Abstract Privacy Policy Framework: Addressing Privacy Problems in SOA

Laurent Bussard and Ulrich Pinsdorf

European Microsoft Innovation Center, Aachen, Germany
{LBussard,Ulrich.Pinsdorf}@microsoft.com

Abstract. This paper argues that privacy policies in SOA needs a lifecycle model. We formalize the lifecycle of personal data and associated privacy policies in Service Oriented Architectures (SOA), thus generalizing privacy-friendly data handling in cross-domain service compositions. First, we summarize our learning in two research projects (PrimeLife and SecPAL for Privacy) by proposing generic patterns to enable privacy policies in SOA. Second, we map existing privacy policy technologies and ongoing research work to the proposed abstraction. This highlights advantages and shortcomings of existing privacy policy technologies when applied to SOA.

1 Motivation

Service Oriented Architectures (SOA) aim 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 [10]. A common principle is to dynamically bind services hosted in different security domains and by different legal entities. We refer to this as "cross-domain service composition" [7]. In many cases, a distributed system might involve the processing of personal data and thus requires privacy-enhancing technologies [16].

Service composition enables new features but increases risk for the privacy of their users: First, data subjects may no longer be aware of what data relating to them are handled by what entity for what purpose. Data subjects might even not be aware of the involvement of further legal entities at all. Second, the use of standardized data formats and interfaces makes it easy for involved parties to link different sets of personal data and generate profiles on data subjects. Mechanisms to specify data handling are required.

Many individual technologies have been developed to address data handling in distributed systems [24,23,8,6,1,17]. Yet, they typically focus only on specific technical aspects or scenarios. In this paper we take a step back and look at the whole lifecycle of private data and its associated privacy policies in a distributed system, thus generalizing privacy-friendly data handling in cross-domain service compositions. We distill lessons learnt from working on privacy-enhancing technologies for distributed systems in the projects *PrimeLife* [18] and *S4P* [6]. Our

J. Camenisch and D. Kesdogan (Eds.): iNetSec 2011, LNCS 7039, pp. 104–118, 2012.

result is a *generic framework* that defines the general processing steps to achieve privacy compliance and proper data handling in SOA. We explicitly address downstream data sharing [8] by repeated application of the same principle, i.e. the abstract framework can be "chained". The framework deliberately abstracts from concrete technologies and policy languages. However, we compare existing technologies with the proposed abstract framework in Sect. 5. This validates the feasibility of our approach and makes it possible to compare advantages and shortcomings of existing technical solutions.

The remainder of this paper is structured as follows. Section 2 gives a general overview of involved parties and high-level protocol steps. The subsequent two sections refine the protocol steps for the provider of personal data (Sect. 3) and the consumer of personal data (Sect. 4). Section 5 analyses prior art; the abstract framework is instantiated with existing standards and ongoing research in order to compare their use in SOA. Finally, we conclude with the lessons learnt in Sect. 6.

2 Abstract Privacy Framework

In distributed systems, such as Service Oriented Architectures (SOA), involved parties (users and services) can provide personal data and/or consume personal data. For the sake of readability, we define two roles *PII Provider* and *PII Consumer*. Note that we use PII (personally identifiable information) as a short notation for the broader concept of personal data. A party can have both roles: a service can act as PII Consumer when collecting data and as PII Provider when forwarding collected data; a user can act as PII Provider of her own personal data and as PII Consumer of third parties' data. Figure 1 presents major privacy challenges of SOA: multi-hop data sharing, aggregation of data, privacy-aware discovery and late binding, distributed enforcement, and distributed audit.

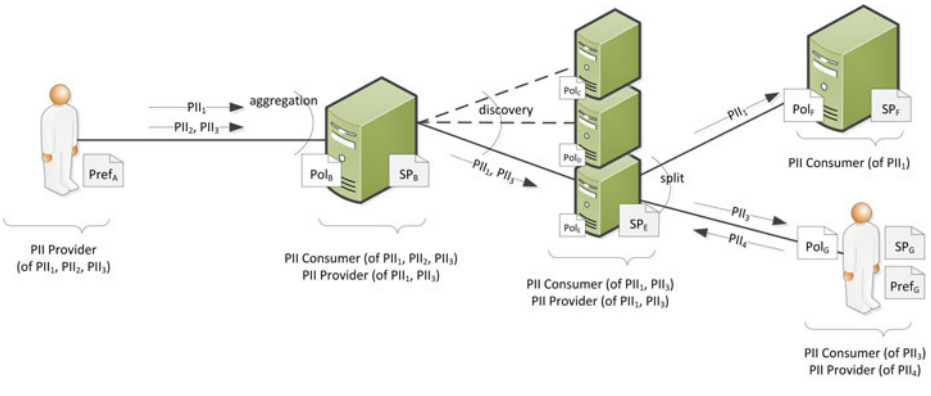

Fig. 1. Privacy Policies in Service Oriented Architecture

2.1 Outline

Figure 2 shows the Abstract Privacy Framework in a visual representation. The figure contains 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. Both top-level components contain a best-practice workflow. The combined workflow is an ideal, technology-agnostic, and 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 manifestation. The remainder of this section describes all top-level components from Fig. 2. Sections 3 and 4 go into more details for the PII Provider and resp. the PII Consumer.

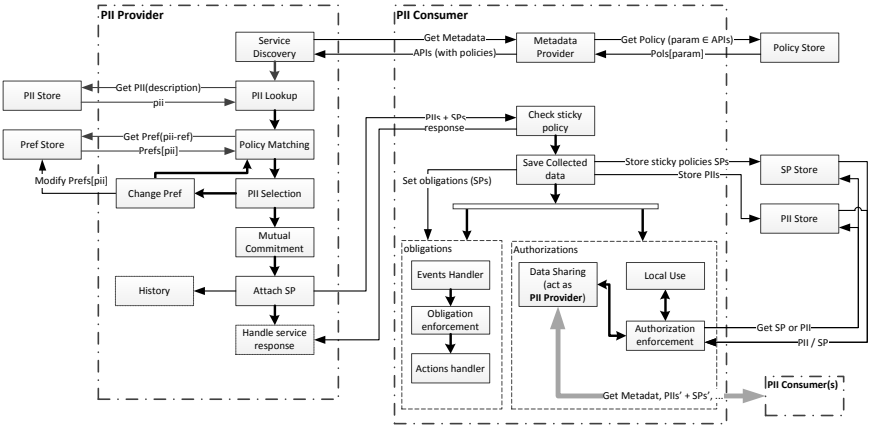

Fig. 2. Overview of abstract privacy policy framework

PII Provider: *PII Provider P* shares 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.

PII Consumer: *PII Consumer C* collects personal data provided by PII Providers. In most scenarios, a service (or data controller) is acting as PII Consumer. PII consumers are liable for making sure that collected data are properly handled. 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.

PII Store: *PII Store* PII_P is the database containing each piece of 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).

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 willing 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. $Prefs[pii]$ 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 any time, a PII Provider can create or modify her preferences.

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 a business process. They may depend on the PII Consumer, e.g. when this one is authenticated. Moreover, policies are generally locally defined but often mention external policies (e.g. the policies of downstream data controllers).

The main differences between Policy Store and Preference Store are: 1) Policies and preferences are expressed in different languages (which can be very similar in some technologies). 2) Policies are generally associated to parameters (e.g. any e-mail address collected with this interface) while preferences are associated to types (e.g. all my e-mail addresses) or instances (e.g. alice@contoso.com).

Sticky Policy Store: *Sticky Policy Store* SP_C is very similar to *Preferences Store* (see section 2.1). 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 $\forall pii_C \in PII_C \cdot Prefs_C[pii_C] \neq \emptyset$.

The main differences between sticky policy store and and preference stores are: 1) The data subject (user) can decide to change how her data must be handled and modify her preferences while the PII consumer cannot modify sticky policies associated with collected data (at least cannot make them more permissive). 2) The sticky policies generally define obligations that must be locally enforced (e.g. delete data within one year).

3 A Closer Look at PII Provider Role

This section provides more insight on technical components necessary to fulfill the "PII Provider" role in Figure 2.

Service Discovery: *Service Discovery* is the process of finding 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 send only one of them.

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 in HTML Forms), generic preferences are used or new preferences are created.

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 $param \leftarrow pii$.

Privacy constraint p_2 is a valid enforcement of p_1 (denoted $p_1 \trianglerighteq p_2$) when enforcing p_2 cannot violate p_1. In [6], this notion is expressed in terms of lower- and upper-bounds behaviors. In [8], $\trianglerighteq$ means "more permissive than", i.e. more rights and/or less obligations. 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.

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. $\forall(param_{sel} \leftarrow pii_{sel}) \cdot Prefs_P[pii_{sel}] \trianglerighteq Pols_C[param_{sel}]$.

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 [2], and history of previously released data. It is not possible to present all aspects to end-users. As a result, "meta-preferences" may be necessary to reduce the number of options.

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. 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).

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. $sp \trianglerighteq Pols_C$. In other words, $\forall(param \leftarrow pii) \cdot Prefs_S[pii] \trianglerighteq sp_{param \leftarrow pii} \trianglerighteq Pols_C[param]$. 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.

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 [14] could bind sticky policies (licenses) to personal data (document).

Domain Specific Languages: Multiple representation of a policy language can be envisioned: XML representation, 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 [21].

4 A Closer Look at PII Consumer Role

This section provides more insight on components necessary to fulfill the "PII Consumer" role in Figure 2.

Provide Metadata: Each PII Consumer C must provide metadata about its service. Data are 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 keeping them one week, and 2) enforce (sticky) policies.

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. $sp_{param \leftarrow pii} \trianglerighteq Pols_C[param]$. This check ensure that a malicious PII Provider cannot provide sticky policies with insufficient rights or with too strict obligations. This check can be part of a mutual commitment protocol.

Authorization Decision: Checking authorization before using collected personal data is 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 collected data locally (within PII Consumer's trust domain) and 2) sharing collected data with a third party.

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.

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'. In other words, most of the components described in Sect. 3 are also part of this component.

The main difference between user P providing her personal data pii to service C and C sharing P's personal data pii with another service C' is that in case of mismatch between P's preferences $Prefs_P[pii]$ and C's policy $Pol_C[param]$, P can modify her preferences while C cannot modify sticky policy $SP_{param \leftarrow pii}$ when there is a mismatch between C and C'.

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.

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 $(Delete(thisPII), (AtTime(t_0, t_0 + 365d)))$ where t_0 is set to the transaction time. The action can be triggered at any time between t_0 and one year after t_0.

Action Handler: 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.

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.

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.

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.

5 Instantiation of the Framework

The *Abstract Privacy Policy Framework* defines an ideal setting to enforce privacy policies in *Service Oriented Architectures* (SOA). In this section, 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.

Several parts of the abstract framework are covered by existing privacy-enhancing technologies. Table 1 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.

5.1 Evaluated Technologies

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) [23] and privacy policies are expressed with *P3P* (the Platform for Privacy Preferences Project) [24]. Enforcement may rely on other technology such as *EPAL* (Enterprise Privacy Authorization Language) [3].

The second instantiation of the abstract framework is based on *PrimeLife Policy Language* (PPL) [18, 19] an extension of XACML [22] 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.

The third instantiation of the abstract framework is based on *SecPAL for Privacy* (S4P) [6] an extension of logic-based authorization language SecPAL [4]. 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 [5].

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 [22]. This approach can be considered as "inadequate" [12] to enforce privacy but is largely used. For instance *OAuth* [11] and *User-Managed Access* (UMA) [13] offer remote management of access control policies.

The fifth instantiation is based on the *PRIME Data Handling Policy* (PDH or PRIME-DHP) [1]. This language is focusing on data handling and access control 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) [9], *Obligation Specification Language* (OSL) [17], MPEG-21 REL [25], or *Open Digital Rights Language* (ODRL) [15]. 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.

5.2 Evaluation Results

Each instantiation of the abstract framework has been evaluated with a set of criteria summarizing key concepts of the framework. Results are summarized in Table 1. Details on the choice of the criteria as well as their evaluation can be found in Chapter 6 and Appendix A of a PrimeLife project report [20].

Here are seven criteria related to PII Provider's Preferences. 1) *Simple Syntax*: Privacy preferences are expressed in a human-readable language. Syntax and semantics are well defined and can be processed by machines. 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. 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. 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.

Table 1. 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 ○.

Abstract features / Concrete technologies	A+P		PPL		S4P		AC	PDH	
PII Provider's Preferences (Sect. 2.1)									
- 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. 2.1)	○	◐	◐	●	○	●	○	○	◐
PII Consumer's Policy (Sect. 2.1)									
- 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. 2.1)	◐	◐	◐	●	○	●	○	●	●
PII Store (Sect. 2.1)	○	◐	◐	◐	○	●	●	○	●
Privacy-Aware Service Discovery (Sect. 3)	○	◐	○	●	○	◐	○	○	○
PII Lookup (Sect. 3)	○	◐	◐	●	○	◐	◐	○	◐
Policy Matching (Sect. 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. 3)	○	○	●	●	○	●	○	◐	◐
Change Preferences (Sect. 3)									
- Can Show Mismatches	○	◐	◐	◐	○	●	○	○	○
- Can Suggest Modifications	○	◐	◐	◐	○	◐	○	○	○
Sticky Policy (Sect. 3)									
- Optional Sticky Policy	●	●	○	●	●	●	○	◐	◐
- Can be Expressive	○	○	●	●	○	●	●	●	●
- Supports Signature or Commitment	○	○	○	●	○	●	◐	○	●
- Can Change Sticky Policy	○	○	◐	●	○	◐	●	○	○
- Can Store and Retrieve Sticky Policy (Sect. 2.1)	○	○	◐	●	○	●	●	◐	●
Attach Sticky Policy (Sect. 3)	○	○	◐	●	○	●	◐	◐	●
High-Level Policy Language (Sect. 3)									
- Same Language for Preferences and Policies	○	○	●	●	◐	◐	○	○	○
- Language Expressiveness	○	◐	●	●	●	●	●	●	●
- Clear Separation of Obligations and Rights	○	○	●	●	◐	◐	●	●	●
Check Sticky Policy (Sect. 4)	○	○	○	●	○	●	●	●	●
Authorization Decision (Sect. 4)									
- Enforces Local Use, e.g. Purpose (Sect. 4)	○	●	●	●	○	●	◐	●	●
- Enforces Access Control when Sharing (Sect. 4)	○	◐	●	●	○	●	●	●	●
- Checks Downstream Data Handling when Sharing	○	○	●	●	○	●	○	○	○
- Attach (New) Sticky Policy when Sharing	○	○	●	●	○	●	○	◐	◐
Composing Sticky Policies (Sect. 4)	○	◐	○	◐	○	◐	◐	○	○
Obligations (Sect. 4)									
- Supports Enforcement of Obligations	○	◐	●	●	●	●	◐	◐	●
- Checks Rights of Enforcing Obligations	○	○	○	◐	●	●	○	◐	◐
- Specifies Action Handler (Sect. 4)	○	◐	●	●	○	●	●	◐	◐
- Specifies Event Handler (Sect. 4)	○	◐	●	●	○	●	◐	◐	◐
Log and Audit (Sect. 4)	○	◐	◐	◐	○	●	◐	◐	◐
Trust Model (Sect. 4)	○	○	○	◐	◐	●	◐	○	◐
Protocol independent (HTTP, WS)	○	●	●	●	◐	●	●	◐	●
Policy for Implicit PII (e.g. IP address)	●	●	◐	●	○	●	○	○	●

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. 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. 7) *Can Retrieve Applicable Preferences (Sect. 2.1)*: 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).

We define six criteria related to PII Consumer's Policy. 1) *Simple Syntax*: Privacy policies are expressed in a human-readable language. Syntax and semantics are well defined and can be processed by machines. 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. 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. 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. 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. 6) *Can Retrieve Applicable Policy (Sect. 2.1)*: 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.

One criterion is related to *PII Store*: 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).

One criterion focuses on *Privacy-Aware Service Discovery*: This technology provides mechanisms to discover services based on functional properties and on non-functional properties such as privacy.

One criterion compares *PII Lookup*: This technology offers mechanisms to gather pieces of personal data that are required by a given interface of the PII Consumer.

We defined five criteria related to Policy Matching. 1) *Has Logic Foundations*: The evaluation whether privacy policies do fulfill privacy preferences has logic foundations. 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 matching phase. 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 while matching. 4) *Takes Downstream Properties into Account (One Hop)*: Not only the privacy policy of the data controller is taken into account while matching but also the policy of third parties, which may get subsequently access to the personal data. 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 matching phase.

One criterion is related to *PII Selection*: 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).

Two criteria focus on mechanisms to Change Preferences. 1) *Can Show Mismatches*: In case of mismatch, the root causes of the mismatch can be identified and highlighted. 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.

Here are five criteria related to Sticky Policy. 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. 2)*Can be Expressive*: The sticky policy can express complex constraints with conditions. 3) *Supports Signature or Commitment*: The sticky policy can be signed by one or more parties to ensure non-repudiation of agreed privacy constraints. 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. 5) *Can Store and Retrieve Sticky Policy (Sect. 2.1)*: There is a mechanism to store sticky policies and to query the sticky policy associated with a given piece of personal data.

One criterion focuses on mechanisms to Attach Sticky Policy: Mechanism to attach the sticky policy to data on the wire and in databases. Mechanisms such as Enterprise Rights Management (e.g. [14]) would be an example where personal data cannot be decrypted without acknowledging the sticky policy (i.e. licenses).

We define three criteria for High-Level Policy Language. 1) *Same Language for Preferences and Policies*: Privacy preferences, policies, and sticky policies are expressed in a common language that avoid semantics mismatches. 2) *Language Expressiveness*: The common language is expressive and allows the specification of conditions, nested or recursive policies, and variables. 3) *Clear Separation of Obligations and Rights*: Obligations and rights are clearly expressed to handle, for instance, the right to store personal data for 3 months, the obligation of storing data for 3 months, and the obligation of deleting data within 3 months.

One criterion focuses on mechanisms to *Check Sticky Policy*: 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.

We define four criteria for Authorization Decision. 1) *Enforces Local Use, e.g. Purpose (Sect. 4)*: Before using collected data, the PII Consumer can verify that actions are authorized according to sticky policies. 2) *Enforces Access Control when Sharing (Sect. 4)*: Authorization of sharing data with a third party takes into account the sticky policy and attributes (e.g. certificates) of the third party. 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. 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.

We define one criterion for *Composing Sticky Policies*: 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 $pii_{1,2} = f(pii_1, pii_2)$.

Four criteria focus on Obligations. 1) *Supports Enforcement of Obligations*: There are mechanisms to automatically enforce obligations that can be specified in (sticky) policies. 2) *Checks Rights of Enforcing Obligations*: Mechanisms to define lower bound and upper bound of behavior. 3) *Specifies Action Handler (Sect. 4)*: There are mechanisms to parse and execute actions associated with obligations. It is possible to extend the set of actions that are handled. 4) *Specifies Event Handler (Sect. 4)*: 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.

We define one criterion for *Log and Audit*: 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.

Here is one criterion for *Trust Model*: 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.

We define one criterion for *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).

We define one criterion for *Policy for Implicit PII*: It is possible to specify how PII Consumer handles personal data that are implicitly collected (e.g. IP address).

6 Conclusions

The Abstract Privacy Policy Framework defines key features required to enforce privacy policies in Service Oriented Architecture (SOA). Future work will extend the framework and refine each component.

In this paper, the abstract framework has been instantiated with different technologies in order to compare their suitability in SOA:

It appears that using P3P [24], APPEL [23], and EPAL [3] 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 are 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 [11], UMA [13]) is not sufficient even when obligations can be specified (e.g. XACML [22]). 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 [1] provides more features than P3P but does not address the preference side and complex downstream cases.

S4P [6] offers promising features but only the core functionality (evaluation of queries) has been implemented. Tools tools for creating sticky policies, for enforcing policies, and for auditing execution traces need to be developped.

Finally PPL [18] supports a large part of the abstract privacy policy framework. This is not surprising since PrimeLife's work packages on SOA and on Policies are strongly connected. PPL mainly lacks homogeneity and logic foundations to enable reasoning on the policies.

Future work will add columns to Table 1 by evaluating the use of Usage Control and Right Expression Languages to instantiate the abstract framework. Moreover, the number of rows will increase by refining evaluation criteria. This work will also impact the evolution of policy languages we are contributing to.

Acknowledgements. 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), for their feedback on the different instantiations of the abstract framework.

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 216483 for the project PrimeLife.

References

1. Ardagna, C.A., Cremonini, M., De Capitani di Vimercati, S., Samarati, P.: A privacy-aware access control system. J. Comput. Secur. 16(4), 369–397 (2008)
2. Ardagna, C., De Capitani di Vimercati, S., Foresti, S., Paraboschi, S., Samarati, P.: 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)
3. Ashley, P., Hada, S., Karjoth, G., Powers, C., Schunter, M.: Enterprise privacy authorization language, EPAL 1.2 (2003)
4. Becker, M.Y., Fournet, C., Gordon, A.D.: SecPAL: Design and semantics of a decentralized authorization language. Journal of Computer Security (2009)

5. Becker, M.Y., Mackay, J.F., Dillaway, B.: Abductive authorization credential gathering. In: IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY) (July 2009)
6. Becker, M.Y., Malkis, A., Bussard, L.: A Practical Generic Privacy Language. In: Jha, S., Mathuria, A. (eds.) ICISS 2010. LNCS, vol. 6503, pp. 125–139. Springer, Heidelberg (2010)
7. Bussard, L., Nano, A., Pinsdorf, U.: Delegation of access rights in multi-domain service compositions. Identity in the Information Society 2(2), 137–154 (2009), http://www.springerlink.com/content/020524p066765742/
8. Bussard, L., Neven, G., Preiss, F.S.: Downstream usage control. In: IEEE Policy 2010 (July 2010)
9. ContentGuard: XrML 2.0 Technical Overview (2002), http://www.xrml.org/reference/XrMLTechnicalOverviewV1.pdf
10. Coulouris, G., Dollimore, J., Kindberg, T.: Distributed Systems. Concepts and Design, 4th edn. Addison Wesley (2005)
11. Hammer-Lahav, E.: RFC 5849: The OAuth 1.0 Protocol (2010), http://tools.ietf.org/html/rfc5849
12. Kagal, L., Abelson, H.: Access control is an inadequate framework for privacy protection. In: W3C Workshop on Privacy for Advanced Web APIs (July 2010)
13. Kantara Initiative: User managed initiative, http://kantarainitiative.org/confluence/display/uma/
14. Microsoft: Rights Management Services (2009), http://www.microsoft.com/windowsserver2008/en/us/ad-rms-overview.aspx
15. ODRL: Open Digital Rights Language (ODRL), version 1.1 (2002), http://www.odrl.net/1.1/ODRL-11.pdf
16. Pinsdorf, U., Bussard, L., Meissner, S., Schallaböck, J., Short, S.: Privacy in Service Oriented Architectures. In: Camenisch, J., Fischer-Huebner, S., Rannenberg, K. (eds.) Privacy and Identity Management for Life, pp. 383–411. Springer, Heidelberg (2011)
17. Pretschner, A., Schütz, F., Schaefer, C., Walter, T.: Policy evolution in distributed usage control. In: 4th Intl. Workshop on Security and Trust Management. Elsevier (June 2008)
18. PrimeLife Consortium: Draft 2nd design for policy languages and protocols (heartbeat: H5.3.2). Tech. rep. (July 2009)
19. PrimeLife Consortium: Second Release of the Policy Engine (D5.3.2). Tech. rep. (September 2010)
20. PrimeLife Consortium: Infrastructure for Privacy for Life (D6.3.2). Tech. rep (January 2011), http://www.primelife.eu/images/stories/deliverables/d6.3.2-infrastructure_for_privacy_for_life-public.pdf
21. Rahman, S.T.: Analyzing Causes of Privacy Mismatches in Service Oriented Architecture. Master's thesis, RWTH (2010)
22. Rissanen, E.: OASIS eXtensible Access Control Markup Language (XACML) Version 3.0. OASIS committee specification 01, OASIS (August 2010)
23. W3C: A P3P preference exchange language 1.0, APPEL1.0 (2002)
24. W3C: The platform for privacy preferences 1.1 (P3P1.1) specification (2006)
25. Wang, X.: MPEG-21 Rights Expression Language: Enabling Interoperable Digital Rights Management. IEEE MultiMedia 11(4), 84–87 (2004)