A Privacy-Aware Access Control System[*]

C.A. Ardagna, M. Cremonini, S. De Capitani di Vimercati, and P. Samarati
Università degli Studi di Milano
Dipartimento di Tecnologie dell'Informazione
Via Bramante 65, 26013 Crema - Italy
{ardagna,cremonini,decapita,samarati}@dti.unimi.it
Phone: +39.0373.898048 – Fax: +39.0373.898010

Abstract

The protection of privacy is an increasing concern in our networked society because of the growing amount of personal information that is being collected by a number of commercial and public services. Emerging scenarios of user-service interactions in the digital world are then pushing toward the development of powerful and flexible privacy-aware models and languages.

This paper aims at introducing concepts and features that should be investigated to fulfill this demand. We identify different types of privacy-aware policies: *access control*, *release*, and *data handling* policies. The access control policies govern access/release of data/services managed by the party (as in traditional access control), and release policies govern release of *personal identifiable information* (PII) of the party and specify under which conditions it can be disclosed. The data handling policies allow users to specify and communicate to other parties the policy that should be enforced to deal with their data. We also discuss how data handling policies can be integrated with traditional access control systems and present a privacy control module in charge of managing, integrating, and evaluating access control, release, and data handling policies.

Keywords: Access control, Privacy, Data handling policies.

1 Introduction

The increased power and interconnectivity of computer systems available today provide the ability of storing and processing large amounts of data, including personal information of users and customers of commercial and public services. The vast amount of personal information thus available on the Web has led to growing concerns about the privacy of its users. Privacy has been recognized as one of the main reasons that prevents users from using the Internet for accessing on-line services. To this end, many useful *privacy enhancing technologies* (PETs) have been developed for dealing with privacy issues and previous works on privacy protection have focused on a number of related topics [4, 13, 14, 18, 26, 32]. In particular, an important service for helping users to keep the control over their personal information is represented by access control solutions enriched with the ability of supporting privacy requirements [4, 20].

Despite the benefits of such solutions, few proposals have addressed the problem of how to regulate the use of personal information in secondary applications. Users do not always realize that the information they disclose for one purpose (e.g., name, date of birth, and address within an on-line transaction) may also have secondary uses (e.g., access to existing data for grouping together users on the basis of common characteristics such as age or geographic location). Therefore, even if users consent to the initial collection

[*] A preliminary version of this paper appeared under the title "Enhancing User Privacy Through Data Handling Policies," in *Proc. of the 20th Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, Sophia Antipolis, France, July 31-August 2, 2006 [5].

of their personal information, they must also be provided with the possibility to specify whether or not to consent to the future uses of that information in secondary applications. To address this issue, it is widely recognized that a standardized format for privacy policies would allow users to quickly assess whether a particular site's privacy policy satisfies their privacy goals. This idea was behind the development of approaches like the *Platform for Privacy Preferences Project* (P3P) [31] and *PReference Expression for Privacy* (PREP) [2]. P3P is a XML-based language that allows service providers and users to reach an agreement on the release of personal data. PREP is another XML-based language used for representing the user's privacy preferences with Liberty Alliance. Although these approaches represent a first step towards the development of a solution supporting user privacy preferences, a more deeper integration between privacy and access control requirements is needed. A privacy-aware access control solution should encompass two aspects: guaranteeing the desired level of privacy of information exchanged between different parties and controlling the access to services/resources according to this information; and controlling all secondary uses of information disclosed for the purpose of access control enforcement. In addition, a privacy-aware access control solution should be simple and expressive enough to support, among others, the following privacy requirements [21].

- *Openness.* Policies and privacy practices should be transparent and fully understandable for all parties.
- *Individual control.* Users should be able to specify who can see what information about them and when.
- *Collection limitation.* Parties collecting personal data for the purposes of a transaction must gather no more data than what strictly needed for carrying out the transaction itself.
- *Purpose specification.* Those who collect and disseminate personal data must specify the purposes for which they need these data. Collected data must therefore be used only for specified purposes.
- *Consent.* Users should be able to give their explicit and informed consent on how to use their personal data.
- *Data quality.* Those who collect and disseminate personal data must maintain accurate information. Users, therefore, should be able to verify their personal information and modify it when needed.
- *Data security.* Adequate security mechanisms for data protection have to be applied, according to the sensitivity of collected personal data.

In this paper, we present a privacy-aware framework that integrates access control policies together with a new type of privacy policy, called *data handling policy* [5]. A data handling policy regulates how *Personal Identifiable Information* (PII) will be handled at the receiving parties (e.g., information collected through an on-line service may be combined with information gathered by other services for commercial purposes). Here, the term PII is used in its wider meaning and identifies all personal data of users, that is, all the information that can be linked to an identity. We also present the working of a privacy-aware access control module developed in the context of the Privacy and Identity Management for Europe (PRIME) project [22], an European project whose goal is the development of privacy-aware solutions for enforcing security.

The remainder of this paper is organized as follows. Section 2 presents related work. Section 3 illustrates our reference scenario together with some basic concepts about our privacy-aware policies. Section 4 presents the different types of privacy policies we have identified, that is, access control, release, and data handling policies. Section 5 introduces the basic elements of a data handling policy and presents the specific language for expressing such policies. Section 6 discusses the interactions between parties for data and services release. Section 7 describes a privacy-aware access control module that integrates the access control, release, and data handling policies. Finally, Section 8 presents our conclusions and sketches future work.

2 Related work

The works most directly related to ours lay in the area of credential-based access control and privacy-aware languages and models. The first proposals that investigate the application of credential-based access control regulating access to a server were presented by Winslett et al. [25, 29]. In these proposals access control rules are expressed in a logic language, and rules applicable to an access request are communicated by the server to clients. A first attempt to provide a uniform framework for credential-based access control specification and enforcement was presented by Bonatti and Samarati [10], where, accordingly to previous proposals, access rules are specified as logical rules. The paper envisions a system composed of two entities: a *client* and a *server*, interacting through a predefined negotiation process. The server is characterized by a set of resources. Both the client and the server have a *portfolio*, which is a collection of credentials (i.e., statements issued by authorities trusted for making them) and declarations (statements issued by the party itself). Credentials correspond to digital certificates and are guaranteed to be unforgeable and verifiable through the public key of the issuing authority. The communication of requisites that a client has to satisfy to gain access is based on a filtering and renaming process applied to server's policies. The filtering process allows the server to communicate to the client the requisites for an access, without disclosing possible sensitive information on which the access decision is taken. The proposal also allows clients to control the release of their credentials, possibly making counter-requests to the server, and releasing certain credentials only if their counter-requests are satisfied. In [23, 27, 28, 33, 34, 36], the authors introduce different trust negotiation strategies that a party can apply to select the credentials that need to be submitted to the counterpart during a service request evaluation. PeerTrust [17] is a language for expressing access control policies and for trust negotiation. Trust is established gradually by disclosing certificates and requests for certificates. PROTUNE (PROvisional TrUst NEgotiation) [9] is a policy specification language that provides a powerful declarative metalanguage for driving critical negotiation decisions such as the specification of what certificates are needed to gain an access, and where certificates can be retrieved. Ardagna et al. [4] present a privacy-enhanced authorization model and language to support the definition and enforcement of access restrictions based on properties associated with subjects and objects. They also bring forward the idea of exploiting the Semantic Web to allow the definition of access control rules based on generic assertions that are expressed by exploiting the concepts defined in the ontologies that control metadata content. These rules are then enforced on resources annotated with metadata regulated by the same ontologies. Early works on privacy address the problem of designing a privacy-oriented information system, called DORIS, supporting privacy as the individual's right of informational self-determination [7, 8, 11].

Although these works allow the definition of powerful and expressive access control models and languages, they do not regulate the use of personal information in secondary applications, which is the main focus of our work. P3P (Platform for Privacy Preferences) [31] is a XML-based language that addresses the need of a user to assess whether the privacy practices adopted by a service provider comply with her privacy preferences. In short, a service provider can define a P3P policy containing the specification of data recipients, data required, consequences of data release, purposes of data collection, data retention policy, and dispute resolution mechanisms. Users specify their privacy preferences through a policy language, called APPEL [30], and enforce privacy protection by means of a user agent. The user agent compares the privacy preferences of a user with the P3P policies of a service provider and verifies whether the P3P policies conform to the user privacy preferences. Agrawal et al. [1] propose a language, called XPref, for expressing user preferences. APPEL and XPref are however not sufficiently expressive because, for example, they do not support negotiation and contextual information, and they do not allow the definition of attribute-based conditions. P3P is therefore a good starting point but it has some shortcomings that our work tries to address. First, users can only accept or deny the privacy practices defined by a service provider. We believe that a better way to enforce the privacy practices is to offer users a richer, more active role in establishing how their personal information should be used. Our approach is therefore based on the customization of policy templates that allows a user to be actively involved in policies instantiation (see Section 4.2). Second, P3P is based on a fixed vocabulary and does not support the definition of restrictions based on generic properties characterizing the parties of a transaction. By contrast, our solution allows the definition of

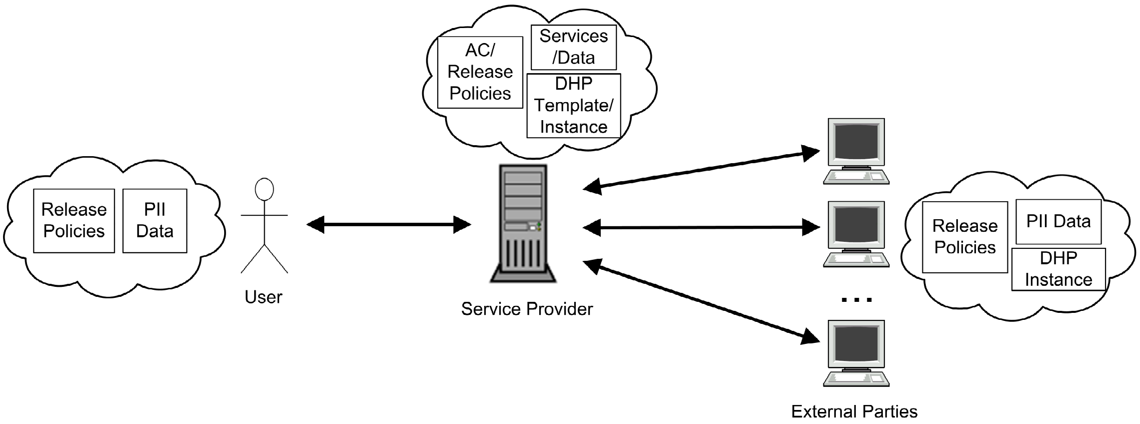

Figure 1: Reference scenario

generic conditions that can be integrated with obligations to define actions to be performed after an access has been granted (e.g., notification, deletion). Third, P3P does not provide protection against chains of releases. Our solution instead allows the definition of policies that are associated with the data at which they refer [19].

3 Scenario and basic concepts

Our reference scenario is a distributed infrastructure that includes three parties (see Figure 1): *i) users* are human entities that request on-line services; *ii) service provider* is the entity that provides on-line services to the users and collects personal information before granting an access to its services; *iii) external parties* are entities (e.g., business partners) to which the service provider may want to share or trade personal information of users. We assume that the functionalities offered by a service provider are defined by a set of objects/services. Each object/service is associated with *metadata* that provide additional contextual information explaining, for example, the owner of the object/service, the creation date, and so on. Metadata are therefore not part of the object/service content. As we will see, metadata can be exploited to define powerful restrictions. We also assume that the data recipients (i.e., both the service provider and external parties) are trusted, meaning that they manage the personal information of users according to the privacy preferences of the users. When a user needs to access a service, she is required to complete a registration process. Registered users are characterized by a unique *user identifier* (user id, for short). When registration is not mandatory, non-registered users are characterized by a *persistent user identifier* (pseudonym) that is generated by the service provider. The pseudonym is automatically sent by the client (e.g., a web browser) to the service provider whenever the user submits a request. In this case, personal information is stored under pseudonyms and not users' real identities.

Since we are considering open environments where the access decision is often based on properties of the requester rather than its specific identity, we assume that each party has a *portfolio* of *declarations* and *credentials* issued and certified by authorities trusted for releasing such credentials. This introduces the distinction between two types of attributes [10]:

- *certified attributes* are specified in a credential that is characterized by a *name*, the *issuer*'s public key, the *subject*'s public key, a *validity period*, a list (possibly empty) of pairs ⟨*attribute_name*, *attribute_value*⟩ representing the certified party's attributes (e.g., name and surname contained in an electronic passport), and a *digital signature*;
- *declared attributes* represent statements produced by a party, with no certification from any legal

authority. A declared attribute is a pair $\langle$*attribute_name*, *attribute_value*$\rangle$ corresponding to the party's attribute (e.g., the professional status communicated by a user during the registration process).

The set of credentials and declarations released by a party to the service provider are stored in a *profile* associated with the party. To define restrictions or to identify a party based on its attributes, we introduce the concepts of *credential term* and *declaration term.* Let $\mathcal{C}$ be a set of credential names and $\mathcal{P}$ be a set of predicates including standard binary built-in mathematical predicates (e.g., `equal`, `notEqual`, `greaterThan`). We define a credential term as follows.

Definition 3.1 (Credential term) Given a credential name `cred_name` $\in \mathcal{C}$, a *credential term* over `cred_name` is an expression of the form `cred_name`(*condition_list*), where *condition_list* is a list of predicates `math-pred`(*attribute_name*,*value*) with `math-pred` $\in \mathcal{P}$ a mathematical predicate, *attribute_name* is the name of an attribute that appears in credential `cred_name`, and *value* is the corresponding attribute's value.

The *condition_list* permits to define a list of conditions that are treated as if ANDed and that allow to define restrictions on a single credential without introducing variables in the language. We then define a binary predicate `credential(`ct,K`)`, where ct is a credential term `cred_name`(*condition_list*), and K is the public key or the name of a trusted authority. Predicate `credential` is evaluated to true if and only if there exists a credential `cred_name` issued by authority K and such that *condition_list* is satisfied.

Example 3.1 *Credential term* `passport`(`equal`(`nationality`*,* 'Italian')) *denotes a* `passport` *credential whose attribute* `nationality` *has value* Italian*. Predicate* `credential`(`passport`(`equal`(`nationality`*,* 'Italian'))*,* K_1) *is then evaluated to true if there exists the* `passport` *credential issued by* K_1 *certifying that the nationality of the credential's subject is* Italian*.*

A declaration term is defined as follow.

Definition 3.2 (Declaration term) A *declaration term* is an expression of the form `predicate_name`(*arguments*), where `predicate_name` $\in \mathcal{P}$ is the name of a generic predicate, and *arguments* is a list, possible empty, of constants or attributes.

We define a unary predicate `declaration(`d`)`, where d is a declaration term `predicate_name`(*arguments*). Predicate `declaration` is evaluated to true if and only if there exists a set of attributes such that `predicate_name`(*arguments*) is satisfied.

Example 3.2 *Declaration term* `equal`(`name`*,* 'Bob') *denotes the* `name` *attribute whose value is* Bob*. Predicate* `declaration`(`equal`(`name`*,*'Bob')) *is then evaluated to trus if there exists attribute* `name` *with value* Bob*.*

Declarations and credentials in a portfolio may be organized into a partial order. For instance, an `identity-document` can be seen as an abstraction for credentials `driver-license`, `passport`, and `identity-card`. Finally, the functionalities offered by a server are defined by a set of services. Intuitively, each service can be seen as an application that clients can execute. Abstractions can also be defined within the domains of users as well as services/objects. Intuitively, abstractions allow to group together users (objects, resp.) with common characteristics and to refer to the whole group with a name.

4 Privacy-aware access control policies

Before introducing our privacy-aware policies, it is important to highlight some desiderata that the privacy-aware policies for the reference scenario should satisfy and that guided our work.

- *Expressivity.* A privacy-aware policy should be expressive enough so that the policy can suit all the privacy needs. It should then be based on generic properties (attributes) associated with subjects (e.g., name, address, occupation) and objects (e.g., owner, creation date). These attributes should be substantiated by certificates (if needed) issued and signed by trusted authorities.
- *Simplicity.* One of the major challenges in the definition of a privacy-aware policy language is to provide expressiveness and flexibility while at the same time ensuring easiness of use and therefore applicability. A privacy-aware policy should then be based on a high level formulation of the rules, possibly close to natural language formulation.
- *Context-based conditions.* A privacy-aware policy should support the definition of generic conditions based on context information, including location information [3], to allow environment factors to influence how and when the policy is enforced. Generally speaking, context information is a set of metadata identifying and possibly describing entities of interest, such as subjects and objects, as well as any ambient parameters concerning the technological and cultural environment (including location), where a transaction takes place.
- *Ontology integration.* Policy definition should be fully integrated with subject and object ontologies in defining access control restrictions. Also, privacy-aware policies should take advantages from the integration with credentials ontology that represents relationships among attributes and credentials.
- *Client-side restrictions.* In addition to traditional server-side access control rules, users should be able to specify restrictions on how the released information can be used by their remote counterpart.
- *Interactive enforcement.* The server may not have all the information it needs to decide whether or not an access should be granted. On the other side, however, the requester may not know which information she needs to present to a (possibly just encountered) server to get access. All this requires a new way of enforcing the access control process, which cannot be assumed anymore to operate with a given prior knowledge and return a "yes/no" access decision. Therefore, the the access control process should provide a way of interactively applying criteria to retrieve additional information. Additional information may be either sufficient or simply necessary for the counterpart to eventually have its request satisfied.

To address these requirements, we introduce three different types of privacy-aware policies: *access control*, *release*, and *data handling policies*. In the following, we describe these policies, focusing however on data handling policies that are not addressed by previous proposals.

4.1 Access control and release policies

Access control policies define authorization rules concerning access to data/services [24]. Authorizations correspond to traditional (positive) rules usually enforced in access control systems. For instance, an authorization rule can require the proof of majority age and a credit card number (condition) to read (action) a specific set of data (object). When an access request is submitted to a service provider, it is evaluated against the authorization rules applicable to it. If the conditions for the required access are evaluated to true, access is permitted. If none of the specified conditions that might grant the requested access can be fulfilled, then the access is denied. Finally, if the current information is insufficient to determine whether the access request can be granted or denied, additional information is needed and the requester receives an undefined response with a list of requests that she must fulfill to gain the access. For instance, if some of the specified conditions can be fulfilled by signing an agreement, then the party prompts the requester with the actions that would result in the required access.

Release policies define the party's preferences regarding the release of its Personal Identifiable Information (PII) by specifying to which party, for which purpose/action, and under which conditions a particular set of PII can be released [10]. For instance, a release policy can state that the credit card information can be released only in the process of a `purchase` action and upon presentation of a nondisclosure agreement (condition) by the party. The release of PII may only be performed if the release policies are satisfied.

As an example of the combined use of access control and release policies, suppose that `Alice` wants to access a service through the `ACME` web site. The access control policy of `ACME` requires users to release their credit card and contact information to access such a service. Since the release policy of `Alice` states that the contact information can be released to other parties without any constraints while credit card information can be released only to parties that are members of the Better Business Bureau (BBB), the on-line transaction can be completed only if `ACME` is able to prove that is a member of BBB.

4.1.1 Basic elements of the language

We are now ready to describe the basic constructs of the language used to define the access control and release policies.[1]

Predicates `declaration` and `credential` previously introduced to distinguish between conditions on data declarations and conditions on credentials (we will elaborate more on this in the following) together with the set $\mathcal{P}$ of predefined predicates constitute the basic literals that can be used in access control and release policies. Attributes associate with users and objects/services are referenced through the usual dot notation. For instance, `Alice.Address` indicates the address of user `Alice`. Here, `Alice` is the pseudonym of the user (and therefore the identifier for the corresponding profile), and `Address` is the name of the property. Also, to refer to the requester (i.e., the subject) and the target (i.e., the object) of the request being evaluated without the need of introducing variables in the language, we use keywords **user** and **object**, respectively, whose appearances in a conditional expression are intended to be substituted with actual request parameters during run-time evaluation of the access control policy.

We have then identified three main basic elements of the language: *subject_expression*, *object_expression*, and *conditions*.

Subject expression. These expressions allow the reference to a set of subjects depending on whether they satisfy given conditions that can be evaluated on the subject's profile. More precisely, a subject expression is a boolean formula of `credential` and `declaration` predicates. The predicates specified as arguments of the `declaration` and `credential` predicates can be: *i)* the standard built-in mathematical predicates, *ii)* location-based predicates, and *iii)* the non predefined predicates that evaluate information stored at the service provider. The following are examples of subject expressions.

- `credential(passport(equal(`**user**`.job,`'professor'`),greater_than(`**user**`.age,`35`)),`K_1`)` denotes users with age greater than 35 who are professors. These properties should be certified by showing the `passport` credential verifiable with public key K_1.
- `declaration(equal(`**user**`.name,`'Bob'`))` denotes users whose name is Bob.

Object expression. These expressions allow the reference to a set of objects depending on whether they satisfy given conditions that can be evaluated on the object's profile. More precisely, *an object expression is a boolean formula of terms of the form* `predicate_name(`*arguments*`)`, where `predicate_name` $\in \mathcal{P}$ and *arguments* is a list, possible empty, of constants or attributes. The predicates specified in an object expression can be: *i)* the standard built-in mathematical predicates, and *ii)* the non predefined predicates that evaluate information stored at the server. The type of predicates that can be specified in the object expression field can be extended by adding, for example, location-based predicates. The following are examples of object expressions.

- `equal(`**object**`.owner,`**user**`)` denotes all objects created by the requester;
- `lesserThan(`**object**`.creation_date,`1971`)` denotes all objects created before 1971.

[1] Note that although semantically different, access control and release policies are syntactically identical.

Conditions. We assume that the type of conditions that can be specified in the *conditions* element are only conditions that can be brought to satisfactions at run-time processing of the request. These conditions can be related to agreement acceptance, payment fulfillment, or registration. Conditions can be associated with data at different levels (i.e., attribute, credentials' attributes and credentials) and can be certified or uncertified. More precisely, *conditions are boolean formula of terms of the form* `predicate_name(`*arguments*`)`, where `predicate_name` $\in \mathcal{P}$ and *arguments* is a list, possible empty, of constants or attributes. Note that the predicates specified in the *conditions* element can be: *i)* trusted-based conditions stating that, for example, the requester should use a trusted platform, *ii)* the standard built-in mathematic predicates, *iii)* location-based conditions defining restrictions based on location information [3]; and *iv)* the non predefined predicates that evaluate information stored at the server. An example of condition is `fill_in_form(`*form1*`)` that checks if the form *form1* has been filled.

4.1.2 Policy and rule definition

Syntactically, access control and release policies are composed by a set of authorization rules defined as follow.

Definition 4.1 (Access control/Release rule) *An access control rule (release rule, resp.) is an expression of the form* ⟨subject⟩ [WITH ⟨subject_expression⟩] CAN ⟨actions⟩ ON ⟨object⟩ [WITH ⟨object_expression⟩] FOR ⟨purposes⟩ [IF ⟨conditions⟩]*, where:*

- subject *identifies the subject to which the rule refers and corresponds to a user identifier or a named abstraction, if abstractions are defined on subjects;*
- subject-expression *is a boolean formula of* `declaration` *and/or* `credential` *predicates that refers to a set of subjects depending on whether they satisfy given conditions that can be evaluated on the user's profile;*
- actions *is the set of actions to which the rule refers (e.g., read, write, and so on);*
- object *identifies the object to which the rule refers and corresponds to an object identifier or a named abstraction, if abstractions are defined on objects;*
- object-expression *is a boolean formula of mathematical predicates that allows the reference to a set of objects depending on whether they satisfy given conditions that can be evaluated on the object's metadata;*
- purposes *is the purpose (e.g.,* `scientific`*) or group thereof to which the rule refers and represents how the data are going to be used by the recipient;*
- conditions *is a boolean formula of generic predicates stating generic conditions that must be satisfied by the access request.*

The release and access control rules have been implemented in XML and comply with the XML Schema illustrated in Appendix A. In the following, for the sake of clarity and conciseness, access control and release policies will be expressed in accordance to Definition 4.1.

Example 4.1 *Suppose that the* `ACME` *company stores credit card information* (`cc_info`) *to provide its services. Table 1 illustrates an access control policy composed of three rules that regulates the access to the* `cc_info` *data. Here, the term '*any*' in the subject field denotes any user of the system.*

Although the definition of access control and release policies permits to protect the access of data and services and the release of personal data, respectively, no solution is provided for regulating how PII must be used and processed after their release. To this purpose, we introduce the data handling policies.

	AC Rules	Description
AC1	any WITH credential(employeeCard(equal(**user**.job,'Director')),ACME) AND declaration(equal(**user**.company,'ACME')) CAN read ON cc_info WITH greaterThan(**object**.expiration,*today*) FOR {marketing,service_release} IF {in_area(**user**.sim,'ACME') AND log_access()}	ACME's directors are authorized to read valid (i.e., not yet expired) cc_info for marketing and service release purposes, if they are located inside the ACME building and the access is logged.
AC2	any WITH credential(employeeCard(equal(**user**.job,'Seller'),equal(**user**.jobLevel,'A')),ACME) AND declaration(equal(**user**.company,'ACME')) CAN read ON cc_info WITH greaterThan(**object**.expiration,*today*) FOR service_release IF log_access()	Sellers of level A of ACME are authorized to read valid cc_info for service release purpose if the access is logged.
AC3	any WITH credential(employeeCard(equal(**user**.job,'BusinessConsultant')),ACME) CAN read ON cc_info WITH greaterThan(**object**.expiration,*today*) FOR reimbursement	ACME's business consultants are authorized to read valid cc_info for reimbursement purpose.

Table 1: An example of a access control policy that protects the cc_info data stored at ACME

4.2 Data handling policies

The development of privacy policies reflecting the privacy preferences expressed by the users for managing their information is a complex task. A viable solution should be simple and expressive enough to support the privacy requirements that include individual control, consent, modifiability, and data security [21].

These requirements imply that personal information collected for one purpose may not be used for any other purpose without the specific informed consent of the user it concerns. A *data handling policy* (DHP, for short) provides the users with the possibility to define how their PII can be used by the service provider and/or external parties. A data handling policy physically follows the data when they are released to an external party, thus building a chain of control coming from the data owner.

In the data handling policy specification, two issues need to be discussed: *by whom* and *how* a policy is defined. With respect to the first issue, different strategies are possible, each one requiring different levels of negotiation between a user and a service provider. We consider the following three strategies.

- *Server-side.* A service provider defines its data handling policies and a user can accept or reject them according to her privacy preferences. This "all or nothing" approach to secondary use is similar to what happen in a P3P-like scenario and therefore has the same drawbacks.

- *User-side.* A user defines its data handling policies and a service provider can accept or reject them according to its privacy preferences. This approach could not be applicable in a real scenario since it is unlikely that service providers are willing to accept whatever privacy preference policy from users.

- *Customized.* When a user requires a service, predefined policy templates are provided by the service provider as a starting point for creating data handling policies. The templates are customized to meet different privacy requirements. A user can directly customize the templates or it can be supported by a customization process that automatically applies some user privacy preferences. If the customized data handling policies will be accepted by the service provider, the personal information provided by the user will be labeled with the customized data handling policies. This represents the most flexible and balanced strategy for the definition of data handling policies.

In summary, server-side and user-side are the opposite endpoints of all possible approaches in the definition of privacy rules that balance between service providers and users needs. Since the customized approach represents a good trade-off between the power given to the service providers and the protection assured to the users, we adopt it in the following.

With respect to the second issue (i.e., how a DHP is defined), we first need to consider how data handling policies relate to traditional access control policies. Each access request has to be first checked against the access control policies of the service provider. If an access request satisfies at least one access control policy, then the service provider has to verify the data handling policies possibly attached to the requested information. In particular, there may exist one or more data handling policies and each of them can impose different restrictions on how such data can be used in secondary applications. Syntactically, access control policies and data handling policies are similar, since a data handling policy regulates which *subject* can execute which *actions* on which *resources* and under which *conditions*. Similarities and relations with access control policies bring to consider two approaches in the definition of a data handling policy.

- *Integrated policies*. Traditional access control policies can be extended by adding a *DHP component* that should specify additional conditions (e.g., research purpose only, no disclosure, delete after 10 days, and so on) regulating under which circumstances the personal information disclosed by the service provider can be used. This approach has the main disadvantage to make the overall policy less clear and requires some specific mechanisms to transfer DHP rules together with data.
- *Stand-alone policies*. Data handling policies can be defined as independent rules. Therefore, a data handling policy should represent the user's privacy preferences and should then include different components that allow to define how the external parties can use personal data. Personal data are then *tagged* with such data handling policies.

Although the stand-alone option can introduce some redundancy in policy definition, it provides a good separation between policies that are used with two different purposes. This clear separation makes data handling policies more intuitive and user-friendly, and implicitly suggests the differences with access control policies. In addition, the definition of stand-alone policies reduces the risks of unprotected data types and, finally, it allows the customization of additional components such as recipients, actions, and so on. In the following, we assume a customized stand-alone approach for the data handling policies specification.

5 Data handling policy language

We start by describing the basic elements of a data handling policy and then we illustrate the syntax of the language together with some examples.

5.1 Basic elements of the language

The basic elements of a data handling policy are: *recipients*, *purposes*, *PII abstraction*, and *restrictions*.

Recipients. A recipient is the external party to which PII can be disclosed [16]. Since external parties may be unknown to the user, she should define to which entities her data may be disclosed without knowing their identity. Our approach supports the definition of recipients according to three options: *identity-based*, *category-based*, and *attribute-based*. Identity-based means that the external parties may be identified by their unique identity. Category-based means that the external parties are grouped into different categories, which represent recipients of different domains and with different access rights on personal data. Categories can be hierarchically organized and within the hierarchy, a category may inherit from its ancestors all permissions. A critical aspect for scalability is represented by the ability of defining an external party based on its *attributes*, instead of its identity. Attribute-based means that the recipient is defined through conditions that it has to satisfy. Conditions (as discussed in Section 3) are evaluated on declarations and credentials of the recipient.

Purposes. The term *purposes* is used to denote those purposes for which the information can be used. Abstractions can be defined within the domain of purposes, so as to refer to purposes showing common characteristics and to refer to a whole group with a name. Abstractions can therefore correspond to generalization/specialization relationships. For instance, `pure research` and `applied research` can be seen as a specialization of `research`.

PII abstraction. *Data types* can be introduced as abstractions of PII to let data handling policies being expressed in terms of data types, rather than single properties of the user only. Data types can be organized hierarchically. For instance, the `cc_info` can be seen as an abstraction for the credit card information, which can include the `number` and `expiration` attributes; the `personal_info` can be seen as an abstraction for the personal information, which can include the `name` and `SSN` attributes; and the `contact_info` can be seen as an abstraction for the contact information that includes the `address` and `telephone` attributes.

Restrictions. A privacy statement specifies restrictions that have to be satisfied before or after access to personal data is granted. If just one condition is not satisfied, the access should not be granted. We distinguish between the following three types of conditions.

- *Provision* represents actions that have to be performed before an access can be granted [6]. For instance, a data handling policy can state that a business partner can read the email address of the users provided that it has *paid a subscription fee*.
- *Obligation* represents actions that have to be either performed after an access has been granted [6] or actions that need to be performed in the future, based on the occurrence of well defined events (e.g., time-based or context-based events [12]). For instance, a data handling policy can state that users will be notified whenever their personal information is disclosed. Another policy can impose a restriction on how long personal data should be retained (data retention).
- *Generic* conditions either evaluate properties of users' profiles, like membership of requester, or represent conditions that can be brought to satisfaction at run-time when the request is processed. Generic conditions may include, for example, conditions on the granularity of data disclosure (e.g., data would only disclosed in aggregate form), on the time frame for possible data disclosure, and so on.

5.2 Data handling policy syntax

A data handling policy has the form "⟨*PII*⟩ MANAGEDBY ⟨*DHP_rules*⟩", where *PII* identifies a PII abstraction and *DHP_rules* identifies one or more rules, composed in OR logic, regulating the access to the PII data to which they refer. Note that in a data handling policy template the *PII* element represents the name of an attribute or the name of a data type; it represents an attribute belonging to a privacy profile, in case of a customized data handling policy. Formally, a DHP rule can be defined as follow.

Definition 5.1 (Data handling rule) *A* DHP rule *is an expression of the form* ⟨*recipients*⟩ CAN ⟨*actions*⟩ FOR ⟨*purposes*⟩ [IF ⟨*gen_conditions*⟩] [PROVIDED ⟨*prov*⟩] [FOLLOW ⟨*obl*⟩]*, where:*

- recipients *can be an identifier, a category, or a boolean formula of the* `credential` *and/or* `declaration` *predicates;*
- actions *is the set of actions;*
- purposes *is the purpose or a group thereof;*
- provisions, obligations*, and* generic *conditions are optional and are boolean expressions of terms of form* `predicate_name(`arguments`)`*, where* `predicate_name` $\in \mathcal{P}$*, and* arguments *is a list, possibly empty, of constants or variables on which predicate* `predicate_name` *is evaluated.*

A data handling rule specifies that *recipients* can execute *actions* on *PII* for *purposes* provided that *prov* is satisfied, *gen_conditions* are satisfied, and with obligations *obl*. From the evaluation point of view generic conditions, provisions, and obligations are different: generic conditions are conditions that an access request has to satisfy before the access is granted; provisions are preconditions that need to be evaluated as pre-requisites before a decision can be taken; obligations are additional steps that must be taken in account after the policy evaluation. For instance, `in_area(user,'New York')`, is a generic predicate requiring **user** to be located within the metropolitan area of New York; `fill_in_form()` and `log_access()` are provision predicates that require to fill in a form and to login the access, respectively; and `notify(`*user*`)` and `delete_after(`*num_days*`)` are obligation predicates that require to notify the *user* after releasing some data and to delete the data after a specific number of days, respectively.

Like for access control and release rules, the data handling rules have been implemented in XML and comply with the XML Schema illustrated in Appendix B. In the following, for the sake of clarity and conciseness, data handling policies will be expressed in accordance to Definition 5.1.

Example 5.1 *Suppose that* `ACME` *provides the* rent-a-car, book-a-flight, *and* flight+hotel *services. Table 2 illustrates an example of customized data handling policies that regulate the secondary use of the personal information of* `Alice` *stored by the* `ACME` *company. In particular,* DHP1 *is composed by three rules that are associated with and protect the* `name` *and the* `contact_info` *of* `Alice`*;* DHP2 *is composed by a single rule that protects the* `cc_info` *of* `Alice`*;* DHP3 *is composed by two rules that protect the* `personal_info` *of* `Alice`.

6 Interplay between parties

The reference scenario in Figure 1 is aimed at supporting two different interactions between the parties (see Figure 2): *User-Service Provider interplay*, when a user submits an access request for a resource managed by the service provider; *External Party-Service Provider interplay*, when an external party submits an access request for personal information of a user stored by the service provider. The access request submitted by a user or an external party can be defined as follow.

Definition 6.1 (Access request) *An* access request *is a 4-uple of the form* ⟨user_id, action, object, purposes⟩*, where* user_id *is the optional identifier/pseudonym of the requester,* action *is the action that is being requested,* object *is the object on which the requester wishes to perform the action, and* purposes *is the purpose or a group thereof for which the object is requested.*

For instance, the access request ⟨*Alice,execute,book_a_flight,service_access*⟩ states that `Alice` wants to *execute* the service *book_a_flight* for the purpose of accessing the requested service (*service_access*). The two types of interactions previously mentioned work as follows.

- *User-Service Provider Interplay.* The User-Service Provider interplay is depicted in Figure 2(a) (step 1-4). Upon the reception of a service request (step 1), the service provider evaluates its access control policies and, if needed, asks some PII to the user (*DataRequest*(PII_U) in step 2). In addition, the service provider presents one or more data handling policy templates to the user for customization (*DHPTemplate*(DHP_S) in step 2). The user receives the service provider request, evaluates her release policies and customizes the data handling policy templates. If the PII request satisfies at least one (user-side) release policy, the user sends back the required PII (PII_U) along with the customized data handling policies (DHP_S) (step 3). Otherwise, if the user's release policies deny the PII release, the transaction aborts. Finally, the service provider stores the user's data ⟨PII_U,DHP_S⟩ and re-evaluates the access control policies based on PII_U. If the evaluation succeeds, the service is granted to the user (step 4). Note that in general, both the release of PII and the data handling policy customization could require multiple negotiation steps [35], rather than a single request-response exchange. The user, for example, may require the service provider to release some PII as well, and, in turn, the service provider may want to specify data handling policies for such a PII.

	Data Handling Policies		
	PII	**DHP Rules**	**Description**
DHP1	Alice.name, Alice.contact_info	declaration(equal(user.type,'BusinessPartners')) CAN read FOR market PROVIDED pay_a_fee()	Business partners of ACME can read for market purpose the name and the contact_info of Alice provided that they have paid a fee.
		declaration(equal(user.type,'BusinessPartners')) CAN {read,write} FOR service_release FOLLOW delete_after(30 days)	Business partners of ACME can read and write for service release the name and the contact_info of Alice. The name and contact_info must then be deleted after thirty days.
		declaration(equal(user.type,'MarketAgencies')) AND credential(IMB_Cert(equal(user.speciality.category, 'computer')), IMB) CAN read FOR statistic	Market agencies specialized for distribution of computers and whose specialization has been certified by the *International Market Board* (IMB) authority can read the name and the contact_info of Alice for statistic purpose.
DHP2	Alice.cc_info	credential(employeeCard(equal(user.job,'Seller')), ACME) AND declaration(equal(user.company, 'ACME')) CAN read FOR service_release IF time(8:30am,6:00pm) PROVIDED log_access()	Sellers of ACME can read credit card information of Alice for service release purpose during the working hours (i.e., from 8:30 am to 6:00 pm) provided that the access is logged.
DHP3	Alice.personal_info	(declaration(equal(user.type,'BusinessPartners')) AND declaration(equal(user.country,'EU'))) OR (declaration(equal(user.type,'GovAuth'))) CAN read FOR research FOLLOW notify(Alice)	European business partners of organization ACME or government authorities can read the personal_info of Alice for research purpose with the obligation of notify Alice at each access.
		declaration(equal(user.type,'Administrative')) CAN read FOR market IF in_area(user, ACMEBuilding)	Administrative staff of ACME can read the personal information of Alice for market purpose only if they are in the building of ACME.

Table 2: An example of data handling policies that protect Alice's data stored by ACME

- *External Party-Service Provider Interplay.* External party-service provider interplay (see Figure 2(b)) is temporally subsequent to the interplay between the user and the service provider. Suppose that an external party asks the access to the personal information of a user, that is, PII_U, stored at the service provider ($\mathit{DataRequest}(\mathrm{PII}_U)$ in step 1). External party-service provider interplay begins with a mutual identification, followed, in the most general case, by a negotiation of the data to be released and of the corresponding data handling policies.[2] Differently to the user-service provider scenario, in this case the service provider must protect the privacy of the user, with respect to the external party, by enforcing the access control policies of the service provider and the data handling policies attached to the requested information. If the evaluation of access control and data handling policies returns a positive answer, the external party obtains the pair $\langle \mathrm{PII}_U, \mathrm{DHP}_S \rangle$ (step 2). Otherwise, the access is denied.

Example 6.1 Let Alice be a user that releases her PII information to the ACME company, together with the data handling policies showed in Table 2. Assume now that Bob, which is a seller of ACME, needs to read

[2] Note that for the sake of clarity Figure 2(b) does not show these steps that are similar to steps 2 and 3 of the user-service provider interplay in Figure 2(a).

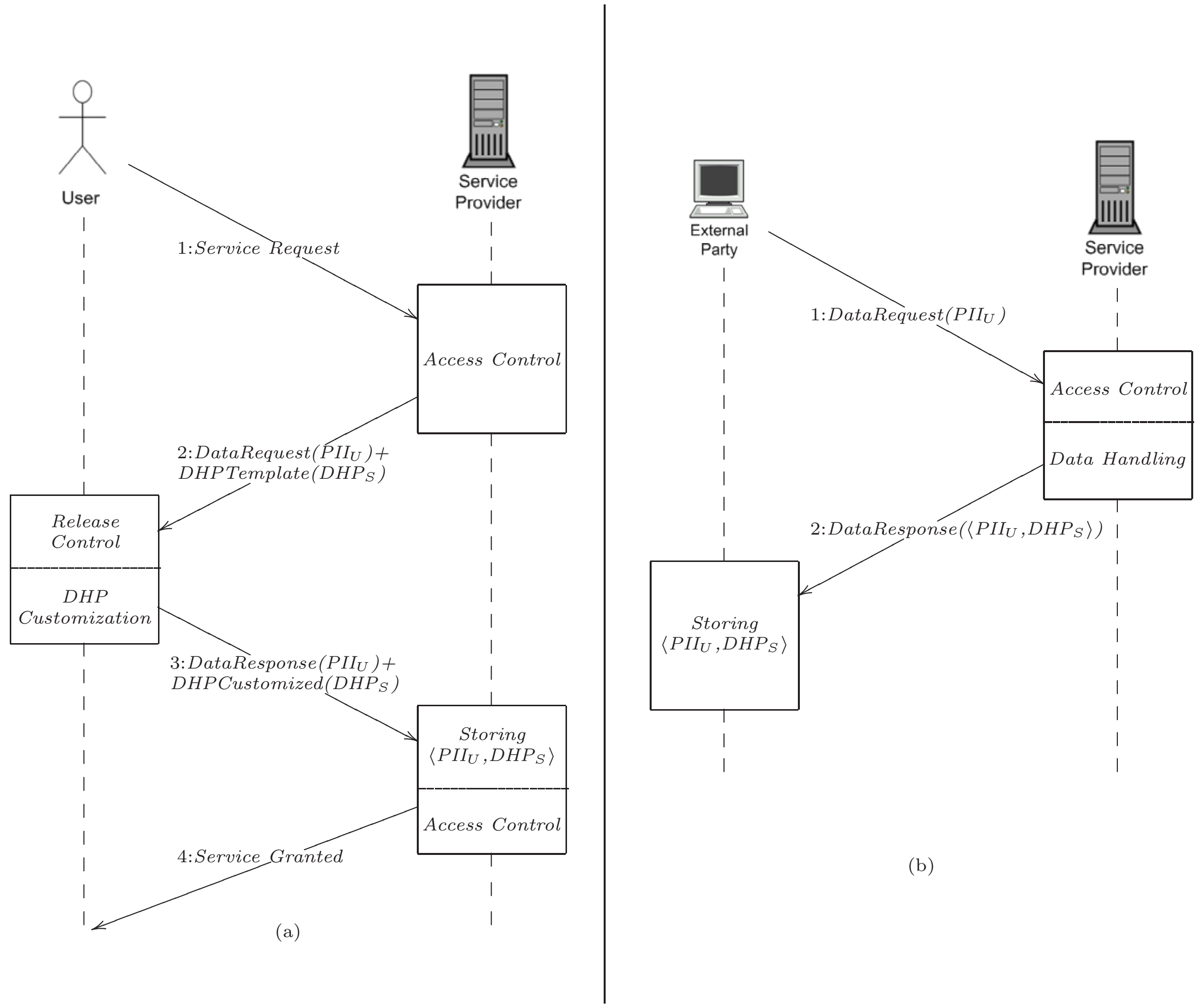

Figure 2: User-Service Provider interplay (a) and External Party-Service Provider interplay (b)

the `cc_info` of `Alice`. `Bob` sends a request of this form: $\langle \mathit{Bob}, \mathit{read}, \mathit{Alice.cc\_info}, \mathit{service\_release} \rangle$. Suppose also that the `ACME`'s access control policy corresponds to the policy in Table 1.

First, the set of applicable access control rules to `Bob`'s request is computed. AC1 and AC2 are selected because their actions, purposes, and object include the action, purposes, and object specified in the access request, respectively, and the object of the access request satisfies the object expression in the access control rules. The access control rules are then evaluated with respect to the `Bob`'s profile showed in Table 3. Since `Bob` is not a *Director*, AC1 evaluates to false, while AC2, which requires an ACME's seller of level A, is evaluated to true and therefore the evaluation process continues.

Then, the data handling policies in Table 2 are evaluated. DHP2 is the only policy that regulates the access to the `cc_info` of `Alice` and the corresponding rule applies to the access request since its actions and purposes include the action and purposes specified in the access request, respectively. DHP2's rule is satisfied by the data in the `Bob`'s profile (see Table 3), only if the request is sent between 8.30 am and 6.00 pm. Suppose that the request is sent at 1.00 pm. `Bob` satisfies this rule and receives the `cc_info` of `Alice` together with the data handling policies in Table 2.

Attribute	Value
Pseudonyms	ciks2ss4skk
First Name	Bob
Last Name	Smith
Age	35
Job	Seller
Company	ACME
JobLevel	A

Table 3: An example of the profile of Bob

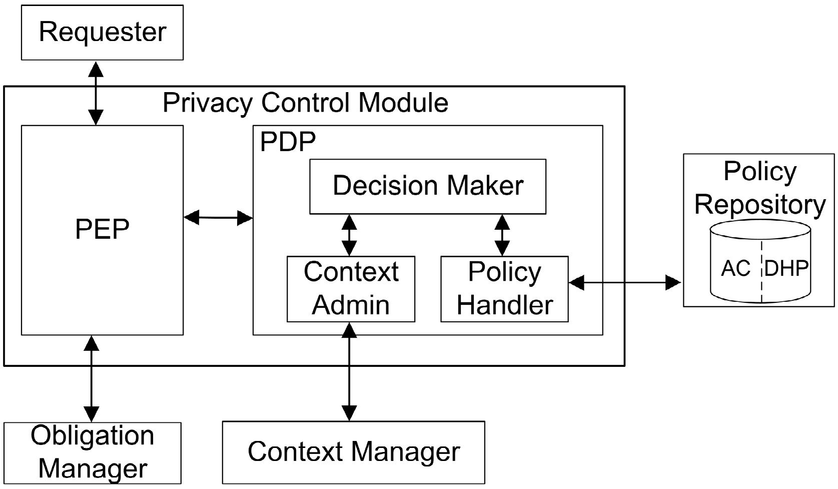

Figure 3: Privacy control module

7 A privacy-aware access control system prototype

The access control, release, and data handling policies previously described have been integrated within a Java-based *privacy control* module, which is a module of the architecture developed in the context of the European PRIME project. Figure 3 illustrates the privacy control module and its relationships with other modules of the PRIME architecture. We developed the *Policy Repository* that contains access control, release, and data handling policies, and its core technology is a relational database that provides functionalities for policy administration such as searching facilities, modify, insert, and delete operations. The *Context Manager* is the component that collects contextual information from various sources and provides the aggregate information in a standardized way. The Privacy Control Module operates on top of the Context Manager and contains two submodules: a *Policy Decision Point* (PDP) and a *Policy Enforcement Point* (PEP). We developed the PDP that is responsible for taking an access decision for all access requests directed to data/services. PDP retrieves and evaluates all policies (both access control, release, and data handling) applicable to a request and includes: a *Decision Maker* that produces the final response possibly combining several access decisions coming from the evaluation of different policies; a *Policy Handler* that is in charge of managing all communications with the *Policy Repository* to retrieve all policies applicable to an access request; a *Context Administrator* that manages access and communication with the *Context Manager* component, which contains the information associated with the requester during a session, and the ontology engine to reason about policy conditions and to find out how to evaluate the policies [15]. The PEP submodule is responsible for enforcing access control decisions by intercepting accesses to resources and granting them, only if they are part of an operation for which a positive decision has been taken. The PEP also interacts with the *Obligation Manager* [12], a component responsible for enforcing obligation conditions specified inside data handling policies.

7.1 Access request enforcement

Given an access request (see Definition 6.1) where the object can be a service or some PII associated with a user, the access request is first received by the PEP submodule and then is evaluated by the PDP submodule in two steps as follows.

Step 1. The access request is evaluated against the applicable access control rules. Note that, if no rule is selected, the access is denied meaning that the default access decision is "no" (*deny-all*). After collecting applicable rules, all conditions are inserted in a Reverse Polish Notation (RPN) stack that is used to increase the evaluation performances. Then, each rule is evaluated. Our implementation is based on an access control policy specified in a disjunctive normal form (DNF), meaning that the rules are ORed and conditions inside the rules are ANDed. At the end of this evaluation step, the system reaches a "yes", "no", or "undefined" access decision. In case of a positive (yes) access decision, the system has to verify whether there exists some restrictions on the secondary use of the requested target (see Step 2). In case of a negative (no) access decision, the access request is denied, and the process terminates. An undefined access decision means that the information submitted by the requester is insufficient to determine whether the request can be granted or denied. Additional information is required by communicating filtered queries to the requester. Such requests are called *claim requests*. Since a claim request can contain sensitive information, a sanitization process is performed before the claim request is sent to the counterpart, which then avoid the release of sensitive information related to the policy itself. For instance, suppose that the rule which has been neither satisfied nor denied yet contains the condition `equal(creditCardCircuit,'VISA')`. The sanitization process can decide to return to the user either the condition as it is or a modified filtered query simply requesting the user to declare its credit card circuit (then protecting the information that access is restricted to VISA credit card). From the implementation viewpoint, a claim request is a Java class that has to be serialized before its release to the involved parties.

Step 2. The PDP submodule sends a request to the Policy Repository to retrieve all applicable data handling rules. This selection is performed by using the action, object, and purposes specified in the access request as keys. For each applicable data handling rule, the system evaluates the conditions specified in the *recipients*, *gen_conditions*, and *prov* fields. Again, a RPN stack is used for the evaluation of the rules. At the end of this evaluation step, the system reaches a "yes", "no", or "undefined" access decision, which is managed as described in Step 1. In particular, in case of a positive (yes) access decision, the requested data/service is released to the counterpart together with the corresponding data handling policies. The requester is then responsible for managing the received data/service following the attached data handling policies.

8 Conclusions and future work

The vast amount of personal information available in today's global infrastructure has lead to growing concerns about the privacy of the users. This paper presents a first step towards the definition of a comprehensive approach to privacy-aware access control. We introduced different types of privacy-aware policies that allow the definition and enforcement of powerful and flexible access restrictions based on generic properties associated with subjects and objects. In particular, we focused on data handling policies, which are policies aimed at regulating the use of personal information in secondary applications. We then described a privacy-aware access control module that integrates access control and data handling policies evaluation and enforcement. Several issues are still open to further investigation: the definition of a solution regulating the secondary use of data by considering their combination with PII released in previous transactions; the definition of enhanced techniques for the dissemination of data handling policies among parties; the composition of data handling policies defined at different levels of PII abstraction; and the definition of a global data handling policy life-cycle management process.

Acknowledgements

This work was partially supported by the European Union within the 6FP project PRIME under contract IST-2002-507591, by the European Union within the 7FP project "PrimeLife", and by the Italian MIUR within PRIN 2006 under project 2006099978.

References

[1] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. An xpath based preference language for P3P. In *Proc. of the 12th International World Wide Web Conference*, Budapest, Hungary, May 2003.

[2] G.-J. Ahn and J. Lam. Managing privacy preferences in federated identity management. In *Proc. of the ACM Workshop on Digital Identity Management*, Fairfax, VA, USA, November 2005.

[3] C.A. Ardagna, M. Cremonini, E. Damiani, S. De Capitani di Vimercati, and P. Samarati. Supporting location-based conditions in access control policies. In *Proc. of the ACM Symposium on Information, Computer and Communications Security (ASIACCS'06)*, Taipei, Taiwan, March 2006.

[4] C.A. Ardagna, E. Damiani, S. De Capitani di Vimercati, and P. Samarati. Towards privacy-enhanced authorization policies and languages. In *Proc. of the 19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, Storrs, CA, USA, August 2005.

[5] C.A. Ardagna, S. De Capitani di Vimercati, and P. Samarati. Enhancing user privacy through data handling policies. In *Proc. of the 20th Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, Sophia Antipolis, France, July/August 2006.

[6] C. Bettini, S. Jajodia, X. Sean Wang, and D. Wijesekera. Provisions and obligations in policy management and security applications. In *Proc. of the 28th VLDB Conference*, Hong Kong, China, August 2002.

[7] J. Biskup and H.H. Brüggemann. The personal model of data: Towards a privacy oriented information system. *Computers & Security*, 7:575–597, 1988.

[8] J. Biskup and H.H. Brüggemann. The personal model of data - towards a privacy oriented information system. In *Proc. of the Fifth International Conference on Data Engineering (ICDE 1989)*, pages 348–355, Los Angeles, California, USA, February 1989.

[9] P.A. Bonatti and D. Olmedilla. Driving and monitoring provisional trust negotiation with metapolicies. In *Proc. of the IEEE 6th International Workshop on Policies for Distributed Systems and Networks (POLICY 2005)*, Stockholm, Sweden, June 2005.

[10] P.A. Bonatti and P. Samarati. A unified framework for regulating access and information release on the web. *Journal of Computer Security*, 10(3):241–272, 2002.

[11] H.H. Brüggemann. Interaction of authorities and acquaintances in the DORIS privacy model of data. In *Proc. of the 2nd Symposium on Mathematical Fundamentals of Database Systems (MFDBS 1989)*, Visegrad, Hungary, June 1989.

[12] M. Casassa Mont and F. Beato. On parametric obligation policies: Enabling privacy-aware information lifecycle management in enterprises. In *Proc. of the 8th IEEE Workshop on Policies for Distributed Systems and Networks (Policy 2007)*, Bologna, Italy, June 2007.

[13] R. Chandramouli. Privacy protection of enterprise information through inference analysis. In *IEEE 6th International Workshop on Policies for Distributed Systems and Networks (POLICY 2005)*, Stockholm, Sweden, June 2005.

[14] L.F. Cranor. *Web Privacy with P3P.* O'Reilly & Associates, 2002.

[15] E. Damiani, S. De Capitani di Vimercati, C. Fugazza, and P. Samarati. Extending policy languages to the semantic web. In *Proc. of the International Conference on Web Engineering*, Munich, Germany, July 2004.

[16] Directive 95/46/ec of the european parliament and of the council of 24 october 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data. *Official Journal L 281*, pages 031–050, 23/11/1995.

[17] R. Gavriloaie, W. Nejdl, D. Olmedilla, K. Seamons, and M. Winslett. No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web. In *Proc. of the 1st First European Semantic Web Symposium*, Heraklion, Greece, May 2004.

[18] G. Karjoth and M. Schunter. Privacy policy model for enterprises. In *Proc. of the 15th IEEE Computer Security Foundations Workshop*, Cape Breton, Nova Scotia, Canada, June 2002.

[19] G. Karjoth, M. Schunter, and M. Waidner. Privacy-enabled services for enterprises. In *Proc. of the 13th International Conference on Database and Expert Systems Applications (DEXA'02)*, Aix-en-Provence, France, September 2002.

[20] J. Kolter, R. Schillinger, and G. Pernul. A privacy-enhanced attribute-based access control system. In *Proc. of the 21st Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, Redondo Beach, CA, USA, July 2007.

[21] Organization for Economic Co-operation and Development. OECD guidelines on the protection of privacy and transborder flows of personal data, 1980. http://www.oecd.org/document/18/0,2340,en_2649_34255_1815186_119820_1_1_1,00.html.

[22] Privacy and identity management for europe (PRIME). http://www.prime-project.eu.org/.

[23] T. Ryutov, L. Zhou, C. Neuman, T. Leithead, and K.E. Seamons. Adaptive trust negotiation and access control. In *Proc. of the 10th ACM Symposium on Access Control Models and Technologies*, Stockholm, Sweden, June 2005.

[24] P. Samarati and S. De Capitani di Vimercati. Access control: Policies, models, and mechanisms. In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design*, LNCS 2171. Springer-Verlag, 2001.

[25] K. E. Seamons, W. Winsborough, and M. Winslett. Internet credential acceptance policies. In *Proc. of the Workshop on Logic Programming for Internet Applications*, Leuven, Belgium, July 1997.

[26] B. Thuraisingham. Privacy constraint processing in a privacy-enhanced database management system. *Data & Knowledge Engineering*, 55(2):159–188, November 2005.

[27] T.W. van der Horst, T. Sundelin, K.E. Seamons, and C.D. Knutson. Mobile trust negotiation: Authentication and authorization in dynamic mobile networks. In *Proc. of the Eighth IFIP Conference on Communications and Multimedia Security*, Lake Windermere, England, September 2004.

[28] W. Winsborough, K. E. Seamons, and V. Jones. Automated trust negotiation. In *Proc. of the DARPA Information Survivability Conf. & Exposition*, Hilton Head Island, SC, USA, January 2000.

[29] M. Winslett, N. Ching, V. Jones, and I. Slepchin. Assuring security and privacy for digital library transactions on the web: Client and server security policies. In *Proc. of the ADL '97 — Forum on Research and Tech. Advances in Digital Libraries*, Washington, DC, USA, May 1997.

[30] World Wide Web Consortium. *A P3P Preference Exchange Language 1.0 (APPEL1.0)*, April 2002. http://www.w3.org/TR/P3P-preferences/.

[31] World Wide Web Consortium. *The Platform for Privacy Preferences 1.1 (P3P1.1) Specification*, July 2005. http://www.w3.org/TR/2005/WD-P3P11-20050701.

[32] M. Youssef, V. Atluri, and N.R. Adam. Preserving mobile customer privacy: An access control system for moving objects and customer profiles. In *Proc. of the 6th International Conference on Mobile Data Management*, Ayia Napa, Cyprus, May 2005.

[33] T. Yu, X. Ma, and M. Winslett. An efficient complete strategy for automated trust negotiation over the internet. In *Proc. of the 7th ACM Computer and Communication Security*, Athens, Greece, November 2000.

[34] T. Yu and M. Winslett. A unified scheme for resource protection in automated trust negotiation. In *Proc. of the IEEE Symposium on Security and Privacy*, Berkeley, California, May 2003.

[35] T. Yu, M. Winslett, and K. E. Seamons. Interoperable strategies in automated trust negotiation. In *ACM Conference on Computer and Communications Security (CCS 2001)*, Philadelphia, Pennsylvania, USA, November 2001.

[36] T. Yu, M. Winslett, and K.E. Seamons. Supporting structured credentials and sensitive policies trough interoperable strategies for automated trust. *ACM Transactions on Information and System Security (TISSEC)*, 6(1):1–42, February 2003.

A Access control and release language: XML Schema

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:Q1="http://www.prime-project.eu/policies/"
 xmlns:xs="http://www.w3.org/2001/XMLSchema" attributeFormDefault="unqualified"
 elementFormDefault="qualified" targetNamespace="http://www.prime-project.eu/policies/"
 xmlns:Q2="https://www.prime-project.eu/ont/XSD-Claim"
 xmlns:request="https://www.prime-project.eu/ont/XSD-ClaimRequest">
  <xs:import namespace="https://www.prime-project.eu/ont/XSD-ClaimRequest"
   schemaLocation="ClaimRequest.xsd" />
    <xs:element name="policy" type="Q1:policyType">
      <xs:annotation>
        <xs:documentation>One policy.</xs:documentation>
      </xs:annotation>
    </xs:element>
  <xs:complexType name="policyType">
    <xs:sequence>
      <xs:element maxOccurs="1" minOccurs="1" name="description" type="Q1:descriptionType" />
        <xs:element name="object" type="xs:anyURI">
          <xs:annotation>
            <xs:documentation>What to access.</xs:documentation>
          </xs:annotation>
        </xs:element>
      <xs:element name="objectExprs" type="Q1:conditionListType" />
      <xs:element name="actions" type="Q1:actionListType">
        <xs:annotation>
          <xs:documentation>Allowed actions.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="rules" type="Q1:ruleListType" />
    </xs:sequence>
    <xs:attribute name="id" type="xs:ID">
      <xs:annotation>
        <xs:documentation>Id of the policy.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="type">
      <xs:annotation>
        <xs:documentation>
          A policy either guarding the disclose of or access to data.
        </xs:documentation>
      </xs:annotation>
      <xs:simpleType>
        <xs:restriction base="xs:anyURI">
          <xs:enumeration value="http://www.prime-project.eu/policies/access" />
          <xs:enumeration value="http://www.prime-project.eu/policies/release" />
          <xs:enumeration value="http://www.prime-project.eu/policies/datahandling" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="authors" type="xs:string" />
  </xs:complexType>
  <xs:complexType name="ruleListType">
    <xs:sequence>
      <xs:element maxOccurs="unbounded" minOccurs="0" name="rule" type="Q1:ruleType" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="ruleType">
    <xs:sequence>
      <xs:element name="subject" type="xs:anyURI" />
      <xs:element name="actions" type="Q1:actionListType" />
      <xs:element name="purposes" type="Q1:purposesListType" />
      <xs:element name="conditions" type="Q1:expressionsType" />
    </xs:sequence>
    <xs:attribute name="id" type="xs:ID" use="required" />
  </xs:complexType>
  <xs:complexType name="groupListType">
    <xs:sequence>
      <xs:element maxOccurs="unbounded" minOccurs="0" name="group" type="Q1:group" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="expressionsType">
    <xs:sequence>
      <xs:element name="subjectExprs" type="Q1:groupListType" />
      <xs:element name="genericExprs" type="Q1:conditionListType" />
      <xs:element name="trustExprs" type="Q1:conditionListType" />
      <xs:element name="lbsExprs" type="Q1:conditionListType" />
      <xs:element name="stateExprs" type="Q1:groupListType" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="conditionType">
    <xs:complexContent>
      <xs:extension base="Q1:predicateType">
        <xs:attribute default="true" name="sanitization" type="xs:boolean" />
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="group">
    <xs:sequence>
      <xs:element maxOccurs="unbounded" minOccurs="1" name="condition" type="Q1:conditionType" />
      <xs:element maxOccurs="1" minOccurs="1" name="evidence"
       type="request:claimrequest.option.group.evidenceListType" />
    </xs:sequence>
    <xs:attribute name="name" type="xs:anyURI" use="required" />
  </xs:complexType>
  <xs:complexType name="annotationType">
    <xs:simpleContent>
      <xs:extension base="xs:anySimpleType">
        <xs:attribute name="name" type="xs:anyURI" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="descriptionType">
    <xs:sequence>
      <xs:element maxOccurs="1" minOccurs="1" name="long" type="xs:string" />
      <xs:element name="annotation" type="Q1:annotationType" />
    </xs:sequence>
    <xs:attribute name="short" type="xs:string" />
  </xs:complexType>
  <xs:complexType name="conditionListType">
    <xs:sequence>
      <xs:element maxOccurs="unbounded" minOccurs="0" name="condition" type="Q1:conditionType" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="actionListType">
    <xs:sequence>
      <xs:element maxOccurs="unbounded" minOccurs="1" name="action" type="xs:anyURI" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="predicateType">
    <xs:sequence>
      <xs:element maxOccurs="unbounded" minOccurs="0" name="argument" type="Q1:argumentType" />
    </xs:sequence>
    <xs:attribute name="name" type="xs:anyURI" />
  </xs:complexType>
  <xs:complexType name="certificationType">
    <xs:sequence>
      <xs:any maxOccurs="unbounded" minOccurs="0" namespace="##any" processContents="lax" />
    </xs:sequence>
    <xs:attribute name="type" type="xs:anyURI" />
  </xs:complexType>
  <xs:complexType name="argumentType">
    <xs:complexContent>
      <xs:extension base="xs:anyType">
        <xs:attribute name="isLiteral" type="xs:boolean" use="required" />
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="purposesListType">
    <xs:sequence>
      <xs:element maxOccurs="unbounded" minOccurs="1" name="purpose" type="xs:anyURI" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="recipientsListType">
    <xs:sequence>
      <xs:element name="recipient" type="xs:anyURI" />
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

B Data handling language: XML Schema

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
 attributeFormDefault="unqualified">
  <xs:complexType name="condition">
    <xs:sequence>
      <xs:element name="argument" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:complexContent>
            <xs:extension base="xs:anyType">
              <xs:attribute name="isLiteral" type="xs:boolean" />
            </xs:extension>
          </xs:complexContent>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="name" type="xs:anyURI" use="required" />
  </xs:complexType>
  <xs:element name="DHP">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="annotation">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="description" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="Description">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Long" type="xs:string" />
              <xs:element name="Short" type="xs:string" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="PII" type="xs:anyURI" />
        <xs:element name="actions">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="action" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="purposes">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="purpose" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute name="name" type="xs:anyURI" use="required" />
                      <xs:attribute name="type" use="optional" default="optional">
                        <xs:simpleType>
                          <xs:restriction base="xs:string">
                            <xs:enumeration value="required" />
                            <xs:enumeration value="optional" />
                          </xs:restriction>
                        </xs:simpleType>
                      </xs:attribute>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="recipients">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="recipient" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="condition" type="condition" minOccurs="0"
                     maxOccurs="unbounded" />
                  </xs:sequence>
                  <xs:attribute name="subject" type="xs:anyURI" />
                  <xs:attribute name="type" use="optional" default="optional">
                    <xs:simpleType>
                      <xs:restriction base="xs:string">
                        <xs:enumeration value="required" />
                        <xs:enumeration value="optional" />
                      </xs:restriction>
                    </xs:simpleType>
                  </xs:attribute>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="obligations">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="condition" type="condition" minOccurs="0"
              maxOccurs="unbounded" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="provisions">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="condition" type="condition" minOccurs="0"
              maxOccurs="unbounded" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="gen_conditions">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="condition" type="condition" minOccurs="0"
              maxOccurs="unbounded" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="disputes">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="dispute" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="remedies" type="xs:string" />
                    <xs:element name="description" type="xs:string" />
                  </xs:sequence>
                  <xs:attribute name="thirdParty" type="xs:anyURI" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="id" type="xs:string" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```