E-P3P Privacy Policies and Privacy Authorization

Paul Ashley
IBM Software Group, Australia
pashley@au1.ibm.com

Satoshi Hada
IBM Tokyo Research Laboratory, Japan
satoshih@jp.ibm.com

Günter Karjoth, Matthias Schunter
IBM Zurich Research Laboratory, Switzerland
{gka,mts}@zurich.ibm.com

ABSTRACT

Enterprises collect large amounts of personal data from their customers. To ease privacy concerns, enterprises publish privacy statements that outline how data is used and shared. The Platform for Enterprise Privacy Practices (E-P3P) defines a fine-grained privacy policy model. A Chief Privacy Officer can use E-P3P to formalize the desired enterprise-internal handling of collected data. A particular data user is then allowed to use certain collected data for a given purpose if and only if the E-P3P authorization engine allows this request based on the applicable E-P3P policy. By enforcing such formalized privacy practices, E-P3P enables enterprises to keep their promises and prevent accidental privacy violations.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Access Controls*; E.3 [**Data Encryption**]: Standards; K.4.1 [**Computers and Society**]: Public Policy Issues—*Privacy*

General Terms

Management,Security,Standardization

Keywords

Privacy Policies, Privacy Manager, E-P3P

1. INTRODUCTION

Enterprises store a variety of personally identifiable information (PII) on their customers for their business. As a consequence enterprises publish privacy statements that promise fair information practices. Written in natural language or described using P3P [12], they merely constitute privacy promises and are not necessarily backed up by technological means.

In order to enforce these privacy promises inside an enterprise, the Platform for Enterprise Privacy Practices (E-P3P) defines the enterprise privacy enforcement system for enterprise-internal privacy policies [8]. Unlike P3P, E-P3P has a well-defined privacy-architecture and semantics. This enables an enterprise to internally enforce the E-P3P privacy policies while promising a P3P statement to its customers. Whenever an enterprise collects, stores, or processes PII, E-P3P can be used to ensure that the data flows and usage practices of an enterprise comply with the privacy statement of that enterprise.

For example, assume that an enterprise's database already stores customer's email addresses and that the E-P3P privacy policy says that the marketing department is allowed to use the stored email addresses, but the sales department is not allowed to do that. When applications access email addresses, the E-P3P policy is always enforced. That is, if the application is used by an employee in the marketing department then the access is authorized, but if it would be used by a sales department employee, then access would be denied.

E-P3P cannot prevent untrusted systems or administrator from using PII in an unauthorized way. However, it can help enterprises to maintain PII in compliance with laws and privacy policies.

In this paper, we describe the formal model of E-P3P privacy policy language, outline the XML syntax of the language with some selected details, and describe the authorization engine processing a policy.

1.1 Related Work

The Platform for Enterprise Privacy Practices (E-P3P) was proposed in [8]. The paper described the basic concept but no detailed syntax and semantics of E-P3P. Most building blocks of E-P3P policies have been introduced as features of access control policies. Examples are obligations [3, 10], purpose hierarchies [4], subject and object hierarchies [6], precedences, or conditions on context [10]. The core of new privacy-specific access control languages [5, 7, 8] are the notions of purpose and purpose-binding together with a privacy-specific combination of other advanced access control features.

E-P3P is similar to APPEL [13] in the sense that both languages define the syntax and semantics for describing privacy control rules. However, they are designed for different purposes. APPEL is used by end users to describe their individual preferences. On the other hand E-P3P is used by enterprises to describe their internal privacy policies.

The Platform for Privacy Preferences (P3P) standard of W3C [12] enables a Web site to declare what kind of data is collected and how this data will be used. A P3P policy may contain the purposes, the recipients, the retention pol-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WPES'02, November 21, 2002, Washington, DC, USA.
Copyright 2002 ACM 1-58113-633-1/02/0011 ...$5.00.

icy, and a textual explanation of why this data is needed. P3P defines standardized categories for each kind of information included in a policy. Unlike P3P, E-P3P defines the privacy practices that are implemented inside an enterprise. Since practices depend on internal details of the enterprise, the syntax and semantics of E-P3P result in much more detailed policies that can be enforced and audited automatically. However, the resulting privacy guarantees can be simplified as a P3P promise that is offered for the users of the services. Therefore, P3P is used to publish an enterprise privacy policy in order to collect PII from the customers while E-P3P is used for internally enforcing the enterprise policy to control accesses to the collected PII. In this sense, E-P3P complements P3P. The automated translation of E-P3P privacy practices into P3P promises has been examined in [11].

Issues relating to the practical deployment of E-P3P in an enterprise are discussed in [2].

1.2 Organization

In Section 2, we describe the formal model of E-P3P privacy policies and privacy authorization. This includes definitions of the terminology as well as the rules of a policy. In Section 3, we discuss some details of the XML-based syntax for E-P3P Privacy Policies that is used to enable interoperability of privacy products. In Section 4, we describe the E-P3P authorization engine and its interfaces. Section 5 concludes the paper.

2. MODEL AND SEMANTICS

E-P3P privacy policies define the purposes for which collected data (PII) can be used, model the consent a data subject can give, and may impose obligations onto the enterprise. In this section, we present a refined and formalized notion of the E-P3P privacy policies that has been outlined in [8]. This definition models the elements and semantics of an E-P3P privacy policy.

A E-P3P privacy policy defines what data users can perform what actions for what purposes on which data categories. For example, a policy can define that the 'marketing department' can read the data category 'customer record' for purpose 'e-mail notification'. In Section 2.1, we define the syntax of an E-P3P privacy policy. In Section 2.2, we define the semantics of processing simple requests. A simple request is an access request with one element of each type. The result of evaluating a simple request is a decision whether the access is allowed or denied. In Section 2.3, we define the semantics of a compound request, which contains multiple elements of each type. Multiple data users indicate that the subject accessing the data can play multiple roles. If one role can access the data then the compound request is granted. Multiple data categories indicate that a subject wants to access data belonging to multiple categories, such as multiple columns of a database. Multiple purposes indicate that a subject wants to access data for different purposes. In these cases, the compound request is granted if all categories/actions/purposes are allowed.

2.1 Elements of a Privacy Policy

An enterprise privacy policy can be formalized by a pair $\textbf{Policy} := (\textbf{Terms}, \textbf{Rules})$ that contains a terminology used in the policy and a set of authorization rules. Unlike P3P, an E-P3P policy self-defines its terminology.

The terminology **Terms** consists of the six elements (DCH, PH, DUH, A, O, C). DCH, PH, and DUH define the hierarchies of data categories, purposes, and data users. A, O, and C define the sets of actions, obligations, and conditions.

Data categories $DCH := (T, \geq_T)$ are structured in a tree hierarchy, where T is the set of data categories and $\geq_T$ defines a relationship between two data categories. $t \geq_T t'$ means that t is an ancestor data category of t' where each parent data category groups its child data categories. Examples are 'email' or 'financial data' that are children of 'any data'. The basic idea of using a hierarchical data structure is that this enables to refine the meaning of elements in a hierarchical sense.

Similarly, purposes and data users are structured in a tree hierarchy $PH := (P, \geq_P)$ and $DUH := (U, \geq_U)$, respectively. Examples of purposes are 'marketing' or 'sales', which are children of 'business purposes'. Examples of data users are 'sales department' or 'marketing department', which are children of the enterprise 'Borderless books'.

The set $\textbf{Rules} := (\mathit{Ruleset}, dr)$ contains a set $\mathit{Ruleset}$ of authorization rules that allow or deny an action as well as an element dr that defines the default ruling (permission) of this rule-set. Allowed default rulings are $\{+, \emptyset, -, \times\}$ representing 'allow', 'don't care', 'deny', and 'error'.

An authorization rule $(i, t, p, u, r, a, \overline{o}, \overline{c}) \in \mathit{Ruleset}$ consists of the following eight elements:

Precedence $i \in \mathbb{Z}$ The precedence of the rule. Higher precedence rules unconditionally overrule lower precedence ones.[1]

Data Category $t \in T$ The category of data that may be accessed. Examples include "contact data" or "medical measurements".

Purpose $p \in P$ The purpose for which data may be used.

Data User $u \in U$ The data user that may use the data. Examples can be an entity inside an enterprise such as "John Doe" or "Marketing Department".

Ruling $r \in \{+, -\}$ The ruling (permission) of the rule. A rule either allows $(+)$ or denies $(-)$ the action. The rulings '$\emptyset$' and '$\times$' are only allowed as a default ruling but not inside a rule.

Action $a \in A$ The action that may be performed on the data. Examples are 'read', 'write', or 'disclose'.

Obligations $\overline{o} \subseteq O$ The duties that the rule mandates. A rule can contain a set $\overline{o}$ of obligations that need to be performed. The empty obligation is denoted by $\emptyset$. This obligation defines that the rule does not impose duties on the enterprise. Examples for obligations are "limited retention", "log this access", or "notify data subject".

Conditions $\overline{c} \subseteq C$ The conditions under which the rule is applied. A rule can contain a set of Boolean conditions that are combined by a logical and. Rules where at least one of the conditions is not satisfied will be

[1]Note that E-P3P is a technical interchange format. Since policies with precedences are difficult to understand, policy editors will usually use other representations for prioritizing rules.

ignored. Examples are opt-in or opt-out choices or conditions on enterprise data (such as a credit-limit). Unconditional rules contain no condition. This is denoted by $\emptyset$.

2.2 Formal Semantics of Simple Requests

A simple request is a request $(t_R, p_R, u_R, a_R) \in T \times P \times U \times A$ with one element of each type. The semantics of a privacy policy **Policy** = (**Terms**, (**Ruleset**, dr)) processes a request based on the set of authorization rules whose conditions are satisfied. This process is split into a pre-processing stage and a request processing stage.

The pre-processing stage creates a set of preliminary authorization rules PA as follows:

Direct Authorization For each rule $(i, t, p, u, r, a, \overline{o}, \overline{c}) \in$ **Ruleset**, a tuple $(i, t, p, u, r, a, \overline{o}, \overline{c})$ is added to PA.

Down-Inheritance For each rule $(i, t, p, u, r, a, \overline{o}, \overline{c}) \in PA$, for every (t', p', u') such that $t \geq_T t'$, $p \geq_P p'$, and $u \geq_U u'$, a tuple $(i, t', p', u', r, a, \overline{o}, \overline{c})$ is added to PA.

Up-Inheritance of Deny For each rule $(i, t, p, u, -, a, \overline{o}, \overline{c}) \in PA$, for every (t', p', u') such that $t' \geq_T t$, $p' \geq_P p$, and $u' \geq_U u$, a tuple $(i, t', p', u', -, a, \overline{o}, \overline{c})$ is added to PA.

Note that 'deny'-rules are inherited both downward and upward along the three hierarchies while 'allow'-rules are inherited only downward.

After these pre-processing steps, a request $(t_R, p_R, u_R, a_R) \in T \times P \times U \times A$ is processed as follows using a copy PA' of PA:

Process Conditions All tuples[2] $(i, t, p, u, r, a, \overline{o}, \overline{c}) \in PA'$ such that $\overline{c}$ contains an unsatisfied condition are removed from PA'.

Denial Takes Precedence If a tuple $(i, t, p, u, -, a, \overline{o}, \overline{c}) \in PA'$ exists, all tuples $(i, t, p, u, +, a, \overline{o}', \overline{c}') \in PA'$ with any $\overline{o}'$, $\overline{c}'$ are removed.[3]

Process the Simple Request For each precedence level i^* (starting from the highest one), select all tuples $(i^*, t_R, p_R, u_R, r, a_R, \overline{o}, \overline{c}) \in PA'$. If such tuples exist, the pair $R := (r, RO)$ is returned, where $RO := \{\overline{o} | (i^*, t_R, p_R, u_R, r, a_R, \overline{o}, \overline{c}) \in PA'\}$. Note that RO is a set of obligation sets. [4] Else, the next lower precedence level $i^* - 1$ is processed.

Apply the Default Ruling If there is no such tuple at any precedence level (i.e., i^* is lower than any precedence of any rule), the default ruling with the empty obligation $\{(\mathrm{dr}, \emptyset)\}$ is returned.

If the policy is non-deterministic and the semantics returns a set RO containing a set with multiple sets of obligations. In this case, the implementation can choose any contained element (i.e., any of the obligation sets) to be enforced.

[2]We talk about tuples since one rule is usually decomposed into many tuples.

[3]Processing "denial takes precedence" is performed only on the remaining rules with satisfied conditions. As a consequence, it must be done at run-time.

[4]If the policy is deterministic, this set RO will contain only one element. Non-deterministic policies produce sets with identical rulings but different obligations.

2.3 Formal Semantics of Compound Requests

A compound request is a request $(\overline{t_R}, \overline{p_R}, \overline{u_R}, \overline{a_R})$ with sets of elements of each type. The semantics is defined by decomposing the compound into simple requests and then recomposing the answer. The intuitive semantics of a compound request is to check whether **any** of the data users is allowed to access **all** categories, for **all** purpose, and **all** actions. For a given data user that is allowed or denied to perform the action, the resulting obligations are collected from all applicable rules and then returned. Note that this semantic definition is not optimized for performance. Since implementations need not implement this exact algorithm, they can further optimize.

2.3.1 Processing a Single Data User

Let

$$\mathrm{auth} : T \times P \times U \times A \to \{\times, +, -, \emptyset\} \times \mathcal{P}(O)$$

be the function defined by the semantics for processing simple requests. We denote $\mathrm{auth}^{\mathrm{r}}(t, p, u, a) \to r$ as the ruling part and $\mathrm{auth}^{\mathrm{o}}(t, p, u, a) \to \overline{o}$ as the obligation part of the result. The processing of a compound request $(\overline{t_R}, \overline{p_R}, u, \overline{a_R})$ for a single data user u is $\mathrm{auth}_{\mathrm{C}}(\overline{t_R}, \overline{p_R}, u, \overline{a_R})$ as defined in Table 1.

2.3.2 Processing Multiple Data Users

Let $\mathrm{auth}_{\mathrm{C}}(\overline{t}, \overline{r}, u, \overline{a}) \to (r, \overline{\overline{O}})$ be the function defined by processing compound requests for a single data user. The processing of a compound request $(\overline{t_R}, \overline{p_R}, \overline{u_R}, \overline{a_R})$ for multiple data users $\overline{u_R}$ is $\mathrm{auth}_{\mathrm{C}}(\overline{t_R}, \overline{p_R}, \overline{u_R}, \overline{a_R})$ as defined in Table 1.

3. SELECTED DETAILS OF THE E-P3P LANGUAGE

In this section, we outline some implementation details of the E-P3P XML format that might be interesting for designing similar languages.

3.1 Language Details

E-P3P is an XML-based language. The formal syntax in Section 2.1 has been translated into an XML schema that can be used to validate E-P3P policies. An E-P3P policy has one section defining the terminology and one section defining the rules. Rules cross-reference element definitions by means of ID/IDREF pairs. A rule may specify one or more identifiers for each element (data user, data category, purpose, and action), and allows or denies the cross-products of the specified identifiers. Besides terms and rules, an E-P3P policy is augmented with policy-information such as an issuer or an expiry date.

3.2 Deployment Mapping

Unlike P3P [12], E-P3P does not assume a pre-defined terminology but defines all the identifiers that can be used in authorization rules. The reason for this decision is that E-P3P will be used by a variety of enterprises in different sectors. There exists no terminology such that one fits all. As a consequence, each policy first declares its terminology.

In order to enforce a policy, the privacy enforcement system is required to understand the particular terminology used by the policy to be enforced. In order to enable interoperation, multiple systems need to agree on the terminol-

$$\text{auth}_{\text{C}}(\overline{t_R}, \overline{p_R}, u, \overline{a_R}) := \begin{cases} (\times, \emptyset) & \text{if} & \exists (t,p,u,a) \in (\overline{t_R}, \overline{p_R}, u, \overline{a_R}) : \text{auth}^{\text{r}}(t,p,u,a) = \times \\ (\emptyset, \emptyset) & \text{elseif} & \forall (t,p,u,a) \in (\overline{t_R}, \overline{p_R}, u, \overline{a_R}) : \text{auth}^{\text{r}}(t,p,u,a) = \emptyset \\ (+, \overline{\overline{O}}) & \text{elseif} & \forall (t,p,u,a) \in (\overline{t_R}, \overline{p_R}, u, \overline{a_R}) : \text{auth}^{\text{r}}(t,p,u,a) \in \{\emptyset, +\} \\ & \text{with } \overline{\overline{O}} := \left\{ \begin{array}{l} \text{auth}^{\text{o}}(t,p,u,a) \mid (t,p,u,a) \in (\overline{t_R}, \overline{p_R}, u, \overline{a_R}); \\ \qquad\qquad\qquad \mid \text{and auth}^{\text{r}}(t,p,u,a) = + \end{array} \right\} \\ (-, \overline{\overline{O}}) & \text{else} & \\ & \text{with } \overline{\overline{O}} := \left\{ \begin{array}{l} \text{auth}^{\text{o}}(t,p,u,a) \mid (t,p,u,a) \in (\overline{t_R}, \overline{p_R}, u, \overline{a_R}); \\ \qquad\qquad\qquad \mid \text{and auth}^{\text{r}}(t,p,u,a) = - \end{array} \right\} \end{cases}$$

$$\text{auth}_{\text{C}}(\overline{t_R}, \overline{p_R}, \overline{u_R}, \overline{a_R}) := \begin{cases} (+, \overline{\overline{O}}) & \text{if} & \exists u \in \overline{u_R} : \text{auth}_{\text{C}}(\overline{t_R}, \overline{p_R}, u, \overline{a_R}) = (+, \overline{\overline{O}}) \\ (-, \overline{\overline{O}}) & \text{elseif} & \exists u \in \overline{u_R} : \text{auth}_{\text{C}}(\overline{t_R}, \overline{p_R}, u, \overline{a_R}) = (-, \overline{\overline{O}}) \\ (\times, \emptyset) & \text{elseif} & \exists u \in \overline{u_R} : \text{auth}_{\text{C}}(\overline{t_R}, \overline{p_R}, u, \overline{a_R}) = (\times, \emptyset) \\ (\emptyset, \emptyset) & \text{else} & \exists u \in \overline{u_R} : \text{auth}_{\text{C}}(\overline{t_R}, \overline{p_R}, u, \overline{a_R}) = (\emptyset, \emptyset) \end{cases}$$

Table 1: Semantics for Compound Requests with Single and Multiple Data Users

ogy before actually exchanging privacy policies. This agreement on a common terminology can be enterprise- or sector-specific. Each enterprise then needs to map its (local) terminology onto the commonly agreed one using a so-called deployment mapping. Negotiating and agreeing such ontologies is currently done off-line among the enterprises using the language. However, by including the terminology, E-P3P can be used to verify that both enterprises agreed on a common terminology.

Take for example the data category. In an enterprise application, PII may be stored in an XML format, in a relational database, or in another proprietary format. In all cases, the application needs to define which part of the stored PII should map to each data category in the particular terminology.

3.3 Using XSLT as a Condition Language

Privacy authorizations often depend on context data that is stored outside the authorization engine. Examples are opt-in or opt-out choices that may be part of a customer record or information about the person accessing the data. In order to enable a policy to allow or deny access based on this data. In order to re-use language standards and to minimize the implementation effort by using existing tools, we have decided to use XSLT to formalize the conditions. Another reason is that XSLT can easily be extended by adding user-defined functions that can interface to the environment of an enterprise. The basic idea is that context data is provided as XML. This XML is then translated into `<TRUE/>` if and only if the Boolean condition is satisfied.

3.3.1 Defining Conditions

An E-P3P policy defines conditions that can evaluate containers. A container definition defines the context data required to evaluate conditions. It contains a list of application-specific attribute names that can be multi-valued and abstract from the actual data model of the enterprise.

A condition definition then defines a particular condition. It identifies the containers whose data it evaluates. In addition, it contains an XML stylesheet that implements the condition. Once the condition is defined, it can be referenced via their identifiers in any rule.

3.3.2 Evaluating Conditions

When a condition specified in a rule is evaluated, we need to look at the condition definition. Before actually applying the XSLT stylesheet to the context data, the attributes of the required containers are structured in a two-level XML format as follows:

1. Create a top-level element called "XmlADI", where ADI stands for access decision information.
2. For each specified container, add a child element with the container's identifier as its element name.
3. For each attribute name/value pair contained by the container, add a child element that has the attribute identifier as its element name and the value as its textual content. Recall that an attribute can have multi values.

These containers can be validated using the syntax defined in the corresponding container definitions. After creating the `<XmlADI>` element, the XSLT stylesheet is then used to translate the ADI document into either `<TRUE/>` or anything else. If the resulting XML is `<TRUE/>`, the condition is evaluated to be true. Otherwise, it is evaluated to be false.

3.3.3 Example

We now consider an example that illustrates the use of XSLT to implement Boolean conditions. Consider the following two container definitions. The two containers contain data about the user who currently accesses the data and the patient record of the patient who's data is accessed.

```
<container id="DataUserInfo">
  <attribute id="DataUserID"
             maxOccurs="1"
             minOccurs="1"
             simpleType="xsd:string"/>
  <attribute id="WorkingOnStations"
             maxOccurs="unbounded"
             minOccurs="1"
             simpleType="xsd:string"/>
  <attribute id="OnDuty"
             maxOccurs="1"
             minOccurs="1"
             simpleType="xsd:boolean"/>
</container>
<container id="PatientRecord">
```

```
  <attribute id="Station"
             maxOccurs="1"
             minOccurs="1"
             simpleType="xsd:string"/>
  <attribute id="PrimaryDoctorID"
             maxOccurs="unbounded"
             minOccurs="1"
             simpleType="xsd:string"/>
</container>
```

We use the <XmlADI> element to enclose the list of containers as described above. The ADI document corresponding to the above container definitions may look as follows:

```
<XmlADI>
  <DataUserInfo>
    <DataUserID>Jane Doe</DataUserID>
    <WorkingOnStations>50B</WorkingOnStations>
    <WorkingOnStations>ER</WorkingOnStations>
    <OnDuty>true</OnDuty>
  </DataUserInfo>
  <PatientRecord>
    <Station>50B</Station>
    <PrimaryDoctorID>John Doe</PrimaryDoctorID>
    <PrimaryDoctorID>Bill Doc</PrimaryDoctorID>
  </PatientRecord>
</XmlADI>
```

We now evaluate this data using the following XSLT-based condition. The condition expressed by the stylesheet is that the rule only applies if the data user (a nurse) is working on a station in the same area where the patient lies and that the nurse needs to be on duty.

```
<condition id="check">
  <evaluates-container refid="DataUserInfo">
  <evaluates-container refid="PatientRecord">
  <xsl:stylesheet
    version="1.0"
    <xsl:output method="xml" indent="no"/>
    <xsl:strip-space elements="*"/>
    <xsl:template match="/XmlADI">
      <xsl:if test="
       PatientRecord/Station
         =DataUserInfo/WorkingOnStations
       and DataUserInfo/OnDuty">
        <TRUE/>
      </xsl:if>
    </xsl:template>
  </xsl:stylesheet>
</condition>
```

3.4 Configurable Obligation Elements

Obligations are duties that are imposed by the privacy policy onto the enterprise. Examples are auditing, logging, limited retention, notifications, or collecting additional consent.

An obligation usually corresponds to executing an implementation that is external to the authorization engine. In order to enable flexibility, the E-P3P rules can identify obligations while specifying multi-valued parameters to be input to this external implementation. The syntax of each parameter is defined in the same way as the attributes in a container definition are defined. In the following example, the 'retention' obligation specifies a parameter 'days' that is used to signal that data shall be deleted after the number of days specified by the parameter.

```
<obligation id="retention">
  <parameter
    id="days"
    simpleType="xsd:positiveInteger"
    minOccurs="1"
    maxOccurs="1"/>
</obligation>
```

This obligation with the parameter instances is then used in an E-P3P rule as follows:

```
<obligation refid="retention">
  <parameter refid="days">30</parameter>
</obligation>
```

The advantage of having parameterized obligations is to enable a wider range of behaviors with one pre-installed obligation and with only one piece of code that implements it.

4. THE E-P3P AUTHORIZATION ENGINE

In this section, we describe the E-P3P authorization engine. We focus on the authorization algorithm and the authorization interface. The described algorithm only handles simple requests but an extension for compound request processing is quite straightforward.

4.1 Authorization Algorithm

The E-P3P authorization engine receives access requests, which consist of a single data user, a single data category, a single purpose, a single action, and context data as defined in the policy. It outputs a ruling ('allow', 'deny', 'none', or 'error'), the rule identifier that mandates the ruling, and the list of obligations that are specified in the rule. In case the privacy policy mandates the default ruling, both the rule identifier and the obligation list are empty.

The algorithm presented in Section 2 defines the E-P3P semantics. However, there are more efficient ways to implement this algorithm. In the following, we describe an authorization algorithm to evaluate an E-P3P policy according to the E-P3P policy semantics, which requires less computation time. This algorithm also consists of a pre-processing phase, which optimizes the policy data structure, and a request processing phase, in which the access decision is computed.

4.1.1 Pre-processing Phase

Goal of the access decision function is to find in the E-P3P rule set an authorization rule $(i, t, p, u, r, a, \overline{o}, \overline{c}) \in \mathit{Ruleset}$, which allows (denies) the given access request, thereby checking that there is no other rule that would deny (allow) the same request. Our approach to simplify the processing of an access request is the pre-computation of the inheritance effects followed by a clustering of related rules to avoid a sequential search through the rule set.

Recall that 'allow'- and 'deny'-rules are inherited downward along the hierarchies of data user, data category and purpose and also 'deny'-rules are inherited upward. In the pre-processing phase, the authorization engine first removes this inheritance. Then it creates a hash table that maps a 4-tuple (data user, data category, purpose, action) to the list of rules associated with this 4-tuple. When filled the table can be used to retrieve rules using such 4-tuples as search keys. Specifically, the pre-processing is done as follows:

1. For every rule, transform it to *simple* rules in the cross-product, where a rule is called simple if it specifies a single data user, a single data category, a single purpose, and a single action.

2. Add all the simple rules to the hash table. For every simple request, add the authorization rules generated via inheritance to the hash table.

3. For each 4-tuple key, sort the list of the associated rules so that 'deny'-rules appear before 'allow'-rules and rules with higher precedence appear before rules with lower precedence.

Note that the sorting of the associated rules enables efficient request processing. When a matching rule is found, there is no need to search through the rest of the associated rule set as no other rule can change the result.

4.1.2 Request Processing

A given simple access request is evaluated as follows:

1. From the hash table, retrieve the *sorted* list of rules that matches the input 4-tuple.

2. If the list is not empty, then process all the rules in their order. For each rule, do the following:

   (a) Verify all the conditions (if any). If a required container is missing output 'error' and stop. If the condition is not satisfied skip the rule else continue processing the rule.

   (b) If the rule is a 'deny'-rule return its rule identifier and its list of obligations. The resulting ruling is 'deny'.

   (c) If the rule is an 'allow'-rule return its rule identifier and its list of obligations. The resulting ruling is 'allow'.

3. As no error occurred nor a rule was found that allowed respectively denied the request, return the default-ruling with no rule identifier and no obligation list.

To process a compound request, the compound request is first decomposed into simple requests and the individual answers are then combined into the answer as defined in Section 2.3.

4.2 Design Details and Java Interfaces

We have defined Java APIs to the authorization engine. The in- and outputs correspond to the in- and outputs of the algorithm as described in Section 4.

The Java APIs are structured into four main interfaces named Policy, ContainerProvider, AuthznEngine, and AuthznResult. The major idea behind the interface design is that the ContainerProvider interface defines a call-back method to dynamically retrieve context data that is needed to instantiate the XmlADI documents to evaluate the XSLT-based conditions. That is, the call-back method is invoked from the AuthznEngine to evaluate given requests. For efficiency reasons, we decided to use this call-back interface: Usually, only some conditions will be evaluated and only some containers will be needed. As a consequence, requiring data for all the containers as input would decrease the efficiency.

Our interfaces are independent of any standard XML interfaces including DOM, SAX, and JAXP. We implemented a JAXP- and DOM-based implementation using Xerces and Xalan. Below is pseudo Java code describing how to use these interfaces for a request processing.

```
//Create a Document object representing
//an E-P3P policy.
 org.w3c.dom.Document ep3p = ...

//Construct a Policy instance, where we assume
//that we have a DOM-based implementation named
//PolicyImpl. The pre-processing is done here.
 Policy policy=new PolicyImpl(ep3p);

//Get the engine to evaluate the E-P3P policy.
 AuthznEngine engine = policy.getAuthznEngine();

//Once the engine is created, you can use it
//to evaluate requests as many times as you like.
//However, you have to create a container
//provider before that.
 ContainerProvider containerCallBackObject =
   new ContainerProvider(...);
 AuthznResult result = engine.isAuthorized(
     "SalesDepartment",
     "TeleMarketing",
     "Read",
     "CustomerRecord",
     containerCallBackObject);
//Now you can examine the result.
```

In our implementation, once the hash table is created in the pre-processing phase, the expected time to evaluate a policy is constant independent of the number of rules. Actually, a request evaluation for a simple policy using our implementation shows that more than 98 percent of the evaluation time is consumed by Xalan to evaluate the XSLT-based conditions. As a consequence, one may investigate alternatives to XSLT-based conditions. Alternative approaches are the definition of predicates and functions as done by XACL [9] or the use of well-defined call-back methods that evaluate the conditions of the policy.

5. CONCLUSION

In this article, we described the semantics of the E-P3P language for enterprise privacy policies. Formalized and strictly enforced privacy practices enable enterprises to provide the level of privacy promised using privacy statements. In addition, accidental or fraudulent violations of privacy can be reduced to a large extend.

The authorization engine requires the identification of data user, action, and purpose of an privacy-relevant access. As a consequence, we assume that an enterprise deploys an authentication system that identifies these elements on top of E-P3P.

An issue that can be difficult in practice is determining the purpose of an access. *Privacy-enabled* applications can provide the purpose and will respect the decision of the E-P3P engine. Determining purpose and enforcing the decision for legacy systems is more challenging. If application and storage system interact using a pre-defined interface such as SQL [1], the system is required to identify purposes based on other accessible attributes of the context of the request. As a consequence, the resolution of purpose may be coarser than desired.

Acknowledgments

Many IBM colleagues have contributed to the development of the E-P3P language. In particular, we would like to thank Calvin Powers for requirements and continuous guidance.

This work has been partially funded by the IBM Privacy Institute (see `www.research.ibm.com/privacy`).

6. REFERENCES

[1] I. 9075:1992. Information technology — database languages — sql, 1992. ISO Standard.

[2] P. Ashley, M. Schunter, and C. Powers. From privacy promises to privacy management — a new approach for enforcing privacy throughout an enterprise. In *ACM New Security Paradigms Workshop (NSPW2002)*, page to appear, Virginia Beach, VA, USA, Oct. 2002. ACM Press. To appear.

[3] C. Bettini. Obligation monitoring in policy management. In *3rd International IEEE Workshop on Policies for Distributed Systems and Networks (Policy 2002)*, pages 2–12, Monterey CA, June 2002.

[4] P. Bonatti, E. Damiani, S. D. C. di Vimercati, and P. Samarati. A component-based architecture for secure data publication. In *17th Annual Computer Security Applications Conference*, 2001.

[5] S. Fischer-Hübner. *IT-Security and Privacy : Design and Use of Privacy-Enhancing Security Mechanisms*. Number 1958 in Lecture Notes in Computer Science (LNCS). Springer Verlag, Berlin, 2001.

[6] S. Jajodia, P. Samarati, M. L. Sapino, and V. Subrahmanian. Flexible support for multiple access control policies. *ACM Transactions on Database Systems*, 26(4):216–260, June 2001.

[7] G. Karjoth and M. Schunter. A privacy policy model for enterprises. In *15th IEEE Computer Security Foundations Workshop*, pages 271–281. IEEE Computer Society Press, 2002.

[8] G. Karjoth, M. Schunter, and M. Waidner. The Platform For Enterprise Privacy Practices – Privacy-enabled Management Of Customer Data. In *Proceedings of the Privacy Enhancing Technologies Conference*, page to appear, San Francisco, CA, April 14-15 2002.

[9] M. Kudo and S. Hada. XML document security based on provisional authorizations. In *7th ACM Conference on Computer and Communications Security*, pages 87–96. ACM Press, 2000.

[10] OASIS. eXtensible Access Control Markup Language (xacml), 2002. Available at `www.oasis-open.org/committees/xacml`.

[11] M. Schunter and E. Van Herreweghen. Enterprise privacy practices vs. privacy promises — how to promise what you can keep. Research Report RZ 3452 (# 93771), IBM Research, Sept. 2002.

[12] W3C. Platform for Privacy Preferences. Available at `www.w3.org/P3P`.

[13] W3C. A P3P Preference Exchange Language 1.0 (APPEL1.0), 2002. Available at `www.w3.org/TR/P3P-preferences`.