Abstract— Whereas access control describes the conditions that have to be fulfilled before data is released, usage control describes how the data has to be treated after it is released. Usage control can be applied to digital rights management, where the data are usually copyright-protected media, as well as in privacy, in which case the data are privacy-sensitive personal information. An important aspect of usage control for privacy, especially in light of the current trend towards composed web services (so-called mash-ups), is downstream usage, i.e., with whom and under which usage control restrictions data can be shared. In this work, we present a two-sided XML-based policy language: on the one hand, it allows users to express in their preferences in a fine-grained way the exact paths that their data is allowed to follow, and the usage restrictions that apply at each hop in the path. On the other hand, it allows data consumers to express in their policies how they intend to treat the data, with whom they intend to share it, and how the downstream consumers intend to treat the data.

Keywords—privacy; policy languages; usage control; data sharing;

I. INTRODUCTION

Many web services today are so-called service mash-ups. A mash-up is a service that acts as a front-end for a composition of multiple subservices that are offered by different companies. For example, a travel booking mash-up may offer an integrated interface to book flights, hotels, and rental cars. In the background, however, it invokes the web service APIs of different specialized airline, hotel, and car rental subsidiaries to collect offers. The best offers are presented to the user, who selects an offer, enters her booking and payment information, and confirms to let the mash-up make all the bookings for her through the subsidiaries’ APIs.

Service mash-ups are important for leveraging online service APIs to create new functionality, but pose significant privacy risks for their users. The users cannot keep track of who stores what information about them, and often they do not even know the subsidiaries their data is shared with.

To overcome this problem, service providers publish their privacy policies to inform the users about how the gathered data is used. Human-readable privacy policies, e.g., have the disadvantage of mostly being written in complex language influenced by the legal profession and thus being ignored by users. Even if clear privacy policies are presented, they often remain vague about the sharing of information with third parties. For example, the privacy policy of Expedia.com\(^1\), a popular online travel service, mentions the following with regard to sharing personal information with suppliers:

_We do not place limitations on our suppliers’ use or disclosure of your other personal information [i.e., other than the email address]. Therefore, we encourage you to review the privacy policies of any travel supplier whose products you purchase through this site._

Machine-interpretable privacy policy languages such as EPAL [2] and P3P [17] is a promising approach, in particular when used in combination with a privacy preference language such as APPEL [16]. In the latter language, users can express how they expect their data to be treated, so that an automated or semi-automated matching procedure can decide on the acceptability of a proposed P3P policy.

Unfortunately, EPAL and P3P are both rather constrained in expressivity regarding sharing personal information with third parties, or downstream usage as we call it here. EPAL leaves the definition of specific actions and obligations up to enterprise-defined vocabularies, and is hence silent about downstream usage. Support in P3P is limited to specifying which of six classes of third-party recipients the information will be shared with; it is up to the server to classify his recipients into one or more of the classes.

In this work, we investigate how to structure a policy language suitable for downstream usage control. The difficulty here is that downstream usage control involves a mixture of what is typically considered access control (who is allowed to receive the data, e.g., by roles or owned credentials) and usage control (how is the recipient supposed to treat the data, e.g., usage purposes or retention period). We consider the most general setting here, where the user states in her privacy preferences, for each hop in a chain of downstream recipients, how they have to treat her data and to whom they can further forward it. At the same time, each recipient specifies in his privacy policy how he intends to treat the data and to whom he intends to forward it. We propose XML-based languages to express both the user’s preferences and the servers’ policies, and describe an automated matching algorithm to determine whether the proposed policies are

allowed by the specified preferences. The user and the servers can specify downstream usage restrictions up to any number of hops (not necessarily the same number). Optionally, they can either specify that the last restrictions in the chain are valid for all subsequent hops, or that after that hop no further downstream usage is allowed. Moreover, for situations where the server does not know at the time of data collection to whom he may forward the data, we propose an alternative matching algorithm at which a server declares his willingness to impose any restrictions on further downstream usage that the user may specify.

This work relies on the trust model of P3P and EPAL: each service is willing to enforce its privacy policy. Authors assume that service providers are appropriately enforcing their policies to protect their reputation and/or that services are regularly audited and certified by trusted third parties. Proving the correctness of policy enforcement, for instance using a trusted stack (certified TPM, trusted OS, and trusted application), is out of the scope of this paper.

II. RELATED WORK AND CONTRIBUTIONS

Our work is closely related to rights expression languages (RELS), privacy policy languages, and usage control. We give a brief overview of each of these lines of work and how they relate to the language we propose.

Rights Expression Languages: Even if scenarios and trust models are different, there are clear similarities between digital right management (DRM), enterprise right management (ERM), and privacy policies as described in this paper. Indeed, in each case, a data provider attaches constraints, in the form of a license or a sticky policy, to data sent to a data consumer. The domain-specific vocabularies for privacy policies and RELs may be rather different, but the same overall language structure can be used for both.

There are three main differences between our approach and the state of the art ERM and DRM languages MPEG-21 REL [18], XrML [7], and ODRL [11]. First, in RELs it is the data provider who unilaterally creates the license and imposes it onto the consumer, without any kind of matching procedure. In privacy settings, the provider (i.e., the user) usually does not have such power. Second, RELs focus mainly on rights (e.g., play, print) and constraints on these rights (e.g., temporal, fees, device), but obligations remain underspecified. Third, the constraints on downstream data sharing (e.g., sell, lease, share) are not nearly as expressive as in our model.

Privacy Policy Languages: In the introduction we already discussed the shortcomings of the privacy policy languages EPAL [2], P3P [17], and APPEL [16] when it comes to expressing restrictions on downstream usage. SecPAL for Privacy (S4P) [4] is a logic-based language to specify privacy policies and preferences. In S4P, matching is about evaluating queries with a set of assertions, while in the work presented in this document, matching is done by comparing statements. Ardagna et al. [1] describe a data handling policy language that allows for specifying data recipients, usage purposes and obligations. The data recipient can specify global rules on who may receive the data, but transitive downstream usage restrictions cannot be expressed.

Usage Control: The subject of usage control [8], [12] is concerned with how data is treated after its release. The Obligation Specification Language (OSL) [9] supports a wide range of usage control requirements related to time, cardinality, purpose, and events. OSL is logic-based, so that a sequence of events can automatically be checked for compliance with the specified policy. The OSL extensions for policy evolution [13] target a use case similar to ours as they allow for expressing which consumers (indicated by their roles) have to adhere to which rights and duties. However, in OSL the data provider unilaterally creates the policy to be adhered to, whereas in our language it is the result of matching a consumer’s policy against the provider’s preferences. Moreover, in our language one can describe the full downstream path that the data is allowed to follow, including who can share it with whom, and how many hops it can take. For example, in our language a patient could impose one usage control policy when a health insurance company obtains her medical record through the hospital, and another policy when it obtains the record from the patient directly. In OSL both cases are treated the same.

Provenance Access Control: As highlighted in [14], we consider that metadata associated with personal data may also be subject to usage control. For instance, the obligation of notifying the user may contain an e-mail address.

III. DESCRIPTION OF SOLUTION

A. Example Scenario

Alice is a privacy-aware user who regularly shops online, but who is concerned about what happens to the data that she provides about herself. For example, she’s willing to provide her postal address to online shops so that the goods can be delivered, but she realizes that most shops rely on external shipping companies. She wants to impose a detailed set of usage control preferences though, where the restrictions on the shipping company depend on who the front-end service is from which it obtained the address. Namely, when obtained through a book shop, she is fine with the shipping company using her address for statistics, but when obtained through any other store, which may include liquor or lingerie stores, she is not. More precisely, she wants to enforce the following preferences:

- Book shops can collect her address for the purpose of statistics, contact, and account administration. They must delete it after two years and are allowed to forward to shipping companies who can use it for shipping and statistics, have to delete it after two weeks, and can
further forward it to shipping companies under the same restrictions.

- Any shop can collect her address for the purpose of account administration provided they delete it within one year. They can further forward it to shipping companies who can use it for shipping only, have to delete it after two weeks, and are not allowed to share the data with anyone.

Alice regularly buys books at the online book shop bookshop.com because its privacy policy matches her preferences. Namely, bookshop.com states that it will

- use her address for statistics and account administration, that it will delete the address after one year, and that it will forward the address only to shipping.com.

The policy of shipping.com states that

- the address will be used for shipping and statistical purposes, and will be deleted after one week.

When buying a book at bookshop.com, Alice can safely give her address away since bookshop.com’s and shipping.com’s privacy policies match her own preferences. However, when buying a bottle of wine at liquor.com, who also use shipping.com for shipping, the transaction will be refused, as Alice’s preferences in this case do not allow shipping.com to use her address for statistical analysis.

B. Language Model

The abstract scenario we consider is one where two parties, typically a user and a server, engage in an interaction where one of the parties, typically the server, requests some personally identifiable information (PII) from the other party. We will from now on call the party that provides the data the data provider and the party that requests the data the data consumer. Moreover, we consider a scenario where at a later point in time, the data consumer may want to forward the PII to a third party, called the downstream data consumer.

Both the data provider and the data consumer have their own policies expressing the required and proposed treatment of the PII, respectively. These policies contain access control and usage control policies. A piece of PII is only sent to a data consumer after (1) the access control requirements have been met, and (2) a suitable usage control policy has been agreed upon.

We distinguish three kinds of policies:

- Preferences: In his preferences the data provider describes, for specific pieces of PII, which access control requirements a data consumer has to satisfy in order to obtain the PII, as well as the usage control requirements according to which the PII has to be treated after transmission. These requirements may include downstream usage requirements, meaning the requirements that a downstream data consumer has to fulfill in order to obtain the PII from the (primary) data consumer.

- Policy: The policy is the data consumer’s counterpart of the data controller’s preferences. In a policy the data consumer contains, for specific pieces of PII to be obtained, his certified properties (roles, certificates, etc.) that can be used to fulfill access control requirements, and a usage control policy describing how he intends to use the PII.

- Sticky policy: The sticky policy describes the mutual agreement concerning the usage of a transmitted piece of PII. This agreement is the result of a matching process between a data providers’s preferences and a data consumer’s policy. Technically a sticky policy is quite similar to preferences as described above, but it describes a mutual agreement between the data provider and the data consumer that cannot be changed. After receiving the PII, the data consumer is responsible for storing and enforcing the sticky policy.

To illustrate our ideas we employ a simple XML-based language to express preferences, policies, and sticky policies. In the following we first introduce our language and then focus on how to express downstream usage requirements. Note that we provide the full language schemas in an extended version of this paper [6].

1) Preferences model: Figure 1 shows Alice’s preferences for book shops expressed in our policy language. The Preferences root element contains multiple Preference elements, each describing to which PII it applies and what the respective access and usage controls are. An attribute sticky indicates whether these preferences are in fact a sticky policy for the PII (cf. III-B3), acting as an explicit reminder that they cannot be changed. Alternatively, one could keep all sticky policies separately in a read-only policy store.

A Preference can refer to the applicable PII by their data type, meaning that the Preference applies to all PII of this type, or by a unique identifier pointing to a single instance of PII. We assume that a typing mechanism and unique naming scheme for PII are in place. A complete language would probably offer more powerful mechanisms to specify applicability, allowing for example attribute expressions or temporal constraints.

Our language is strictly limited to positive statements, in the sense that it explicitly lists the permitted information exchanges, and assumes that all other exchanges are forbidden. Apart from this being a safe privacy-conservative choice, it also simplifies the matching procedure. However, it means that one cannot express conditions of the form “do not forward to X” or “do not use for purpose Y”.

If multiple Preference elements apply to a single piece of PII, then satisfying the conditions in either of them results in a match. In other words, Preferences are combined according to “or” semantics. This makes it possible to define more permissive exceptions to general preferences.

Within a Preference, access and usage control require-
Access control requirements are expressed in terms of properties that a data consumer must have in order to be granted access. Properties are stated in terms of attributes as certified by some certification authority (CA). Empty access control requirements mean that anybody who commits to fulfilling the usage control requirements is granted access. The simple access control language that we use here could in a real system, e.g., be substituted with a complete role-based [15], attribute-based [5], [10], or logic-based [3] access control language.

Usage control requirements are expressed by distinguishing between Rights and Obligations. A right states an action that the data consumer is allowed to perform on the data, but doesn’t have to perform to comply with the policy. An obligation states an action that a data consumer is obliged to perform. We model two types of rights and two types of obligations here:

- **UseDownstream**: The right to forward the PII under given conditions to further data consumers. This is a crucial element in our policy language; we come back to its exact structure and meaning later.
- **UseForPurpose**: The right to use the PII for a specific purpose.
- **DeleteWithin**: The obligation to delete the PII within a given amount of time.
- **NotifyOnAccess**: The obligation to notify the user when the PII is accessed.

The complete language supports more rights and obligations, however, those four suffice to illustrate the ideas of this work. Multiple rights and obligations within a UsageControl element are combined by “and” semantics, meaning that the data consumer obtains all the specified rights and has to adhere to all of the specified obligations.

2) **Policy model**: A data consumer’s policy states which usage control requirements he is willing to adhere to when requesting a specific resource from a data provider. In addition the policy states the properties the data consumer is willing to disclose for fulfilling the data provider’s access control rules for that resource.

![Figure 1. Excerpt from Alice’s preferences.](image-url)

The language schema of the server policy resembles the schema of the preferences, exhibits the following differences though. The top-level elements are Policies and Policy, where the Policy has no sticky attribute; the AccessControl element contains the properties in Property elements rather than access requirement rules; and the ACUC element in the UseDownstream right has different cardinality as explained later. In the following we provide example policies for the book shop and the shipping company.

The shop’s policy stated in Figure 2 expresses that for collecting addresses, it is willing to authenticate as a shop or a book shop certified by CAx, and as bookshop.com certified by CAz. The address will be deleted after one year and used for statistics and account administration. Further, the shop wants to be able to forward it under the policy of the shipping company (that is specified below).

![Figure 3. shipping.com’s relevant policies.](image-url)

3) **Sticky policy model**: Sticky policies follow the same schema as preferences, but have the sticky attribute in the Preference element set to true. This is to indicate that this sticky policy originates from another party and must...
Figure 2. Excerpt from bookshop.com’s policies.

Figure 3. Excerpt from shipping.com’s policies.

thus not be modified. Note that a data consumer may, in addition to the sticky policy, also have own preferences for forwarding previously received PII. Those preferences can, however, be changed and are therefore not sticky.

C. Downstream usage

The crucial aspect of our policy language is that it allows both the data provider and the data consumer to express to whom and under what conditions PII can or will be forwarded. These conditions are expressed in UseDownstream elements.

We first focus on UseDownstream elements occurring in the data provider’s preferences. Each UseDownstream element contains exactly one ACUC child element. This ACUC element either contains a fully specified pair of access and usage control requirements, or another ACUC element is referenced with the reference attribute.

In the former case, the access control requirements specify to which downstream data consumers the PII can be forwarded, while the usage control requirements specify how these downstream consumers are supposed to treat it. Usage requirements can on their turn also contain UseDownstream elements that specify to whom and under what conditions the downstream consumer can further forward the PII, which on their turn can contain UseDownstream elements as well, etc. This mechanism enables the data provider to restrict the forwarding of his PII up to an arbitrary number of “hops”. We refer to this approach as nested downstream usage control.

In the case where an ACUC element references the content of another ACUC element, for the sake of simplicity we insist that it can only refer to its closest ancestor ACUC element, i.e., the ancestor four levels higher in the XML tree. (We impose this restriction since it simplifies the matching procedure and there seem to be no convincing use cases for “cyclically recursive” policies with cycle length greater than one.) This means that the data consumer can then forward the PII under the same restrictions that were imposed on himself. We therefore call this approach recursive usage control. Note that our policy language allows to combine nested and recursive usage control by defining a chain of nested usage controls for the first number of hops and a final recursive usage control for any further hops.

In a data consumer’s policy, each UseDownstream element contains at most one ACUC element. If present, it contains a set of properties describing to whom he plans to forward the PII, and a usage control policy describing how that downstream consumer will treat the data. This usage control policy could contain further UseDownstream elements, describing the next hops up to an arbitrary nesting degree. Alternatively, the reference attribute can be used to point to another ACUC element. This element may even be hosted directly by a downstream consumer (where we assume the reference to act as a URL).

In many situations, the downstream consumer or his policy are not be known at the time the PII is transmitted to the primary consumer. Rather than specifying all intended hops in full detail, the data consumer can indicate his willingness to enforce any restrictions imposed by the data provider by setting the allowLazy attribute of the UseDownstream element to true and omitting the ACUC element. The matching between the data provider’s preferences and the downstream consumer’s policy is then done by the primary consumer at the time the PII is forwarded to the downstream consumer. We refer to the next section for more details on lazy matching. When the allowLazy attribute occurs in Preferences, it indicates whether the data provider allows the restrictions expressed in the child ACUC element to be matched lazily.

```xml
<Policy>
  <Applicability>
    <DataType>Address</DataType>
  </Applicability>
  <ACUC id="ACUCaddress@Shipping">
    <AccessControl>
      <Property>CertifiedAsBy{ship,CAx}</Property>
    </AccessControl>
    <Obligations>
      <DeleteWithin>P7D</DeleteWithin>
      <Obligations>
        <Rights>
          <UseForPurpose>statistics</UseForPurpose>
          <UseForPurpose>shipping</UseForPurpose>
          <Rights>
            <UseDownstream allowLazy="false">
              <ACUC reference="ACUCaddress@Shopping"/>
            </UseDownstream>
          </Rights>
          <Obligations>
            <UseDownstream allowLazy="false">
              <ACUC reference="ACUCaddress@Shopping"/>
            </UseDownstream>
          </Obligations>
        </Rights>
      </Obligations>
    </UsageControl>
  </Policy>
  ...
</Policies>
```
Finally, in the UseDownstream element a maxDepth attribute can be set to an integer or to unbounded to indicate how often a piece of PII can be forwarded at most. The intended behavior concerning this limit can be explained with a counter contained in sticky policies. The counter in a sticky policy that is attached to forwarded PII is decreased by one w.r.t. the previous sticky policy or w.r.t. maxDepth in case the PII is forwarded by the primary data provider.

IV. MATCHING

Given a data provider’s preferences and a consumer’s policies, matching aims at automating the process of deciding whether the provider can safely transmit a piece of personal data. We introduce a ’more or equally permissive than’ operator to match preferences with policies. We say there is a match when the preferences are more or equally permissive than the policy.

A. Matching Privacy Preferences and Policies

To explain the matching procedure, we use a set-based representation of the XML structure described in the previous section. A Preferences element is represented by a set Pref$\subseteq$Prefs containing an element Pref $\in$ Pref$\subseteq$Prefs for each of its Preference child elements in the XML structure. Pref$\subseteq$App represents a set containing all the PII owned by the user that is covered by the Applicability element, and Pref$\subseteq$ACUC designates the contained ACUC child elements. The set ACUC$\subseteq$Pref$\subseteq$Prefs is the set of access control rules (e.g., CertAsBy) contained in the Rule elements of the embedded AccessControl, while ACUC$\subseteq$UC is the set of usage controls in terms of rights (UC$\subseteq$Rights) and obligations (UC$\subseteq$Obls), specified by the Rights and Obligations elements, respectively. We use an analogous notation for the consumers’ preferences.

Intuitively, preferences Pref$\subseteq$Prefs are more or equally permissive than policies Pol$\subseteq$Prefs, denoted Pref$\subseteq$Pols, if the access control properties in Pol$\subseteq$Prefs satisfy the rules in Pref$\subseteq$Prefs and if Pol$\subseteq$Prefs asks for less rights and promises to adhere to stricter obligations than specified in Pref$\subseteq$Prefs. We “overload” the notation of the $\subseteq$ operator to compare not only preferences with policies, but also to compare rights, obligations, access control policies as well as usage control policies.

To determine if there is a match between preferences Pref$\subseteq$Prefs and policies Pol$\subseteq$Pol$\subseteq$Pols, it is verified if for each ACUC$\subseteq$ACUC$\subseteq$Pref it is verified if for each ACUC$\subseteq$ACUC$\subseteq$Pref$\subseteq$Prefs with a more or equally permissive ACUC$\subseteq$ACUC pair:

$$\text{Pref} \subseteq \text{Prefs} \iff \forall \text{Pol} \in \text{Pol} \subseteq \text{Pols} \exists \text{Pref} \in \text{Pref} \subseteq \text{Prefs} \exists \text{PII} \in \text{PIIs} \cdot \text{PII} \in (\text{Pol} \cdot \text{App} \cap \text{Pref} \cdot \text{App}) \cdot \text{Pref} \cdot \text{ACUC} \subseteq \text{Pol} \cdot \text{ACUC}$$  \hspace{1cm}(1)

Above, PIIs is the set of all pieces of personal information that the user possesses, and PII can be any specific piece of PII in that set. In the following, we use the notations $\ast$Pref$\subseteq$Pref and $\ast$Pols$\subseteq$Pols to denote elements within preferences and policies respectively.

Pairs of access control and usage control policies are matched as follows:

$$\text{ACUC}_{\ast \text{Pref}} \subseteq \text{ACUC}_{\ast \text{Pols}} \iff (\text{ACUC}_{\ast \text{Pref}} \cdot \text{AC} \supseteq \text{ACUC}_{\ast \text{Pols}} \cdot \text{AC}) \land (\text{ACUC}_{\ast \text{Pref}} \cdot \text{UC} \supseteq \text{ACUC}_{\ast \text{Pols}} \cdot \text{UC})$$  \hspace{1cm}(2)

Note that (2) is evaluated multiple times during the evaluation of (1). For example, ACUC$\subseteq$Pref is instantiated subsequently with Pol$\subseteq$ACUC for all Pol$\subseteq$Pols. The access control mechanism we employ is based on certified properties such as roles or IDs. To match access control requirements, it is verified if for each Rule in the preferences there is a corresponding Property in the policy:

$$\text{AC}_{\ast \text{Pref}} \subseteq \text{AC}_{\ast \text{Pols}} \iff \forall r \in \text{AC}_{\ast \text{Pref}} \cdot \exists r' \in \text{AC}_{\ast \text{Pols}} \cdot r = r'$$  \hspace{1cm}(3)

This could be extended to cover more sophisticated access control mechanisms such as claim-based access control or hierarchical roles. However, the matching becomes more complex when doing so (e.g., as environment attributes such as time of day cannot be pre-evaluated). Usage control requirements are matched as follows:

$$\text{UC}_{\ast \text{Pref}} \supseteq \text{UC}_{\ast \text{Pols}} \iff \forall \text{R} \in \text{UC}_{\ast \text{Pref}} \cdot \text{Rights} \cdot \exists \text{R}' \in \text{UC}_{\ast \text{Pols}} \cdot \text{Rights} \cdot \text{R}' \subseteq \text{R} \land \forall \text{O} \in \text{UC}_{\ast \text{Pref}} \cdot \text{Obls} \cdot \exists \text{O}' \in \text{UC}_{\ast \text{Pols}} \cdot \text{Obls} \cdot \text{O}' \supseteq \text{O}$$  \hspace{1cm}(4)

The matching of rights and obligations is specified for the different types of rights and obligations individually. In the following we give example specifications for the ones introduced in Section III-B1. If obligations R$\subseteq$Obls and R$\subseteq$Obls specify that the user must be notified when her PII is accessed, R$\subseteq$Obls is evaluated with the appropriate $\supseteq$ operator, i.e., (6) in this case. Letting obligation DeleteWithin with duration t be denoted as DelWithin(t), matching is done as follows:

$$\text{DelWithin}(t) \subseteq \text{DelWithin}(t') \iff t \geq t'$$  \hspace{1cm}(5)

Letting obligation NotifyOnAccess with contact information c be denoted as NotifyOnAccess(c), matching is done as follows (where $\ast$ represents any string):

$$\text{NotifyOnAccess}(c) \subseteq \text{NotifyOnAccess}(c') \iff c' = \ast \lor c = c'$$  \hspace{1cm}(6)

Letting right UseForPurpose with purpose p be denoted as UseForPurpose(p), matching is done as follows:

$$\text{UseForPurpose}(p) \subseteq \text{UseForPurpose}(p') \iff p = p'$$  \hspace{1cm}(7)

For the sake of readability, support for hierarchical purposes is not described here.
As downstream usage is the focus of this paper, the following two subsections explain the details of the matching procedure for downstream usage rights.

B. Proactive Matching of Downstream Rights

This section provides the intuition behind proactively matching a downstream structure, i.e., matching structures where both the preferences and the full chain of downstream usage policies are known at the time of matching.

For a given pair $ACUC$, let $[ACUC]$ be the “local” ACUC, meaning containing only those restrictions and obligations that do not affect downstream usage. Using this notation, we can represent the structure of an ACUC policy with downstream usage as a directed graph where each node represents a hop in the downstream usage. Each node is labeled with the local ACUC policy describing how the data are to be treated locally. Each edge represents the permission (in case of a provider’s preferences) or intention (in case of a consumer’s policy) to forward the data under the restrictions specified by the ACUC of the endpoint of the edge. By the restrictions that we imposed on the connection among ACUC pairs, the structure of the graph is similar to that of a tree where the leaf nodes can optionally have a loop, representing recursion in the downstream usage policy. Intuitively, matching two ACUC pairs $ACUC_{pref}$ taken from a provider’s preferences and $ACUC_{pol}$ specified in a consumer’s policy is done by simultaneously going over the nodes in the two tree representations of $ACUC_{pref}$ and $ACUC_{pol}$ and verifying that it is possible to cover each branch of the policy-side tree with a more or equally permissive branch on the preference side.

Letting $UseDS(ACUC)$ denote a $UseDownstream$ element with an $ACUC$ child element represented by $ACUC$, the matching for downstream usage rights works according to the rule:

$$UseDS(ACUC) \equiv UseDS(ACUC') \iff ACUC \equiv ACUC'$$ (8)

We assume that an $ACUC$ specifying recursive downstream sharing right with depth $d$ is “folded out” into a graph of $d+1$ nested $ACUC$. Formula (8) is thus sufficient to handle nested as well as recursive access and usage control. Obviously, to improve performance when matching recursive $ACUC$ and to avoid infinite loops in case of unlimited recursion depth, the concrete implementation of the matching algorithm must keep track of which combinations of $ACUC$ elements have already been matched against one another and must take depth into account. We refer to the extended version [6] for a complete example of a proactive matching process.

C. Lazy Matching

In the previous section we focused on proactive matching, i.e., matching where all downstream policies are known beforehand. It is, however, not always possible to collect all policies during matching. For this reason, we also introduce lazy matching, which only takes into account the properties and policies of the data consumer, but not those of any downstream data consumers. Rather, the data consumer expresses that he is willing to impose whatever usage restrictions on downstream consumers that the data provider may specify.

Both types of matching imply that the sticky policy that the data consumer associates to the data must at least enforce the preferences of the data provider. On one hand, proactive matching allows to minimize the rights and maximize the obligations that are transferred as the matching procedure can already take the downstream consumers and their policies into account. On the other hand, lazy matching offers more flexibility and is the only option in dynamic settings, where either the downstream consumers or their policies are not known yet at the moment of matching, or where the access control policy depends on environment variables that will only be known when the data is actually forwarded.

To support lazy matching, we redefine formula (8) to take the allowLazy attribute into account, which is represented by a boolean value lazy here. Matching for downstream usage then follows the rule:

$$UseDS(lazy, ACUC) \equiv UseDS(lazy', ACUC') \iff (lazy \land lazy') \lor (ACUC \equiv ACUC')$$ (9)

D. Creating Sticky Policies

A sticky policy specifies the commitment of the data consumer towards the data provider w.r.t. treatment of her shared data. We envision a sticky policy $SP$ to be produced in case a matching procedure is successful. This sticky policy then never violates the preferences and is compliant with the policy, i.e., $Prefs \supseteq SP \supseteq Pols$ always holds. (Note that we assume that matching preferences with preferences is analog to the matching of preferences with policies.)

A privacy-conservative matching algorithm will choose a sticky policy that is as close as possible to the policy proposed by the consumer. Note that a sticky policy includes only those preferences that contain rights the data consumer gets or obligations he has to adhere to w.r.t. a given piece of PII.

Clearly, when a primary data consumer forwards the data to a downstream consumer, he must also attach a sticky policy. The data consumer may have its own preferences $Prefs'$ regarding data sharing. On top of his own preferences, the data consumer must enforce the sticky policy $SP$ associated to this piece of data. In other words, the primary data consumer has to find a downstream sticky policy $SP'$ so that $Prefs' \supseteq SP' \supseteq Pols'$ and, informally, $SP \supseteq SP' \supseteq Pols'$, where $Pols'$ is the policy of the downstream data consumer. More precisely, the latter evaluation is done by first extracting downstream preferences ($Prefs_{PS}$) from the sticky policy ($SP$) as depicted below, where $A \leftarrow B$ denotes the assignment to $A$ of the value of $B$:
∀Pref ∈ SP · (∀R · (R ∈ Pref.ACU.UC.Rights ∧ R = UseDS(·, ·))) ·
(Pref_DS.App ← Pref.App,
Pref_DS.ACU ← R.ACU,
Pref_SP ← PrefPref DS

Next Pref DS is used for matching the downstream policy (i.e., Pref DS ⊇ Pols') and to create the downstream sticky policy SP' so that Pref DS ⊇ SP' ⊇ Pols'.

V. CONCLUSION AND FUTURE WORK

This paper presents a simple yet expressive language to specify privacy policies and user preferences which may express downstream usage requirements. Our paper also describes a matching algorithm that, given a server’s privacy policy, helps a user to decide whether her personal data can be shared with the server according to her preferences.

In our language we employ disjunctive semantics when combining multiple preference elements. We are currently investigating if deviating from this semantics allows policies and preferences to be expressed in a more compact way. Moreover, we also study how the matching algorithms have to be adapted when employing more advanced access control mechanisms than the one described in this paper.

Finally, we are developing tools to facilitate the enforcement of policies. Those tools only help honest services to fulfill their commitments. Proving correct enforcement remains out of our scope.

The main ideas of our language have been integrated into the PrimeLife Policy Language (PPL). A prototype engine implementation is currently being developed by the PrimeLife project.

ACKNOWLEDGMENT

The authors would like to thank Ulrich Pinsdorf and Mario Verdicchio for their constructive feedback. 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