

Semester Project:

**Assisting users to provide information
required for a Web Service**

Author: Muriel Gummy
Assistants: David Portabella
 Ion Constantinescu
Supervisor: Martin Rajman

June 27, 2005

Table of contents

Introduction	3
1. General goal description	4
2. Introduction to Xforms	5
3. Introduction to Web Services	8
4. System architecture	9
5. Semantic Framework	12
6. Several approaches	13
6.1 XML and XML Schemas.....	13
6.2 RDF-S and OWL.....	17
6.3 XSL Transformation.....	22
7. Results	26
8. Conclusion	27
Bibliography	28

Introduction

In this document, we will discuss the problem of gathering input data needed by a Web Service to be executed and consider some possible solutions to solve it.

In the first section, we will present in details the need for a solution to this issue and the challenges that it raises. Then, we will make a brief introduction about two technologies related to this problematic, Xforms and Web services. This will be followed by a presentation about the bases of the Semantic Web as designed by the World Wide Web Consortium. In the sixth section, we will develop several approaches to solve our problem and discuss their advantages and disadvantages. Finally, we will present the results of this work, concluded by a short assessment.

1. General goal description

As Web Services play an increasing role nowadays, making their use easier has become a key issue. Some steps have already been performed to that purpose, like Semantic Web Services. They are based on a mechanism of semantic description that enable better automatic discovery, invocation and composition of services. These measures tend to facilitate the process of Web Services by software agents. The semantic description contains among others a profile indicating the required input data to call the service and the output data that will be produced.

However, the gathering of input data is a preliminary and independent step that may sometimes be an issue in itself. Indeed, a service can request very accurate information that may be difficult to obtain. Let us imagine the following simple situation: a flight booking service needs to be provided with the three-letters codes of departure and arrival airports in order to be executed. However, airport codes do not necessarily belong to the background knowledge of human beings, whereas airports names more often do. Or, in case of information retrieved from a data base, airport codes may not be present, whereas airports names are. In other words, the available information may not be directly the one that is required by a service, making its invocation impossible. The information about the airport name is not useless though. Actually, the desired information can be derived from it: practically speaking, this is done by calling some specific Web Service able to convert an airport name into an airport code. Now, the problem has been solved with the assumption that at least the *airport name* was known. But, what if it is not the case? What if only the departure/arrival *city name* is known? Then, we must consider another approach to collect the wanted information.

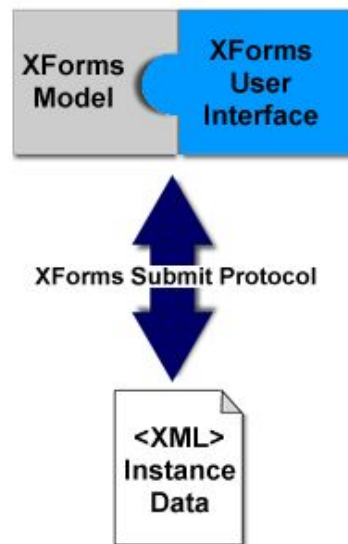
This small example perfectly illustrates the fact that there is seldom one single possible way of retrieving the needed data. It normally depends on the initial available information, if present. Besides, if the data come from a human being and not from a data base, the user may have preferences or constraints regarding the information he is willing or able to give. Thereby, the main issue is how to retrieve the required data from scratch or from partial and indirect information.

From now on, it becomes necessary to provide some assistance to accomplish this task. This project aims at the development of a tool, called personal assistant, to help a user calling a Web Service more easily. The key idea is to build a system in charge to compute a set of sequences of questions leading to the desired information. Then, the user would have the possibility to select one alternative, according to what data he is ready or able to give. Asking questions will be done by means of forms that the system will propose to the user. More especially, we will focus on the recent XForms technology recommended by the W3C. This technology is presented in the next section.

2. Introduction to XForms

Xforms are a recommendation of the World Wide Web Consortium (W3C) and are intended to replace HTML forms, currently part of the HTML standard. HTML forms suffer from several weaknesses, especially through their strong need for scripting to handle events and their lack of flexibility.

Xforms bring two major novelties: first, the data collected from users are XML-structured. Second, Xforms separate the logic part from the presentation part: they are made of two components, called Xforms model and XForms user interface, as shown in Fig 2.1. This two features offer important enhancements in comparison with HTML forms. For example, XForms allow multiple device support, since the interface can be customized for all kind of devices and not only web browsers. Furthermore, XML is a standard data exchange format and hence the device independence and the internationalization of Xforms is ensured.



2.1 Xforms components

Xforms model encapsulates the description of the instance data. To be precise, it contains the XML structure that will format the output data. This structure must be embedded within the following tags:

```
<model>
  <instance> Structure to be defined </instance>
</model>
```

For example, this is a possible structure for describing a Person entity:

```

<model>
  <instance>
    <person>
      <name/>
      <address/>
    </person>
  </instance>
</model>

```

Tags containing no subelement are intended to receive the submitted data. Fig. represents a possible Xform corresponding to the previous model (Fig 2.2).

2.2 Xforms interface

Once the user submits his entries, an Xform processor will produce the XML instance data corresponding to the model, with the user entries nested in the previously empty elements `<name>` and `<address>`, as follows:

```

<person>
  <name>Jean Dupont</name>
  <address>Avenue d'Ouchy 7, 1006 Lausanne</address>
</person>

```

There is a bunch of standard Xforms controls designed to be used in XHTML files. The main controls are: `<input>`, `<textarea>`, `<range>`, `<select>`, `<select1>`, etc. Each control is coupled with a label element defining the legend of the control and an attribute `ref` binding it to the element of the model in which the user entry will be nested.

Here is the code of the previous Xform user interface:

```

<input ref="/person/name">
  <label>Enter your name:</label>
</input>
<br/>
<input ref="/person/address">
  <label>Enter your address:</label>
</input>

```

```
<submit submission="some_id">  
  <label>Submit</label>  
</submit>
```

For a detailed specification of Xforms, please refer to [1].

3. Introduction to Web Services

Web Services are resources that provide an abstract set of functionality allowing machine-to-machine interaction over a network. In other words, Web Services offer an standard interface of functionality, specified by a Web Service Description (WSD), processable by software agents.

The Web Service Description is written in an XML-base language, WSDL standing for Web Service Description Language. The service description describe a sort of protocol to invoke the service. The protocol defines how the requester and the provider will interact and what are the expected behavior of the service in response to requests. Practically speaking, requesters and providers exchange messages to respectively invoke a service and respond to a request. As requesters and providers are software agents, it is necessary that both sides agree on the meaning of the messages. Thus, in addition to describe the mechanism of the interaction between requesters and providers, WSD define also a semantic regarding the purpose and the effects of this interaction.

However, it appears that WSD are somehow limited and unable to fulfil certain more complex tasks about Web Services such as:

- ◆ Describe what the service has to offer to allow automatic **discovery**
- ◆ Describe what a service needs to be executed to allow automatic **invocation**
- ◆ Describe how services can interoperate to allow automatic service **composition**

That is what Semantic Web Services have been designed for. Semantic Web Services are expressed by means of OWL-S, standing for Ontology Web Language for Service. Simple OWL is a language to define ontologies. An ontology is a formal description of concepts and the relationships between them. Concepts are formally represented by classes and the relationships by properties. The relationships can be hierarchical within a domain of knowledge (e.g a Dog is a sub class of Animal) or cross-domains, binding two different concepts (e.g a Person is the owner of a House).

Then, OWL-S is a particular case of ontology describing a service resource. Consequently, OWL-S has a very stronger semantic than WSD. The service ontology should be able to express answers to the following questions:

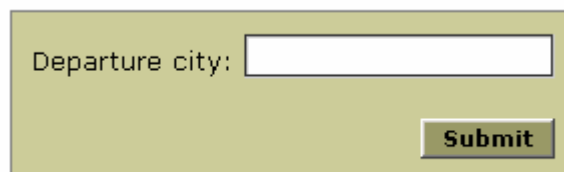
- ◆ *What does the service require of the user(s), or other agents, and provide for them?*
- ◆ *How does it work?*
- ◆ *How is it used?*

For the purpose of this document, only the first question is of interest. It is handled by the ServiceProfile ontology, which has two main properties: hasInput and hasOutput. These properties allows to describe what inputs are needed by the service and what outputs will be produced. Generally, inputs and outputs are specified by ontology classes too.

4. System architecture

As presented in the first section, the aim is to design a system able of computing several sequences of questions asked to a user in order to obtain some specific information. To implement the question asking task, we have chosen to use XForms. The system will propose XForms to the user one after the other and he will have to fill them. However, the user is not supposed to supply each bit of information on his own: the system is expected to use Web Services in order to acquire additional information that the user does not know or does not directly possesses.

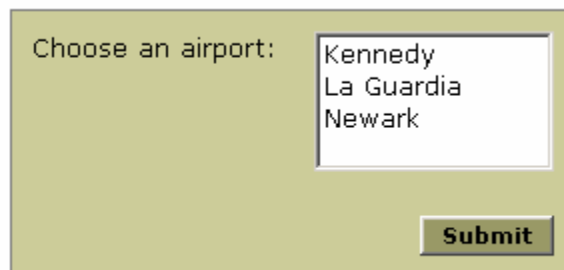
Let us consider the flight booking service example again. Let us also imagine that our system, whose target information is an airport name, interacts with a user who only knows in which city his plane should take off. The ideal scenario is the following: a first XForm will ask him to enter his wanted departure city (Fig 4.1). Once the information is submitted, the system is expected to propose as next form a list of airports located in the city entered by the user. In order to do that, the system has to call a Web Service able to produce a list of airports according to the city retrieved from the previous form. The output of the service, the airports list, become the input data of the next XForm, displayed as a menu (Fig 4.2). Then, the user will select an item from the menu and the system will eventually end up with the targeted information. Intermediary Web Services are therefore a necessary layer in our system architecture: they allow producing information that was not in possession of the user at the start of the process, and they also allow creating XForms with dynamic content, that is, Xforms adapting their displayed content to previous entries of the user. By the way, we can notice also that, in this approach, Xforms behave like Web Services: they require input data, possibly null, and, after their execution – in this case, data submission from the user – they return a value as output.



Departure city:

Submit

4.1 First XForm



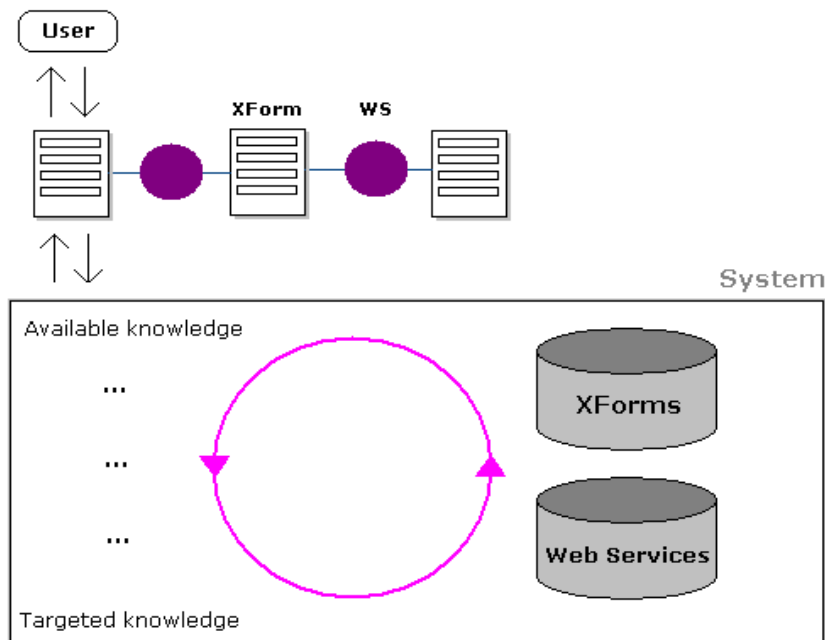
Choose an airport:

Kennedy
La Guardia
Newark

Submit

4.2 Second XForm

If the content of XForms can be dynamically built, on the other hand the system will not create the whole form from scratch: it will need a set of pre-existing *questioning frames*, that is, generic XForms. Actually, one can imagine a system operating without them, but it will make the whole process a lot more complex. The point here is to have a repository containing these generic XForms, as well as a repository of Web Services links. Then, the task of the system is to compute a consistent sequence of XForms/Web Services allowing the system to end up with desired input data of the target Web Service.



4.3 Systems architecture

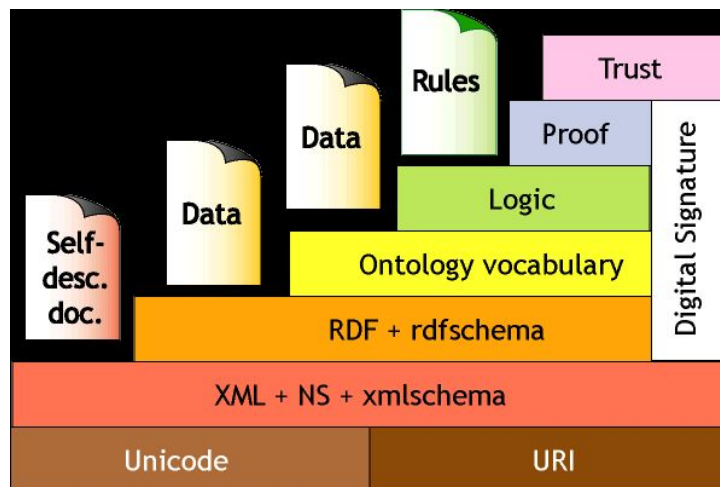
How can this be achieved? The system has to perform a planning task, more especially a state-base planning. Thus, actions correspond to XForms/Web Services, preconditions to inputs, and effect to outputs. Each Web Service already specifies what are its inputs and outputs. But, in the opposite, regular XForms have no such input or output fields defined. That is, the system has no way to know in advance what kind of data does an XForm require as input and what kind of data will outcome. As an illustration, let us consider the above example again: the first XForm requires no input and produces a city name, whereas the second requires a list of airports names and produces a single airport name. In other words, the system needs to know the type of the input/output data. This information should thus be somehow made available for the system. In the semantic framework of the W3C, data types are commonly represented by ontology classes, as presented in section 2.

	XForm 1	XForm2
Input(s)	-	Set of airports
Output(s)	City name	Airport name

As a matter of fact, we have to find a mechanism to handle these new features of XForms. Furthermore, it is not sufficient to know a list of inputs and outputs data types: each one of them should be associated to its corresponding spot in the actual XForm. The systems must be informed where to collect which data in the XForm. Thus, our mechanism should also provide a way to identify what part of the forms will contain the input/output data. Several approaches have been considered to implement this task; they are presented in details in the next section.

5. Semantic Web Framework

The Semantic Web framework of the W3C is organized in various layers. The architecture can be regarded as a stack of technologies (Fig 5.1) and is also known as the Semantic Web pyramid. Regarding the markup languages, XML and XML Schemas lay at the base of the pyramid. The immediate layer built on top of XML consists of RDF and RDF Schema (RDF-S), which is itself inferior to the Ontology layer (OWL).



5.1 Semantic Web architecture: Stack of technologies

These technologies are summarized as follows by the W3C:

XML provides a surface syntax for structured documents, but imposes no semantic constraints on the meaning of these documents.

XML Schema is a language for restricting the structure of XML documents and also extends XML with datatypes.

RDF is a data model for objects ("resources") and relations between them, provides a simple semantics for this data model, and these datamodels can be represented in an XML syntax.

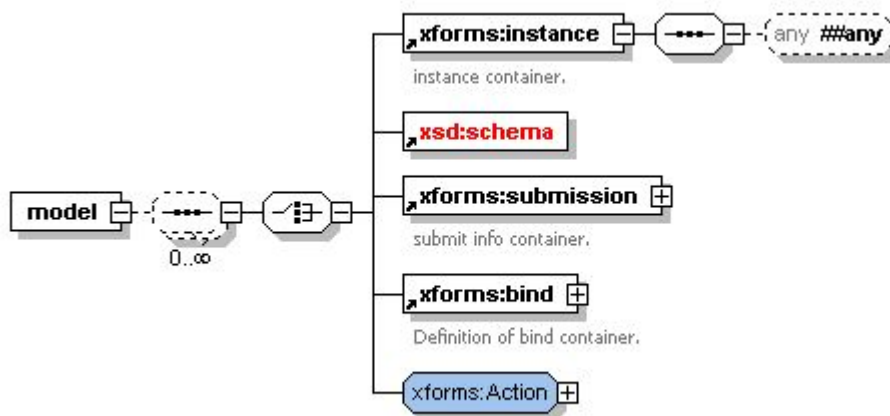
RDF Schema is a vocabulary for describing properties and classes of RDF resources, with a semantics for generalization-hierarchies of such properties and classes.

OWL adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes.

6. Several approaches

6.1 XML and XML Schemas

The first basic idea was to focus on the bottom layer of the pyramid by modifying the raw XML structure of XForms: practically speaking, this means extending or changing the XML Schema of XForms in order to integrate new allowed elements. The added tags or attributes will be used to identify which parts of the XForm involve input or output and specify what OWL classes they correspond to. Fig 6.1.1 is an excerpt of the XML Schema of XForms as defined by the W3C:



6.1.1 Graphical view of XForms XML Schema

The *model* element is followed by a sequence, possibly repeated, of a choice of one or more of the following elements: *instance*, *schema*, *submission*, *bind* or *Action*. The *instance* element can be followed by any tag: it enables to extend the content of the instance with elements not specified in the schema. Hence, this part will be freely defined by the creator of the XForm.

Naively, we can think of allowing an *input/output* tag within the *instance* element, completed by a *class* attribute. Or we could also identify the input/output directly with an attribute such as *type* (with value *input* or *output*) followed by a *class* attribute. Examples of such XForms models:

```
<xf:model>
  <xf:instance>
    <person>
      <output class="http://examples.org/Person#firstName">
        <fName/>
      </output>
      <output class="http://examples.org/Person#lastName">
```

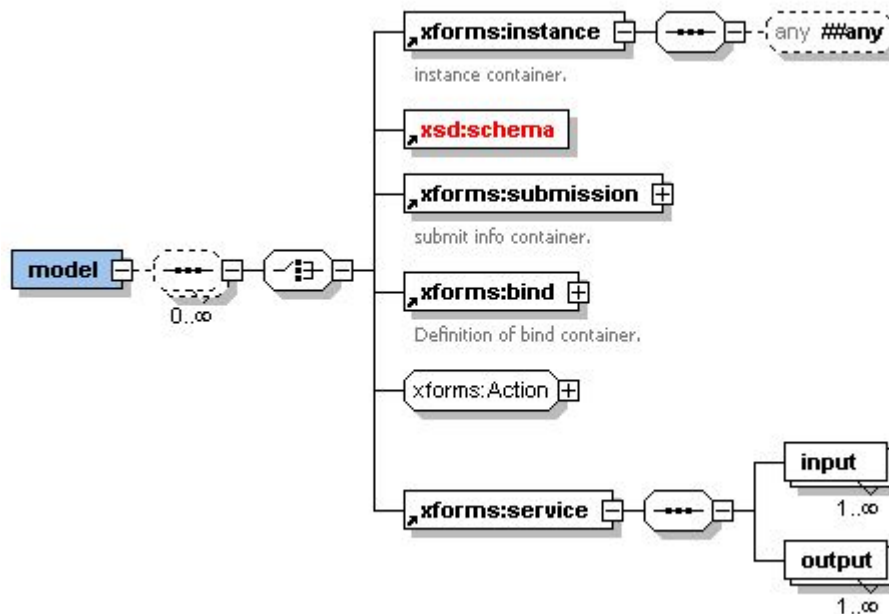
```
        <lName/>
      </output>
    </person>
  </xf:instance>
</xf:model>
```

```
<model>
  <instance>
    <person>
      <fName type="output" class="http://examples.org/Person#firstName">
        <lName type="output" class="http://examples.org/Person#lastName"/>
      </person>
    </instance>
  </model>
```

6.1.2 XML view of XForms XML schema, model part

This approach has the advantage to mark out the data in place and hence their value and their associated class will be easily extracted from the XML instance data file, once submitted. However, this cannot formally be done, because XML schemas do not allow defining elements within an *any* element nor is it possible to add attributes for elements within the *any* tag. In fact, the content of an *any* element can not be defined at all, even partially.

Another solution would be to add a *service* element as one of the possible choices in addition to *instance*, *schema*, *submission*, *bind* or *Action*. The service element would contain two sub-elements: *input* and *output*. That means that for each instance, we could associate a service and indicate what are its inputs and outputs. Input and output tag would contain the class information.



6.1.3 Modified XForms XML schema

Example:

```

<model>
  <instance>
    <person>
      <name>
        <firstName/>
      </name>
    </person>
  </instance>
  <service>
    <input>http://examples.org/person.owl#lastName</input>
    <output>http://examples.org/person.owl#firstName </output>
  </service>
</model>

```

Unlike the two first solutions, XML schemas do not prevent from realizing this solution. But nevertheless, it is a lame solution. The model intends to give a structure to the instance data, so in a sense it would be a misuse to include service information into the model part.

Actually, it appears that all approaches tending to modify the raw XML structure of XForms are awkward and unsatisfactory, due to the structural nature of XForms. Indeed, we have seen earlier in this document that the big novelty of XForms was to separate the model from its representation or user interface. But actually, XForms outputs are related to the model since the latter is supposed to define the XML structure in which instance data will be embeded, while Xform inputs - such as a list of airports - have nothing to do with the model and are therefore intended to be displayed to the user in a form control. Hence, information

about input and output classes should not be present in the same components of an XForm.

In some cases, the model component and the interface component may even not be in the same file. Indeed, an instance can be stored in an independant XML file and only linked at by a model, typically located in XHTML pages as well as the form controls where information about Xform input should be embedded. Our planning system should then parse these various files to retrieve the desired information. Obvioulsy, it is not very convenient. The best would be to have all information in a single file.

Another major drawback is that it is not possible with XML schemas to formally require an OWL class as a data type, because data types for XML schemas are limited. At best, we can make compulsory to have some restricted string type, but it is obviously another lack in the approach of XML schemas.

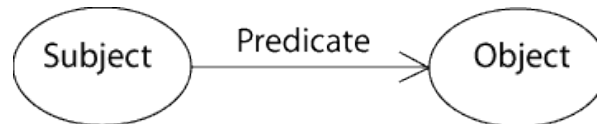
For all these reasons, it is not a proper solution to include information about input and output as an additional part of the XML structure of Xforms.

6.2 RDF-S and OWL

As seen in the previous section, XML schemas did not prove powerful enough to achieve our goal. In this new approach, we are going to start from an upper level of abstraction.

RDF stands for Resource Description Framework and provides an XML-based syntax to describe all sorts of resources, e.g web pages, author, etc. The RDF data model lays on the fundamental concept of triples. A triple consists of:

- Subject: the resource being describe
- Predicate: a property of the resource
- Object: the value of the property



6.X Triples in RDF

Let us have a look at a very simple example:

```
<rdf:Description about="http://mypage.ch/ThisReport.pdf">
  <author> Muriel Gummy </author>
</rdf:Description>
```

This triple asserts that the resource identified by the URI `http://mypage.ch/ThisReport.pdf` has a property *author* whose value is "Muriel Gummy". Such an assertion is called an RDF statement. Thus, the RDF data model provides a mechanism to describe resources and the relationships between them, but no means to declare the properties themselves or the type of the resources being described.

RDF Schemas (RDF-S) fulfil this function. RDF-S defines the following concepts, similarly to the object-oriented concepts:

Core classes:

- ◆ `rdfs:Resource`

This class represents all resources: everything described in RDF is instance of that class. Equivalent to the `Object` class in Java.

- ◆ `rdf:Property`

This class represents the subset of resources that are properties. The resource author would be of class `rdfs:Property`.

- ◆ `rdfs:Class`

This class represent the subset of resources that are classes.

Core properties:

- ◆ `rdf:type`

This property indicates that a resource is an instance of a class.

- ◆ `rdfs:subClassOf`

This property indicates that a class is subordinate to another class.

- ◆ `rdfs:subPropertyOf`

This property indicates that a property is subordinate to another property.

Core constraints:

- ◆ `rdfs:range`

This property indicates that the value of a property must be an instance of a given class.

- ◆ `rdfs:domain`

This property indicates the class to which a property can be applied.

These concepts allow to attach semantics to a resource description, that is, they provide information about the interpretation of the statements given in an RDF data model

Ontology Web Language (OWL) is built on top of RDF and RDF-S. Basically, the mechanism is the same but OWL extends vocabulary for describing properties and classes: new concepts are added like disjointness of classes or cardinality of properties. An ontology language can formally describe the meaning of some specific terminology.

From now on, the idea is to create an RDF Schema or an ontology to describe an Xform resource, and, more especially, their inputs and outputs. Indeed, semantic descriptions enable software agents to reason about them: each resource defined by the ontology/RDF-schema has an exact meaning that can be understood, interpreted and processed by computers. That is exactly what our system is intended to do. To plan a successful sequence of Xforms and Web Services, the systems must actually browse through the repository, and then identify matching inputs and outputs of the available resources.

Here is a simple RDF Schema for Xforms:

```

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://www.daml.org/2001/10/html/xform-rdfs">
  <rdfs:Class rdf:ID="XForm"/>
  <rdf:Property rdf:ID="parameter">
    <rdfs:domain rdf:resource="#XForm"/>
    <rdfs:range rdf:resource="owl#Class"/>
  </rdf:Property>
  <rdf:Property rdf:ID="input">
    <rdfs:subPropertyOf rdf:resource="#parameter"/>
  </rdf:Property>
  <rdf:Property rdf:ID="output">
    <rdfs:subPropertyOf rdf:resource="#parameter"/>
  </rdf:Property>
</rdf:RDF>

```

We can write an ontology in OWL equivalent to the RDF schema, but in this case it is not necessary to have OWL, because we do not make use of the additional specific features of OWL for such a simple resource description.

```

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://www.daml.org/2001/10/html/xform-ont">
  <owl:Class rdf:ID="XForm"/>
  <owl:ObjectProperty rdf:ID="parameter">
    <rdfs:domain rdf:resource="#XForm"/>
    <rdfs:range rdf:resource="owl#Class"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="input">
    <rdfs:subPropertyOf rdf:resource="#parameter"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="output">
    <rdfs:subPropertyOf rdf:resource="#parameter"/>
  </owl:ObjectProperty>
</rdf:RDF>

```

Basically, the schema/ontology contains one class Xform and three properties, parameter,

input and output. The parameter property is the super property of the input and output properties and is used to factorize the domain and the range that are common to both subproperties. Their domain is obviously Xform, and their range is OWL class.

Now, let us imagine an example of a simple airport ontology:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://www.daml.org/2001/10/html/airport-ont">

  <owl:Class rdf:ID="Airport"/>
  <owl:Class rdf:ID="Name"/>
  <owl:DataProperty rdf:ID="name">
    <rdfs:domain rdf:resource="#Airport"/>
    <rdfs:range rdf:resource="#AirportName"/>
  </owl:DataProperty>
  <owl:DataProperty rdf:ID="airportNameValue">
    <rdfs:domain rdf:resource="#AirportName"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DataProperty>
  <owl:Class rdf:ID="AirportNameBag"/>
    <rdfs:subClassOf
      rdf:resource="rdf#Bag"/>
  </owl:Class>
  <owl:ObjectProperty rdf:ID="AirportNameMember">
    <rdfs:subProperty
      rdf:resource="rdfs#member"/>
    <rdfs:domain rdf:resource="#AirportName"/>
    <rdfs:range rdfs:resource="#AirportNameBag"/>
  </owl:ObjectProperty>
</rdf:RDF>
```

We define a class *Airport* and a class *AirportName* with a property *name* binding an *Airport* entity to an *AirportName* entity. Then, we subclass the class *rdf:Bag* to create a class *AirportBag* that will represent a set of airports. Finally, we add a property *AirportMember* for this class as a subproperty of *rdfs:member*, specifying that it can be applied on an instance of *AirportBag* and whose value will be an *Airport*.

From now on, we can describe a specific Xform resource, corresponding to a previous example. The resource is an instance of the predefined class Xform. Through the properties input and output, we state that this Xform takes an *AirportList* instance as input and produce a result of class *Airport*.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:xf="http://www.daml.org/2001/10/html/xform-ont#"
  xmlns:airp="http://www.daml.org/2001/10/html/airport-ont#">

  <xf:Xform rdf:ID="http://examples.org/AirportXForm.xml">
    <input rdf:resource="airp#AirportList" />
    <output rdf:resource="airp#Airport" />
  </xf:form>

</rdf:RDF>
```

Thanks to the RDF schema/ontology, any software agent is now able to interpret this statement in order to make a correct and successful planning. Our goal is thus partially reached. Actually, there is still one lack in this approach: inputs and outputs are not related at all with the concrete Xform. Indeed, the RDF description is very useful for the planning, that is, the task that has to be done as a first step, before starting the interaction with the user. But, once a consistent and successful sequence of Xforms/Web Services is computed, the system has to put the plan into operation and hence deals with the actual input and output data. The current RDF description will not be helpful at all for this task. How will the system identify which part of the submitted instance data correspond to which output in the RDF description? How to find the matching pairs of instance data and RDF properties?

As things are at present, our RDF description is unable to solve this problem and consequently we will have to modify it. The goal is then to bind the RDF input and output resources to their corresponding XML tag in the Xform. The main idea would be to have triple whose subject would be an XML tag, property would be input or output and value would be an OWL class. However, each component of a triple must be a declared RDF resource (except for value of DataProperty), so this solution is impossible.

We also considered of the ID – IDREF basic XML mechanism, but this is not a working solution. First, this mechanism only works for elements in the same file, which is not the case in the present situation. Second, this mechanism does not simply refer to a tag or an element, it is a way of avoiding the redundancy of content and therefore, the result is the equivalent to a copy of the content of the pointed element. Obviously, this is not the behaviour that we expect.

Unfortunately, it seems that there is not way to make an RFD file reference an XML fragment.

6.3 XSL Transformation

In the previous section, we have seen that describing an Xform resource with RDF plus RDF-S or OWL raised some problems for the later processing because the system was not able to identify inputs and outputs in the XML instance data with the ones in the RDF description.

To remedy this problem, the idea was to take the reverse point of view: instead of trying to relate RDF resources to their matching in Xform, we could try to start from the Xforms data and then associate the desired information to them, similarly to our first approach at the XML layer, but with the difference that we would keep the semantics. Concretely, the approach would be the following: we would turn an instance data file into an RDF description of it. This can be done by means of an XSL transformation.

XSLT is an XML-based language able to transform an XML document into an other XML document. In our case, it would mean more especially transforming an XML document into an RDF document, which is only a particular case of XML document.

XSLT lays on the use of templates coupled with Xpath. Templates are sets of rules for the transformation. Xpath is a syntax for defining parts of an XML document. With Xpath, we can define expression to select or filter definite nodes (root, element, attribute, text, comment, processing-instructions, namespace) of an XML document. Then, XSLT allows to apply transformation templates on the nodes selected by an Xpath expression.

Here are some special symbols of Xpath:

```

/ : root of the document
. : current node
e : some tag <e>
* : any élément
f/e : a tag <e> having a child taf <f>
f//e : a tag <e> descendant of <f>
@a : value of attribute a
x|y : x or y
e[s] : any element e having a child s
e[n] : the n-th element of <e>
e[f='v'] : element e whose subelement f has a value v
e[@a='v'] : element e whose attribute a has value v
id('v') : element with the attribute ID="v"
text() : text element

```

XPath syntax defines also a certain amount of axes, that is, expression to select a set of nodes relative to the current node: ancestor, child, parent, descendant, following-siblings, self, etc.

The main commands of XSLT are:

- ◆ `<xsl:template match=XPath_expr>` : defines the transformation rules applicable to all the nodes filtered by the Xpath expression following the attribute *match*.
- ◆ `<xs:apply-templates select=XPath_expr>` : applies to the current node or its children filtered by the *select* attribute all matching templates.

Here is the needed XLS transformation to turn the instance data into an RDF description:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xf="http://www.examples.org/XForms/xform-rdf-schema#">

  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

  <xsl:template match="/">
    <xsl:call-template name="mainTemplate"/>
  </xsl:template>

  <xsl:template name="mainTemplate">
    <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns="http://www.examples.org/XForms/instances#"
      xml:base="http://www.examples.org/XForms/tests/"
      xmlns:xf="http://www.examples.org/XForms/xform-rdf-schema#"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
      <xsl:apply-templates/>
    </rdf:RDF>
  </xsl:template>

  <xsl:template match="*">
    <rdf:Description>
      <xsl:attribute name="rdf:about"><xsl:value-of select="name(current())"/></xsl:attribute>
      <xsl:for-each select="/*">
        <xf:defines>
          <xsl:attribute name="rdf:resource">
            <xsl:text>#</xsl:text>
            <xsl:value-of select="name(current())"/>
          </xsl:attribute>
        </xf:defines>
      </xsl:for-each>
      <xsl:if test="child::text()">
        <xsl:call-template name="leafHandling">
          <xsl:with-param name="outputXMLName">town</xsl:with-param>
          <xsl:with-param name="OWLClass">http://examples.org/onto.owl#C1</xsl:with-param>
        </xsl:call-template>
        <xsl:call-template name="leafHandling">
          <xsl:with-param name="outputXMLName">state</xsl:with-param>
          <xsl:with-param name="OWLClass">http://examples.org/onto.owl#C12</xsl:with-param>
        </xsl:call-template>
      </xsl:if>
    </rdf:Description>
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="text()">
```

```

</xsl:template>

<xsl:template name="leafHandling">
  <xsl:param name="outputXMLName"/>
  <xsl:param name="OWLClass"/>
  <xsl:if test="$outputXMLName = name(current())">
    <xf:hasValue>
      <xsl:attribute name="rdfs:Literal"><xsl:value-of select="."/></xsl:attribute>
    </xf:hasValue>
    <xf:isOutput>
      <xsl:attribute name="rdf:resource">
        <xsl:value-of select="OWLClass"/>
      </xsl:attribute>
    </xf:isOutput>
  </xsl:if>
</xsl:template>
</xsl:stylesheet>

```

A first template applied to the root of the document creates the opening and closing tags for RDF, and the call `<xsl:apply-templates>`. The second template is matched due its Xpath filters that accepts all element nodes. For each encountered element of the source file, this template will create an RDF resource and add it a property `<xf:defines>` whenever the current element has direct subelements. When the current elements has text node as child, then it means that the child node contains the submitted data. For such elements, we add two properties: `hasValue` which contains the textual value of the output, and `isOutput` wich contains the OWL class of the data.

Here is the XML instance data before transformation:

```

<location>
  <city>College Park</city>
  <state>MD</state>
</location>

```

Here is the RDF document resulting from the transformation:

```

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns="http://www.examples.org/XForms/instances#"
  xml:base="http://www.examples.org/XForms/tests/"
  xmlns:xf="http://www.examples.org/XForms/xform-rdf-schema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

  <rdf:Description rdf:about="location">
    <xf:defines rdf:resource="#city"/>
    <xf:defines rdf:resource="#state"/>
  </rdf:Description>

  <rdf:Description rdf:about="city">
    <xf:hasValue rdfs:Literal="College Park"/>
    <xf:isOutput rdf:resource="http://examples.org/onto.owl#C1"/>

```

```
</rdf:Description>

<rdf:Description rdf:about="state">
  <xf:hasValue rdfs:Literal="MD"/>
  <xf:isOutput rdf:resource="http://examples.org/onto.owl#C2"/>
</rdf:Description>

</rdf:RDF>
```

Now, we have a document that contains the output data associated with its corresponding OWL class. However, this solution, although it solves the problem that we had before, generates a new problem: the XSL transformation can only be applied once the Xform has been submitted. As a consequence, the planning task becomes now impossible. With this approach, we removed the difficulty of the previous solution, but we also lost its positive contribution that solved the planning task.

Thus, none of the various attempts we made so far has been successful.

7. Results

Unfortunately, we could not find any satisfying solution that would be simultaneously standard-compliant and effective.

As time was missing, we eventually decided for a simpler option: we built a small web application that, instead of computing itself the sequence of Xforms/Web Services, takes as input parameter the workflow of the operations. The workflow contains the results of an hypothetical planning. Therefore, it indicates the sequence of Xforms/WebServices to invoke, and also the relationship .

```
<FlowDescription name="ZipCodeFinder">
  <formInput>http://localhost:8080/xforms/PlaceXForm.xml</formInput>
  <webservice>
    http://www.mindswap.org/2004/owl-s/1.0/ZipCodeFinder.owl
  </webservice>
  <formOutput>http://localhost:8080/xforms/CodeForm.xml</formOutput>

  <mappingInput fromPath="/location/city" wsInputParameter="City"/>
  <mappingInput fromPath="/location/state" wsInputParameter="State"/>
  <mappingOutput toPath="code" wsOutputParameter="zipcode-ont:zip"/>
</FlowDescription>
```

The process is the following: a first servlet retrieve the desired workflow given as parameter. It then retrieve the first Xform model and returns an Java Server Page displaying the corresponding Xform. The user submit the data and the control is given to another servlet. The servlet extracts the output and sends the required data to the following Web Service. The output of the Web Service is given as input for the next Xform.

8. Conclusion

The results of this project have not totally reached the initial expectations. Indeed, no satisfying solution could be found to implement an effective planning system. Nevertheless, the present work has allowed to point at some design difficulties to develop application in the Semantic Web framework. The study of the various semantic technologies has revealed great features and enhancements, but on the other hand, some lacks have been discovered too.

A solution might though be found thanks to a new recommendation of the W3C: Xpointer. During this project, this technology has not been actually studied, but it appears that Xpointer define a way to point at XML fragments, that is, parts of XML files even external. Unfortunately, there are nearly no support to that technologies at the moment. Most famous API for RDF do not take Xpointer into account. In a near future, when these technologies will be mature, it would be interesting to direct future attempts into this direction.

Bibliographie

Web Sites

[1] W3C: www.w3.org

[2] W3C schools: www.w3schools.com

[3] Preparing for Semantic Web Services: www.sitepoint.com/article/semantic-web-services

[4] OWL-S: <http://daml.semanticweb.org/services/owl-s/1.0/owl-s.html>

[5] Mindswap: <http://www.mindswap.org/2004/owl-s/services.shtml>