# Describing the P3P base data schema using OWL

Giles Hogben,
European Commission,
Joint Research Centre,
Via Enrico Fermi 1,
21020 VA, Ispra, Italy
+39 0332789187

giles.hogben@jrc.it

## ABSTRACT

This paper describes use cases and requirements for a privacy policy data schema. It describes problems with existing schemas in relation to these requirements (P3P 1.0, P3P 1.1 and RDFS schema for P3P). It proposes and motivates the use of an OWL schema to describe the same semantics, which fulfils all the requirements and may be used in a semantic web based privacy and identity management context. It describes the advantages which this gives to a policy evaluation engine based on such a schema and describes some of the reasoning use cases addressed in modelling the schema.

Modelling the schema using OWL appears simple at first sight, because the entire schema can be constructed with OWL-Lite predicates or using one custom predicate. However, the fact that modal logical statements must be made about data types in the schema (e.g. Organization x May Collect Data of type Y) makes reasoning over the typing schema challenging. The paper also looks at syntactic and semantic validation using the schema as well as extensions and modifications to the vocabulary items supported.

## Categories and Subject Descriptors

D.3.3 [**Programming Languages**]: Language Constructs and Features – *abstract data types.*

E.1 **[Data Structures]:** *Distributed data structures, graphs and networks.*

## General Terms

Standardization, Languages, Theory

## Keywords

Policy engineering, P3P and policies, Semantic Web Groundings

**This work was supported by the IST PRIME project; it represents the view of the authors only.**

## 1. Introduction

P3P [1] is a policy framework for describing web site privacy practices using XML. The main body of a P3P policy is made up of a set of statements about data collection practices. Each statement refers to the practices claimed for a certain type of data, described by "data elements" which are typed and validated according to a special P3P data schema. The vast majority of existing policies use the P3P Base Data Schema [2], the base typing schema provided by P3P for this purpose. The exact specification of this schema is outlined in [3] .

The P3P 1.0 base data schema is intended to provide a base set of data types to cover the most common categories of personal data about which P3P privacy policy statements might be made. The schema also provides extensibility mechanisms for expressing custom types. The fact that it is one of the only mechanisms to offer this functionality for such a broad range of data types has meant that the P3P data schema has also been adopted for several other use-cases which were unforeseen by the P3P working group.

This paper shows how an OWL [4] based semantics can be used in these use cases to fulfil many of the requirements that are problematic for the P3P base data schema. The schema is designed to fit into the policy architecture framework proposed in "P3P Using the Semantic Web (OWL Ontology, RDF Policy and RDQL Rules)" [5]

One important problem resolved by this paper is that P3P makes statements about data types in the schema which use modal logic (e.g. Organization x *May* Collect Data of type Y). This makes reasoning over the typing schema challenging. The paper presents a solution for achieving this using available OWL reasoning tools.

## 2. Use cases

Our motivation for creating an OWL data schema for privacy policy languages was broader than the usage scenarios envisaged for P3P and our schema is designed to cover scenarios envisaged for both P3P and nascent enterprise privacy management standards such as EPAL [6] and the technologies being developed by the PRIME project [7], as well as to satisfy identity management requirements such as those for automated form filling and pseudonym management. In practice, the P3P data schema has already been used beyond its design remit in many projects [7],[8],[9] and it is therefore an urgent need to provide a schema which can satisfy these broader requirements.

The schema we propose should allow the description of data types in the following policy contexts:

a. Requesting data or credentials (the auto-form filling/Xforms [10] scenario). The data typing schema is used to describe the type of data to be inserted into a form field.

b. Describing data or credentials (metadata). The data typing schema is used to describe data or credential instances.

c. Describing data practices (P3P type scenario) according to data types. The data typing schema is used to describe types of data to which certain data handling practices may be applied.

d. Application of access control rules. The schema is used to describe types of data to which groups of access control rules should be applied. For example it should be able to describe the type used in the natural language rule: "Do not give user emaill addresses to third parties".

## 3. Requirements on a privacy and Identity Management data schema

An analysis of the above use cases has led to the following specific requirements:

1. Data types must describe data (i.e. the object is the information), not properties of individuals. This is needed to allow for types of data which are personal but do not necessarily apply to individuals. It is also correct semantically as data handling policies for example, make statements about data and not about individuals and their properties.

   This implies that data types must be modelled as classes rather than properties. So for example "email" means "data of type email" rather than "the email property of user x". This allows the model to be centred around statements about data collection practices rather than statements about individuals and their properties. It is more difficult to use OWL to provide meta-information about properties than it is about classes. The semantics of properties also breaks down when it comes to data types such as "user". If user is a property, what does it refer to? [11] breaks the schema down into classes and instances so that "user" is a class, while "prefix" is the value of a property, but this seems unnecessarily complex as all the types in the P3P schema can be described as classes of data.

2. The schema should distinguish between abstract (cannot be instantiated) and concrete types. This gives the possibility to use the schema for data and credential requests such as automated form filling. It is not possible to use the P3P base data schemas for automatic form requests because it does not satisfy this requirement. But if types are designated abstract and concrete status, then an application can ask for say "user, online data" and a reasoning engine can drill down the schema to dig out the concrete types "home page, email address etc…

3. It should be easily possible within the semantics to apply meta-data both to instances of data types and to the types themselves. This requirement is derived from the need both to describe data literals, and to make statements about classes of data when describing data handling practices. This is another strong reason to model data as classes and not as properties, because it is much more natural to apply metadata to classes rather than to properties.

4. The schema should be able to describe both literals (data submissions) and classes of data.

5. The semantics of the OWL base data schema should not conflict with any semantics which can be inferred from the P3P Base Data Schema unless this can be shown to be inconsistent with other requirements. The vocabulary used in the P3P Base Data Schema semantics is based on a standards process and thereby represents a consensus on the actual data types required for describing most data. Although the syntax and semantics is poorly expressed, the actual taxonomy represented has considerable value.

6. The number of classes defined should be minimized. As with any data model, redundancy is to be avoided and the description of classes should be as normalized as possible.

7. The schema should provide validation functionality for allowed data types and for the syntax of instances of a designated type. If the schema is to be used for typing instances, it is natural to provide syntactic validation functionality.

8. The schema should use standardized, well-defined syntax. In order to foster adoption.

9. The schema should have a well-defined semantics. This makes it easy to apply the schema to new use cases.

## 4. Existing data schemas in relation to requirements

### 4.1 P3P1.0 base data schema

Some literature exists outlining problems with the Base Data Schema [11],[12]. [12] cites the over complexity of the syntax and proposes an XML schema version of the syntax which has now been incorporated into the P3P1.1 working draft [13].

In relation to the above requirements, the P3P1.0 data schema has the following specific problems:

1. (Requirement 2) It does not distinguish between abstract and instantiatable types.

2. (Requirement 7.) There is no provision for validation of instance data.

3. (Requirement 8) The schema uses a highly complex and obscure custom syntax which:

   a. Does not use mechanisms available in XML syntax, which are commonly used to model semantics. For example it does not use nesting to indicate subclass or other class relationships, but rather a convoluted custom syntax involving string matching.

   b. Is not well defined – the syntax used for defining the relations between allowed data types can only be deduced by examining the base data schema and examples. It does not follow directly from the specification document. To take one example out of several:

   Data Structures are abstract types (for example "POSTAL") which appear in the schema, but are never actually allowed as types in data elements. They serve to group concrete elements together. Nowhere in the specification document is it stated that in a data schema, data structures refer to their child elements by parsing the data element name, splitting it by "." delimiters and then taking the first token!

   Another example is that, according to [1], the categories of the data schema (broad classes of data types) follow a "bubble-up rule". The meaning of this phrase is not precisely explained in the P3P specification, but by examining the base data schema, one can deduce that it means data types which can be expanded into further structures must inherit any categories which are valid for those structures. In fact, however, not all the categories quoted in the P3P base data schema do follow a "bubble-up rule". For example, the postal.name data structure is not (according to the official specification [2]) assigned to the category demographic of its child data structure, personname prefix.

Many of these problems were not picked up because the syntax is so obscure.

4. (Requirement 9) The semantics is also not well defined. There is a confusion between classes of data and properties of individuals. For example, "user.employer" :"Name of User's Employer" seems to model an object (the user) and its properties. But "dynamic.cookies" "Use of HTTP Cookies" models an abstract class of data (dynamic.cookies) and not the "cookies" property of a "dynamic" object. Furthermore the specification does not define whether syntax such as "user.email" is meant to represent a set of user's email addresses – or the intersection of the class of user data with the class of email data. This has important implications when trying to describe instance data.

   Furthermore the semantics of the dot relationship between the data types is not made clear. The specification says that elements "include" other elements, implying that the relation is equivalent to "subclass" but elements are also included by several disjoint classes, making this incoherent. It is one of the aims of this paper to make clear the exact semantics of the base data schema in order to model it using OWL.

## 4.2 The P3P 1.1 Data Schema

The P3P 1.1 Data Schema (still in draft at the time of writing) [13] addresses some of the problems outlined in 4.1

a. The P3P 1.1 Data Schema prescribes a standardized XML syntax for describing the relationships between data elements. Abstract "data structures" are abandoned, and relationships are described simply by nesting tags within each other. Custom schemas can be created by referencing another XML schema.

b. A more precise semantics for the elements can also be derived from the specification of this document. That " for an element <B> to be defined as an allowed child of element <A> means if the policy states that it may collect data of type <A>, *then it can also be taken to state that it may also collect data of type <B>.* "

The use of XML rather than description logic syntax is however fundamentally limited because

a. XML semantics is only informal and is based on a questionable interpretation of its syntax.

b. In practical terms, semantics expressed using a custom interpretation of XML syntax such as in the P3P 1.1 Data Schema cannot be interfaced with reasoning engines in the way that RDF + OWL can. Much of the utility of the data schema is lost because reasoning is proceduralized in program code which then cannot be reused.

c. Since the structure of the schema is not well suited to representation as a tree (as opposed to a directed graph), a custom syntax has to be used to represent the structure.

## 4.3 The RDFS Schema for P3P

[14] is a previous attempt at producing a P3P data schema using Description Logic syntax (RDFS). The RDF Schema for P3P models data types as properties and describes a different class for every possible combination of basic data types. While it does provide a well-defined "p3p:extends" relation between data types, it also describes all possible properties created by this extension relation. This is highly redundant as the extension relation is then contained in the syntax of the class names. It also has over 350 classes of data instead of less than 80 classes which are used to compose these.

Furthermore, the definition of the extends relation as "Extends another dataElementComponent" suggests a parallel with object oriented design, which is not consistent with the semantics. (Does a user's email extend the properties of a user?).

Finally, the RDFS schema's use of properties rather than classes does not fulfil requirement 1.

## 5. Modelling Class Relationships in OWL

OWL provides a syntax which fulfils all the above requirements. In using OWL, we implement the base data schema semantics in the context of a semantic web enabled privacy architecture as described in [5]. We chose OWL instead of other object oriented modelling languages because it gives a standard XML based syntax which provides the functionality required by the semantic web based architecture in which the schema is used.

## 5.1 Reasoning use cases

We begin by describing a reasoning use case and then go on to show how this can be implemented using an OWL-based semantics which accurately reflects the intended semantics of the P3P1.0 base data schema.

Identity management and access control systems typically need to reason over policies or requests for broad data types which correspond to specific data types in a store. Some important reasoning use cases are as follows:

5.1.1. A typical statement of collection practices specifies that the service may collect any data which is in both User and Name classes (i.e. specializing Name as a User, not a Business, name)

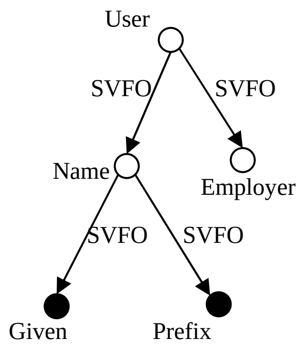

Figure 1. Classes related by SVFO

The diagram represents this scenario. The reasoner is required to deduce that this implies that the service may also collect the data classes Given and Prefix (concrete types are filled in black, inferences dotted lines).

5.1.2. A policy states that a company collects any data of type User, whereas a preference rule refers to protecting Online data. The reasoner needs to infer that if a service might collect User data, it might also collect Online data.

5.1.3. A policy gives sensitivity ratings to data types which determine their release by an identity management policy. The reasoner selects the type with the maximum or minimum rating in a given context.

Formally speaking, 5.1.1 and 5.1.2 require a system of modal logic since it is describing possibilities. However, we show below that it is possible to produce the required entailments using an ordinary propositional logic system such as prolog.

## 5.2 Modelling the entailments using the structure of the P3P 1.0 Data Schema

The P3P1.0 specification states: "*P3P1.0 Data elements are organized into a hierarchy based on the data element name as specified by the data schema. A data element automatically "includes" all of the data elements below it in the hierarchy. For example, the data element representing "the user's name" includes the data elements representing "the user's given name", "the user's family name", and so on. Thus the data elements user.name.given, user.name.family, and user.name.nickname are all children of the data element user.name, which is in turn a child of the data element user.*"

It is important to note that the exact meaning of "includes" here is not specified. It appears to mean "subclasses" but, if one examines the structure and semantics of the schema, this cannot be the case because data elements such as personname are used as part of disjoint classes such as User and Business.

Data schemas often need to reuse a common group of data elements. P3P 1.0 data schemas support this through named data structures. A data structure is a named, abstract definition of a group of data elements. The name of the data structure itself (e.g. postal) is never actually used in a data element. We quote the P3P 1.0 Specification's example:

```
<DATA-STRUCT name="date.ymd.year"
  short-description="Year" />
<DATA-STRUCT name="date.ymd.month"
  short-description="Month"/>
<DATA-STRUCT name="date.ymd.day"
  short-description="Day"/>
```

The structure of the P3P base data schema is, as [11] correctly points out, not a forest, but a semi-lattice, as elements are used repeatedly in different contexts. Figure 2 below is a Venn diagram showing a fragment of the schema classes, which illustrates the relation that holds between the data elements. The figure shows the Classes User and ThirdParty, which both include some (>1) values from Cert, Personname, Bdate and Gender.

All data elements in the P3P base data schema which are "included" are in fact related as shown. That is if A "includes" (B and C) then A contains some values from B and some values from C and no other values unless otherwise stated (note that in fig 2, User is shown outside of Cert, Personname etc… because it also "includes" other data elements.)

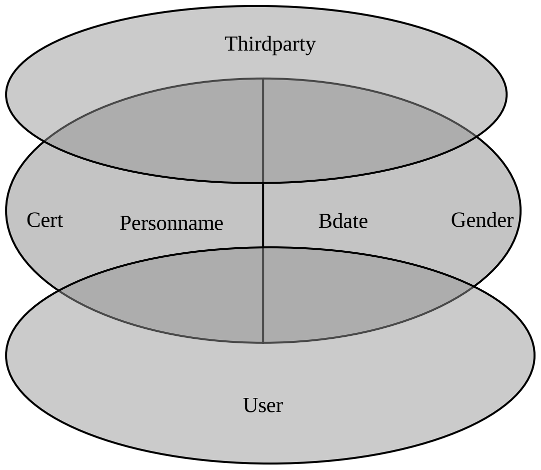

Figure 2: Fragment of schema classes as Venn Diagram

## 5.3 OWL semantics

If we model all data elements as classes of data (as shown in figure 2), then a single relationship, "SomeValuesFromOnly" can be used to define the entire P3P base data schema using OWL.

In formal set theoretic notation, then, we wish to express a relation R between three classes A, B and L (as shown in figure 2), where L is an RDF collection of classes:

If A <R> L then,

$$\forall li \in L\,, A = \bigcup (A \cap li)$$

and

$$\forall li \in L, A \cap li \neq \{\};$$

(A is the union of the intersection of A with each member, li, of L, and no intersection is null). Or alternatively, in terms of class members,

$$\forall l \in L, \exists i (i \in (A \cap l))$$

and

$$\neg \exists a, a \in A, \forall li \in L, a \notin l$$

Informally, this means that if A <R> L, where L is a list of classes, then A is made up of some values from every class which is a member of L and no other values. For example suppose L is made up of Login, Name, Bdate and Gender. Then suppose we state that (User <R> L), then User is made up of Login data, Name data, Bdate data and Gender data. Note that the Venn diagram does not show all the classes in User and therefore User has some values not in Login, Name, Bdate or Gender. Note however, that these classes are *not* subclasses of User data as they also share members with other classes which are disjoint from User.

We found that the relation <R> can in fact be expressed in OWL-Lite using the following syntax:

```
<owl:Class rdf:ID="A">
<owl:equivalentClass rdf:parseType="Collection">
<owl:Restriction>
<owl:onProperty rdf:resource="&rdf;type" />
<owl:someValuesFrom rdf:resource="#B" />
</owl:Restriction>
<owl:Restriction>
<owl:onProperty rdf:resource="&rdf;type" />
<owl:someValuesFrom rdf:resource="#C" />
</owl:Restriction>
</owl:equivalentClass>
</owl:Class>
```

This uses a restriction on the property "type" to say that some the class A is made up of some values of type B and some values of type B. The equivalent class predicate ensures that there are no other values included. Using this syntax, in combination with rules defining typical inferences to be made over the class graph for various policy predicates, we found that all the necessary deductions can be made. In order to increase reasoning efficiency, we decided to abbreviate the above syntax to the equivalent syntax:

```
<owl:Class rdf:ID="A">
  <customNS:SVFO rdf:parseType="Collection">
        <B/>
        <C/>
  </customNS:SVFO>
</owl:Class>
```

(SVFO stands for "Some Values from Only", which is an abbreviation of the relations expressed in OWL-Lite above applying to a list of objects)

These two syntaxes are equivalent and the second is only a performance enhancement. We do not therefore specify the use of one syntax preferably to the other if performance considerations are addressed in some other way (e.g. introducing custom procedural code into reasoning engines).

Furthermore, as in P3P1.1, we have also removed the "data structure" names such as "postal" which are never referred to and therefore complicate the schema unnecessarily. Structural information can be included in labels if required for readability.

The whole schema hierarchy is then modelled using relations such as:

```
<owl:Class rdf:ID="User">
  <customNS:SVFO rdf:parseType="Collection">
        <Personname/>
        <Cert/>
        ……
  </customNS:SVFO>
</owl:Class>
<owl:Class rdf:ID="Personname">
  <customNS:SVFO rdf:parseType="Collection">
        <Given/>
        <Prefix/>
        ……
  </customNS:SVFO>
</owl:Class>
```

Note that the categories of the base data schema can also be modelled using this syntax, since they are just another class to which some of the other data types stand in relation SVFO. The syntax for integrating categories is more succinct and readable than other syntaxes because it is only necessary to list the categories and their allowed SVFO relations. The categories then stand as an orthogonal system to the main hierarchy of types.

For example,

```
<owl:Class rdf:about="#Political-data-category">
   <customNS:SVFO rdf:resource="#Cookies"/>
   <customNS:SVFO rdf:resource="#Miscdata"/>
   <rdfs:subClassOf rdf:resource="#Categories"/>
</owl:Class>
```

## 6. Concrete and abstract types

Many applications need to know whether a data type can be instantiated or not. For example if an application requests "User data", this cannot be instantiated and the application must first derive the concrete types inferred from the request. For this reason, all concrete classes are designated as type Instantiatable. If a type is not designated as instantiatable, then it is assumed to be abstract.

## 7. Shortcut classes

In order to abbreviate the syntax of typing instance data, we provide a set of shortcut classes for all possible instantiatable classes. For example for data of type User, Name and Given, the RDF syntax for typing an instance would be very verbose, so we define the class

```
<owl:Class rdf:ID="User.Name.Given">
 <rdf:type rdf:resource="#Instantiateable"/>
 <owl:intersectionOf rdf:parseType="Collection">
 <owl:Class rdf:about="#User"/>
 <owl:Class rdf:about="#Name"/>
 <owl:Class rdf:about="#Given"/>
 </owl:intersectionOf>
</owl:Class>
```

These classes do not add anything to the semantics of the ontology, but make it quicker and easier to type instance data and to reason over the type ontology.

## 8. Referencing the schema from privacy policies

There are 3 main use cases for referring to types from the schema expressed using this syntax

1. Requesting a type – in a privacy negotiation between an access control system and requester, the access control system may require information or credentials. It therefore needs to send hints as to the credentials required. For example a web service may require a certain certificate in order to allow access to a client. In this case, the service must be able to provide hints to the client as to what is needed to get authorization to use the service.

This can be expressed using the following syntax

1. Requesting typed data (Entity requests the data specifying the user's name).

```
<Entity>
<requests-data-types>
<rdfs:Class>
       <rdfs:subClassOf rdf:resource="User"/>
       <rdfs:subClassOf rdf:resource="Name"/>
</rdfs:Class>
</requests-data-types>
</Entity>
```

```
(Or shorcut class syntax can also be used – see
Sec 7.)
```

2. Typing an instance (Entity Submits data of type User's Given Name). This is expressed using the following syntax:

```
<Entity>
<hasData>
  <User.Name.Pseudonym>
       <rdf:value>Pseud1</rdf:value>
  </User.Name.Pseudonym>
</hasData>
</Entity>
```

3. Describing a practice carried out on a data type (Entity collects any values which are of type User and Name i.e. the class which is the intersection of both these classes)

```
<Entity>
<collectsAny rdf:parseType="Collection">
<rdfs:Class>
       <rdfs:subClassOf rdf:resource="User"/>
       <rdfs:subClassOf rdf:resource="Name"/>
</rdfs:Class>
</ collectsAny>
</ Entity>
```

The following points are worth noting in relation to this syntax:

- Each of these descriptions uses a different semantic to describe the operation on the data, but the data types are always referred to in terms of classes from the ontology. That is using rdf:type or rdfs:subClassOf.

- In order to express a specific type, it is often necessary to use multiple type declarations. For instance a name may be a User name or a Business name so in the data request description, it is declared as being both of type User and Name to make clear this specialization.

- As discussed in section 7, the predicates "collectsAny" and "requests-data-type" are in fact modal predicates and effectively convert DataClassX into a prototypical class representing all possible classes satisfying the subClass properties. This somewhat contradicts the formal semantics of OWL, however it will be shown that the correct deductions can still be derived using prolog style rules to extend the OWL semantics.

## 9. Inferencing over the schema

There are many possibilities for customized reasoning over such a schema, as discussed in section 5.1. We discuss below how the reasoning use case 5.1.1 (and implicitly also 5.1.2) may be implemented. These cases are key to each of the policy use cases described in section 2. This solution has been implemented and tested using the Jena API [15].

### 9.1 Deriving types of data collected or requested from broad types.

The relation "SVFO" (someValuesFromOnly) defined above specifies a directed graph which has a tree structure. Figure 3 represents a typical statement about collection practices as described in 5.1.1 (expanding the possible types of Name data).

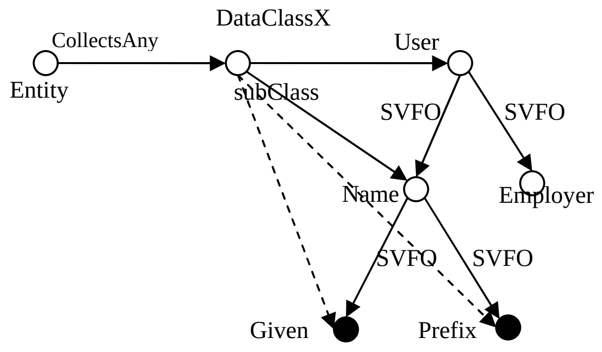

Figure 3: Data Handling Directed Graph

The policy states that the service may collect any data which is in both User and Name classes (i.e. specializing Name as a User, not a Business, name) – we give the elements of this class the temporary name DataClassX. The reasoner then is required to deduce that "Service may collect DataClassX" implies that the service may also collect the data classes Given and Prefix (concrete types are filled in black, inferences dotted lines).

Generally speaking, if X mayCollect DataClassX where DataClassX is some subClass of A (This is a typical statement from a P3P policy), then if A <SVFO> B and B <SVFO> C then it is also true that:

X mayCollect DataClassX where DataClassX subClass of C

The property mayCollect is a modal predicate and effectively converts DataClassX into a prototypical class representing all possible classes satisfying the subClass properties. However the correct deductions can still be derived using prolog style rules.

There are 3 important problems that a reasoner must address in this context:

1. The reasoner must return that DataClassX (the class of data which may be collected) is a subClass only of the lowest node in the SVFO hierarchy which DataClassX is already a subclass of. In the example then, the reasoner must not deduce that the service may collect Employer data. Therefore it cannot simply return that DataClassX is a subclass of – (all classes to which User and Name stand in relation SVFO to). I.e. the reasoner must only expand the graph below Name but not below User.

Using a typical OWL reasoner, this is impossible because it involves a form of negation (or "Unsaid within the context"). In order to determine to expand the SVFO child nodes of Name and not User in the above example, it must determine that Name does not have any SVFO children of which DataX is already a subclass. This can only be done using a forAll(not…) type rule functor.

2. The conclusion (DataClassX subClass Given) invalidates any rule premises searching for Unsaid((Name SVFO ?x), (DataX subClass ?x)). This means that the conclusions invalidate the premises, which is nonmonotonic reasoning. This is solved using marker triples, which tell the reasoner to ignore conclusion triples. This is a workaround which forces the reasoning to be monotonic.

3. Finally, transitivity does NOT hold for SVFO. It is not always true that if X has some values from Y and Y has some values from Z, then X has some values from Z. For example if (User has some values from Only Name and Employer) and (Name has

some values from only Given and Prefix), we cannot deduce that (User has some values from Prefix), because the some values that User has from Name may not be any of those that Name has from Prefix.

What we need from such a property however is the following:

If and only if a service may collect any data of classes Given and X, and Given is in the relation SVFO to class X, where X is relation SVFO to class Y then the reasoner should also return that the service might collect any data in class Y. (a kind of conditional transitivity for SVFO).

All these requirements can be met within the limits of acceptable performance using the proposed OWL ontology in combination with prolog style rules. We used the Jena inference libraries to derive these inferences on a sample policy. The following two Jena rules correctly expand the types based as described above (we have abbreviated the name space declaration for brevity). The question mark syntax indicates universal quantification and all triples are ANDed in the premises and conclusions:

Rule 1. The following complex rule ensures that the reasoner deduces that A is a subclass of SVFO child nodes of any class, X such that N requests-data-types A and A subClassOf X where there is no class Y such that X <SVFO> Y (problems 1 and 2.)

```
[(?N ns:mayCollect ?A),
(?A rdfs:subClassOf ?X),
unSaidSpecial(?A,ns:someValuesFrom,rdf:type,?X)
->
[r3:(?A rdfs:subClassOf ?E)
(?X rdf:type ns:marker)
<- (?X ns:someValuesFrom ?E)]]
```

Rule 2. The following rule ensures that all SVFO children of a class are returned as being of the same as the policy node, as long as they have been previously marked using the second rule (problem3).

```
[(?A rdfs:subClassOf ?D) <-
(?A rdfs:subClassOf ?B)
(?B rdf:type ns:marker)
(?B ns:someValuesFrom ?C)
(?C ns:someValuesFrom ?D)
]
```

The Rule builtin, unSaidSpecial provides the required negation functor described above (problem 1) and is defined as follows:

```
unSaidSpecial(A,P,Q,X)
```

True iff for all(Y), (X,P,Y) there is no triple st (A,Q,Y)

Note that using the shortcut classes (see sec 7.), this reasoning step can be performed much more simply for the case of finding instantiatable types, however for the case of matching preferences without the benefit of shortcut classes, this reasoning is still necessary.

## 10. Validation using the OWL format

As well as reasoning functionality, most applications require some validation functionality. This is of two kinds:

### 10.1 Synactic validation

This is available for concrete types such as "email". For example the schema can specify that the concrete type email must contain an @ sign – this can then be used to validate form entries for example. This is achieved simply by specifying the rdfs:range of instantiatable types described by the schema as being over an xsd datatype.

```
e.g.
<rdfs:range rdf:resource="&xsd;dateTime"/>
```

This example shows a builtin data type. OWL does not specify a mechanism for referencing user-defined xsd data types, but it does not prohibit their usage. The OWL specification has this to say about the question of user defined XML schema datatypes:

"*Because there is no standard way to go from a URI reference to an XML Schema datatype in an XML Schema, there is no standard way to use user-defined XML Schema datatypes in OWL.* "

If we specify a mechanism for referring to custom data types in a resource, we are therefore able to define a namespace containing syntactic validation constraints on the concrete types for the OWL data schema.

For example the following could be used to validate an email address:

```
<rdfs:range rdf:resource="&PII-DS-
XML;emailAddress"/>
```

Can be specified to refer to the simpletype in the schema as follows:

```
<simpleType name='emailAddress'>
     <restriction base='duration'>
      <pattern value='\w*@\w*\.\w*((\.\w*)*)?'/>
     </restriction>
</simpleType>
```

### 10.2 Semantic validation

A data type assignation breaks semantic validation rules if it refers to a type of data which cannot exist. OWL is not a language which is well adapted to making negation based statements of this kind. However, we have added disjointness relations for classes which should not be assigned simultaneously to data types (i.e. they have no common values). For example if a policy describes a class which is a subclass of both User and Business this should be flagged as invalid. More sophisticated semantic validation constraints may be added later. for example, a user's login can have only one value. This may also involve the use of custom rules within the reasoner module.

## 11. Changes to the P3P data schema vocabulary

Based on input from other researchers, we have also altered the available classes of the P3P data schema. For example, the following alterations have been made.

1. Name is a single class rather than dividing it into user name and business name. It is then specialized using business and name classes.
2. We have added classes corresponding to fields in electronic credentials,for example electronic identity card, drivers' licence and passport fields.

3. We have taken into account recommendations on identity document fields given in the recent ICPP study on identity management systems [16].
4. The techniques used to model credential metadata have also added other classes and predicates, which are out of the scope of this paper. For example we have added classes for describing proof methods for assertions made by credentials which fit into the typing schema.

## 12. Conclusion

OWL can be used to satisfy the requirements on data schemas for privacy and identity management policies within and beyond the use case scenarios of P3P. Some modification of the rulebase for reasoning over OWL is needed to deal with the modal "may collect values from" and "requests values from" predicates required by these scenarios, but this is possible using standard semantic web libraries. OWL data schemas can also provide required type validation functionality.

## 13. REFERENCES

[1] Platform for Privacy Preferences Specification, Cranor et al. ,Platform for Privacy Preferences, W3C Recommendation, http://www.w3.org/tr/p3p

[2] *P3P Base data schema*, part of W3C Recommendation on P3P, http://www.w3.org/TR/P3P/base

[3] Cranor et al. *P3P Base Data Schema specification*, http://www.w3.org/TR/P3P/#Data_Schemas

[4] *Web Ontology Language*, W3C recommendation, see http://www.w3.org/TR/owl-semantics/

[5] Giles Hogben, *P3P Using the Semantic Web (OWL Ontology, RDF Policy and RDQL Rules)*, W3C Working Group Note 3 September 2000, http://www.w3.org/P3P/2004/040920_p3p-sw.html

[6] Powers, C., Schunter, M., Enterprise Privacy Authorization Language (EPAL 1.2), W3C Member Submission 10 November 2003, http://www.w3.org/Submission/EPAL/ (for references to P3P data schema, see http://www.w3.org/2003/p3p-ws/pp/ibm2.html)

[7] Privacy and Identity Management in Europe, European Research Project, see http://www.prime-project.eu.org

[8] Claus,S. and Doring,S. *P3P Based Negotiation of Personal Data*, Technische Universitat Dresden, Fakultat Informatik, D-01062 Dresden, Germany

[9] Electronic Commerce Modeling Language (ECML), http://www.faqs.org/ftp/rfc/pdf/rfc3505.txt.pdf

[10] Eds Dubinko et al, *XForms 1.0*, W3C Recommendation 14 October 2003, http://www.w3.org/TR/xforms/

[11] E. Damiani, S. De Capitani di Vimercati, C. Fugazza, and P. Samarati: *Semantics-aware Privacy and Access Control:Motivation and Preliminary Results*, Proceedings of 1st Italian Semantic Web Workshop, 10th December 2004

[12] Giles Hogben , A technical analysis of problems with P3P v1.0 and possible solutions, Position paper for "Future of P3P" workshop, Dulles, Virginia, USA, 12-13 November 2002, http://www.w3.org/2002/p3p-ws/pp/jrc.html

[13] Cranor, Dobbs, Egelman, Hogben et al., *The Platform for Privacy Preferences 1.1 (P3P1.1) Specification W3C Working Draft* 4 January 2005 http://www.w3.org/TR/2005/WD-P3P11-20050104/

[14] McBride, B., Wenning, R., Cranor, L., *An An RDF Schema for P3P*, W3C Note 25 January 2002, http://www.w3.org/TR/p3p-rdfschema

[15] Jena open source semantic web libraries, http://jena.sourceforge.net/

[16] Independent Centre for Privacy Protection (ICPP) / Unabhängiges Landeszentrum für Datenschutz (ULD), Schleswig-Holstein and Studio Notarile Genghini (SNG), *Identity Management Systems (IMS)*: *Identification and Comparison Study*.