RZ 3485 (# 93951) 03/03/2003
Computer Science 69 pages

Research Report

Enterprise Privacy Authorization Language (EPAL)

Paul Ashley,[1] Satoshi Hada,[2] Günter Karjoth,[2] Calvin Powers,[1] and Matthias Schunter[2]

[1]IBM Tivoli Software

[2]IBM Research
Zurich Research Laboratory
8803 Rüschlikon
Switzerland

Enterprise Privacy Authorization Language (EPAL)

IBM Research Report

Latest public version:
http://www.zurich.ibm.com/security/enterprise-privacy/epal
Editor:
Matthias Schunter, IBM Research, mts at zurich.ibm.com
Authors (alphabetically):
Paul Ashley, IBM Tivoli Software
Satoshi Hada, IBM Research
Günter Karjoth, IBM Research
Calvin Powers, IBM Tivoli Software
Matthias Schunter, IBM Research
This Version:
$Id: index.html,v 1.73 2003/03/04 13:42:50 mts Exp $
Other Versions:
1.73: Published as IBM Research Report RZ 3485 (#93951), 03/03/2003.
1.72: First stable version on www.zurich.ibm.com

Abstract

This is the Enterprise Privacy Authorization Language (EPAL) technical specification. EPAL is a formal language for writing enterprise privacy policies to govern data handling practices in IT systems according to fine-grained positive and negative authorization rights. It concentrates on the core privacy authorization while abstracting data models and user-authentication from all deployment details such as data model or user-authentication.

An EPAL policy defines lists of hierarchies of data-categories, data-users, and purposes, and sets of (privacy) actions, obligations, and conditions. Data-users are the entities (users/groups) that use collected data (e.g., travel expense department or tax auditor). Data-categories define different categories of collected data that are handled differently from a privacy perspective (e.g., medical-record vs. contact-data). Purposes model the intended service for which data is used (e.g., processing a travel expense reimbursement or auditing purposes).

Actions model how the data is used (e.g., disclose vs. read). Obligations define actions that must be taken by the environment of EPAL (e.g., delete after 30 days or get consent). Conditions are Boolean expressions that evaluate the context (e.g., "the data-user must be an adult" or "the data-user must be the primary care physician of the data-subject").

These elements are then used to formulate privacy authorization rules that allow or deny actions on data-categories by data-users for certain purposes under certain conditions while mandating certain obligations.

EPAL aims at formalizing enterprise-internal privacy policies. While P3P formalizes privacy promises to be advertized (i.e., business to consumer), EPAL formalizes privacy authorization for actual enforcement within an enterprise or for business-to-business privacy control. Feedback and comments are welcome and should be sent to Matthias Schunter `mts (at) zurich.ibm.com`.

Table of Contents

---

1. Introduction

The Enterprise Privacy Authorization Language (EPAL) is an interoperability language for exchanging privacy policy in a structured format between applications or enterprises. This document formally describes EPAL, including concepts, syntax, and semantics. To help readers understand the structure and capabilities of the language, it is presented in several forms. First, a brief overview of the language is given in Section 2 in textual form, explaining the

major structures of the language and how they fit together. Secondly, the detailed syntax of the language will be specified in Sections 3 and 4 by parts of the EPAL schema along with some examples of EPAL. Thirdly, the semantics of an EPAL policy is described in Section 5. Data-types that are re-used in mulriple places are defined in Section 6. The formal definition of the EPAL syntax is given by the XML Schema for EPAL as an appendix.

1.1 Mission Statement

The EPAL Working Group exists to develop a interoperability language for the representation of data handling policies and practices within and between privacy-enabled enterprise tools, which serve to

- enable organizations to be demonstrably compliant with their stated policies;
- reduce overhead and the cost of configuring and enforcing data handling policies; and
- leverage existing standards and technologies.

1.2 Objectives

The goals for the EPAL language are the following.

- Provide the ability to encode an enterprise's privacy-related data-handling policies and practices.
- A language that can be imported and enforced by a privacy-enforcement systems.

There are certain situations for which EPAL is not intended as an appropriate language. EPAL was not designed for the following applications:

- Directly encoding or enforcing specific privacy legislation in a generic and completely application and enterprise independent way.
- Manipulation of EPAL by data subjects to set their preferences: E.g., providing information that enables the creation of user-preferences forms based on the information given in an EPAL policy.
- Creation of new mechanisms or syntax for data representation: EPAL should abstract from the data model and should not provide it's own syntax for describing schemas for data.
- The limitation of features in EPAL for the sake of conformance to the assumptions, data structures, or features of any particular pre-existing product or tool.

1.3 Applications

The following applications expose the design considerations raised during the development of EPAL. For EPAL to satisfy the demands of each use case, particular requirements must be inferred. These case-specific requirements and the accompanying assumptions are listed below.

1.3.1 Rule Creation by a Privacy Administrator

A privacy administrator, who has a legal background and no programming experience, creates EPAL rules to model her firm's privacy policy. This EPAL policy serves as input to a variety of privacy management, audit, and enforcement software tools.

Implications on EPAL requirements:

- EPAL rules must be expressive enough to reflect the complexity of privacy policy.
- EPAL rules must be editable to manage the model of the firm's evolving privacy policy.
- The processes of rule creation and rule editing should be intuitive.
- Conflicting rules are possible, defined as two EPAL rules in the same privacy policy that may return different rulings for some request.
- Some mechanism must be available to resolve conflicting rules.
- Assumption: Non-technical users will write EPAL with some rule creation/management tool.

1.3.2 Interoperability of Privacy Software Products

To allow privacy software from different companies to work together, EPAL should be used as a common format to exchange a privacy policy. For example, a privacy creation tool from one company may create an EPAL policy, and a privacy enforcement tool from another company may read-in the EPAL policy and then enforce it.

Implication on EPAL requirements:

- There can be no ambiguity in the rules. All of the elements and rules must be clearly defined so that there is no confusion in the meaning of terms.
- All extensions to the language must be carefully considered so that interoperability is still possible even when using extensions.
- EPAL must be independent of actual products. In particular, it must not require a specific data model or format in which the collected data is stored.

1.3.3 Privacy Enforcement

A company has implemented an privacy enforcement system in their software package. The privacy enforcement system must enforce the given set of EPAL rules and the obligations that are defined by the policy.

Implications on EPAL requirements:

- Rules must consist of abstract types of objects, and not the actual instances of objects. A mapping from the types to the instances is implementation specific, and beyond the scope of EPAL. This makes EPAL deployable in legacy systems.

- A set of EPAL rules must be designed to enable software to generate a ruling *in real-time* from a given request.
- The privacy enforcement system must be able to either enforce obligations itself or else distribute obligations to the application that is responsible for enforcing them.

1.3.4 A Privacy Audit

An independent auditor uses EPAL to model a client's privacy policy and to perform a privacy audit. The client's goal is to have demonstrated compliance with privacy legislation and recommended industry-specific practices. Two options exist for performing a privacy audit. One method is to generate rulings against the EPAL-modeled privacy policy from run-time requests. These rulings are logged and subsequently analyzed for requests which violated the privacy policy. There is no interference with requests during an audit. An alternative method is completely separate from the client's run-time system. The auditor also models the requests. Compliance with the EPAL-modeled privacy policy is measured against simulated requests.

Implications on EPAL requirements:

- EPAL rule sets must be designed to enable software to generate a ruling given a request and a set of EPAL rules.
- To reduce the cost of the privacy audit, it may be desirable to define the EPAL rules with a "deny" ruling. Together with a default-ruling "allow", any action that is not explicitly denied by an EPAL rule is then allowed by default.
- Assumption: Conflicting rules in a set of EPAL rules must first be resolved before performing a privacy audit. This could be a feature of the rule creation and management tool.

1.4 Requirements

Requirements describe the necessary expressiveness of EPAL, enable and/or facilitate deployment, and guide the design of the structure of the language.

1.4.1 Expressiveness Requirements

EPAL must be expressive and flexible to capture evolving privacy legislation and customized privacy policy in a useful, computer-readable format. The following requirements aim to fulfill these goals.

- The user may specify a default ruling of "allow" or "deny" and may have no default ruling.
- Each rule can allow a set of data users to perform an action in a set of action on a category in a set of category for any purpose in a set of purposes.
- Each rule may contain conditions that must be true for the rule to be active.
- Each rule may specify obligations that must be performed if the action is

performed.
- Conflicts may be resolved by assigning precedence.
- The author of EPAL must be provided with some flexible means of including comments, parameters, and unforeseen additional information.

1.4.2 Deployment-Independence Requirements

EPAL is not designed for any specific target environments or existing software privacy tools. Nevertheless, certain requirements must be satisfied to deploy EPAL.

- The EPAL elements, which constitute a rule, are abstract types. These types are mapped to actual instances of elements during implementation. Not only does this simplify EPAL, but it also ensures that EPAL is deployable in all existing and legacy environments. This implies the following special cases:
  - EPAL does not define how users are authenticated.
  - EPAL does not define how privacy-categories and policies are associated with collected data.
  - EPAL does not define how enterprise activities are mapped onto purposes and privacy-relevant actions.
  - EPAL does not define how data needed for evaluating conditions is stored.
  - EPAL does not define how obligations are implemented.
- It must be feasible to generate a ruling for a given request from a large set of rules in real-time. Examples of rulings include "allow", "deny", and "not-applicable". The processing of rules and the generation of a ruling is implementation-dependent.

1.4.3 Language Requirements

Since EPAL is intended to be an interoperability language, the syntax must satisfy the following requirements.

- The syntax of EPAL must be consistent with widely accepted computer-readable languages to foster simple parsing.
- An EPAL-encoded privacy policy should be intuitively understood, not rendered incomprehensible by the syntax or terminology.
- EPAL must be extensible to accommodate future changes while supporting backwards compatibility whenever possible.

1.5 Requirements on the Environment using EPAL

An EPAL privacy policy and an engine that evaluates it is not sufficient for privacy enforcement. To be deployment independent, EPAL defines an abstract authorization interface that outputs a decision and obligations on input of purpose-identifiers, action-identifiers, data-user-identifiers, data-category-identifiers, and context data. Implicitly, these authorization inputs pose a list of requirements on the environment of the EPAL engine

- The environment must be able to identify the policy and data-categories that govern a particular protected resource that can be accessed.
  Note that EPAL only defines how each data-category is handled. Without this additional information how an actual piece of collected data has been categorized, an EPAL policy cannot be used to evaluate whether collected data is handled in a privacy-friendly way.
- The environment must know the elements used in the policy. This includes:
  - How to reliably authenticate each data-user of the policy.
  - How to correctly map the current activity onto a purpose and privacy-relevant operation of the policy.
- The environment must provide the context data to be input to the EPAL evaluation engine from the actual data-model of the enterprise.
  If the context contains choices, this requires being able to identify the data subject (i.e., the individual that provided the data) and retrieve the corresponding opt-in and -out choices.
- The environment must provide implementations for all obligations of the policy.

1.6 Documentation Conventions and Notation

The normative syntax is given by the XML schema in the appendix. Fragments of this XML Schema are used to define the XML data structures of EPAL inside the specification.

```
<!-- All schema fragments are shown in a box like this. -->
```

Examples illustrate the XML data structures of EPAL as XML code.

```
<!-- All examples are shown in a box like this. -->
```

In general, `a fixed space font` is used to represent excerpts of an EPAL file within the text of this document. Angled brackets <> enclose XML elements. Double quotes "" enclose XML types. Single quotes delimit strings.

1.7 Acknowledgements

We thank the many reviewers within IBM that gave valuable input. In particular we would like to thank Michael Backes, Kathy Bohrer, Michiharu Kudoh, Birgit Pfitzmann, and Michael Waidner.

2 EPAL Overview and Example

This section briefly describes EPAL and its capabilities. An EPAL policy categorizes the data an enterprise holds and the rules which govern the usage of data of each category. Since EPAL is designed to capture privacy policies in many areas of responsibility, the language cannot predefine the elements of a

privacy policy. Therefore, EPAL provides a mechanism for defining the elements which are used to build the policy.

An EPAL policy is essentially a set of privacy rules. A rule is a statement that includes a ruling, a data user, an action, a data category, and a purpose. A rule may also contain conditions and obligations. Rules in a policy are ordered with descending precedence. Rules with a higher precedence unconditionally override rulings of rules with lower precedence. For example, an enterprise may have the following rule in its privacy policy.

privacy policy (informal)	*Allow a sales agent or a sales supervisor to collect a customer's data for order entry if the customer is older than 13 years of age and the customer has been notified of the privacy policy. Delete the data 3 years from now.*
ruling	allow
data user	sales department
action	store
data category	customer-record
purpose	order-processing
condition	the customer is older than 13 years of age
obligation	delete the data 3 years from now

Rules are used to determine if a request is allowed or denied. A request contains a data user, an action, a data category, and a purpose. Continuing with the same enterprise as above, consider the following request.

request (informal)	*A person acting as a sales agent and an employee requests to collect a customer's email for order entry.*
data user	sales department
action	store
data category	customer-record
purpose	order-processing

The above rule allows the request, so the sales agent would be permitted to store the customer's contact information. Additional rules can then govern how this stored data may be used.

2.1 Example of EPAL in Use

As in introduction to EPAL, let us consider one common scenario that makes use of EPAL.

A company *BestShoesRUs* sells shoes to consumers:

1. The shoes are sold through stores throughout North America and Asia (thirty of them),
2. Through a mail catalog that is mailed out to subscribers,
3. Via a 1-800 type phone service (24 hours a day), and
4. Via the Internet (http://www.bestshoesrus.example.com).

BestShoesRUs has always been concerned with its customers privacy. However, as the business has expanded it has found it more difficult to present and enforce a consistent policy across the business. Customers have started to complain that they are getting unwanted marketing material, and some customers have challenged that *BestShoesRUs* has revealed their private data to third parties.

To facilitate better privacy practices *BestShoesRUs* has decided to create an enterprise wide policy using EPAL. The goal of the policy is to define a set of common rules for the handling of PII (Personal Identifiable Information) data that is collected from consumers. This way no matter how the data is collected (store, mail, phone, Internet, North America and Asia) that data is treated in a consistent manner.

To begin the process the CPO and her staff need to define an EPAL policy. The CPO obtains legal advice to obtain requirements for the policy based on legislation and industry regulation. The CPO also contracts some consultants specializing in privacy to get some guidelines around industry accepted practices for handing PII data. Using these two inputs, and also tailoring it for the business requirements of *BestShoesRUs* the CPO and her staff defines the policy. This policy will be the basis for all of the handling of the PII data across the whole business.

Once this policy is defined the IT staff need to ensure it is enforced. There are a number of aspects to this:

- **Notice**: The policy is clearly articulated for the customers. This means publishing a privacy notice (e.g. text and P3P) on the *BestShoesRUs* web site, ensuring that all marketing material and the mail catalog describes the policy in text, and that when the phone sales people are collecting PII data that they make sure the customer is aware of the policy.
- **Consent**: The consent choices that are defined by the EPAL policy can be presented to the customer. The customer can review and update the given consent. For example, the customer should be able to make selections on whether they want to receive email, postal mail or phone calls from *BestShoesRUs* or decide to opt-out of marketing altogether. These consents must be available via the catalog forms, web, phone and the stores.
- **Enforcement**: At all places in the infrastructure where PII data is

collected or used, procedures and software must be in place to enforce the EPAL policy.

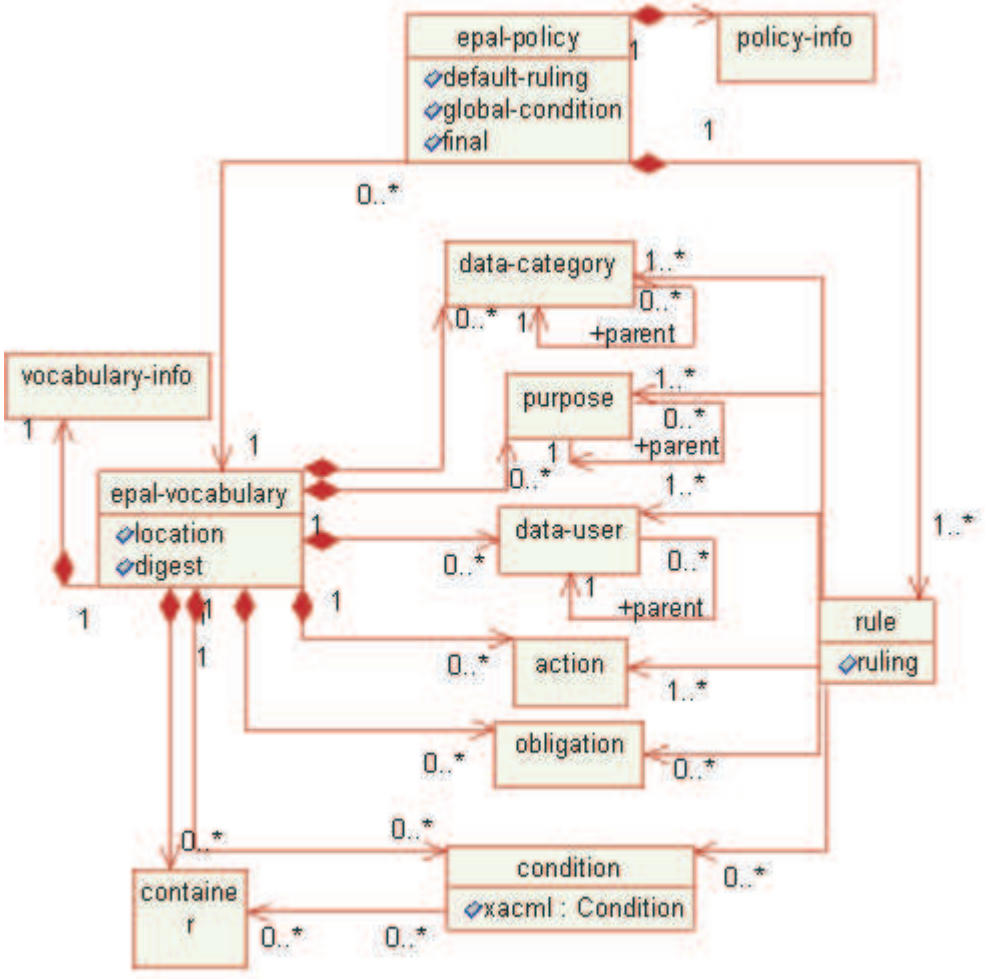
The EPAL policy will be expected to change over time because of new regulations, industry practices and business requirements. New versions of the policy will need to be created.

BestShoesRUs will need to acquire software tools for creation of policy, notice, consents and enforcement. These tools may be acquired from different software vendors. The EPAL policy will be used as the medium for exchanging policy across the tools.

2.2 UML Overview on the EPAL Syntax

The following picture shows a non-normative UML (the Unified Modeling Language) overview over the elements of an EPAL policy. Sub-elements are depicted as aggregation by value while references using id/refid pairs are depicted as ordinary relations. The element `<epal-vocabulary>` and `<epal-policy>` are the two top-level elements of EPAL.

Figure: Non-normative High-level UML Overview of an EPAL policy



2.3 XML Schema of EPAL

EPAL is well-formed XML, conforming to XML 1.0. In addition to well-formedness, EPAL must be validated by the XML Schema for EPAL. The Schema formally defines the EPAL elements. The cardinality, the type, and the (required or optional) use of each element is stated. Any sequencing restrictions of sub-elements are stated. The type, the (required or optional) use, the permitted values, and any default values for each attribute are stated. Some attributes have values which must be unique IDs. An ID must be unique throughout the EPAL policy.

Section 3 and 4 describe the contents of the XML Schema for EPAL. The EPAL namespace URI is "http://www.research.ibm.com/privacy/epal".

Conventional XML namespace prefixes are used throughout the listings in this specification to stand for their respective namespaces as follows, whether or not a namespace declaration is present in the example:

Prefix	Namespace URI	Notes
epal	http://www.research.ibm.com/privacy/epal	The EPAL namespace defined in this specification
xs	http://www.w3.org/2001/XMLSchema	The W3C XML Schema namespace
xsi	http://www.w3.org/2001/XMLSchema-instance	The W3C XML Schema namespace for instances
xacml	urn:oasis:names:tc:xacml:1.0:policy	The XACML policy namespace
xacml-context	urn:oasis:names:tc:xacml:1.0:context	The XACML context namespace

3. EPAL Vocabularies

This section describes the syntax of EPAL vocabularies that define the language to express sector-specific privacy policies. Enterprises usually use a single vocabulary. For exchanging policy data with other enterprises, both enterprises can agree and share a subset of this vocabulary.

3.1 `<epal-vocabulary>` Top-level Element for an EPAL Vocabulary

Syntax: The `<epal-vocabulary>` element defines all elements that will later be referenced in rules. The `<epal-vocabulary>` element contains the following

sub-elements in this fixed order:

1. One `<vocabulary-information>` element of type "infoType" that describes the vocabulary.
2. Zero or more `<data-user>` elements that define the data users of the vocabulary.
3. Zero or more `<data-category>` elements that define the data categories of the vocabulary.
4. Zero or more `<purpose>` elements that define the purposes of the vocabulary.
5. Zero or more `<action>` elements that define the privacy-relevant actions of the vocabulary.
6. Zero or more `<container>` elements that define external context data that can be evaluated by conditions.
7. Zero or more `<obligation>` elements that define the obligations that can be returned by the vocabulary.

The required attribute "version" defines the version of EPAL that has been used.

Semantics: The `<epal-vocabulary>` element lists all terms that must be known by the system interpreting this vocabulary. In addition, it defines what data may be needed for evaluating the conditions contained in this vocabulary.

Design Note: Separating vocabularies from the actual rules serves the following purposes:

1. It enables to fix a vocabulary (and the correspondoing deployment) while allowing flexibility in the rules.
2. It enables compact rules that only reference the elements in the vocabulary.
3. It enables verbose definitions that contain additional information such as human-readable explanations.
4. It enables an authorization engine to verify that it is able to interpret a policy by only looking at the vocabularies.
5. It enables a separation of the stable parts (vocabularies) from the more volatile and customizable parts (rules).
6. It enables more flexible policies since the policy self-defines its terminology. I.e., unlike P3P, the terminology is defined by the actual policy instead of defining it in the spec.

The disadvantage is that for interoperability, one needs to agree on a common vocabulary in order to be able to exchange policies.

Schema Fragment

```
<xs:element name="epal-vocabulary">
<xs:complexType>
    <xs:sequence>
```

```
        <xs:element name="vocabulary-information" minOccurs="1" maxO
        <xs:element name="data-user" minOccurs="0"  maxOccurs="unbou
        <xs:element name="data-category" minOccurs="0" maxOccurs="un
        <xs:element name="purpose" type="epal:hierarchicalType" minO
        <xs:element name="action" type="epal:describedObjectType" ma
        <xs:element name="container" minOccurs="0"  maxOccurs="unbou
        <xs:element name="obligation" minOccurs="0"  maxOccurs="unbo
    </xs:sequence>
    <xs:attribute name="version" type="xs:string" default="1.0"/>
</xs:complexType>
</xs:element>
```

Example: Structure of a <epal-vocabulary> element and its sub-elements

```
<epal-vocabulary version="1.0" xmlns="http://www.research.ibm.com/pr
  <vocabulary-information id="NCName">
    <short-description language="en">short-description</short-descri
    <long-description language="en">long-description</long-descripti
    <issuer>
      <name>name</name>
      <organization>organization</organization>
      <e-mail>e-mail</e-mail>
      <address>address</address>
      <country>country</country>
    </issuer>
    <location>http://www.ibm.com</location>
    <version-info end-date="2001-12-31T12:00:00" last-modified="2001
  </vocabulary-information>
  <data-user id="NCName" parent="NCName">
    <short-description language="en">short-description</short-descri
    <long-description language="en">long-description</long-descripti
    <property id="NCName">
      <value>value</value>
    </property>
    <contact-info>
      <name>name</name>
      <organization>organization</organization>
      <e-mail>e-mail</e-mail>
      <address>address</address>
      <country>country</country>
    </contact-info>
  </data-user>
  <data-category id="NCName" parent="NCName">
    <short-description language="en">short-description</short-descri
    <long-description language="en">long-description</long-descripti
  </data-category>
  <purpose id="NCName" parent="NCName">
    <short-description language="en">short-description</short-descri
    <long-description language="en">long-description</long-descripti
  </purpose>
  <action id="NCName">
    <short-description language="en">short-description</short-descri
    <long-description language="en">long-description</long-descripti
  </action>
  <container id="NCName">
    <short-description language="en">short-description</short-descri
    <long-description language="en">long-description</long-descripti
```

```
      <attribute auditable="true" id="NCName" maxOccurs="1" minOccurs=
      </attribute>
    </container>
    <obligation id="NCName">
      <short-description language="en">short-description</short-descri
      <long-description language="en">long-description</long-descripti
      <parameter id="NCName" maxOccurs="1" minOccurs="1" simpleType="h
      </parameter>
    </obligation>
  </epal-vocabulary>
```

3.2 <vocabulary-information>: Information about the Vocabulary

The <vocabulary-information> of type "infoType" contains information about the vocabulary in the form of three sub-elements. The "id" attribute defines the Name that identifies the vocabulary. The <issuer> describes the issuer, the <version-info> the version and other management information. The "id" attribute of vocabulary-information together with the "revision-number" attribute must uniquely identify a vocabulary for each issuer.

Example: Vocabulary Information inside an EPAL vocabulary

```
  <vocabulary-information
    id="sample-policy">
    <short-description language="en">Example policy information</short
    <long-description language="en">long-description</long-description
    <!-- Entity issuing the policy -->
    <issuer>
      <name>EPAL Working Group</name>
      <organization>IBM Research</organization>
      <e-mail>testmail@zurich.ibm.com</e-mail>
      <address>Saumerstr. 4, 8838 Ruschlikon</address>
      <country>Switzerland</country>
    </issuer>
    <!-- Policy version. This is so the enterprise
         can track versions of its policy. -->
    <version-info
      start-date="2002-03-19T00:00:00"
      test="true"
      last-modified="2002-03-19T00:00:00"
      revision-number="1.02A"
      end-date="2004-03-19T00:00:00"/>
  </vocabulary-information>
```

3.3 <data-user>: Definition of the Data User Hierarchies

Data users are required in a privacy context to represent an actor, a receiver, an initiator, or a data subject. Each data-user in an EPAL policy represents a category of individuals that use data. Data users that represent data subjects (i.e., the owner who's data is collected) define the access rights for the person who provided the data.

Syntax: Each `<data-user>` element defines a data-user that can be referenced in a rule. The data-user element inherits the mandatory attribute "id", the optional attribute "parent", and elements `<short-description>`, `<long-description>`, `and <property>` from the "hierarchicalType". It extends this type by adding an optional element `<contact-info>` of "contactInfoType".

Semantics: The list of `<data-user>` elements defines the entities accessing the data. A `<data-user>` describes an entity accessing some data and can reference its parent. This includes individuals, groups of individuals, enterprises, organizations, or software agents. Data users are structured in a list of hierarchies. The elements are ordered in hierarchies by a "parent" attribute that contains the value of the "id" of its parent `<data-user>` . We mandate that this graph corresponds to one or more trees, i.e., there are no cycles. The meaning of the hierarchy is a grouping, i.e., the parent groups all its children. By default rulings="allow" inherit down while authorizations of rules with ruling="deny" inherit up and down along the element hierarchies.

Design Notes:

- Data users represent categories of users that are distinct from a privacy perspective.
- EPAL requires a data-user for all authorization queries. Therefore, vocabulary authors should always define a data user like "anyOther" that covers all data users that are not explicitly defined by this vocabulary.
- Unlike 'data-users', 'allow'-rules in a role hierarchy inherit up instead of down (i.e., unlike groups, parent-roles get more powerful).
- If (dynamic) roles are needed, these can either be implemented as a condition that restricts access to certain parties or else by dynamically assigning data-users to actual users at run-time. In the first case, the condition under which a role is assigned would be explicit in the policy. In the second, it would not be visible. E.g., restricting access to a patient's primary care physician can be done by a condition that allows the data-user "medical-personnel" to access a record if and only if the user-id of the data-user equals the physician-id stored in the patient's record.
- Note that most policies should include a distinguished data-user called 'data-subject' that represents the data subject that provided the data. This data-user can then be used to define the access rights of the individual. However, we did not define this data user since this would mandate that the implementation keeps track of who has provided what piece of data.
- The algorithm for processing compound requests uses the sequence of the `<data-users>` as precedence to resolve non-determinism: If a user can acts as data-user "marketing-manager" and as "marketing-department" and is allowed to perform the actions/purposes on category, the first data-user "marketing-manager" would be used to determine the obligations.

Schema Fragment

```
<xs:element
```

```
    name="data-user"
    minOccurs="0"
    maxOccurs="unbounded">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension
          base="epal:hierarchicalType">
          <xs:sequence
            minOccurs="0"
            maxOccurs="1">
            <xs:element
              name="contact-info"
              type="epal:contactInfoType"
              minOccurs="0"
              maxOccurs="1"></xs:element>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
```

3.4 <data-category>: Definition of the Data Category Hierarchies

The data categories used in privacy policies are often high-level classifications of data, such as customer contact information, medical record, or statistics. These categories are used to distinguish classifications of collected data that need to be treated differently from a privacy point of view. Organizing the data categories into a hierarchy improves the expressiveness of rules. For example, a rule that mentions a high-level data category applies to all parts. This reduces the number of rules in an EPAL policy.

Syntax: Each <data-category> element defines a data-category that can be referenced in a rule. The data-category element is of type "hierarchicalType". i.e., it inherits the mandatory attribute "id", the optional attribute "parent", and elements <short-description>, <long-description>, and <property> .

Semantics: A <data-category> represents some data of a given data classification. The elements are ordered in one or more hierarchies by a "parent" attribute that contains the value of the "id" attribute of its parent <data-category> . We mandate that this hierarchy represents one or more trees, i.e., there must not be any cycles. The meaning of the hierarchy is a grouping, i.e., the parent groups all its children. By default (unless superseded by rules with higher precedence) authorizations of rules with ruling="allow" inherit down while authorizations of rules with ruling="deny" inherit up and down. Another example is an exception to a rule that can treat a part of the data category. Exceptions enable an EPAL policy to resemble the privacy policy from which it was derived.

Design Notes:

- EPAL does not deal with the actual data that is collected. Instead, it uses

<data-category> to represent and categorize the data to be accessed. The granularity of the categories is determined by the enterprise.
- If actual instances of the data are needed to be evaluated by conditions, they must be defined separately as an data structure called the EPAL container.
- The system using EPAL is required to be able to determine the applicable policy as well as the applicable <data-category> for any protected resource. This can, e.g., be done by assigning a <data-category> to each column of a relational data-base while storing an identifier of the applicable policy with each customer-record.
- If multiple categories apply to a certain resource, this is covered by rules and/or requests containing multiple categories.
- EPAL requires a data-category for all authorization queries. Therefore, vocabulary authors should always define a data category like "otherData" that covers all data that is not explicitly defined by this vocabulary.

Schema Fragment:

```
<xs:element
  name="data-category"
  minOccurs="0"
  maxOccurs="unbounded"
  type="epal:hierarchicalType">
</xs:element>
```

3.5 <purpose>: Definition of the Purpose Hierarchies

Purpose is an important concept in privacy. Information is often disclosed only for particular purposes. Privacy policies must state the purposes for which data is used or is going to be used. The purposes mentioned in privacy policies are often high-level, such as marketing. More specific rules may deal with more specific purposes that fall under the domain of a high-level purpose, such as direct marketing by a third party via e-mail. Organizing the purposes into a hierarchy improves the expressiveness of rules and allows for compact policies.

Syntax: Each <purpose> element defines one purpose that can be referenced in rule definitions. Each <purpose> element is of type "hierarchicalType", i.e., it inherits the mandatory attribute "id", the optional attribute "parent", and elements <short-description>, <long-description>, and <property> .

Semantics: A <purpose> represents the purpose for which a rule authorizes or denies access. The elements are ordered in one or more hierarchy by a "parent" attribute that contains the value of the "id" attribute of its prospective parent. We mandate that this graph represents one or more tree, i.e., there are no cycles. The meaning of the hierarchy is a grouping, i.e., the parent groups all its children. By default (unless superseded by rules with higher precedence) authorizations of rules with ruling="allow" inherit down while authorizations of rules with ruling="deny" inherit up and down.

Design Notes:

- Unlike access control, the `<purpose>` is part of an EPAL authorization query. Without knowing the purpose of an access, authorization cannot be decided.
- As a consequence, any system using EPAL must be able to determine a purpose before asking the EPAL engine to evaluate a given policy.
- How to determine the purpose depends on the application. Privacy-aware applications may even state their current purpose or an activity identifier that can be used to easily derive it.
- Vocabulary authors should always define a purpose "otherPurpose" that covers all purposes that are not explicitly defined by this vocabulary.

Schema Fragment

```
<xs:element
  name="purpose"
  type="epal:hierarchicalType"
  minOccurs="0"
  maxOccurs="unbounded">
</xs:element>
```

3.6 `<action>`: Definition of a Privacy-relevant Action to be Executed on Data

An enterprise performs certain actions on its data resources. A privacy policy specifies which actions are allowed and which are denied under particular circumstances.

Syntax: Each `<action>` element defines one privacy-relevant action that can be referenced in rule definitions. Each `<action>` element is of type "describedObjectType", i.e., it inherits the mandatory attribute "id", and elements `<short-description>`, `<long-description>`, and`<property>` .

Semantics: An `<action>` represents the action that is authorized. `<action>` elements are *not* structured in a hierarchy.

Design Notes:

- Like `<data-category>`, these privacy-relevant actions abstract from the actual operation on the data. E.g., a file access as well as an SQL query may translate into a <action id="read"/> action. An actual application of EPAL needs to map its own terminology to the privacy-policy-specific one by means of a deployment mapping.
- Vocabulary authors should define an action "otherAction" that covers all actions that are not explicitly defined by a vocabulary.

Schema Fragment

```
<xs:element
  name="action"
  type="epal:describedObjectType"
  minOccurs="0"
  maxOccurs="unbounded">
</xs:element>
```

3.7 <container>: Abstract Definition of the Data to be Evaluated by Conditions

Privacy authorization can depend on conditions based upon the context of the request. For example, some data may be used for marketing only if the person is not a minor or a parent has given consent. The element <container> defines the data structure that contains context data that can be evaluated by conditions. Any condition then defines a list of containers that contain the input that is evaluated by this condition. Based on the actually input data the condition then returns 'true' or 'false'.

Syntax: Each <container> element extends the type "describedObjectType", i.e., it inherits the mandatory attribute "id", and elements <short-description>, <long-description>, and<property>. This type is extended by adding a list of one or more "attribute" elements of type "containerAttributeDefinitionType". Each of these elements specifies a piece of data that is contained in the container by means of attributes "simpleType", "minOccurs", "maxOccurs", "origin", and "auditable". The "origin" attribute can be used to categorize attributes. The Boolean "auditable" defines whether this attribute must be collected in an auditable way.

Semantics: Each container defines that the given list of attribute-name/attribute-values pairs is available at the time of authorization. An attribute is multi-valued, i.e., an attribute has multiple values (as specified by the minOccurs and maxOccurs attributes). The EPAL authorization engine requires that the values-lists validate against the given attribute specification of the given container. Once validated, the resulting data is then used to evaluate the rules. Note that we require that all containers that are needed are available, i.e., if a rule is applicable but the container-data needed to evaluate the condition is missing, this is defined to be an error.

When evaluating a condition, all the container attribute values needed to evaluate the condition are translated into a <xacml-context:Request> document as follows:

1. Create the root <xacml-context:Request> element.
2. For each container attribute value, do as follows:
    1. Create an <xacml-context:Attribute> element whose "AttributeId" attribute specifies a URI of the form urn:ibm:epal:1.0:container-attribute:[policy identifier]:[container identifier]:[attribute identifier] and "DataType" attribute specifies the data type. For example, when the policy, container, and attribute identifiers are "sample-policy",

"DataUserInfo", and "DataUserID" respectively, the URI is "urn:ibm:epal:1.0:container-attribute:sample-policy:DataUserInfo:DataUserID". Also, the data type must be the same as the one defined in the container definition.

2. For each attribute value, add an `<xacml-context:AttributeValue>` element representing the container attribute value to the created `<xacml-context:Attribute>` element.
3. According to the value of the "origin" attribute, add the created `<xacml-context:Attribute>` element to the corresponding XACML element. The mapping between the origin values and XACML elements are summarized in the following table. For example, if the origin value is "data-subject" then it is added to the `<xacml-context:Resource>` element.

Origin values	XACML elements
data-user	`<xacml-context:Subject>`
data-subject	`<xacml-context:Resource>`
filled-form	`<xacml-context:Resource>`
resource	`<xacml-context:Resource>`
action	`<xacml-context:Action>`
other	`<xacml-context:Environment>`

The resulting `<xacml-context:Request>` document is used to evaluate the `<xacml:Condition>` specified in the condition.

Design Notes:

- Note that preparing the container to be input is done by the system using EPAL. In our XML-based interface, the context data is part of the authorization interface. In the Java-based interface, it is provided by a call-back interface on request.
- Grouping attributes into containers should roughly reflect the cost for fetching the attributes, i.e., the costs for fetching attributes in the same container will usually be less than fetching attributes in different containers. The grouping of attributes was inspired by information services that sell information in containers. E.g., buying a credit-rating will deliver a container containing additional information (such as name, age) virtually 'for free'. As a consequence, conditions should try to minimize the number of containers they evaluate.
- In some cases, the available containers depend on the data that is accessed. E.g., a container with 'birthdate' may only be available when accessing a customer record. As a consequence, a condition evaluating birthdate can only be associated with rules defining access to "customer record". Ensuring such consistency between the policy and its

environment is out of the scope of EPAL. Whenever an EPAL implementation needs a context that is not available, it returns an error.

Schema Fragment

```
<xs:element
  name="container"
  minOccurs="0"
  maxOccurs="unbounded">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension
        base="epal:describedObjectType">
        <xs:sequence
          minOccurs="1"
          maxOccurs="1">
          <xs:element
            name="attribute"
            type="epal:containerAttributeDefinitionType"
            minOccurs="1"
            maxOccurs="unbounded">
          </xs:element><
        /xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <!-- Declared attribute id's must be unique within each container
  <xs:unique
    name="uniqueAttribute">
    <xs:selector
      xpath="attribute"/>
    <xs:field
      xpath="@id"/>
  </xs:unique>
</xs:element>
```

Example: Two Containers

```
<container
  id="DataUserInfo">
  <short-description>Information about the Data User</short-descript
  <long-description>More Infos.</long-description>
  <attribute
    id="DataUserID"
    maxOccurs="1"
    minOccurs="1"
    origin="data-user"
    simpleType="http://www.w3.org/2001/XMLSchema#string">
    <short-description>The ID of the data user accessing the data.</
  </attribute>
  <attribute
    id="WorkingOnStations"
    maxOccurs="unbounded"
    minOccurs="1"
    origin="data-user"
    simpleType="http://www.w3.org/2001/XMLSchema#string">
```

```
    <short-description>A multi-valued attribute describing the stati
  </attribute>
</container>

<container
  id="PatientRecord">
  <short-description>The patient record of the patient who's data is
  <attribute
    id="Station"
    maxOccurs="unbounded"
    minOccurs="1"
    origin="resource"
    simpleType="http://www.w3.org/2001/XMLSchema#string">
    <short-description>A multi-valued attribute describing the stati
  </attribute>
  <attribute
    id="PrimaryCarePhysicianID"
    maxOccurs="unbounded"
    minOccurs="1"
    origin="resource"
    simpleType="http://www.w3.org/2001/XMLSchema#string">
    <short-description>A multi-value input defining the IDs of all p
  </attribute>
</container>
```

Example: Data that Validates under these Containers

The following is an example XACML request context translated from container data that is valid under the above container definitions.

```
<xacml-context:Request xmlns:xacml-context="urn:oasis:names:tc:xacml
  <xacml-context:Subject>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>John Doe</xacml-context:Attribut
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>50B</xacml-context:AttributeValu
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>Emergency Room</xacml-context:At
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>49A</xacml-context:AttributeValu
    </xacml-context:Attribute>
  </xacml-context:Subject>
  <xacml-context:Resource>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>50B</xacml-context:AttributeValu
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>John Doe</xacml-context:Attribut
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>Bull Doc</xacml-context:Attribut
    </xacml-context:Attribute>
  </xacml-context:Resource>
```

```
</xacml-context:Request>
```

3.8 <obligation>: Definition of an Obligation

Legislation and privacy policies may state that when a certain action is performed, the enterprise is obligated to take some additional steps. An example is that all accesses against a certain type of data for a given purpose must be logged. Or children's data shall be deleted within 30 days unless parent consent is obtained. In EPAL such consequential actions are called obligations. EPAL is not designed to encode the logic of an obligation. The system which evaluates a request against an EPAL policy must be capable of executing all obligations given the unique name of the obligation.

Syntax: Each <obligation> element defines one obligation that can be referenced in rule definitions. <obligation> elements are *not* structured in a hierarchy.
The <obligation> elements extend the type "describedObjectType", i.e., it inherits the mandatory attribute "id", and elements <short-description>, <long-description>, <property> . This type is extended by a list of <parameter> elements of type "attributeDefinitionType" that specify parameter values to be specified inside the rules.

Semantics: An obligation is opaque and is returned after the rule is processed. An obligation inside a rule can contain values for its parameters to be returned together with the obligation identifier. An obligation parameter is multi-valued. These parameter values are of types as specified by the parameter definition. An obligation shall be returned if the rule was applied (i.e., either no condition was attached or the attached condition was satisfied). This holds for "allow" as well as "deny"-rules.

Design Notes:

- Like <data-category>, obligations abstract from the details.
- An opaque obligation "retention" with a parameter <days> 30</days> may be used to signal that data shall be deleted after 30 days. Note that the actual implementation of the obligation needs additional inputs (such as the actual data-instance that is accessed). These parameters are not passed through the EPAL engine, i.e., the environment will be required to input additional context into the implementation of the obligations.
- We assume that obligations do not interfere with each other. If a rule returns multiple obligations, we assume that it is possible to enforce them all.

Schema Fragment

```
<xs:element
  name="obligation"
  minOccurs="0"
```

```
    maxOccurs="unbounded">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension
          base="epal:describedObjectType">
          <xs:sequence
            minOccurs="1"
            maxOccurs="unbounded">
            <xs:element
              name="parameter"
              type="epal:attributeDefinitionType"
              minOccurs="0"
              maxOccurs="unbounded">
            </xs:element>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
    <!-- Declared parameter id's must be unique within each obligation
    <xs:unique
      name="uniqueParam">
      <xs:selector
        xpath="parameter"/>
      <xs:field
        xpath="@id"/>
    </xs:unique>
  </xs:element>
```

Example: An Obligation

```
  <obligation
    id="retention">
    <short-description>A test obligation</short-description>
    <long-description>long-description</long-description>
    <property
      id="Name">
      <value>property</value>
    </property>
    <parameter
      id="days"
      simpleType="http://www.w3.org/2001/XMLSchema#integer"
      minOccurs="1"
      maxOccurs="1"/>
  </obligation>
```

4. EPAL Policies

This section describes the syntax of an EPAL policy that defines privacy rules based on a given `<epal-vocabulary>` . While the `<epal-vocabulary>` defines the language to express sector-specific privacy policies, the policy defines rules that formalize what is allowed and what is denied.

4.1 `<epal-policy>`: Top-level Element for an EPAL Privacy Policy

Syntax: The <epal-policy> element defines the actual privacy rules based on a given vocabulary. The <epal-policy> element contains the following sub-elements in this fixed order:

1. One <policy-information> element of type "infoType" that describes the policy.
2. One <epal-vocabulary-ref> element of type "importStatementType" that refers to the <epal-vocabulary> to be used.
3. Zero or more <condition> elements that formalize conditions that evaluate containers of the vocabulary and that can then be attached as pre-conditions to implement instance-based decisions in rules.
4. Zero or more <rule> elements that define the actual privacy rules of the policy that define the actual authorizations of the policy.

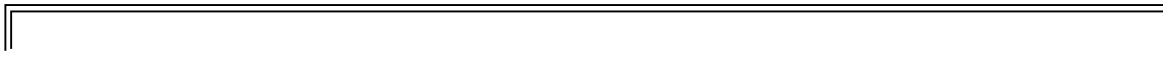
The <epal-policy> element has an optional attribute "global-condition" that defines a global pre-condition for this policy, a required attribute "version" that defines the EPAL version, and a required attribute "default-ruling" that defines the default ruling of this policy. Default-rulings are 'allow', 'deny', or 'not-applicable'. The optional Boolean attribute "final" that indicates that the rulings of this policy must not be overruled by other policies.

Semantics: The default ruling specifies the ruling that is literally returned if no rule is applicable. The ruling 'allow' indicates that the action is allowed if none of the rules was applicable, 'deny' indicates that the action is denied, and 'not-applicable' indicates that this policy does not care even though the request was in the domain of the policy (but others may). The global condition refers to a condition-ID. If this condition is true, the default ruling is returned. If "final='true'" then this fact should be output and used for detection of real conflicts: While conflicts between ordinary policies can be resolved (e.g., by 'denial-takes-precedence'), conflicts between two final policies must be detected and logged.

Design Notes:

- The default ruling 'not-applicable' allows a policy to implement a 'don't care'. This can simplify combining different policies where each policy only covers a given scope while 'not caring' about other authorizations.
- The indication that a policy is final will be output together with the ruling. This corresponds to two additional rulings 'must allow' and 'must deny' that can be evaluated by policy combination algorithms.
- The global condition is a means to select the right policy out of a set of policies that may are applicable.
- The 'final' Boolean attribute can be used to implement legal requirements (personal data by minors must never be stored unless consented by parent) or access promises to a data subject (you are allowed to review your data). This allows the distinction between enterprise-internal privacy design decisions vs. external contracts or promises.

Schema Fragment



```
<xs:element name="epal-policy">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="policy-information" minOccurs="1" maxOccurs=
      <xs:element name="epal-vocabulary-ref" minOccurs="1" maxOccurs
      <xs:element name="condition" minOccurs="0" maxOccurs="unbounde
      <xs:element name="rule" minOccurs="0" maxOccurs="unbounded">..
    </xs:sequence>
    <xs:attribute name="version" type="xs:string" default="1.0"/>
    <xs:attribute name="global-condition" type="xs:NCName"
      use="optional"/>
    <xs:attribute name="default-ruling" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="allow"/>
          <xs:enumeration value="deny"/>
          <xs:enumeration value="not-applicable"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="final" type="xs:boolean"
        use="optional"
        default="false"/>
  </xs:complexType>
</xs:element>
```

Example: An <epal-policy>

```
<epal-policy default-ruling="allow" global-condition="NCName" versio
  <policy-information id="NCName">
    <short-description language="en">short-description</short-descri
    <long-description language="en">long-description</long-descripti
    <property id="NCName">
      <value>value</value>
    </property>
    <issuer>
      <name>name</name>
      <organization>organization</organization>
      <e-mail>e-mail</e-mail>
      <address>address</address>
      <country>country</country>
    </issuer>
    <location>http://www.ibm.com</location>
    <version-info end-date="2001-12-31T12:00:00" last-modified="2001
  </policy-information>
  <epal-vocabulary-ref digest="0" canonicalizationAlgorithm="http://
  <condition id="NCName">...</condition>
  <rule id="NCName" ruling="allow">
      <short-description language="en">short-description</short-desc
      <long-description language="en">long-description</long-descrip
      <property id="NCName">
        <value>value</value>
      </property>
      <data-user refid="NCName"/>
      <data-category refid="NCName"/>
      <purpose refid="NCName"/>
      <action refid="NCName"/>
```

```
        <condition refid="NCName"/>
        <obligation refid="NCName">
          <parameter refid="NCName">
            <value>value</value>
          </parameter>
        </obligation>
    </rule>
  </epal-policy>
```

4.2 <policy-information>: Information about the Policy

The <policy-information> of type "infoType" contains information about the vocabulary in the form of three sub-elements. The "id" attribute defines the id of the policy. The <issuer> describes the issuer, the <version-info> the version and other management information. The "id" attribute of policy-information together with the "revision-number" attribute must uniquely identify a policy for each issuer.

Example: Policy Information inside an EPAL policy

```
  <policy-information
    id="sample-policy">
    <short-description language="en">Example policy information</short
    <long-description language="en">long-description</long-description
    <!-- Entity issuing the policy -->
    <issuer>
      <name>EPAL Working Group</name>
      <organization>IBM Research</organization>
      <e-mail>testid@zurich.ibm.com</e-mail>
      <address>Saumerstr. 4, 8838 Ruschlikon</address>
      <country>Switzerland</country>
    </issuer>
    <!-- Policy version. This is so the enterprise
         can track versions of its policy. -->
    <version-info
      start-date="2002-03-19T00:00:00"
      test="true"
      last-modified="2002-03-19T00:00:00"
      revision-number="1.02A"
      end-date="2004-03-19T00:00:00"/>
  </policy-information>
```

4.3 <epal-vocabulary-ref>: Reference to the EPAL Vocabulary to be Used

Syntax: The <epal-vocabulary-ref> element identifies the vocabulary to be used for the rules and conditions of this policy. It inherits the following attributes from "importStatementType".

1. "location" a mandatory attribute of type "anyURI" that points to the vocabulary file to be used.
2. "id" an optional attribute of type "NCName" that signals the expected "id"

of the imported vocabulary (the name is stored at vocabulary-information/@id).
3. "revision" an optional attribute of type "string" that signals the expected revision of the imported vocabulary (the revision is stored at vocabulary-information/version-info/@revision-number).
4. "digest", "digestAlgorithm", and "canonicalizationAlgorithm "determine value and algorithms for verifying a digest.
5. It can be augmented by elements `<long-description>`, `<short-description>`, and `<property>` as described for the describedObjectType.

Semantics: The `<epal-vocabulary-ref>` element points to the vocabulary that will be read by an EPAL engine. If the "id" or "revision" of the vocabulary at the given "location" are not identical to the ones in the `<epal-vocabulary-ref>` element or if the computed digest is not correct, the policy is invalid and no further processing can be done.

Design Notes:

- The digest can be used for auditability. Signing a policy authenticates the vocabulary as well if this digest is included.

4.4 `<condition>`: Definition of Conditions evaluating Container Attributes

Legislation and privacy policies make statements based on conditions. Rules must enforce these statements while accounting for the associated conditions. It is assumed that a condition can be evaluated to a boolean value.

Syntax: The `<condition>` element inherits the mandatory attribute "id", and the elements `<short-description>`, `<long-description>`, and `<property>` from the "describedObjectType". It extends this type by adding a list of at least one element `<evaluates-container>` of type "referringObjectType". The mandatory attribute "refid" of each `<evaluates-container>` element contains the value of an "id" of a container that is evaluated by this condition. The second element that is added is exactly one additional element `<xacml:Condition>`, which specified a particular condition written in XACML's condition language.

Semantics: Each `<condition>` represents one condition on the containers. Before actually evaluating the condition, the container definitions should be used to validate the input container data and the container data is translated to an `<xacml-context:Request>` document (see above). Then the condition is evaluated on the `<xacml-context:Request>` document. Only if all conditions in a rule are satisfied, the rule can be used. Otherwise, it is ignored.

Design Notes:

- The reason why we used XACML's condition language as a condition language is that it is a standardized language. Another reason was that XACML's condition language can easily be extended by adding user-

defined functions that can interface to the environment of an enterprise.
- Multiple conditions in a rule are satisfied if all conditions are satisfied (Boolean AND).

Schema Fragment

```
<xs:element
  name="condition"
  minOccurs="0"
  maxOccurs="unbounded">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension
        base="epal:describedObjectType">
        <xs:sequence>
          <xs:element
            name="evaluates-container"
            minOccurs="1"
            maxOccurs="unbounded"
            type="epal:referringObjectType"/>
          <xs:element
            minOccurs="1"
            maxOccurs="1"
            ref="xacml:Condition"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

Example: Two Conditions Evaluating the Given Containers

```
<condition
  id="NurseMatchesPatient">
  <short-description>This condition checks whether a nurse
       is allowed to access patient data.</short-description>
  <long-description>This access is only allowed if the nurse
    is on duty and if the nurse is working on a station where the pa
  <evaluates-container refid="DataUserInfo"/>
  <evaluates-container refid="PatientRecord"/>
  <xacml:Condition
    xmlns:xacml="urn:oasis:names:tc:xacml:1.0:policy"
    FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
    <xacml:Function FunctionId="urn:oasis:names:tc:xacml:1.0:functio
    <xacml:ResourceAttributeDesignator AttributeId="urn:ibm:epal:1.0
    <xacml:SubjectAttributeDesignator AttributeId="urn:ibm:epal:1.0:
  </xacml:Condition>
</condition>

<condition
  id="isPrimaryCarePhysician">
  <short-description>This condition checks whether a doctor is the
        patient's primary care physician.</short-description>
  <evaluates-container refid="DataUserInfo"/>
  <evaluates-container refid="PatientRecord"/>
  <xacml:Condition
```

```
      xmlns:xacml="urn:oasis:names:tc:xacml:1.0:policy"
      FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
      <xacml:Function FunctionId="urn:oasis:names:tc:xacml:1.0:functio
      <xacml:SubjectAttributeDesignator AttributeId="urn:ibm:epal:1.0:
      <xacml:ResourceAttributeDesignator AttributeId="urn:ibm:epal:1.0
    </xacml:Condition>
  </condition>
```

4.5 <rule>: Definition of an Authorization Rule

A rule states that certain data-users are allowed or denied to perform an action on data of given categories for given purposes if and only if the given conditions are satisfied, and that, consequently, particular obligations must be executed. This statement is encoded as a <rule> element in EPAL. Rules have three types: 'allow' and 'deny-rules define whether the action is allowed or not while defining obligations that must be performed. 'obligate'-rules define obligations that must be output without actually defining a ruling.

Syntax: The <rule> element extends the type "describedObjectType", i.e., it inherits the mandatory attribute "id", and elements <short-description>, <long-description>, <property> . The "describedObjectType" is extended by the required attribute "ruling" with values 'allow', 'deny', or 'obligate' as well as the following sub-elements in this fixed order:

1. One or more <data-user> elements that define the data users of the policy. Each <data-user> is of "referringObjectType". Its "refid" attribute must refer to an "id" attribute of a <data-user> in the referenced vocabulary.
2. One or more <data-category> elements that define the data categories of the policy. Each <data-category> is of "referringObjectType". Its "refid" attribute must reference an "id" attribute of a <data-category> in the referenced vocabulary.
3. One or more <purpose> elements that define the purposes of the policy. Each <purpose> is of "referringObjectType". Its "refid" attribute must reference an "id" attribute of a <purpose> in the referenced vocabulary.
4. One or more <action> elements that define the privacy-relevant actions of the policy. Each <action> is of "referringObjectType". Its "refid" attribute must reference an "id" attribute of an <action> in the referenced vocabulary.
5. Zero or more <condition> elements that identify the pre-conditions for applying this rule. Each <condition> is of "referringObjectType". Its "refid" attribute must reference an "id" attribute of a <condition> in the referenced vocabulary.
6. Zero or more <obligation> elements that define the obligations that are mandated by this rule. Its "refid" attribute must reference an "id" attribute of an <obligation> in the referenced vocabulary. If the obligation in the vocabulary has parameters, the obligation can provide values by adding a sequence of <parameter> elements. These parameters shall be returned as the parameters of the obligation. Each <parameter> element has a

required attribute "refid" to reference an "id" attribute of a `<parameter>` of the obligation definition and contains `<value>` elements with contents of type "epalSimpleType". The number of `<value>` elements and the contents are required to validate under the `<parameter>` definition in the obligation definition.

Design Notes:

- For any parameter with id1 inside an obligation with id2 there must be a corresponding parameter in the definition of obligation id2.
- **The rules in a policy are ordered by precedence. The first rule has highest precedence; the last the lowest.**
- The order of elements of the same type inside a rule is not relevant.

Schema Fragment

```
<xs:element name="rule" minOccurs="0"
  maxOccurs="unbounded">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="epal:describedObjectType">
        <xs:sequence>
          <xs:element name="data-user"
            type="epal:referringObjectType"
            minOccurs="1"
            maxOccurs="unbounded"></xs:element>
          <xs:element name="data-category"
            type="epal:referringObjectType"
            minOccurs="1"
            maxOccurs="unbounded"></xs:element>
          <xs:element name="purpose"
            type="epal:referringObjectType"
            minOccurs="0"
            maxOccurs="unbounded"></xs:element>
          <xs:element name="action"
            type="epal:referringObjectType"
            minOccurs="1"
            maxOccurs="unbounded"></xs:element>
          <xs:element name="condition"
            type="epal:referringObjectType"
            minOccurs="0"
            maxOccurs="unbounded"></xs:element>
          <xs:element name="obligation" minOccurs="0"
            maxOccurs="unbounded">
            <xs:complexType>
              <xs:complexContent>
                <xs:extension
                  base="epal:referringObjectType">
                  <xs:sequence minOccurs="0"
                    maxOccurs="1">
                    <xs:element name="parameter"
                      minOccurs="0"
                      maxOccurs="unbounded">
                      <xs:complexType>
                        <xs:sequence>
```

```
                        <xs:element name="value"
                          type="epal:epalSimpleType"
                          minOccurs="0"
                          maxOccurs="unbounded"/>
                      </xs:sequence>
                      <xs:attribute name="refid"
                        use="required"
                        type="xs:NCName"></xs:attribute>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
              </xs:extension>
            </xs:complexContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="ruling">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="allow"/>
            <xs:enumeration value="deny"/>
            <xs:enumeration value="obligate"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
</xs:element>
```

5. Privacy Authorization - Semantics of an EPAL Policy

We now describe the precise meaning of an EPAL policy. This is done by describing the meaning of direct authorization and inheritance, and by defining the in- and outputs of authorization algorithms. We have two kinds of algorithms. One is for simple request processing and the other is for compound request processing.

Note that we do not mandate any particular implementation or interface to the algorithm. We give an example implementation that uses XML inputs and outputs as an appendix. In a separate deliverable, we will provide a Java reference implementation that uses a Java-interface.

5.1 Simple Requests

5.1.1 Required Inputs

A simple authorization request contains

- a single data-user *U* that is defined in the vocabulary,
- a single data-category *T* that is defined in the vocabulary,
- a single purpose *P* that is defined in the vocabulary,
- a single action *A* that is defined in the vocabulary, and

- the container data required by this policy that must be valid against the container definition.

The tuple (*U*, *T*, *P*, *A*) is called the authorization-quadruple.

5.1.2 Desired Outputs

The algorithm outputs

- a ruling that is either "allow", "deny", or "not-applicable".
- a boolean flag 'final' that signals whether the policy that created the ruling is final or not. The returned 'final' flag is always identical to the 'final' attribute of the epal-policy.
- A single rule "id" that mandated this ruling (or the empty string if the ruling was the default ruling).
- An "obligationSet" that contains pairs of (rule-id, obligation) that contain an obligation to be enforced and the id of the rule that mandated it.

5.1.3 Operational Semantics

The algorithm determines the ruling by processing each rule with descending precedence. Before starting, it initializes the set "obligationSet" to the empty set. Then, it checks that the global-condition is satisfied. If not, the default-ruling is returned.

For each rule, the algorithm checks:

1. Are the elements of the authorization request in the scope of this rule?
2. Is the condition satisfied?

If these conditions are satisfied, the algorithm acts on the rule. If all rules have been processed (but no 'allow' or 'deny' satisfied these two conditions), the algorithm returns the default-ruling and the current "obligationSet". We now describe the two conditions in more detail:

Scope Check: The scope checks are based on the hierarchies defined in the used <epal-vocabulary>.
A request is in the scope of a rule where the "ruling" attribute is either "allow" or "obligate", iff

- The data-category "id" in the request is either identical to the "id" of one of the data-categories in the rule or else identical to the "id" of a child of a data-category in the rule, and
- The data-user "id" in the request is either identical to the "id" of one of the data-users in the rule or else identical to the "id" of a child of a data-user in the rule, and
- The purpose "id" in the request is either identical to the "id" of one of the purposes in the rule or else identical to the "id" of a child of a purpose in the rule, and
- The action "id" in the request is identical to the "id" of one of the actions in

the rule.

A request is in the scope of a rule where the "ruling" attribute is "deny", iff

- The data-category "id" in the request is either identical to the "id" of one of the data-categories in the rule or else identical to the "id" of a child **or parent** of a data-category in the rule, and.
- The data-user "id" in the request is either identical to the "id" of one of the data-users in the rule or else identical to the "id" of a child **or parent** of a data-user in the rule, and.
- The purpose "id" in the request is either identical to the "id" of one of the purposes in the rule or else identical to the "id" of a child **or parent** of a purpose in the rule, and.
- The action "id" in the request is identical to the "id" of one of the actions in the rule.

If the request is not in the scope of the rule, the next rule is processed. Else, the algorithm continues processing this rule:

Checking the Conditions: If the rule has no conditions, the algorithm continues by acting on this rule. If the rule has one or more conditions, The conditions that are associated with the rule are evaluated. If any of the conditions is not satisfied, the algorithm continues by processing the next rule. Else, it acts on this rule.

Acting on a rule: Pairs with each obligation of the rule and the rule-id are added to the set "obligationSet". If the ruling is "allow" or "deny", the ruling and the obligationSet are returned. If the ruling is "obligate", the (rule-id, obligation)-pairs are added to the "obligationSet" and the algorithm continues by processing the next rule.

Design Notes:

- How to use the given container provider to evaluate a condition is left to the implementers. An implementer might use the container provider to create an `<xacml-context:Request>` document and then evaluate the XACML condition on the document. Another implementer might directly process the container provider without creating an `<xacml-context:Request>` document.
- Upward inheritance of 'deny'-rules enables us to process authorization queries containing non-leaf elements. If one would only allow leafs of the hierarchy in the authorization request, up-inheritance would no longer be needed.
- The ruling 'obligate' can be used to enforce obligations without defining a ruling. An example is "If there ever is a medical-emergency access, I require a notification." Note that if only 'obligate'-rules apply, then the default ruling and these obligations are returned.
- Authorizing the cross-product aims at compact policies. This is similar to P3P statements. However, it's not easy to understand. As a consequence, a policy editor might not use this feature in the user-

interface but compact the policies only on export.
- The parameter of the obligation is included such that the policy can define options of the obligation. E.g., retention-time for a "limited-retention" obligation.

5.2 Compound Requests

The intuitive semantics of a compound request is to check whether **any** of the data-users is allowed to access **all** categories, for **all** purposes, 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.

5.2.1 Required Inputs

A compound authorization request contains

- a non-empty set of data-users *U*,
- a non-empty set of data-categories *T,*
- a non-empty set of purposes *P*,
- a non-empty set of actions *A*, and
- a container provider as defined above

5.2.2 Desired Outputs

The algorithm outputs

- A ruling that is either "allow", "deny", or "not-applicable".
- A set of rule "id"s that mandated this ruling (or the empty set if the ruling was the default ruling).
- A set of pairs (obligation, rule-id-set) with disjoint obligations (obligations with different parameters are considered to be disjoint) and a list of rule-"id"s that mandated this particular obligation.

5.2.3 Operational Semantics

Processing a Single Data User: The compound query for a particular data-user is decomposed into simple requests as follows:

1. The algorithm creates three empty sets of obligation/rule-id-set pairs that will be used to collect all distinct obligations and the set of rule-id's that mandated it. We call these sets denyObligations and allowObligations, and NAObligations. In addition, it creates two sets denyRules and allowRules of rule-id's.
2. The algorithm performs one simple request for each authorization quadruple in the cross product of the fixed data-user, and the purpose, action, and category sets using the same container provider.
   If the ruling is "allow", the returned is added to the set allowRules. If obligations are returned, the (rule-id, obligation)-pairs are added to the set allowObligations. If the ruling is "deny", the (rule-id, obligation)-pairs are added to the sets denyRules and denyObligations, respectively. If the

ruling is 'not-applicable', the obligations are added to the NAObligations set.
3. If all rulings are 'not-applicable', the ruling is 'not-applicable' with the obligations in the NAObligations-set and with an empty rule-id-set.
4. If all rulings are either 'allow' or 'not-applicable' and at least one ruling is 'allow', the algorithm returns "allow". The allowRules rule-id-set as well as a union of the allowObligations and NAObligations sets.
5. Else (if all rulings are 'deny' or 'not-applicable'), the ruling is 'deny'. The denyRules rule-id-set as well as the union of the denyObligations and NAObligations sets are returned.

Processing Multiple Data Users: Multiple data users are processed in the order that they are defined inside the definition section of the policy:

1. If the compound processing for **any** data user is 'allow', select **the first one** of such data users and return the 'allow' ruling with the rule-id's (allowRules) and obligation/rule-id-set pairs (allowObligations) that are returned for this user.
2. If the compound processing for **any** data user is 'deny', select **the first one** of such data users and return the 'deny' ruling with the rule-id's (denyRules) and obligation/rule-id-set pairs (denyObligations) that are returned for this user.
3. Else, return the 'not-applicable' ruling with an empty rule-id-set and the NAObligations-set defined by the first data user..

Design Notes:

- If multiple data-users in the set result in a ruling "allow" or "deny", the algorithm must choose the data user that is the topmost among these data users inside the definition section of the policy. This data user is then used to determine the set of returned obligations and the set of applied rules.
- Any algorithm implementing compound requests is free to do this without actual composition into simple requests as long as the behavior is preserved.
- All simple requests are processed using the same containers, i.e., one assumes that data of the data-user, object, or any other data in the containers is valid for all elements in the compound request.
- Note that processing of compound requests does not use precedence to resolve conflicts. i.e., a lowest-level 'deny' on any element in the cross-product for a given data-user overrules any highest-level 'allow'.

6 EPAL Data Types

We now list some complex data types that have been used for multiple elements. These data-types define sub-elements that are common to all actual elements that implement or refine these types.

6.1 "`identifiedObjectType`": Elements with unique identifiers

Any element of this type has a required attribute "id" of type "NCName". This attribute contains an identifier that can be used to reference the element. This type is further refined by the "describedObjectType" and used in the rules to reference the objects that are covered by a rule.

Schema Fragment

```
<!-- Elements which have an identifier should be of this type -->
<xs:complexType
  name="identifiedObjectType">
  <xs:attribute
    name="id"
    type="xs:NCName"
    use="required">
  </xs:attribute>
</xs:complexType>
```

6.2 "referringObjectType": Elements with reference identifiers

Any element of this type has a required attribute "refid" of type "NCName". This attribute contains an identifier that references another element that contains the same value in its "id" attribute. E.g., a data-user inside a rule contains a refid field that references the id attribute of a data-user inside the referenced vocabulary.

Schema Fragment

```
<!-- Elements which refer to an identifier should be of this type --
<xs:complexType
  name="referringObjectType">
  <xs:attribute
    name="refid"
    type="xs:NCName"
    use="required">
  </xs:attribute>
</xs:complexType>
```

6.3 "describedObjectType": Extensible Elements with Descriptions

The "describedObjectType" extends the "identifiedObjectType". Elements of this type contain a sequence of zero or more elements `<short-description>` of type "string" that is extended by an attribute 'language' of simple type "language", zero or more elements `<long-description>` of type "string" that is extended by an attribute 'language' of simple type "language", and zero or more elements `<property>`. Each `<property>` element contains `<value>` elements with contents of type "epalSimpleType".
We require that for any 'language', there must be at most one `<long-description>` and one `<short-description>` element.

Property elements can be used to extend the EPAL policy language or to store any external information that is needed by particular implementations. Examples of properties include implementation hints, or expressions describing the element in more detail. The `"id"` can be used by implementers to retrieve properties that relate to their particular implementation. In general, the content of the properties is not standardized. However, standardizing properties can enable a later extension of EPAL without changing the core standard. This type is further refined by the "hierarchicalObjectType" and is used in the EPAL vocabularies to define and describe non-hierarchical elements of an EPAL policy.

Schema Fragment

```
<!-- Identified Elements which have descriptions should be of this t
<xs:complexType name="describedObjectType">
    <xs:complexContent>
        <xs:extension base="epal:identifiedObjectType">
            <xs:sequence>
                <xs:element name="short-description" minOccurs="0" m
                    <xs:complexType>
                        <xs:simpleContent>
                            <xs:extension base="xs:string">
                                <xs:attribute name="language" use="o
                            </xs:extension>
                        </xs:simpleContent>
                    </xs:complexType>
                </xs:element>
                <xs:element name="long-description" minOccurs="0" ma
                    <xs:complexType>
                        <xs:simpleContent>
                            <xs:extension base="xs:string">
                                <xs:attribute name="language" use="o
                            </xs:extension>
                        </xs:simpleContent>
                    </xs:complexType>
                </xs:element>
                <xs:element name="property" minOccurs="0" maxOccurs=
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="value" type="epal:epal
                        </xs:sequence>
                        <xs:attribute name="id" use="required" type=
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

6.4 "`hierarchicalType`": Identified elements structured in a hierarchy

The "hierarchicalType" extends the "describedObjectType" by adding an optional "parent" attribute with type "NCName". The parent attribute points to a

unique "id" attribute of another element.
This type is used in EPAL vocabularies to define hierarchical elements (such as purpose, or data-user) to define and describe the elements of an EPAL policy while structuring them into a hierarchy.

Schema Fragment

```
<!-- Elements which can participate in a hierarchy should have this
<xs:complexType
  name="hierarchicalType">
  <xs:complexContent>
    <xs:extension
      base="epal:describedObjectType">
      <xs:attribute
        name="parent"
        type="xs:string"
        use="optional">
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

6.5 "contactInfoType": A sequence of elements for contact information

An element of type "contactInfoType" contains a sequence of exactly one of each of the following required elements of type "string": <name>, <organization>, <e-mail>, <address>, and <country> .
This type is used for stating the mandatory address of the issuer of a policy and for adding an optional address to a data-user of a policy.

Schema Fragment

```
<!-- Contact information   -->
<xs:complexType
  name="contactInfoType">
  <xs:sequence>
    <xs:element
      name="name"
      type="xs:string"
      minOccurs="1"
      maxOccurs="1"></xs:element>
    <xs:element
      name="organization"
      type="xs:string"
      minOccurs="1"
      maxOccurs="1"></xs:element>
    <xs:element
      name="e-mail"
      type="xs:string"
      minOccurs="1"
      maxOccurs="1"></xs:element>
    <xs:element
```

```
        name="address"
        type="xs:string"
        minOccurs="1"
        maxOccurs="1"></xs:element>
      <xs:element
        name="country"
        type="xs:string"
        minOccurs="1"
        maxOccurs="1"></xs:element>
    </xs:sequence>
  </xs:complexType>
```

6.6 "attributeDefinitionType": Schema-like Definition of Multi-valued Attributes

An element of type "attributeDefinitionType" extends the "describedObjectType" by adding a set of attributes. The inherited attribute "id" defines the name of the defined attribute. The attribute "simpleType" defines the data type of the attribute. The allowed data types are taken from the XACML specification. The attribute "minOccurs" defines the minimal number of values that need to occur in a list. The attribute "maxOccurs" defines the maximal number of values that need to occur in a list. Both are either of type "nonNegativeInteger" or else equal the string 'unbounded'.

Schema Fragment

```
<!-- A Descriptor of an attribute -->
<xs:complexType
  name="attributeDefinitionType">
  <xs:complexContent>
    <xs:extension
      base="epal:describedObjectType">
      <xs:attribute
        name="simpleType">
        <xs:simpleType>
          <xs:restriction
            base="xs:anyURI">
            <xs:enumeration value="http://www.w3.org/2001/XMLSchema#
            <xs:enumeration value="http://www.w3.org/2001/XMLSchema#
            <xs:enumeration value="http://www.w3.org/2001/XMLSchema#
            <xs:enumeration value="http://www.w3.org/2001/XMLSchema#
            <xs:enumeration value="http://www.w3.org/2001/XMLSchema#
            <xs:enumeration value="http://www.w3.org/2001/XMLSchema#
            <xs:enumeration value="http://www.w3.org/2001/XMLSchema#
            <xs:enumeration value="http://www.w3.org/2001/XMLSchema#
            <xs:enumeration value="http://www.w3.org/2001/XMLSchema#
            <xs:enumeration value="http://www.w3.org/TR/2002/WD-xque
            <xs:enumeration value="http://www.w3.org/TR/2002/WD-xque
            <xs:enumeration value="urn:oasis:names:tc:xacml:1.0:data
            <xs:enumeration value="urn:oasis:names:tc:xacml:1.0:data
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute
```

```
          name="minOccurs"
          use="optional"
          default="1">
          <xs:simpleType>
            <xs:union
              memberTypes="xs:nonNegativeInteger">
              <xs:simpleType>
                <xs:restriction
                  base="xs:string">
                  <xs:enumeration
                    value="unbounded"/>
                </xs:restriction>
              </xs:simpleType>
            </xs:union>
          </xs:simpleType>
        </xs:attribute>
        <xs:attribute
          name="maxOccurs"
          use="optional"
          default="1">
          <xs:simpleType>
            <xs:union
              memberTypes="xs:nonNegativeInteger">
              <xs:simpleType>
                <xs:restriction
                  base="xs:string">
                  <xs:enumeration
                    value="unbounded"/>
                </xs:restriction>
              </xs:simpleType>
            </xs:union>
          </xs:simpleType>
        </xs:attribute>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
```

6.7 "containerAttributeDefinitionType": Schema-like Definition of Multi-valued Container Attributes

An element of type "containerAttributeDefinitionType" extends the "attributeDefinitionType" by adding a required attribute "origin" and an optional attribute "auditable". The allowed values for the "origin"attribute are "data-user" (the XACML subject that performs the operation), "data-subject" (the data subject who's data is accessed; in XACML this is part of the resource), "filled-form" (the data has been collected in conjunction with the data that is accessed, i.e., at least from the same data subject under the same policy) "resource" (the resource to be accessed excluding data collected from the data-subject), "action" (data about the action that is performed such as parameters), and "other" (anything else).
The "auditable" attribute is Boolean and defines whether this particular attribute must be collected in an auditable way. In practice, this attribute will be used to distinguish consent like "yes to direct mailings" (auditable) and enterprise-internal context "time of day" (not required to be auditable).

Schema Fragment

```
<xs:complexType name="containerAttributeDefinitionType">
    <xs:complexContent>
        <xs:extension base="epal:attributeDefinitionType">
            <xs:attribute name="origin" use="required">
                <xs:simpleType>
                    <xs:restriction base="xs:NCName">
                        <xs:enumeration value="data-subject"/>
                        <xs:enumeration value="data-user"/>
                        <xs:enumeration value="filled-form"/>
                        <xs:enumeration value="resource"/>
                        <xs:enumeration value="action"/>
                        <xs:enumeration value="other"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="auditable" use="optional" type="
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

6.8 "epalSimpleType": Data allowed in Attributes, Parameters, and Properties.

An element of type "epalSimpleType" contains values of simple types that can be tunneled through the EPAL engine. These primitive types are taken from the XACML specification.

Schema Fragment

```
<!-- Allowed EPAL attribute types;
     This list MUST be in sync with list in attributeDefinitionType.
     Currently, four data types (dateTimeDuration, yearMonthDuration
<xs:simpleType
  name="epalSimpleType">
  <xs:union
    memberTypes="xs:string xs:boolean xs:integer xs:double xs:date x
</xs:simpleType>
```

6.9 "infoType": Information about the Policy or Vocabulary

This type defines the information that is used to describe and identify an <epal-policy> or an <epal-vocabulary> . It contains information about the policy or vocabulary in the form of three sub-elements. This type extends the "describedObjectType" by appending the following sub-elements in the following sequence:

1. The optional <issuer> of type "contactInfoType" contains the contact information of the issuer.
2. The optional <location> element of type "anyURI" points to the location

in which the policy or vocabulary is stored.

3. The required `<version-info>` element contains a required attribute "revision-number" of type "string" that identifies the version of the policy, a required attribute "last-modified" of type "dateTime" that defines when the policy has been modified, an optional attribute "test" of type "boolean" that indicates that this is a test policy, a required attribute "start-date" of type "dateTime" that indicates the start of the lifetime of this policy, an optional attribute "end-date" that defines when the policy expires, and two optional attributes "superseded-by-id" and "superseded-by-revision" that identifies another policy that supersedes this policy. If only an "id" is given, all revisions with this id supersede the given policy, if only a revision is given, the policy with the same but the given revision supersedes this policy.
   Note that the "id" attribute together with the "revision-number" attribute must uniquely identify a policy or a vocabulary for each issuer.

Schema Fragment

```
        <xs:complexType name="infoType">
            <xs:complexContent>
                <xs:extension base="epal:describedObjectType">
                    <xs:sequence>
                        <xs:element name="issuer" minOccurs="0"
                            maxOccurs="1"
                            type="epal:contactInfoType"></xs:element>
                        <xs:element name="location" minOccurs="0"
                            maxOccurs="1"
                            type="xs:anyURI"></xs:element>
                        <xs:element name="version-info" minOccurs="1"
                            maxOccurs="1">
                            <xs:complexType>
                                <xs:attribute name="test" type="xs:boole
                                    default="false"
                                    use="optional"></xs:attribute>
                                <xs:attribute name="start-date"
                                    type="xs:dateTime"
                                    use="required"></xs:attribute>
                                <xs:attribute name="revision-number"
                                    type="xs:string"
                                    use="required"></xs:attribute>
                                <xs:attribute name="last-modified"
                                    type="xs:dateTime"
                                    use="required"></xs:attribute>
                                <xs:attribute name="end-date"
                                    type="xs:dateTime"
                                    use="optional"></xs:attribute>
                                <xs:attribute name="superseded-by"
                                    type="xs:dateTime"
                                    use="optional"></xs:attribute>
                            </xs:complexType>
                        </xs:element>
                    </xs:sequence>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
```

6.10 "importStatementType": Import of a file

This type defines the syntax for importing a file (e.g., an EPAL vocabulary file). It contains the following attributes.

1. "location" a mandatory attribute of type "anyURI" that points to the file to be imported
2. "id" an optional attribute of type "NCName" that signals the expected "id" of the imported file (When the file is a vocabulary file, the name is stored at /epal-vocabulary/vocabulary-information/@id).
3. "revision" an optional attribute of type "string" that signals the expected revision of the imported file (When the file is a vocabulary file, the revision is stored at vocabulary-information/version-info/@revision-number).
4. "digest" a digest value for auditability. This optional attribute (of type xs:base64Binary) specifies the digest of the imported file. This is used to ensure the imported file is the expected one. The digest value is computed by a digest algorithm after a canonicalization algorithm is applied to the file.
5. "digestAlgorithm" is an optional anyURI attribute that specifies the digest algorithm being used. This specification uses SHA-1 as the default algorithm although additional algorithms can be used. The default URI is "http://www.w3.org/2000/09/xmldsig#sha1" defined in the XML-Signature specification.
6. "canonicalizationAlgorithm" is an optional anyURI attribute that specifies the canonicalization algorithm being used. This specification uses Exclusive XML Canonicalization Version 1.0 as the default algorithm although additional algorithms can be used. The default URI is "http://www.w3.org/2001/10/xml-exc-c14n# defined in the Exclusive XML Canonicalization specification.

In addition, it can be augmented by elements `<long-description>`, `<short-description>`, and `<property>` as described for the describedObjectType.

Schema Fragment

```
<xs:complexType name="importStatementType">
     <xs:complexContent>
         <xs:extension base="epal:describedObjectType">
             <xs:attribute name="location" use="required"
                 type="xs:anyURI"></xs:attribute>
             <xs:attribute name="digest" use="optional"
                 type="xs:base64Binary"></xs:attribute>
             <xs:attribute name="digestAlgorithm" use="optional"
                 type="xs:anyURI"
                 default="http://www.w3.org/2000/09/xmldsig#sha1"
             <xs:attribute name="canonicalizationAlgorithm" use="
                 type="xs:anyURI"
                 default="http://www.w3.org/2001/10/xml-exc-c14n#
             <xs:attribute name="revision" use="optional"
                 type="xs:NCName"></xs:attribute>
         </xs:extension>
     </xs:complexContent>
```

```
        </xs:complexType>
```

Example: Structure of a `<import>` element and its sub-elements

```
    <import digest="" canonicalizationAlgorithm="http://www.w3.org/200
```

Appendices

Appendix 1. References (Normative)

CPExchange 1.0
Kathy Bohrer and Bobby Holland, editors. *Customer Profile Exchange (CPExchange) Specification*, 20 October 2000. Available at: http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/

Exclusive XML Canonicalization
World Wide Web Consortium. *Exclusive XML Canonicalization Version 1.0.* Available at: http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/

P3P 1.0
World Wide Web Consortium. *The Platform for Privacy Preferences 1.0 (P3P1.0) Specification*, 28 September 2001. Available at: http://www.w3.org/TR/P3P/

XACML
OASIS. *eXtensible Access Control Markup Language (XACML)*, Committee Specification 1.0 (Revision 1), 12 December 2002, Available at: http://www.oasis-open.org/committees/xacml/

XML 1.0 (Second Edition)
World Wide Web Consortium. *Extensible Markup Language (XML) 1.0, Second Edition.* Available at: http://www.w3.org/TR/2000/WD-xml-2e-20000814

XML Schema Part 1: Structures
World Wide Web Consortium. *XML Schema Part 1: Structures.* Available at: http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/

XML Schema Part 2: Datatypes
World Wide Web Consortium. *XML Schema Part 2: Datatypes.* Available at: http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

XML Schema Requirements
World Wide Web Consortium. *XML Schema Requirements.* Available at: http://www.w3.org/TR/1999/NOTE-xml-schema-req-19990215

XML-Signature
World Wide Web Consortium. *XML-Signature Syntax and Processing.* Available at: http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/

Appendix 2. References (Non-Normative)

XML Schema Language: Part 0 Primer
World Wide Web Consortium. *XML Schema Language: Part 0 Primer.*

Available at: http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/

XSL

World Wide Web Consortium. *Extensible Stylesheet Language (XSL).* Available at: http://www.w3.org/TR/2001/REC-xsl-20011015/

UML Guide

Martin Fowler. *UML Distilled.* The Addison-Wesley Object Technology Series, August 1999.

Robert C. Martin. *UML Tutorial: Part I - Class Diagrams.* Available at: http://www.objectmentor.com/publications/umlClassDiagrams.pdf

XML Guide

Didier Martin, et al. *Professional XML.* Wrox Press Ltd., January 2000.

Appendix 3. Example Authorization Interface

This section specifies one feasible algorithm for evaluating a request given a policy. The algorithm implements the semantic meaning of the data structures. This algorithm is not normative but is intended as an illustration how to implement the semantics of EPAL in an actual algorithm. The algorithm should be capable of running in real-time for the purposes of access control, while maintaining the expressiveness of the data structures for rule creation and management. It is not the intention of this specification to deliver an optimized algorithm for general implementation, but rather to develop an algorithm that may be optimized during implementation without modification to the semantics of the language.

An algorithm is developed for simple requests. Algorithm variants, which respect the language semantics but have more general functionality, are considered subsequently. A compound request could be evaluated by decomposing the request and running the algorithm iteratively for each simple request.

Be sure to follow the hyperlinks from the terminology to the glossary in order to obtain a thorough understanding of the semantics.

This interface enables an authorization engine to consume XML inputs while producing XML outputs. This is particularly useful for testing purposes where authorization engines can consume batches of XML inputs while producing batches of outputs that can then be verified for conformance to the semantics.

The following sections define the syntax of an authorization request and result. The namespace URI for the syntax is "http://www.research.ibm.com/privacy/epal/interface".

Appendix 3.1 `<epal-query>` : EPAL Authorization Request

The top-level element `<epal-query>` defines a data structure that is input to an EPAL authorization engine for performing an actual authorization.

Syntax: Each `<epal-query>` element contains a sequence of one or more

<data-user> elements, one or more <data-category> elements, one or more <purpose> elements, and one or more <action> elements. All these elements are of type "referringObjectType". In addition, it contains a list of <container> elements. Each <container> element contains an attribute "refid" that identifies the container of the policy that is instantiated. In addition, it contains one or more "attribute" elements that contain the actual data to be input to the conditions.

Semantics: Each <epal-query> element corresponds to one authorization query. Each "container" element is an instance of the corresponding "container" of the policy. The contained data will be used to evaluate conditions.

Design Notes:

- The interface of this specific implementation defines that the container data needs to be provided up front. More optimized implementations may retrieve the container on demand by means of a call-back interface. This improves the efficiency in particular if multiple different containers have been defined.
- If there is only one container, inputting the data may be preferable. Nevertheless, even in this case, the effort to retrieve and compose the container data may be wasted if the policy does not contain any applicable rules with conditions.

Schema Fragment

```
<xs:element
  name="epal-query">
  <xs:complexType>
    <xs:sequence>
      <!-- The data user accessing the data-->
      <xs:element
        name="data-user"
        type="epal:referringObjectType"
        minOccurs="1"
        maxOccurs="unbounded"></xs:element>
      <!-- The category of the accessed data -->
      <xs:element
        name="data-category"
        type="epal:referringObjectType"
        minOccurs="1"
        maxOccurs="unbounded"></xs:element>
      <!-- The stated purpose of the access -->
      <xs:element
        name="purpose"
        type="epal:referringObjectType"
        minOccurs="1"
        maxOccurs="unbounded"></xs:element>
      <!-- The stated operation of the access -->
      <xs:element
        name="action"
        type="epal:referringObjectType"
        minOccurs="1"
```

```
          maxOccurs="unbounded"></xs:element>
        <!-- The list of XML container instances each corresponding
        <xs:element
          name="container"
          minOccurs="0"
          maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element
                name="attribute"
                minOccurs="0"
                maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="value" type="epal:epalSimpleTy
                  </xs:sequence>
                  <xs:attribute name="refid" use="required" type="xs
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute
              name="refid"
              type="xs:NCName"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

Example: An EPAL Authorization Query

```
<?xml version="1.0"?>
<epal-query
  xmlns="http://www.research.ibm.com/privacy/epal/interface"
  xmlns:epal="http://www.research.ibm.com/privacy/epal"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.research.ibm.com/privacy/epal/inter
  <data-user refid="SalesDepartment"/>
  <data-category refid="CustomerRecord"/>
  <purpose refid="OrderProcessing"/>
  <action refid="Store"/>
  <container refid="CustomerRecord">
    <attribute refid="CustomerID">
      <value>0123456789</value>
      <value>Alice</value>
    </attribute>
  </container>
</epal-query>
```

Appendix 3.2 `<epal-ruling>` : Authorization Result

The top-level element `<epal-ruling>` defines the data structure that is output by the EPAL authorization engine as an answer to an authorization query.

Syntax: Each `<epal-ruling>` element contains an attribute "ruling" with the

ruling, a Boolean attribute "final" that indicates a decision that must not be overruled, a sequence of zero or more <originating-rule> elements of type "referringObjectType" that identify the rules that mandate this ruling, and zero or more `<obligation>` elements. Each obligations element inherits an attribute "refid" from the "referringObjectType". It extends this type by one or more child elements `<originating-rule>` of type "referringObjectType" that identify the rules that mandate this obligation, and a list of `<parameter>` elements of type "epalSimpleType" that define the parameters for this obligation. The required attribute "simpleType" specifies the epalSimpleType of the parameter as specified in the definition of the policy.

Semantics: The `<epal-ruling>` element describes one authorization answer. The ruling attribute contains the ruling returned by the engine, the originating-rules identify the rules that were used to allow/deny the request. The contained list of obligations identifies the resulting obligations. Each obligation has a "refid" that points to an "id" of an obligation of the policy. In addition, the obligation has a list of `<originating-rule/>` sub-elements that identify all rules that mandated this obligation. A policy with "final='true'" always produce rulings with "final='true'".

Design Notes:

- Note that one needs to know the policy in order to be able to interpret this answer.
- We require that each obligation with a given ID and given parameter values is only contained once, i.e., all rules that mandated an identical obligation should be collected and output in the list of `<originating-rule/>` sub-elements of each obligation.
- The 'final' attribute can be used by to decide whether conflicts between two policies may be resolved or should be treated as an error.

Schema Fragment

```
<xs:element
name="epal-ruling">
<xs:complexType>
  <xs:sequence>
    <!-- The rules that mandated authorized access. -->
    <xs:element
      name="originating-rule"
      minOccurs="0"
      maxOccurs="unbounded"
      type="epal:referringObjectType"/>
    <!-- The returned obligations -->
    <xs:element
      name="obligation"
      minOccurs="0"
      maxOccurs="unbounded">
      <xs:complexType>
        <xs:complexContent>
          <xs:extension
            base="epal:referringObjectType">
```

```
                <xs:sequence>
                  <!-- The rules that mandated this obligation -->
                  <xs:element
                    name="originating-rule"
                    minOccurs="1"
                    maxOccurs="unbounded"
                    type="epal:referringObjectType"/>
                  <!-- The XML input for each returned obligation --
                  <xs:element
                    name="parameter"
                    minOccurs="0"
                    maxOccurs="unbounded">
                    <xs:complexType>
                      <xs:simpleContent>
                        <xs:extension
                          base="epal:epalSimpleType">
                          <xs:attribute
                            name="refid"
                            use="required"
                            type="xs:NCName"></xs:attribute>
                          <xs:attribute
                            name="simpleType"
                            use="required"
                            type="xs:anyURI"></xs:attribute>
                        </xs:extension>
                      </xs:simpleContent>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
              </xs:extension>
            </xs:complexContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute
        name="ruling">
         <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="allow"/>
            <xs:enumeration value="deny"/>
            <xs:enumeration value="not-applicable"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
       <xs:attribute name="final" type="xs:boolean"
        use="optional"
        default="false"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Example: An Authorization Answer

```
<?xml version="1.0"?>
<epal-ruling ruling="allow" final="true"
  xmlns="http://www.research.ibm.com/privacy/epal/interface"
  xmlns:epal="http://www.research.ibm.com/privacy/epal"
```

```
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.research.ibm.com/privacy/epal/inter
    <originating-rule refid="rule1"/>
    <obligation refid="Retention">
      <originating-rule refid="rule1"/>
      <parameter refid="Days" simpleType="xs:integer">2</parameter>
      <parameter refid="Hours" simpleType="xs:integer">48</parameter>
    </obligation>
  </epal-ruling>
```

Appendix 4. Glossary

action
: The action being performed in a rule or a request.

component
: A data user, action, data category, or purpose element.

component set
: A component set is a named set of data users, actions, data categories, or purposes. A component set may not mix different types of components, such as an action and a purpose. A component set may contain another component set of the same type of elements. The wildcard is a component set which contains all declared elements of a single type.

compound request
: A request with more than one data user, action, data category, and/or purpose element. The representation of a component as an element or a component set is irrelevant.

compound rule
: A rule with more than one data user, action, data category, and/or purpose. The representation of a component as an element or a component set is irrelevant.

condition
: The condition that indicates when a rule applies. A condition is evaluated on multiple containers.

container
: The container that contains context data.

data category
: The data on which the action is being performed in a rule.

data subject
: The owner whose data is collected and accessed.

data user
: An individual, group, organization, or agent performing the action in a rule.

namespace
: A namespace is a collection of names, identified by a URI reference, which are used in XML documents as element types and attribute names.

namespace prefix
: A namespace prefix is an alias defined in the namespace declaration.

obligation
: One or more actions in a rule that must be performed if the action is

performed.

precedence

Conflict between two or more conflicting rules may be resolved by indicating an ordering. If each rule conflicts with the others, then a total ordering is required to resolve all conflicts. If some rules do not conflict, then a partial ordering is sufficient to resolve all conflicts. This (partial) ordering is called precedence.

privacy administrator

A role within an enterprise which is involved with the design, assessment, or enforcement of a privacy policy. This role may be occupied by a Chief Privacy Officer, another executive, an external auditor, or a member of the information services/technology department.

purpose

The reason that the action is being performed in a rule.

qualified name

A qualified name has a namespace prefix attached to the name, separated by a colon.

request

Any action or set of actions on data can be formulated as a request. The action or actions are mapped to one or more actions. The data is mapped to one or more data categories. The action is performed by one or more data users, which may be individuals, departments, organizations, enterprises, or software agents. One or more purposes must be attached to the request. Therefore, a request contains one or more data users, action, data categories, and purposes. Each component may be represented by the element or by a component set. A request may be of type simple or compound.

rule

A rule defines if a request is permitted or not. A rule contains a ruling of either "allow" or "deny", one or more data users, actions, data categories, and purposes, one conditions container, and one obligations container. A rule may be of type simple or compound.

ruling

A ruling is the state of permission for a request derived from a rule or a set of rules. Examples of rulings include "allow", "deny", and "not-applicable". The "not-applicable" ruling can be used as a default ruling, but cannot be the ruling of a rule. The "not-applicable" ruling states that the policy does not make a statement about a particular request.

schema

A XML schema that defines the syntax of an XML document.

scope

The scope of a request or a rule is defined by four dimensions: the set of applicable data users, the set of applicable actions, the set of applicable data categories, and the set of applicable purposes.

simple request

A request with a exactly one data user, action, data category, and purpose element. The representation of a component as an element or a component set is irrelevant.

simple rule

A rule with exactly one ruling, data user, action, data category, and purpose. The representation of a component as an element or a

component set is irrelevant.

valid XML
: A valid XML document is well-formed XML *and* it conforms to the structure and vocabulary defined in a DTD or an XML Schema.

well-formed XML
: Well-formed XML conforms to the basic syntax rules of XML, but there are no other syntactic guarantees.

XML vocabulary
: An XML vocabulary is a description of XML data that is used as the medium for information exchange, often within a specific domain of human activity, for example, privacy.

Appendix 5: Examples for XACML's Condition Language

Appendix 5.1 General Examples

Appendix 5.1.1 Example Context

The following table depicts the `<xacml-context:Request>` document that is evaluated. The data instantiates containers with "id" "RequestInfo", "LocalContext", "CreditCard", "MileagePlus", "CustomerRecords", and "Synonyms". We omitted the definitions of these containers.

```
<xacml-context:Request xmlns:xacml-context="urn:oasis:names:tc:xacml
  <!--          -->
  <!-- Subject -->
  <!--          -->
  <xacml-context:Subject>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>0123456776543210</xacml-context:
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>10000</xacml-context:AttributeVa
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>2000</xacml-context:AttributeVal
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>100k</xacml-context:AttributeVal
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>12345678</xacml-context:Attribut
    </xacml-context:Attribute>
  </xacml-context:Subject>
  <!--           -->
  <!-- Resource -->
  <!--           -->
  <xacml-context:Resource>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>nc</xacml-context:AttributeValue
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>John Powers</xacml-context:Attri
```

```
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>1989</xacml-context:AttributeVal
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>04</xacml-context:AttributeValue
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>24</xacml-context:AttributeValue
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>13</xacml-context:AttributeValue
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>true</xacml-context:AttributeVal
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>False</xacml-context:AttributeVa
    </xacml-context:Attribute>
  </xacml-context:Resource>
  <!--        -->
  <!-- Action -->
  <!--        -->
  <xacml-context:Action>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>500</xacml-context:AttributeValu
    </xacml-context:Attribute>
  </xacml-context:Action>
  <!--             -->
  <!-- Environment -->
  <!--             -->
  <xacml-context:Environment>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>2002</xacml-context:AttributeVal
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>04</xacml-context:AttributeValue
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>24</xacml-context:AttributeValue
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>nc</xacml-context:AttributeValue
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>NorthCarolina</xacml-context:Att
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>North Carolina</xacml-context:At
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>NC</xacml-context:AttributeValue
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>N.C.</xacml-context:AttributeVal
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>Northcarolina</xacml-context:Att
    </xacml-context:Attribute>
```

```
      <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
        <xacml-context:AttributeValue>true</xacml-context:AttributeVal
      </xacml-context:Attribute>
      <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
        <xacml-context:AttributeValue>True</xacml-context:AttributeVal
      </xacml-context:Attribute>
      <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
        <xacml-context:AttributeValue>yes</xacml-context:AttributeValu
      </xacml-context:Attribute>
      <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
        <xacml-context:AttributeValue>Yes</xacml-context:AttributeValu
      </xacml-context:Attribute>
      <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
        <xacml-context:AttributeValue>1</xacml-context:AttributeValue>
      </xacml-context:Attribute>
      <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
        <xacml-context:AttributeValue>false</xacml-context:AttributeVa
      </xacml-context:Attribute>
      <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
        <xacml-context:AttributeValue>False</xacml-context:AttributeVa
      </xacml-context:Attribute>
      <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
        <xacml-context:AttributeValue>no</xacml-context:AttributeValue
      </xacml-context:Attribute>
      <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
        <xacml-context:AttributeValue>No</xacml-context:AttributeValue
      </xacml-context:Attribute>
      <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
        <xacml-context:AttributeValue>0</xacml-context:AttributeValue>
      </xacml-context:Attribute>
      <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
        <xacml-context:AttributeValue>Powers</xacml-context:AttributeV
      </xacml-context:Attribute>
      <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
        <xacml-context:AttributeValue>powers</xacml-context:AttributeV
      </xacml-context:Attribute>
      <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
        <xacml-context:AttributeValue>100k</xacml-context:AttributeVal
      </xacml-context:Attribute>
      <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
        <xacml-context:AttributeValue>100K</xacml-context:AttributeVal
      </xacml-context:Attribute>
      <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
        <xacml-context:AttributeValue>13</xacml-context:AttributeValue
      </xacml-context:Attribute>
    </xacml-context:Environment>
  </xacml-context:Request>
```

Appendix 5.1.2 Condition Example: Opt-in

The following example determines whether a particular customer opted in for "ThirdPartyMarketing". One way to do this is to compare the value in CustomerRecord/OptInToThirdPartyMarketing" with the literal value 'True'. This, however, can cause problems if the data is not normalized. One way to fix this is to normalize the data first (i.e., all values in the database that mean 'true' are replaced by 'True'). The way this is done here, is to compare the opt-in field with a list of synonyms. If the field value equals one of the synonyms, this is

considered to be true.

```
<xacml:Condition xmlns:xacml="urn:oasis:names:tc:xacml:1.0:policy" F
  <xacml:Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:
  <xacml:ResourceAttributeDesignator AttributeId="urn:ibm:epal:1.0:c
  <xacml:EnvironmentAttributeDesignator AttributeId="urn:ibm:epal:1.
</xacml:Condition>
```

Appendix 5.1.3 Condition Example: Opt-Out

```
<xacml:Condition xmlns:xacml="urn:oasis:names:tc:xacml:1.0:policy" F
  <xacml:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any
    <xacml:Function FunctionId="urn:oasis:names:tc:xacml:1.0:functio
    <xacml:ResourceAttributeDesignator AttributeId="urn:ibm:epal:1.0
    <xacml:EnvironmentAttributeDesignator AttributeId="urn:ibm:epal:
  </xacml:Apply>
</xacml:Condition>
```

Appendix 5.1.4 Condition Example: Credit-limit

This example illustrates the computational capabilities of XACML's condition language. The example restricts access depending on the remaining credit limit.

```
<xacml:Condition xmlns:xacml="urn:oasis:names:tc:xacml:1.0:policy" F
  <xacml:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:int
    <xacml:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:i
      <xacml:ActionAttributeDesignator AttributeId="urn:ibm:epal:1.0
      <xacml:SubjectAttributeDesignator AttributeId="urn:ibm:epal:1.
    </xacml:Apply>
    <xacml:SubjectAttributeDesignator AttributeId="urn:ibm:epal:1.0:
  </xacml:Apply>
  <xacml:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any
    <xacml:Function FunctionId="urn:oasis:names:tc:xacml:1.0:functio
    <xacml:SubjectAttributeDesignator AttributeId="urn:ibm:epal:1.0:
    <xacml:EnvironmentAttributeDesignator AttributeId="urn:ibm:epal:
  </xacml:Apply>
</xacml:Condition>
```

Appendix 5.1.5 Condition Example: Birthdate Computation (Version 1: In XACML)

The following example determines whether a customer is at least "Synonyms/NoChildAge" old. Since XACML's condition language does not allow required date computations, this needed to be done manually given the "pure" dates.

```
<xacml:Condition xmlns:xacml="urn:oasis:names:tc:xacml:1.0:policy" F
  <xacml:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:int
    <xacml:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:i
      <xacml:EnvironmentAttributeDesignator AttributeId="urn:ibm:epa
```

```
        <xacml:ResourceAttributeDesignator AttributeId="urn:ibm:epal:1
      </xacml:Apply>
      <xacml:EnvironmentAttributeDesignator AttributeId="urn:ibm:epal:
    </xacml:Apply>
    <xacml:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and
      <xacml:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:i
        <xacml:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function
          <xacml:EnvironmentAttributeDesignator AttributeId="urn:ibm:e
          <xacml:ResourceAttributeDesignator AttributeId="urn:ibm:epal
        </xacml:Apply>
        <xacml:EnvironmentAttributeDesignator AttributeId="urn:ibm:epa
      </xacml:Apply>
      <xacml:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:i
        <xacml:EnvironmentAttributeDesignator AttributeId="urn:ibm:epa
        <xacml:ResourceAttributeDesignator AttributeId="urn:ibm:epal:1
      </xacml:Apply>
    </xacml:Apply>
    <xacml:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and
      <xacml:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:i
        <xacml:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function
          <xacml:EnvironmentAttributeDesignator AttributeId="urn:ibm:e
          <xacml:ResourceAttributeDesignator AttributeId="urn:ibm:epal
        </xacml:Apply>
        <xacml:EnvironmentAttributeDesignator AttributeId="urn:ibm:epa
      </xacml:Apply>
      <xacml:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:i
        <xacml:EnvironmentAttributeDesignator AttributeId="urn:ibm:epa
        <xacml:ResourceAttributeDesignator AttributeId="urn:ibm:epal:1
      </xacml:Apply>
      <xacml:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:i
        <xacml:EnvironmentAttributeDesignator AttributeId="urn:ibm:epa
        <xacml:ResourceAttributeDesignator AttributeId="urn:ibm:epal:1
      </xacml:Apply>
    </xacml:Apply>
  </xacml:Condition>
```

Appendix 5.1.6 Condition Example: Birthdate Computation (Version 2: With External Computation)

The following example only re-uses an externally computed value to determine the age. The basic idea is that the environment needs to fetch the dates anyway. Before actually wrapping it into a container for the EPAL system, it does the date-computation. In Java, this is much easier than in XACML.

```
  <xacml:Condition xmlns:xacml="urn:oasis:names:tc:xacml:1.0:policy" F
    <xacml:ResourceAttributeDesignator AttributeId="urn:ibm:epal:1.0:c
    <xacml:EnvironmentAttributeDesignator AttributeId="urn:ibm:epal:1.
  </xacml:Condition>
```

Appendix 5.2 Medical Examples

Appendix 5.2.1 Example Context

```
```

```
<xacml-context:Request xmlns:xacml-context="urn:oasis:names:tc:xacml
  <xacml-context:Subject>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>John Doe</xacml-context:Attribut
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>50B</xacml-context:AttributeValu
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>Emergency Room</xacml-context:At
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>ER</xacml-context:AttributeValue
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>49A</xacml-context:AttributeValu
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>true</xacml-context:AttributeVal
    </xacml-context:Attribute>
  </xacml-context:Subject>
  <xacml-context:Resource>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>50B</xacml-context:AttributeValu
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>John Doe</xacml-context:Attribut
    </xacml-context:Attribute>
    <xacml-context:Attribute AttributeId="urn:ibm:epal:1.0:container
      <xacml-context:AttributeValue>Bull Doc</xacml-context:Attribut
    </xacml-context:Attribute>
  </xacml-context:Resource>
</xacml-context:Request>
```

Appendix 5.2.2 Condition Example: Nurse on Duty

The following example determines whether a nurse is allowed to read a particular patient's record without any obligations. The conditions are that the nurse is sometimes working on a station where the patient sometimes lies and that the nurse needs to be on duty.
Note that in a real-world hospital scenario, there will usually be higher-precedence rules allowing medical personnel emergency access with the obligation that an alarm will be raised (e.g., by writing to a dedicated log file).

```
<xacml:Condition xmlns:xacml="urn:oasis:names:tc:xacml:1.0:policy" F
  <xacml:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any
    <xacml:Function FunctionId="urn:oasis:names:tc:xacml:1.0:functio
    <xacml:ResourceAttributeDesignator AttributeId="urn:ibm:epal:1.0
    <xacml:SubjectAttributeDesignator AttributeId="urn:ibm:epal:1.0:
  </xacml:Apply>
  <xacml:SubjectAttributeDesignator AttributeId="urn:ibm:epal:1.0:co
</xacml:Condition>
```

Appendix 5.2.3 Condition Example: Primary Care Physician

This condition checks that the data-user is in fact a primary care physician on the patient. This concept is sometimes called "dynamic roles". Note that if the condition would not be included in the policy (e.g., by assigning roles externally), it would be harder to review the conditions under which such roles can be played.

```
<xacml:Condition xmlns:xacml="urn:oasis:names:tc:xacml:1.0:policy" F
  <xacml:Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:
  <xacml:SubjectAttributeDesignator AttributeId="urn:ibm:epal:1.0:co
  <xacml:ResourceAttributeDesignator AttributeId="urn:ibm:epal:1.0:c
</xacml:Condition>
```

Appendix 6 Technological Context of EPAL

The Platform for Privacy Preferences (P3P)

The Platform for Privacy Preferences from the Worldwide Web Consortium (see P3P) enables a web-site to describe what kind of data is collected and how this data will be used. A P3P policy may contain the purposes, the recipients, the retention period, 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, EPAL defines the privacy-practices that are implemented *inside* an enterprise. Since this depends on internal details of the enterprise, it results in much more detailed policies that can be enforced and audited automatically. However, the resulting privacy guarantees can sometimes be simplified as a P3P promise that is offered for the users of the services.

CPExchange

The Customer Profile Exchange Specification (CPExchange, see CPExchange) defines a data format for disclosing customer data from one party (customer/enterprise) to another. It defines basic and complex data types for many different kinds of personal data (e.g., fields for address, name, hobbies, ...). It enables the specification of privacy meta-information as an option. The privacy meta-information includes the exchange partners, the applicable jurisdiction, and a privacy declaration (based on P3P). Both data types are designed in UML and then translated into XML in a standardized way.

The main focus of CPExchange lies in standardizing the data exchange format. The privacy meta-information is less expressive than EPAL. Consequently, data disclosed using CPExchange may be controlled with EPAL policies instead of using their privacy meta-data.

XACML

XACML is a general purpose and extensible access control language. Access control is a tool to define and later decide whether a user U is allowed to perform an action A on an object O. XACML lacks the privacy-specific notion of

purposes. Unlike XACML, EPAL has an explicit notion of purposes and a syntax that simplifies the formalization of privacy policies.

Appendix 7. Complete XML Schema for EPAL

```
<?xml version="1.0"?>
<!-- %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% -->
<!--                                                                -->
<!-- The Enterprise Privacy Authorization Language (EPAL)           -->
<!--                                                                -->
<!--  Authors:                                                      -->
<!--      Paul Ashley <pashley at us.ibm.com>                          -->
<!--      Satoshi Hada <satoshih at jp.ibm.com>                        -->
<!--      G&uuml;nter Karjoth <gka at zurich.ibm.com>                  -->
<!--      Calvin Powers <cspowers at us.ibm.com>                       -->
<!--      Matthias Schunter <mts at zurich.ibm.com>                    -->
<!--                                                                -->
<!--  Abstract: This schema defines the structure of EPAL           -->
<!--      privacy policies.                                         -->
<!--                                                                -->
<xs:schema xmlns:epal="http://www.research.ibm.com/privacy/epal"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xacml="urn:oasis:names:tc:xacml:1.0:policy"
  targetNamespace="http://www.research.ibm.com/privacy/epal"
  xml:lang="en"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  version="b">

  <!-- %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% -->
  <!--                                                                -->
  <!-- Import other namespaces                                        -->
  <!--                                                                -->

  <xs:import namespace="urn:oasis:names:tc:xacml:1.0:policy"
    schemaLocation="cs-xacml-schema-policy-01.xsd"/>

  <!-- %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% -->
  <!--                                                                -->
  <!-- DECLARATION OF REUSABLE TYPES                                  -->
  <!--                                                                -->
  <!-- Types needed often are declared here.                          -->

  <!-- Elements which have an identifier should be of this type -->
  <xs:complexType name="identifiedObjectType">
    <xs:attribute name="id" use="required" type="xs:NCName"></xs:att
  </xs:complexType>

  <!-- Elements which refer to an identifier should be of this type
  <xs:complexType name="referringObjectType">
    <xs:attribute name="refid" use="required" type="xs:NCName"></xs:
  </xs:complexType>


  <!-- Identified Elements which have descriptions should be of this
  <xs:complexType name="describedObjectType">
    <xs:complexContent>
      <xs:extension base="epal:identifiedObjectType">
```

```
      <xs:sequence>
        <xs:element name="short-description" minOccurs="0"
          maxOccurs="unbounded">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attribute name="language" use="optional"
                  default="en"
                  type="xs:language"></xs:attribute>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
        <xs:element name="long-description" minOccurs="0"
          maxOccurs="unbounded">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attribute name="language" use="optional"
                  default="en"
                  type="xs:language"></xs:attribute>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
        <xs:element name="property" minOccurs="0"
          maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="value" type="epal:epalSimpleType"
                minOccurs="0"
                maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="id" use="required" type="xs:NCName
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- Elements which can participate in a hierarchy should have thi
<xs:complexType name="hierarchicalType">
  <xs:complexContent>
    <xs:extension base="epal:describedObjectType">
      <xs:attribute name="parent" use="optional" type="xs:NCName">
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- Contact information   -->
<xs:complexType name="contactInfoType">
  <xs:sequence>
    <xs:element name="name" type="xs:string" minOccurs="1"
      maxOccurs="1"></xs:element>
    <xs:element name="organization" type="xs:string" minOccurs="1"
      maxOccurs="1"></xs:element>
    <xs:element name="e-mail" type="xs:string" minOccurs="1"
      maxOccurs="1"></xs:element>
```

```
    <xs:element name="address" type="xs:string" minOccurs="1"
      maxOccurs="1"></xs:element>
    <xs:element name="country" type="xs:string" minOccurs="1"
      maxOccurs="1"></xs:element>
  </xs:sequence>
</xs:complexType>

<!-- A Descriptor of an attribute -->
<xs:complexType name="attributeDefinitionType">
  <xs:complexContent>
    <xs:extension base="epal:describedObjectType">
      <xs:attribute name="simpleType">
        <xs:simpleType>
          <xs:restriction base="xs:anyURI">
            <xs:enumeration
              value="http://www.w3.org/2001/XMLSchema#string"/>
            <xs:enumeration
              value="http://www.w3.org/2001/XMLSchema#anyURI"/>
            <xs:enumeration
              value="http://www.w3.org/2001/XMLSchema#boolean"/>
            <xs:enumeration
              value="http://www.w3.org/2001/XMLSchema#hexBinary"/>
            <xs:enumeration
              value="http://www.w3.org/2001/XMLSchema#base64Binary
            <xs:enumeration
              value="http://www.w3.org/2001/XMLSchema#double"/>
            <xs:enumeration
              value="http://www.w3.org/2001/XMLSchema#dateTime"/>
            <xs:enumeration
              value="http://www.w3.org/2001/XMLSchema#date"/>
            <xs:enumeration
              value="http://www.w3.org/2001/XMLSchema#integer"/>
            <xs:enumeration
              value="http://www.w3.org/TR/2002/WD-xquery-operators
            <xs:enumeration
              value="http://www.w3.org/TR/2002/WD-xquery-operators
            <xs:enumeration
              value="urn:oasis:names:tc:xacml:1.0:data-type:x500Na
            <xs:enumeration
              value="urn:oasis:names:tc:xacml:1.0:data-type:rfc822
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="minOccurs" use="optional" default="1">
        <xs:simpleType>
          <xs:union memberTypes="xs:nonNegativeInteger">
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="unbounded"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:union>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="maxOccurs" use="optional" default="1">
        <xs:simpleType>
          <xs:union memberTypes="xs:nonNegativeInteger">
            <xs:simpleType>
              <xs:restriction base="xs:string">
```

```
                <xs:enumeration value="unbounded"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:union>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- A Descriptor of a container attribute -->
<xs:complexType name="containerAttributeDefinitionType">
  <xs:complexContent>
    <xs:extension base="epal:attributeDefinitionType">
      <xs:attribute name="origin" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:NCName">
            <xs:enumeration value="data-subject"/>
            <xs:enumeration value="data-user"/>
            <xs:enumeration value="filled-form"/>
            <xs:enumeration value="resource"/>
            <xs:enumeration value="action"/>
            <xs:enumeration value="other"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="auditable" use="optional" type="xs:boole
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- Allowed EPAL attribute types;
       This list MUST be in sync with list in attributeDefinitionT
       Currently, four data types (dateTimeDuration, yearMonthDura
<xs:simpleType name="epalSimpleType">
  <xs:union
    memberTypes="xs:string xs:boolean xs:integer xs:double xs:date
</xs:simpleType>


<!-- Information on a policy or vocabulary -->

<xs:complexType name="infoType">
  <xs:complexContent>
    <xs:extension base="epal:describedObjectType">
      <xs:sequence>
        <xs:element name="issuer" minOccurs="0" maxOccurs="1"
          type="epal:contactInfoType"></xs:element>
        <xs:element name="location" minOccurs="0" maxOccurs="1"
          type="xs:anyURI"></xs:element>
        <xs:element name="version-info" minOccurs="1" maxOccurs="1
          <xs:complexType>
            <xs:attribute name="test" type="xs:boolean"
              default="false"
              use="optional"></xs:attribute>
            <xs:attribute name="start-date" type="xs:dateTime"
              use="required"></xs:attribute>
            <xs:attribute name="revision-number" type="xs:string"
              use="required"></xs:attribute>
```

```
            <xs:attribute name="last-modified" type="xs:dateTime"
              use="required"></xs:attribute>
            <xs:attribute name="end-date" type="xs:dateTime"
              use="optional"></xs:attribute>
            <xs:attribute name="superseded-by" type="xs:dateTime"
              use="optional"></xs:attribute>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>


<!-- Importing policies and vocabularies -->

<xs:complexType name="importStatementType">
  <xs:complexContent>
    <xs:extension base="epal:describedObjectType">
      <xs:attribute name="location" use="required" type="xs:anyURI
      <xs:attribute name="digest" use="optional"
        type="xs:base64Binary"></xs:attribute>
      <xs:attribute name="digestAlgorithm" use="optional"
        type="xs:anyURI"
        default="http://www.w3.org/2000/09/xmldsig#sha1"></xs:attr
      <xs:attribute name="canonicalizationAlgorithm" use="optional
        type="xs:anyURI"
        default="http://www.w3.org/2001/10/xml-exc-c14n#"></xs:att
      <xs:attribute name="revision" use="optional" type="xs:NCName
    </xs:extension>
  </xs:complexContent>
</xs:complexType>


<!-- %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% -->
<!--                                                                -->
<!-- DECLARATION OF THE <epal-vocabulary> TOPLEVEL ELEMENT          -->
<!--                                                                -->

<xs:element name="epal-vocabulary">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="vocabulary-information" minOccurs="1"
        maxOccurs="1"
        type="epal:infoType"></xs:element>
      <xs:element name="data-user" minOccurs="0" maxOccurs="unboun
        <xs:complexType>
          <xs:complexContent>
            <xs:extension base="epal:hierarchicalType">
              <xs:sequence minOccurs="0" maxOccurs="1">
                <xs:element name="contact-info"
                  type="epal:contactInfoType"
                  minOccurs="0"
                  maxOccurs="1"></xs:element>
              </xs:sequence>
            </xs:extension>
          </xs:complexContent>
        </xs:complexType>
      </xs:element>
```

```
      <xs:element name="data-category" minOccurs="0"
        maxOccurs="unbounded"
        type="epal:hierarchicalType"></xs:element>
      <xs:element name="purpose" type="epal:hierarchicalType"
        minOccurs="0"
        maxOccurs="unbounded"></xs:element>
      <xs:element name="action" type="epal:describedObjectType"
        minOccurs="0"
        maxOccurs="unbounded"></xs:element>
      <xs:element name="container" minOccurs="0" maxOccurs="unboun
        <xs:complexType>
          <xs:complexContent>
            <xs:extension base="epal:describedObjectType">
              <xs:sequence minOccurs="1" maxOccurs="1">
                <xs:element name="attribute"
                  type="epal:containerAttributeDefinitionType"
                  minOccurs="1"
                  maxOccurs="unbounded"></xs:element>
              </xs:sequence>
            </xs:extension>
          </xs:complexContent>
        </xs:complexType>
        <!-- Declared attribute id's must be unique within each co
        <xs:unique name="uniqueAttributeDef">
          <xs:selector xpath="attribute"/>
          <xs:field xpath="@id"/>
        </xs:unique>
      </xs:element>
      <xs:element name="obligation" minOccurs="0"
        maxOccurs="unbounded">
        <xs:complexType>
          <xs:complexContent>
            <xs:extension base="epal:describedObjectType">
              <xs:sequence minOccurs="1" maxOccurs="unbounded">
                <xs:element name="parameter"
                  type="epal:attributeDefinitionType"
                  minOccurs="0"
                  maxOccurs="unbounded"></xs:element>
              </xs:sequence>
            </xs:extension>
          </xs:complexContent>
        </xs:complexType>
        <!-- Declared parameter id's must be unique within each ob
        <xs:unique name="uniqueParamDef">
          <xs:selector xpath="parameter"/>
          <xs:field xpath="@id"/>
        </xs:unique>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="version" type="xs:string" default="1.0"/>
  </xs:complexType>

  <!-- Key and References for data-users/@id -->
  <xs:key name="data-user-identifier">
    <xs:selector xpath="./epal:data-user"></xs:selector>
    <xs:field xpath="@id"></xs:field>
  </xs:key>

  <!-- Key and References for data-category/@id -->
```

```
    <xs:key name="data-category-identifier">
      <xs:selector xpath="./epal:data-category"></xs:selector>
      <xs:field xpath="@id"></xs:field>
    </xs:key>

    <!-- Key and References for purpose/@id -->
    <xs:key name="purpose-identifier">
      <xs:selector xpath="./epal:purpose"></xs:selector>
      <xs:field xpath="@id"></xs:field>
    </xs:key>

    <!-- Key and References for action/@id -->
    <xs:key name="action-identifier">
      <xs:selector xpath="./epal:action"></xs:selector>
      <xs:field xpath="@id"></xs:field>
    </xs:key>

    <!-- Key and References for container/@id -->
    <xs:key name="container-identifier">
      <xs:selector xpath="./epal:container"></xs:selector>
      <xs:field xpath="@id"></xs:field>
    </xs:key>
    <xs:keyref name="container-reference"
      refer="epal:container-identifier">
      <xs:selector xpath="./epal:condition/epal:evaluates-container"
      <xs:field xpath="@refid"></xs:field>
    </xs:keyref>

    <!-- Key and References for condition/@id -->
    <xs:key name="condition-identifier">
      <xs:selector xpath="./epal:condition"></xs:selector>
      <xs:field xpath="@id"></xs:field>
    </xs:key>

    <!-- Key and References for obligation/@id -->
    <xs:key name="obligation-identifier">
      <xs:selector xpath="./epal:obligation"></xs:selector>
      <xs:field xpath="@id"></xs:field>
    </xs:key>
  </xs:element>

  <!-- %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% -->
  <!--                                                                  -->
  <!-- DECLARATION OF THE <epal-policy> TOPLEVEL ELEMENT     -->
  <!--                                                                  -->

  <xs:element name="epal-policy">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="policy-information" minOccurs="1"
          maxOccurs="1"
          type="epal:infoType"></xs:element>
        <xs:element name="epal-vocabulary-ref" minOccurs="1"
          maxOccurs="1"
          type="epal:importStatementType"/>
        <xs:element name="condition" minOccurs="0" maxOccurs="unboun
          <xs:complexType>
            <xs:complexContent>
              <xs:extension base="epal:describedObjectType">
```

```
        <xs:sequence>
          <xs:element name="evaluates-container" minOccurs="
            maxOccurs="unbounded"
            type="epal:referringObjectType"/>
          <xs:element minOccurs="1" maxOccurs="1"
            ref="xacml:Condition"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<xs:element name="rule" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="epal:describedObjectType">
        <xs:sequence>
          <xs:element name="data-user"
            type="epal:referringObjectType"
            minOccurs="1"
            maxOccurs="unbounded"></xs:element>
          <xs:element name="data-category"
            type="epal:referringObjectType"
            minOccurs="1"
            maxOccurs="unbounded"></xs:element>
          <xs:element name="purpose"
            type="epal:referringObjectType"
            minOccurs="0"
            maxOccurs="unbounded"></xs:element>
          <xs:element name="action"
            type="epal:referringObjectType"
            minOccurs="1"
            maxOccurs="unbounded"></xs:element>
          <xs:element name="condition"
            type="epal:referringObjectType"
            minOccurs="0"
            maxOccurs="unbounded"></xs:element>
          <xs:element name="obligation" minOccurs="0"
            maxOccurs="unbounded">
            <xs:complexType>
              <xs:complexContent>
                <xs:extension base="epal:referringObjectType
                  <xs:sequence minOccurs="0" maxOccurs="1">
                    <xs:element name="parameter" minOccurs="
                      maxOccurs="unbounded">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="value"
                            type="epal:epalSimpleType"
                            minOccurs="0"
                            maxOccurs="unbounded"/>
                        </xs:sequence>
                        <xs:attribute name="refid"
                          use="required"
                          type="xs:NCName"></xs:attribute>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:extension>
```

```
                    </xs:complexContent>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
              <xs:attribute name="ruling">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                    <xs:enumeration value="allow"/>
                    <xs:enumeration value="deny"/>
                    <xs:enumeration value="obligate"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:attribute>
            </xs:extension>
          </xs:complexContent>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="final" type="xs:boolean"
      use="optional"
      default="false"/>
    <xs:attribute name="version" type="xs:string" default="1.0"/>
    <xs:attribute name="global-condition" type="xs:NCName"
      use="optional"/>
    <xs:attribute name="default-ruling" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="allow"/>
          <xs:enumeration value="deny"/>
          <xs:enumeration value="not-applicable"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>
</xs:schema>
```