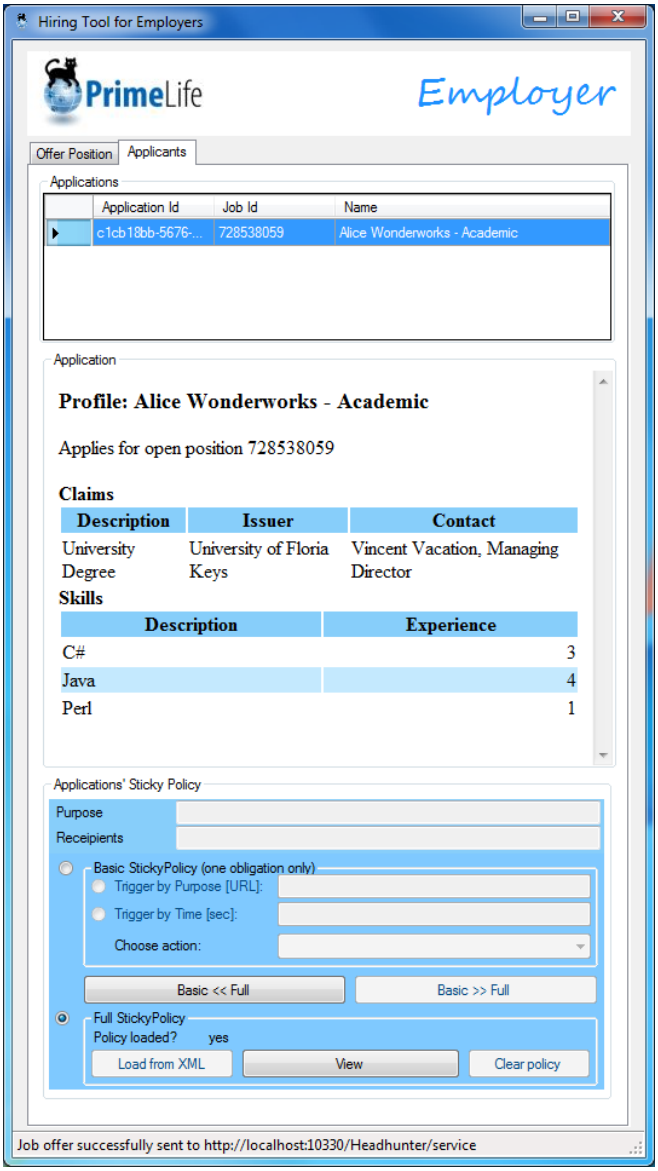


Figure 25: Employer tool showing a received job application

Please note that we use a hand-crafted editor to create, modify or view PPL policies. Figure 26 shows that component explicitly. It also can be found as part of the employer tool, the domain expert tool and the eCV portal tool emulator, and is discussed in more detail in section 9.3.1.

On closing this overview of user interface, we also would like to show additional screens that can be seen on privacy policy mismatching cases.

In case the applicant, which is represented by the eCV portal tool emulator, does

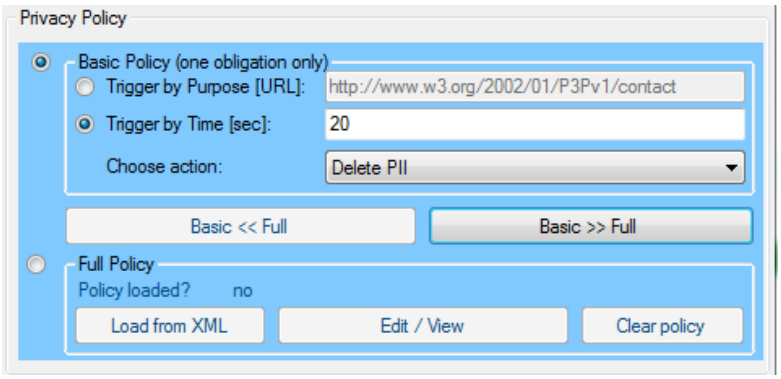


Figure 26: PPL Editor user interface.

not allow downstream sharing its data, the headhunter will not be able to contact any domain expert and will stop proceeding the particular application any further. This is illustrated at figure 27.

Besides, it could also be the case that although downstream sharing is allowed the preference of the applicant does not match with any domain expert's policy. In that case the applicants SMD will be triggered while the headhunter shows the screen of figure 28.

For further details on the eCV scenario and its different show cases, please refere to chapter 7.

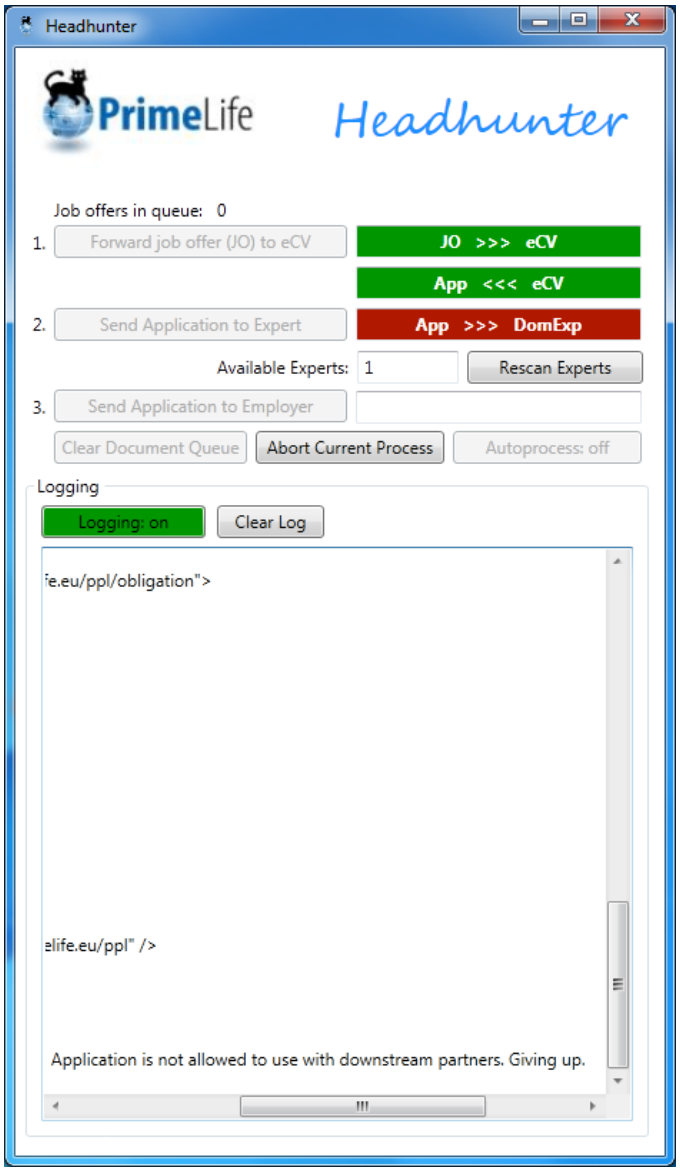


Figure 27: Headhunter tool's UI signaling that applicant does not allow downstream data sharing

Figure 28: Headhunter tool that needs to contact the applicant to resolve policy matching conflict

Bibliography

[AC] AT&T Corp and Carnegie Mellon University. Privacy bird.

[ACDCdVS08] C. A. Ardagna, M. Cremonini, S. De Capitani di Vimercati, and P. Samarati. A privacy-aware access control system. *J. Comput. Secur.*, 16(4):369–397, 2008.

[ADF+10] C.A. Ardagna, S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, and P. Samarati. Minimizing disclosure of private information in credential-based interactions: A graph-based approach. In *Proc. of the 2nd IEEE International Conference on Information Privacy, Security, Risk and Trust (PASSAT 2010)*, Minneapolis, Minnesota, USA, August 2010.

[AHK+03] Paul Ashley, Satoshi Hada, Günter Karjoth, Calvin Powers, and Matthias Schunter. Enterprise privacy authorization language (EPAL 1.2), 2003.

[BFG09] Moritz Y. Becker, Cedric Fournet, and Andrew D. Gordon. SecPAL: Design and semantics of a decentralized authorization language. *Journal of Computer Security*, 2009.

[BGS+07] Johann Bizer, Rüdiger Grimm, Steffen Staab, Sebastian Meissner, Daniel Pähler, Christoph Rigelstein, Martin Rost, Jan Schallaböck, and Felix Schwagereit. Chancen und risiken von service-orientierten architekturen in virtuellen architekturen. Project Report, 2007.

[BMB10] Moritz Y. Becker, Alexander Malkis, and Laurent Bussard. A practical generic privacy language. In *Sixth International Conference on Information Systems Security (ICISS 2010)*. Springer, December 2010.

[BMD09] Moritz Y. Becker, Jason F. Mackay, and Blair Dillaway. Abductive authorization credential gathering. In *IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY)*, July 2009.

[BNP09] Laurent Bussard, Anna Nano, and Ulrich Pinsdorf. Delegation of access rights in multi-domain service compositions. *Identity in the Information Society*, 2(2):137–154, December 2009.

[BNP10] Laurent Bussard, Gregory Neven, and Franz-Stefan Preiss. Downstream usage control. In *IEEE Policy 2010*, July 2010.

[BNS10] Laurent Bussard, Gregory Neven, and Jan Schallaböck. Data handling: Dependencies between authorizations and obligations. W3C Workshop on Privacy and data usage control, October 2010.

[CDK05] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems. Concepts and Design.* Addison Wesley, 4 edition, 2005.

[CK05] Luis Felipe Cabrera and Chris Kurt. *Web Services Arcitecture and Its Specifications: Essentials for Understanding WS-*.* Microsoft Press, 2005.

[Con02] ContentGuard. XrML 2.0 Technical Overview. `http://www.xrml.org/reference/XrMLTechnicalOverviewV1.pdf`, 2002.

[Fie00] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures.* PhD thesis, 2000. AAI9980887.

[HL10] Eran Hammer-Lahav. RFC 5849: The OAuth 1.0 Protocol, 2010.

[JSS08] Ethan K. Jackson, Wolfram Schulte, and Janos Sztipanovits. The power of rich syntax for model-based development. Technical report, 2008.

[KA10] Lalana Kagal and Hal Abelson. Access control is an inadequate framework for privacy protection. W3C Workshop on Privacy for Advanced Web APIs, July 2010.

[Kan] Kantara Initiative. User managed initiative.

[KWWD08] Christina Köffel, Erik Westlund, Peter Wolkerstorfer, and Sandra Dittenberger. The PrimeLife list of personas. Internal report, PrimeLife Consortium, 2008.

[Mic09] Microsoft. Rights Management Services, 2009.

[MS09] Sebastian Meissner and Jan Schallaböck. Requirements for privacy-enhancing service-oriented architectures. Public project deliverable H6.3.1, PrimeLife Consortium, November 2009.

[ODR02] ODRL. Open Digital Rights Language (ODRL), version 1.1, 2002.

[Pri09a] PrimeLife Consortium. Draft 2nd design for policy languages and protocols (heartbeat: H5.3.2). Technical report, July 2009.

[Pri09b] PrimeLife Consortium. Identity Management Infrastructure Protocols for Privacy-enabled SOA (D6.1.1). Technical report, September 2009.

[Pri10a] PrimeLife Consortium. Second Release of the Policy Engine (D5.3.2). Technical report, September 2010.

[Pri10b] PrimeLife Consortium. UI Prototypes: Policy Administration and Presentation - Version 2 (D4.3.2). Technical report, June 2010.

[Pri11] PrimeLife Consortium. Advancement and Integration of Concepts for Secure and Dynamic Creation of Mobile Services (D6.3.1). Technical report, January 2011.

[PSSW08] A. Pretschner, F. Schütz, C. Schaefer, and T. Walter. Policy evolution in distributed usage control. In *4th Intl. Workshop on Security and Trust Management*. Elsevier, June 2008.

[Rag10] Dave Raggett. Machine interpretable privacy policies – a fresh take on p3p. W3C Workshop on Privacy and data usage control, October 2010.

[Rah10] Sharif Tanvir Rahman. Analyzing causes of privacy mismatches in service oriented architecture. Master's thesis, RWTH, 2010.

[Ris10] Erik Rissanen. OASIS eXtensible Access Control Markup Language (XACML) Version 3.0. OASIS committee specification 01, OASIS, August 2010.

[SUN] SUN Microsystems Inc. Model-view-controller.

[W3C02] W3C. A P3P preference exchange language 1.0 (APPEL1.0), 2002.

[W3C03] W3C. P3P: Beyond HTTP, April 2003.

[W3C06] W3C. The platform for privacy preferences 1.1 (P3P1.1) specification, 2006.

[W3C07] W3C. The Simple Object Access Protocoll 1.2 (SOAP1.2) specification, 2007.

[W3C10] W3C. Contacts API, December 2010.

[Wan04] Xin Wang. MPEG-21 Rights Expression Language: Enabling Interoperable Digital Rights Management. *IEEE MultiMedia*, 11(4):84–87, 2004.

[YLA04] Ting Yu, Ninghui Li, and Annie I. Antón. A formal semantics for P3P. In *Proceedings of the 2004 workshop on Secure web service*, SWS '04. ACM, 2004.