11

Privacy Models and Languages: Access Control and Data Handling Policies

Claudio Agostino Ardagna, Sabrina De Capitani di Vimercati, and Pierangela Samarati

Università degli Studi di Milano

11.1 Introduction

The huge amount of personal information available on the Web has led to growing concerns about the privacy of its users, which has been recognized as one of the main reasons that prevents users from using the Internet for accessing online services. Users in fact prefer not to be under the control of anyone at anytime. In this context, the concept of *privacy control* is introduced, and it should encompass three main aspects: to guarantee the desired level of privacy of information by controlling the access to services/resources; to control secondary use of information disclosed for the purpose of access control enforcement; to deal with the specific management of related privacy obligations [Cas04b] (e.g., data retention, data deletion, notifications).

In this privacy-oriented scenario, access control systems may help users in keeping control over their personal information. Access control solutions should then be enriched with the ability of supporting privacy requirements [ADDS05, BDDS01], as for instance: *i) interchangeable policy format,* parties need to specify protection requirements on the data they make available using a format both human- and machine-readable, easy to inspect and interchange; *ii) interactive enforcement,* the evaluation phase should provide a way of interactively applying criteria to retrieve the correct reports, possibly managing complex user interactions, such as, the acceptance of written agreements and/or online payments for each report; *iii) metadata support,* privacy-aware access control systems should allow to specify access restrictions based on conditions on metadata describing (meta)properties of the stored data and the users.

J. Camenisch, R. Leenes, and D. Sommer (Eds.): Digital Privacy, LNCS 6545, pp. 309–329, 2011.

Traditional access control systems, which are based on regulations (*policies*) that establish who can, or cannot, execute which actions on which resources [SD01], result limiting and do not satisfy the above requirements. Although recent enhancements allow the specification of policies with reference to generic attributes/properties of the parties and the resources involved (e.g., XACML [eXt05]), access control systems are not designed for enforcing privacy policies. Also, few proposals have tried to address the problem of how to regulate the use of personal information in secondary applications. The consideration of privacy issues introduces the need for rethinking authorization policies and models, and the development of new paradigms for access control policy specification and enforcement. Two main issues to be looked at are:

1. access control needs to operate even when interacting parties wish to remain anonymous or to disclose only specific attributes about themselves;
2. data collected/released during access control, as well as data stored by the different parties, may contain sensitive information on which privacy policies need to be applied.

In the following of this chapter, we will provide further details about different types of privacy policies managed in PRIME. Chapters 12 and 13 will then investigate more in detail privacy obligations and assurance policies, respectively.

11.2 Privacy Policy Categories

To fully address the requirements introduced by the need of a privacy-aware access control system, a new model together with the following different types of privacy policies have been introduced.

Access Control Policies

Access control policies define authorization rules concerning access to data or services [SD01]. Authorizations correspond to traditional (positive) rules usually enforced in access control systems. For instance, an authorization rule can require a user of 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, 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

Release policies define the preferences of each party regarding the release of its PII. They specify to which party, for which purpose/action, and under which conditions a particular set of PII can be released [BS02a]. For instance, a release policy can state that credit card information can be released only in the process of a purchase and to trusted partners. The release of PII may only be enforced if the release policies are satisfied.

Data Handling Policies and Obligations

Data handling policies [ADS06, ACDS08] regulate how PII will be handled at the receiving parties (e.g., information collected through an online service may be combined with information gathered by other services for commercial purposes). Users specify these policies to define restrictions on secondary use of their personal information, thus controlling the information also after its release. Data handling policies will be attached to the PII or data they protect, and transferred as *sticky policies* to the counterparts [KSW02b]. A specific type of data handling policy is the obligation policy (see Chapter 12) dictating privacy constraints and expectations on the lifecycle management of personal data. For example, these policies might prescribe constraints on data deletion, data transformation, notifications, and the like.

Assurance Policies

Assurance policies describe enterprise assurance properties relating to how personal data will be transferred, processed, and protected (see Chapter 13). They can help in checking compliance to law, data subjects' preferences and enterprise guidelines, and can refer to trust, assurance, and contextual properties.

The next sections briefly describe a scenario illustrating few examples of policies that have been managed in PRIME, focusing on access control and data handling policies. Obligation and assurance policies will be discussed more in detail in the following chapters.

11.3 Scenario

Our reference scenario is a distributed infrastructure that includes three parties (see Figure 11.1):

- *users* are human entities that request online services;
- *service provider* is the entity that provides online services to the users and collects personal information before granting an access to its services;

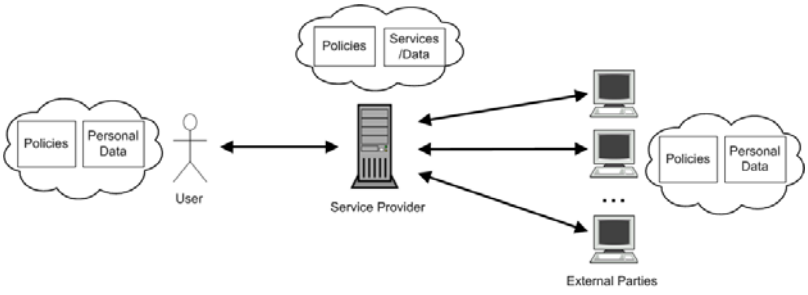

Fig. 11.1 Reference scenario

- *external parties* are entities (e.g., business partners) to which the service provider may want to send or trade personal information of users. Alternatively, they may be involved in the process of checking and providing evidence that the service provider's assurance policies are valid.

Although each party can act interchangeably as a user, a service provider, or an external party during different transactions, they usually have well defined, fixed roles when one specific access request is considered. We assume that the service provider collects personal data that are necessary to provide access to services and stores them into *profiles* associated with each user. A profile can therefore be seen as a container of pairs of the form $\langle attribute\_name, attribute\_value \rangle$, where *attribute_name* is the name of the attribute provided by the user and *attribute_value* is its value.

The set of messages exchanged between a user and a service provider is called *negotiation* [BS02a]. A negotiation always starts with a user request to a service provider and ends explicitly (done or stop) or implicitly, for example, assumed after a certain timeout period. A negotiation intuitively corresponds to the classical concept of *session*. We assume anonymous communications to be in place. Therefore, at the beginning of a negotiation, the user is unknown to the service provider, meaning that no information about the user has been collected by the service provider. During a negotiation a user may, similarly to a service provider, require the counterpart to fulfill some requirements (i.e., provide information) for it to proceed. In others words, a bidirectional negotiation is considered where both parties can require the other party to provide them with certain information or digital certificates necessary for service request or fulfillment. Each party has a *portfolio of credentials* (third-party endorsed attribute data such as digital certificates) and *declarations* (unsigned data). Access to services and release of portfolio information are managed according to the rules specified by the parties.

The same discussion is still valid when an external party wants to access personal information of the users stored at the service provider. The only differences are that, in this case, the service provider must be responsible for protecting the privacy of users' data, and users are assumed to trust the

service provider to faithfully maintain and manage their personal information, according to their privacy requirements. The process of negotiation enables users (or entities acting on their behalf) to have a better understanding of which type of data is going to be required to fulfill a transaction, for which purposes, and under which constraints.

In the PRIME vision, the negotiation and PII disclosure phases should also enable users to specify their privacy preferences in terms of data handling and management. Ultimately, these preferences dictate constraints and obligations to be fulfilled by the data receiving party. This scenario has implications on how to represent these preferences and how to factor them into privacy-aware obligation policies. PRIME also wants to enable users to check upfront the properties and capabilities of a data receiving party (e.g., an organization, a service provider, and so forth), before engaging in any data disclosure. This is part of the 'assurance checking' process, aiming at increasing the level of trust a user has in an organization.

In the remainder of this chapter and in the next two chapters, we illustrate the basic concepts and principles of our models for access control, obligation and assurance policies.

11.4 Access Control Model and Language

An access control language is used to specify both access control and release policies.[1] In this section, we give an overview of the functionalities and syntax provided by our access control model and language.

11.4.1 Basic Concepts

11.4.1.1 Portfolio and Profiles

In open environments, the decision to grant access to a resource is often based on different attributes of the requester rather than its specific identity. Here, we assume that each party has a *portfolio* of *declarations* and *credentials* [GEB], which is used to gain (or offer) services [BS02a]. The portfolio may also represent views of certificates that are not actually stored at the party site but can be obtained if needed [SAB+]. This way, the model allows a party to refer to the set of all its possible credentials without need of maintaining a copy of each of them. The definition of portfolio introduces the following types of attributes [BS02a].

- *Certified attributes* are specified in an electronic credential that is characterized by the credential *name*, the *issuer*'s public key, the *subject*'s public

[1] Although semantically different, access control and release policies are syntactically identical.

key, a *validity period*, a list (possibly empty) of pairs ⟨*attribute_name*, *attribute_value*⟩ representing the certified attributes (e.g., name and surname contained in an electronic passport), and a *digital signature*.
- *Declared attributes* represent self-certified statements of a party, with no certification from any legal authority. A declared attribute is a pair ⟨*attribute_name*, *attribute_value*⟩ (e.g., the professional status of a user communicated by the user herself).

The set of certified and declared attributes released by a party to the service provider is then stored in the 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 $\mathcal{Q}$ a set of predicates including standard built-in mathematical predicates (e.g., `equal`, `notEqual`, `greaterThan`). We define a credential term as follows.

Definition 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 expressions of the form `math-pred`(*attribute_name*,*value*) with `math-pred` $\in \mathcal{Q}$ a mathematical predicate, *attribute_name* the attribute name as it appears in the credential `cred_name`, and *value* the corresponding attribute value.

Expression *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 an authority K and such that *condition_list* is satisfied.

Example 1. An example of credential term is `identity-card`(`equal`(occupation,'Student')) denoting an `identity-card` credential whose attribute occupation has value Student. Predicate `credential`(`identity-card`(`equal`(occupation,'Student')),K_1) is then evaluated to true if there exists an `identity-card` credential issued by K_1 certifying that the occupation of the credential subject is Student.

A declaration term is defined as follow.

Definition 2 (Declaration term). A *declaration term* is an expression of the form `predicate_name`(*arguments*), where `predicate_name` $\in \mathcal{Q}$ 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 2. An example of declaration term is `equal`(name,'Alice') denoting the name attribute whose value is Alice. The corresponding declaration predicate is then `declaration`(`equal`(name,'Alice')).

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 `driver-license`, `passport`, and `identity-card`.

11.4.1.2 Ontologies and Abstractions

Our model provides the support for *ontologies* that permit to make generic assertions on subjects and objects [DDFS04]. More precisely, we use three ontologies: a *subject* ontology, an *object* ontology, and a *credential* ontology. A subject ontology contains terms that can be used to make generic assertions on subjects and to define relationships among them. An object ontology contains domain-specific terms that are used to describe resource content. Finally, a credential ontology represents relationships among attributes and credentials (`part-of` and `is-a` relationships), and more complex relationships between attributes and abstractions. The credential ontology is then used to establish which credentials can be provided to fulfill a declaration or credential request, according to the principle of the *minimum disclosure*.

11.4.2 Functionalities

Before presenting the access control model and language used to specify the *access control policies* protecting server-side resources and the *release policies* regulating access to personal information of the parties, we summarize their main functionalities as follows.

- *Attribute-based restrictions.* The language supports the definition of powerful and expressive policies based on properties (attributes) associated with subjects (e.g., name, address, occupation) and objects (e.g., owner, creation date). The language includes some operators for comparing attribute values and could be extended by adding nonstandard functions.
- *XML-based syntax.* The language provides an XML-based syntax for the definition of powerful and interoperable access control and release policies.
- *Credential definition and integration.* The language supports requests for certified data, issued and signed by authorities trusted for making the statement, and uncertified data, signed by the owner itself.
- *Anonymous credentials support.* The language supports definition of conditions that can be satisfied by means of zero-knowledge proof [CL01c, CV02].

- *Support for context-based conditions.* The language allows the definition of conditions based on context information (including the physical position of the users [ACD+06]). It further integrates 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 where a transaction takes place.
- *Ontology integration.* Policy definition is fully integrated with subject and object ontologies in defining access control restrictions. Also, the language takes advantage of the integration with a credential ontology that represents relationships among attributes and credentials, and between credentials.

11.4.3 Description of the Access Control Language

We describe our access control model discussing the basic constructs of the language used to define access control and release policies. First, the following predicates constitute the basic literals that can be used in access control and release policy specification:

- a binary predicate `credential(`ct,K`)`, where ct is a *credential term* (see Definition 1), and K is the name or the public key of a trusted certification authority (CA);
- a predicate `declaration(`d`)`, where d is a list of *declaration term* (see Definition 2);
- a set of standard built-in mathematical predicates, such as `equal()`, `greater_than()`, `lesser_than()`, and so forth;
- a set of state-based, location-based, trust-based predicates of the form `predicate_name(`*arguments*`)`;
- a set of non-predefined predicates.

Then, three basic elements of the language have been identified: *subject_expression*, *object_expression*, and *conditions*. Below, single properties belonging to user and object profiles are referenced through the dot notation. 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. 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. Beside the formal definition of the access control model and language, we also provide some examples of *subject_expression*, *object_expression*, and *conditions* using the XML-based syntax defined for access control and release policy specification. This syntax has been used in the development of our privacy-aware access control system prototype shown in Section 14.

Subject Expression

These expressions allow to refer 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` (see Definition 1 and 2). The following are examples of subject expressions:

- `credential(passport(equal(`**user**.nationality,'Italian'), `greater_than(`**user**.age,18)),K_1) denoting requests made by Italian users of age. These properties should be certified by showing the `passport` credential verifiable with public key, or released by CA, K_1;
- `declaration(equal(`**user**.name,'John')) denoting requests made by a user whose name is John.

Based on the syntax provided in Appendix 30.1, an example of *subject expression* is provided in the following where *any* user[2] must provide her name (i.e., *name.given*) and surname (i.e., *name.last*) proved by an X.509 *identity-document* released by the Italian public administration, and must be of age (no certification is requested) to gain the access to a particular object.

```
<subject>any</subject>
<subjectExprs>
 <group>
   <condition name="exist">
     <argument isLiteral="false">name.given</argument>
   </condition>
   <condition name="exist">
     <argument isLiteral="false">name.last</argument>
   </condition>
   <evidence>
     <issuer>ItalianPublicAdministration</issuer>
     <proofMethod>X.509</proofMethod>
     <type>identity-document</type>
   </evidence>
 </group>
 <group>
   <condition name="greaterThan">
     <argument isLiteral="false">age</argument>
     <argument isLiteral="true">18</argument>
   </condition>
   <evidence/>
 </group>
</subjectExprs>
```

[2] The `subject` element defines an identifier or an abstraction that refer to a set of users.

The `group` element groups different conditions that have to be satisfied by the same `evidence` element (i.e., the same credential), which in turn defines restrictions on the certification type. This avoids that a request for *name.given* and *name.last* is satisfied by two different credentials.

Object Expression

These expressions allow to refer 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 *arguments* is a list, possible empty, of constants or attributes. The following are examples of object expressions:

- `equal(`**object**.owner,**user**`)` denoting all objects created by the requester;
- `lesserThan(`**object**.validity,today`)` denoting all valid objects;
- `greaterThan(`**object**.age,35`)` denoting all objects whose attribute age is greater than 35.

Based on the syntax provided in Appendix 30.1, an example of *object expression* is provided in the following, specifying the set of all credit cards (i.e., *cc_info*)[3] with VISA circuit and whose expiration date was before December 2000.

```
<object>cc_info</object>
<objectExprs>
 <condition name="equal">
   <argument isLiteral="false">circuit</argument>
   <argument isLiteral="true">VISA</argument>
 </condition>
 <condition name="lesserThan">
   <argument isLiteral="false">expiration</argument>
   <argument isLiteral="true">12/00</argument>
 </condition>
</objectExprs>
```

Conditions

Conditions element specifies conditions that can be brought to satisfactions at run-time processing of the request. More precisely, *a condition element is a boolean formula of terms of the form* `predicate_name(`*arguments*`)`, where *arguments* is a list, possible empty, of constants or attributes. Four different types of conditions can be stated inside a rule: *i) state-based conditions*: restrictions based on the environment state; *ii) location-based conditions*: restrictions based on location information of individuals; *iii) trust-based conditions*: restrictions based on the assurance/trust of the environment; *iv) others conditions*: conditions that do not belong to any of the other classes.

[3] `object` element defines an object identifier or abstraction.

Each condition type is defined by means of an ad-hoc XML element (see Appendix 30.1): `stateExprs`, `lbsExprs`, `trustExprs`, `genericExprs`. In the following, we provide an example of location-based condition stating that, at access control time, the country under which the roaming phone of the requester (i.e., *SIM*) is registered should be 'Italy' to have the request satisfied.

```
<lbsExprs>
 <condition name="equal">
   <argument isLiteral="false">SIM</argument>
   <argument isLiteral="true">Italy</argument>
 </condition>
</lbsExprs>
```

The same syntax of `lbsExprs` element is used for all types of conditions.

11.4.3.1 Policy and Rule Definition

An access control policy (release policy, resp.) is composed by one or more rules, composed in OR logic between them, directly associated with an object component and the related set of actions. Syntactically, an access control policy (release policy, resp.) can be formalized as follows.

Definition 3 (Access control policy). *An access control policy is an expression of the form* ⟨actions⟩ *ON* ⟨object⟩ WITH ⟨object_expression⟩ IF ⟨rules⟩*, where:*

- actions *is the set of actions to which the rules refer (e.g., read, write, and so on);*[4]
- object *identifies the object to which the rules refer and corresponds to an object identifier or a named abstraction of values, if abstractions are defined on objects;*
- object_expression *is a boolean expression that allows the reference to a set of objects depending on whether they satisfy given conditions that can be evaluated on the object's profile;*
- rules *is a set of rules as defined in Definition 4.*

An access control rule (release rule, resp.) represents the basic element used to regulate the access to the objects with which it is associated. Syntactically, an access control rule (release rule, resp.) can be formalized as follows.

Definition 4 (Access control rule). *An access control rule is an expression of the form* ⟨subject⟩ WITH ⟨subject_expression⟩ CAN ⟨actions⟩ FOR ⟨purposes⟩ IF ⟨conditions⟩*, where:*

[4] Note that the *actions* field can be refined in the rules. Abstractions can also be defined on actions, specializing actions or grouping them in sets.

- subject *identifies the subject to which the rule refers and corresponds to a user identifier or a named abstraction of values, if abstractions are defined on subjects;*
- subject_expression *is a boolean expression that allows the reference 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);*
- purposes *is the purpose or a group thereof to which the rule refers, and represents how the data are going to be used by the recipient;*
- conditions *is a boolean expression of conditions that an access request to which the rule applies has to satisfy.*

Example 3. In the following, for sake of clarity and conciseness, access control and release policies are provided in the simplified form shown in Table 11.1, rather than with our complete XML-based syntax (see Appendix 30.1). As an example, suppose that a `hospital` provides a set of services to its patients. Patients release their data to the `hospital` to gain access to the services. The `hospital` defines the access control policies in Table 11.1 to protect the access to the data stored locally. In particular, AC1 is composed by two rules that regulate access to valid `cc_info`. An access control policy is evaluated to true if at least one rule is satisfied, that is *subject_expression* and *conditions* of the rule are satisfied. AC2 is composed by a single rule that regulates access to `personal_info` of patients.

To conclude, although the definition of access control and release policies permits to protect access to data and services, and release of personal data, respectively, no solution is provided for regulating how PII must be used and processed after its release. To this aim, in the next section, we introduce a data handling model and language.

11.5 Data Handling Model and Language

A privacy-aware access control solution supporting restrictions on secondary use should be simple and expressive enough to support, among others, the following privacy requirements [Dir95, Org80]: *i) openness,* privacy practices should be transparent and fully understandable for all parties; *ii) individual control,* users should be able to specify who can see what information about them and when; *iii) collection limitation,* parties collecting personal data for the purpose of a transaction must gather no more data than what is strictly needed; *iv) purpose specification,* entities who collect and disseminate personal data must specify the purposes for which they need these data; *v) consent,* users should be able to give their explicit and informed consent on how to use their personal data.

Table 11.1 An example of access control policies (release policies, resp.)

	Access Control Policies			
	object	**act**	**AC Rules**	**Description**
AC1	`cc_info` WITH `greaterThan(` **object**.expiration, *today*)	read	any WITH [`credential(employeeCard(equal(` **user**.job,'Secretary')),K_H) AND `declaration(equal(` **user**.company, 'Hospital'))] CAN `read` FOR `service_release` IF [`in_area(` **user**.sim,'Hospital') AND `log_access()`]	The secretaries of the `Hospital` are authorized to read valid (i.e., not yet expired) `cc_info` for service release purpose, if they are located inside the `hospital` and access is logged.
			any WITH [`credential(employeeCard(` `equal(` **user**.job,'BusinessConsultant')), K_H)] CAN `read` FOR `reimbursement`	The business consultants of the `Hospital` are authorized to read valid `cc_info` for reimbursement purpose.
AC2	`personal_info` WITH `equal(` **object**.physicianID, **user**.ID)	read	any WITH [`credential(employeeCard(equal(` **user**.job,'Primary Physician'), K_H) AND `declaration(equal(` **user**.company,'Hospital'))] CAN `read` FOR `service_release`	A primary physician of the `hospital` can read `personal_info` of her patients for service release purpose.

Our privacy-aware access control solution is based on *data handling policies* [ACDS08, ADS06] (DHPs, for short), respectful of the above requirements, which provide the users with the possibility to define how their PII can be subsequently used by the service provider and/or external parties. 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 (i.e., by whom a DHP is defined), three different strategies are possible, each one requiring different levels of negotiation between a user and a service provider: *server-side*, *user-side*, and *customized*. Server-side and user-side are the opposite endpoints of all possible approaches in the definition of privacy rules that balance between service provider and user needs. The customized approach, instead, represents a trade-off between the power given to the service providers and the protection assured to the users. In particular, when a user requires a service, a predefined policy template is provided by the service provider as a starting point for creating data handling policies. The template is customized by the user to meet different privacy requirements. A user can directly customize the template or it can be supported by a customization process that automatically applies the privacy preferences of the user. If the customized data handling policy will be accepted by the service provider, the personal information provided by the user will be labeled with this policy. This represents the most flexible and balanced strategy for the definition of a data handling policy, and we therefore adopt it.

With respect to the second issue (i.e., how a DHP is defined), data handling policies are defined as independent rules and represent the privacy preferences of the users. DHPs should then include different components that allow users to define how the external parties can use their personal data. Personal data are *tagged* with such data handling policies. 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* under which *conditions* and following some *obligations*. Although the stand-alone option can introduce some redundancy in policy definition, it provides a better 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. Also, the definition of standalone policies reduces the risks of unprotected data types and allows for the customization of additional components such as recipients and actions. Finally, an additional motivation to prefer stand-alone data handling policies is that some of the conditions (e.g., some obligations) do not necessarily depend on access control events, and then cannot just be enforced by an access control system. For instance, the obligation condition "delete data after 10 days" is enforced independently from the fact that the data have ever been accessed. In this case, stand-alone data handling policies enable building a solution with multiple enforcement points, some of them outside the control of the access control system (e.g., an obligation manager), but orchestrated/configured by it.

In the following, we assume a customized stand-alone approach for data handling policy specification.

11.5.1 Description of the Data Handling Language

The following predicates constitute the basic literals that can be used in data handling policy specification:

- a binary predicate `credential(`ct,K`)`, where ct is a *credential term* (see Definition 1), and K is the name or the public key of a trusted certification authority (CA);
- a predicate `declaration(`d`)`, where d is a list of *declaration term* (see Definition 2);
- a set of standard built-in mathematic predicates, such as `equal()`, `greater_than()`, `lesser_than()`, and so forth;
- a set of provision and obligation predicates of the form `predicate_name(` *arguments*`)`;
- a set of non predefined predicates.

Five basic elements have then been identified: *recipients*, *purposes*, *PII abstraction*, *restrictions*, and *obligations*. As for our access control language, we provide an XML-based syntax for data handling policy specification (see Appendix 30.2).

Recipients

A recipient is an external party to which PII of users can be disclosed by the service provider [Dir95]. 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 based on their *attributes*, instead of their *identity*. Similarly to *subject_expression* in access control policies, *recipients* is a boolean formula of `credential` (see Definition 1) and `declaration` (see Definition 2).

Based on the syntax provided in Appendix 30.2, an example for a *recipients* element is provided in the following where an external party can access user data provided that it belongs to the *Public Administration* or it is a *Non-profit Organization*.

```
<recipients>
 <recipient>
   <condition name="equal">
     <argument isLiteral="false">type</argument>
     <argument isLiteral="true">PublicAdministration</argument>
   </condition>
 </recipient>
 <recipient>
   <condition name="equal">
     <argument isLiteral="false">type</argument>
     <argument isLiteral="true">Non-ProfitOrg</argument>
   </condition>
 </recipient>
</recipients>
```

The `recipients` element groups different `recipient` that are composed in OR logic among them, that is, an external party has to satisfy at least one of the `recipient` elements.

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 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 specializations of `research`.

PII abstraction

Data types can be introduced as abstractions of PII to let data handling policies be expressed in terms of data types, rather than single properties of the user only. Data types can be organized hierarchically. For instance, in Figure 11.2, `cc_info` can be seen as an abstraction for the credit card information, which can include the `number`, `circuit`, and `expiration` attributes.

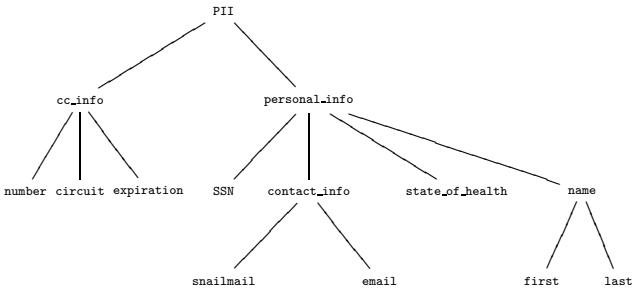

Fig. 11.2 An example of PII abstraction

Restrictions

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

- *Generic* conditions either evaluate properties of recipients' profiles, like membership of requester, or represent conditions that can be brought to satisfaction at run-time when the request is processed.
- *Provision* are preconditions that need to be evaluated as pre-requisites before a decision can be taken [BJWW02].

Syntactically, generic conditions and provision are boolean expressions of terms having the form `predicate_name(`*arguments*`)`, where *arguments* is a list, possibly empty, of arguments on which predicate `predicate_name` is evaluated. 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(`*form*`)` and `log_access()` are provision predicates that require to fill in a form and to log the access, respectively.

Each condition type (i.e., generic condition and provision) is defined by means of an ad-hoc XML element (see Appendix 30.2): `gen_conditions` and `provisions`. In the following we provide an example of generic conditions restricting the access from *10 am* to *2 pm*, and an example of provision stating that before access is given, it must be logged.

```
<gen_conditions>
 <condition name="time">
   <argument isLiteral="true">10am</argument>
   <argument isLiteral="true">2pm</argument>
 </condition>
</gen_conditions>
```

```
<provisions>
 <condition name="log_access">
   <argument/>
 </condition>
</provisions>
```

Obligations

Obligations are defined as complex policies inside data handling policies. They represent actions that have to be performed either after an access has been granted [BJWW02] or in the future based on the occurrence of well-defined events [Cas04b, CB07a] (e.g., time-based or context-based events). For instance, an obligation can state that users will be notified whenever their personal information is disclosed. Another obligation can impose a restriction on how long personal data should be retained (data retention). A full discussion of obligation policies is provided in Section 12.

11.5.1.1 Policy and Rule Definition

Syntactically, a data handling policy has the form "⟨*PII*⟩ MANAGEDBY ⟨*DHP_rules*⟩", where: *PII* identifies a PII abstraction that represents the name of an attribute or a data type, in case of a data handling policy template, a set of pairs of the form ⟨*attribute_name,attribute_value*⟩ belonging to a privacy profile, in case of a customized data handling policy; and *DHP_rules* identifies one or more rules, composed in OR logic, governing the use of PII to which they refer. Syntactically, a *DHP_rules* can be formalized as follow.

Definition 5 (Data handling rule). *A DHP_rules is an expression of the form* ⟨*recipients*⟩ CAN ⟨*actions*⟩ FOR ⟨*purposes*⟩ [IF ⟨*gen_conditions*⟩] [PROVIDED ⟨*prov*⟩] [FOLLOW ⟨*unique_obl_id*⟩], *where:*

- recipients *can be an identifier, a category, or a boolean formula of* `credential` *and/or* `declaration` *predicates;*
- actions *is the set of actions;*
- purposes *is the purpose or a group thereof;*
- provisions *and* generic *conditions are optional boolean expressions of terms having the form* `predicate_name(`arguments`)`, *where* arguments *is a list, possibly empty, of arguments on which* `predicate_name` *is evaluated;*
- obligations *which are referred inside a data handling policy through a* unique_obl_id.

A data handling rule specifies that *recipients* can execute *actions* on *PII* for *purposes* provided that *prov* and *gen_conditions* are satisfied, and with obligations *obl*.

Table 11.2 An example of data handling policies that protect `Alice`'s data

	Data Handling Policies		
	PII	**DHP Rules**	**Description**
DHP1	`Alice.cc_info`	[`credential(employeeCard(equal(`**user**.job, 'Secretary'),`equal(`**user**.jobLevel,'A')), K_H) AND `declaration(equal(`**user**.company,'Hospital'))] CAN `read` FOR `service_release` IF `time`(8:30am,6:00pm) PROVIDED `log_access()`	Secretaries of the `hospital` whole level is A 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.
DHP2	`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 `obl-id-001`	European business partners of the `hospital` or government authorities can read `personal_info` of `Alice` for research with obligation `obl-id-001`.
		[`credential(identity-document(` `equal(`**user**.name.given,'John'), `equal(`**user**.name.last,'Doe')),K_H) AND `declaration(equal(`**user**.job,'Doctor'))] CAN `read` FOR `service_release` IF `in_area(`**user**.sim,'Hospital')	Doctor John Doe can read the personal information of `Alice` for service release purpose only if he is in the `hospital` area.

Example 4. An example of data handling policies is provided in the simplified form shown in Table 11.2. Suppose that a `Hospital` provides services to its patients. Table 11.2 shows an example of customized data handling policies that regulate the secondary use of personal information of `Alice` stored by the `Hospital`. In particular, *DHP1* is composed of a single rule that protects the `cc_info` of `Alice`; *DHP2* is composed of two rules that protect the `personal_info` of `Alice`.

11.6 Related Work

A number of research works about privacy and identity management have been presented in the last few years. The lines of research closely related to the work in this chapter are in the areas of credential-based access control models, trust negotiation solutions, and privacy-aware models and languages.

Access control models based on digital credentials make decisions about whether or not a requesting party may execute an access on the basis of properties that this party may have. These properties can be proven by presenting one or more certificates [BS02a, NLW05, YWS03]. The first proposals that investigate the application of credential-based access control to regulate access to a server are done by Winslett et al. [SWW97, WCJS97]. Access

control rules are expressed in a logic language, and rules applicable to a service access can be communicated by the server to clients. A first attempt to provide a uniform framework for attribute-based access control specification and enforcement is presented by Bonatti and Samarati [BS02a]. The framework includes an access control model and a language for expressing access control and release policies, and a policy-filtering mechanism to identify the relevant policies for a negotiation. Access rules are specified as logical rules, with some predicates explicitly identified. Also, this proposal permits to reason about certified attributes, modeled as credential expressions, and declared attributes (i.e., unsigned statements). Communication of requisites to be satisfied by the requester is based on a filtering and renaming process applied to server policies, which exploits partial evaluation techniques in logic programs. Other works (e.g., [GPSS05]) have also investigated solutions for providing authentication and access control based on biometric systems and information [GLM+04]. In this context, Cimato et al. [CGP+08] propose a privacy-aware biometric authentication technique that uses multiple biometric traits.

Besides solutions for uniform frameworks supporting credential-based access control policies, different automated trust negotiation proposals have been developed [SWY01, YW03, YWS01]. Trust is established gradually by disclosing credentials and requests for credentials [GNO+04]. In [RZN+05, WSJ00, YW03, YWS03], the authors investigate trust negotiation issues and strategies that a party can apply to select those credentials to submit to the opponent party during a negotiation. Trust-management systems (e.g., Keynote [BFIK98], PolicyMaker [BFL96], REFEREE [CFL+97], and DL [LGF00]) use credentials to describe specific delegation of trusts among keys and to bind public keys to authorizations. They therefore depart from the traditional separation between authentication and authorization by granting authorizations directly to keys (bypassing identities).

In the last few years, as the need of privacy increases, a number of useful *privacy enhancing technologies* (PETs) have been developed for dealing with privacy issues. In this context, access control solutions enriched with the ability of supporting privacy requirements have been provided and some privacy-aware models and languages have been defined. The first objective of such solutions is to build an infrastructure that, on one side, regulates and restricts access to data, and, on the other side, allows users to protect their privacy by keeping a level of control over their data after their release to third parties. In this context, important issues to be considered concern the definition, management, and enforcement of privacy obligations. While the management of obligations can be a reasonably easy task when the events that trigger them are well defined and simple to capture, it becomes more complex in the case of privacy obligations triggered by the occurrence of events and conditions non-necessarily related to time or known transactions.

Relevant work has been done by W3C with its Platform for Privacy Preferences Project (P3P) [Cra02, Wor02]. P3P addresses the need of a user to

assess that the privacy practices adopted by a service provider comply with her privacy requirements. P3P provides an XML-based language and a mechanism for ensuring that users can be informed about privacy policies of the server before the release of personal information. Users specify their privacy preferences through a policy language, called A P3P Preference Exchange Language 1.0 (APPEL) [W3C02], and enforce privacy protection by means of a user agent, which compares and verifies whether the P3P policy conforms to user privacy preferences. P3P is important to shape (aspects of) the trust that people might have on the enterprise by verifying which privacy requirements they can fulfill. However, P3P is mainly a "front-end" mechanism, in the context of Web Services. In its current form it is "passive", that is, it only checks if people expectations are matched against promises made by the enterprise. It does not address the problem of allowing users to express fine grained privacy policies and obligations; nor provide mechanisms to deal with the execution and fulfillment of these privacy policies and obligations, and related constraints by enterprises. Last but not least, it does not define an enterprise framework for dealing with privacy policies.

Focusing on the problem of privacy management for enterprises, the Enterprise Privacy Architecture (EPA) [KSW02b] encompasses a policy management system, a privacy enforcement system, and an audit console. EPA is aimed at improving trust in enterprises e-business and provides a new approach to privacy that tries to help organizations in understanding how privacy impacts business processes. Specifically, the work in [SA02] introduces additional architectural details about EPA along with an interpretation of the concept of privacy obligations. This concept is framed in the context of privacy rules (policies) defined for authorization purposes. This approach is further refined and described in the Enterprise Privacy Authorization Language (EPAL) specification [AHK+03, AHKS02]. In general, EPAL consists of an XML-based markup language and an architecture aimed at formalizing, defining, and enforcing enterprise-internal privacy policies. It addresses the problem on the server side and supports the need of a company to specify access control policies, with reference to attributes/properties of the requester, which protect private information of its users. The current EPAL specification does not provide a format (or description) for obligations; obligations are purely a placeholder in the policy rule.

XACML by OASIS [eXt05] proposes an XML-based language to express and interchange access control policies. XACML is designed to express authorization policies in XML against objects that are themselves identified in XML. Also, XACML specifies the syntax and format of obligations, which by definition are included in the access control policies. In addition to the language, XACML defines both an architecture for the evaluation of policies and a communication protocol for message exchange.

Despite the benefits of all these works, none provides a complete solution for protecting the privacy of users and regulating the use of personal information in secondary applications. Our work tries to fill in this gap and provides

an access control infrastructure that supports users acting in distributed environments in the protection of their privacy and in the management of their information when released to external parties.

11.7 Conclusions

The definition of a privacy-aware access control system that regulates access to data/services still preserving the privacy of the involved parties is an important research direction and a practical pressing need. Existing proposals and traditional access control systems focus on the server-side needs of securing access to their resources. As a consequence, access control models and languages turn out to be very limited from a privacy point of view. We have defined an access control model and language for restricting access to resources/data managed by a service provider and release of PII managed by the users, which take advantage by integration with credentials, ontologies, and context information. Afterwards, we have defined a data handling model and language allowing users to pose restrictions on the secondary use of their private data when they are released to external parties.

In the next chapters, we will focus on related obligation policies and on assurance control policies that capture users' preferences and ensure that users' constraints and expectations (as well as constraints dictated by laws and legislation) can be explicitly represented in a language, and automatically enforced and checked by organizations. Furthermore, we will describe a prototype providing functionalities for integrating access control and data handling policy evaluation and enforcement together with a solution for obligation management and enforcement, and a solution for privacy compliance checking.