SWAD-Europe: WP6.3b Pragmatic Methods for Mapping Semantics from XML Schemas (documentation for 6.1)

Project name:
Semantic Web Advanced Development for Europe (SWAD-Europe)
Project Number:
IST-2001-34732
Workpackage name:
SWAD-Europe: XML and Semantic Web Integration research prototypes
Workpackage description:
http://www.w3.org/2001/sw/Europe/plan/workpackages/live/esw-wp-6.html
Deliverable title:
Pragmatic Methods for Mapping Semantics from XML Schemas
URI:
Authors:
Brian Matthews, CCLRC.
Ian Johnson, CCLRC.
Abstract:

Previous work in WP 5 of SWAD has discussed the possibility of automatic extraction of semantics useful for the Semantic Web from legacy XML Documents and Schemas. The general conclusion of this workpackage was that there was no overall approach to this which was automatable although in particular cases and styles it is possible to generate mappings.

In this document we discuss instead a more pragmatic

approach where a mapping between OWL Ontologies and XML Schemas is developed by an expert user, leading to a schema map which can be used to further automate operations on instances. This work is the basis of an implementation which is currently underway.

Status:

First draft: 2004-02-01

Version 1: 2004-06-06

Comments on this document should be sent to the public SWAD-Europe mailing list, public-esw@w3.org,

---

Introduction

This paper is part of SWAD-E Workpackage 6.3.

Previous work in WP 5 of SWAD-Europe has discussed the possibility of automatic extraction of semantics useful for the Semantic Web from legacy XML Documents and Schemas.

In particular, it was discussed whether in particular cases and styles of presentataion it might be possible to generate mappings which would extract RDF triples from XML Document instances; deliverable 5.2 [SWAD5.2] presented *Schematron* and *XSLT*

based mapping system suitable for extracting RDF triples from documents which were presented in "Layered Normal Form" which implements a striped syntax [XNF]. That is, the outermost element represents an object, the first level child elements represent properties, the second level children again represent objects and so on. Leaf nodes represent properties and their values. In this Normal Form, XML attributes are not used

However, the overall conclusion of WP 5 was that in general there was no automatable approach to extracting triple data from XML Schemas. The number of possible forms of expression in XML Schema is too broad for any general rules to be established which will work for an arbitrary XML document.

In this document we discuss instead a more pragmatic approach where a mapping between OWL Ontologies (or potentially an RDF Schema) and XML Schemas is developed by an expert user, leading to a schema map which can be used to further automate operations on instances (that is between XML Documents and RDF triples). This work is the basis of an implementation which is currently underway.

Use Cases

The requirements for such a mapping depends on the use to which we are putting this mapping. Some use cases where we might use the mapping include the following cases.

1. **Designing Web Systems.**

   Designing an XML base representation for use in (say) a web service system between "mutually semantics aware" systems. That is, we are using an ontology as part of the design process of generating data formats (presumably including database schemas and XML schemas and others) to capture the information.

   Logically the ontology comes first, but you may have different XML formats for different purposes. For example,

an XML format used for moving information between databases or applications may represent more complete information than an XML format used to give a presentation of a report. So at design stage it makes sense to build the XML schemas logically from the data model, but you don't need the ontology at runtime.

In this case all the information in the resulting schemas would be represented in the ontologies, but not necessarily have all the information in the schemas that is in the ontology.

Further, as the Ontology comes first, the most logical approach is to derive the XML Schema systematically from the ontology. This is a different yet related approach to the one undertaken in the rest of this document.

2. **Data Exchange.**

   In this case, we are exchanging information between different systems which have not been built on the same data model. The different systems have been constructed independently and they are maintained separately, often by different organisations.

   Here, the recipient would receive data conforming to a legacy XML Schema and would want to convert this to semantically meaningful information, in say RDF triples. For example, you wish to import data from someone else into your system, so you need to produce a mapping from their XML data into your data model. The mismatch could then be quite severe, but as long as "enough" information is extractable then this would not be a problem as is some information could be ignored.

   What is required in this case would be an XSLT script (or equivalent) which extracted the RDF triples conforming to the information whcih is meaningful to the recipient.

3. **Converting between XML formats.**

Converting from one XML Schema format to another from a different user - for data exchange or combination is likely to be a common use. In this case, the "semantic" stage in the process can be ignored at conversion stage - this analysis would have taken place solely in the derivation of the mapping, controlled by a master ontology. XML Schema formats are likely to go into quite a lot of detail, and if the ontology cannot cope with both formats in detail then the transformation may not be possible this way. So here the expressive power of the ontology is likely to be a problem.

In this case, we would expect the mappings (between the ontology and the two XML Schemas) to control the derivation of an XSLT script.

Approach

The idea here is to construct a simple tool to assist the manual mapping between XML Schema and OWL ontologies. We would expect this to load an RDF schema (or OWL Ontology), load an XML Schema and then allow via a point and click interface, a mapping to be built between them.

The mapping itself should be represented as a set of RDF triples, for which would need a standard vocabulary for the mapping language. This mapping has to take into account the context of the information in the XML Schema, so will have to use XPath information to define the context. In the case of properties, the context of the domain and range of the property will also need to be taken into account.

The mapping could then be used in the following ways.

1. Generating Triples

Generating a set of RDF triples from an XML document conforming to the target schema, providing an instance of the ontology. This process is described in Figure 1.

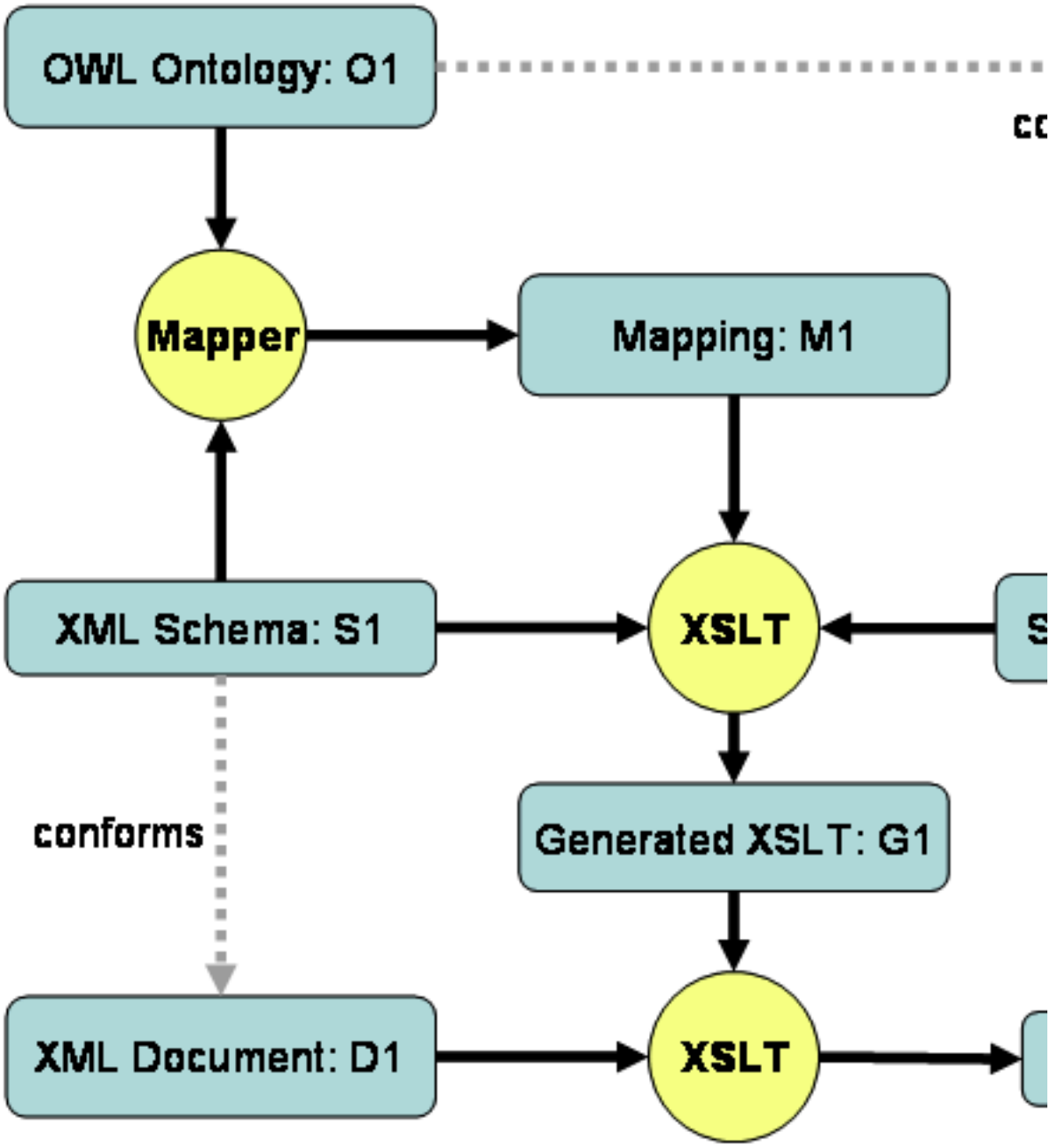


Figure 1: *Using the mapping to generated RDF triples from an XML document.*

This process proceeds as follows.

1. Generate mapping file M1 between Ontology O1 and XML Schema S1.
2. Process M1 using a standard XSLT script X1, also passing in the Schema S1 to generate a new XSLT script G1. G1

encodes the representation of documents whcih conform to S1 as RDF triples.

3. For an XML document D1, conformant to S1, process using XSLT via the generated script G1 to produce RDF triples R1. This will be conformant to the Ontology O1.

2. Generating an XML Representation of Triples

In this case, we derive a representation of triples conforming to the ontology as a XML document in terms of the XML Schema. This is given schematically in Figure 2.

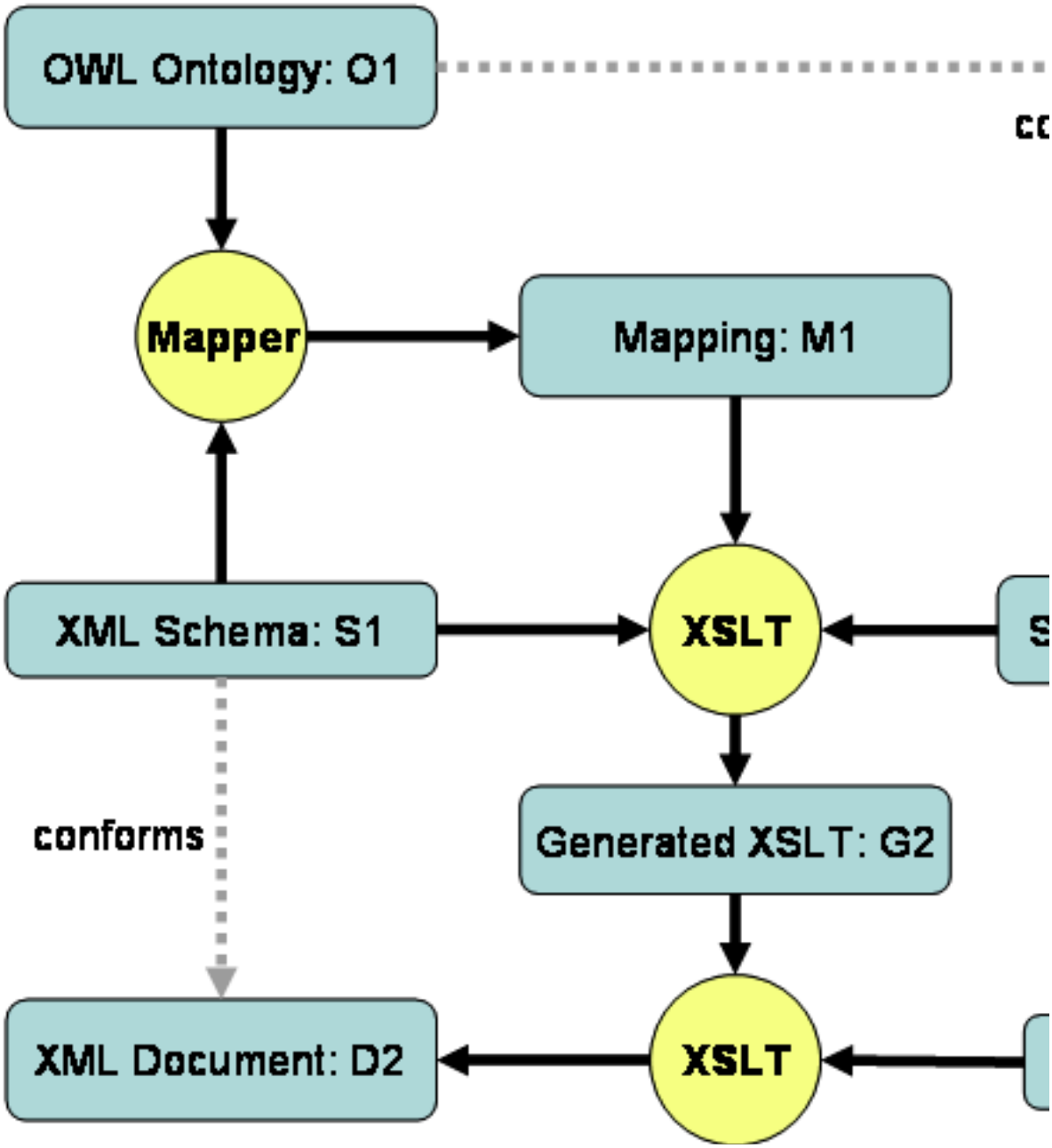


Figure 2: *Using the mapping to generated an XML document from RDF triples.*

This process proceeds as follows.

1. Generate mapping file M1 between Ontology O1 and XML Schema S1.
2. Process M1 using a standard XSLT script X2, also passing in the Schema S1 to generate a new XSLT script G2. G2 encodes the representation as documents whcih conform to S1 of RDF triples.
3. For a set of RDF triples, conformant to O1, process using XSLT via the generated script G2 to produce XML document D2. This will be conformant to the Schema S1.

3. Converting between an XML formats

Having used the map to transform in both directions, we can the combine these to generate a mapping function (in XSLT) between XML documents conformant to different schemas, as in Figure 3.

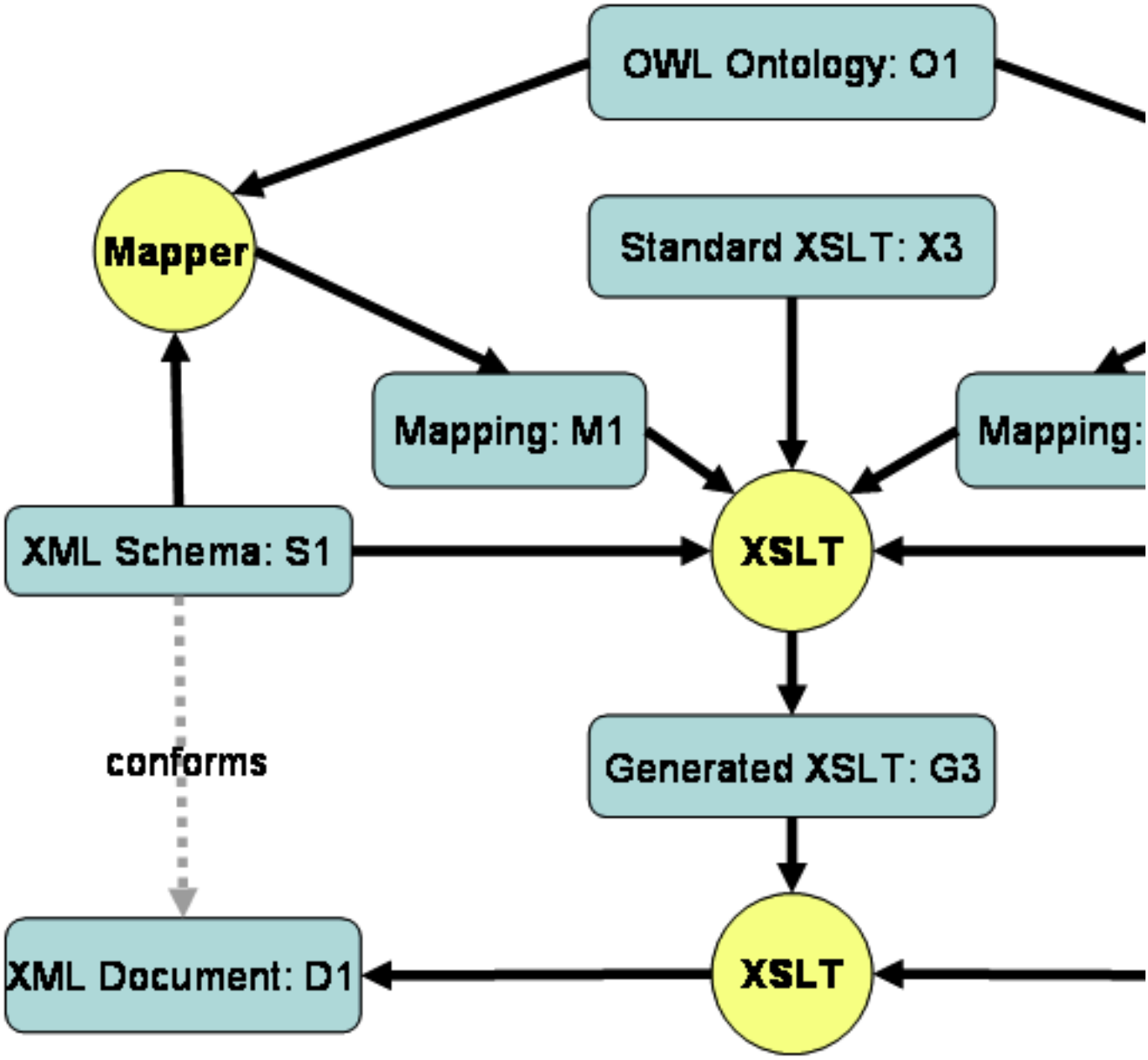


Figure 3: *Using the mapping to convert between XML document formats.*

Documents written under different XML Schemas form a pair of mappings of the respective XML Schema to the same RDF Schema/OWL Ontology.

This process proceeds as follows.

1. Generate mapping file M1 between Ontology O1 and XML Schema S1.
2. Generate mapping file M2 between Ontology O1 and XML Schema S2.
3. Process M1 and M2 using a standard XSLT script X3, also passing in the Schemas S1 and S2 to generate a new XSLT script G3. G2 encodes the mapping between the schemas.
4. Pass a document D2 through the XSLT processor using G3 to produce a XML document D1. This will be conformant to the Schema S1.

Similarly this will also work the other way around.

The resulting transformation between S1 and S2 will be partial in the sense that not all elements will be preserved or generated, but should be *semantics preserving* - the meaning of components which corresponds to parts classes and properties in the ontology will be transformed appropriately.

Other possible uses of the mapping.

Other things which could be integrated into the tool:

- a "heuristics-based" OWL ontology generator - takes an XML Schema and generates a "skeleton" suggested Ontology.
- a similar "XML Schema generator" - generates suggested XML exchange formats.

This should all be integrated eventually into a larger XML and RDF schema management system.

Example 1

We shall use the Purchase Order example. This example is one which is widely known; it is described in the XML Schema Primer [XMLSchemaPrimer]. We have added a namespace declaration.

Thus an instance of the Purchase Order, **po.xml** is as follows:

```
<?xml version="1.0"?>
<purchaseOrder
    xmlns="http://www.w3c.rl.ac.uk/xml/PO"
    orderDate="1999-10-20">
    <shipTo country="US">
        <name>Alice Smith</name>
        <street>123 Maple Street</street>
        <city>Mill Valley</city>
        <state>CA</state>
        <zip>90952</zip>
    </shipTo>
    <billTo country="US">
        <name>Robert Smith</name>
        <street>8 Oak Avenue</street>
        <city>Old Town</city>
        <state>PA</state>
        <zip>95819</zip>
    </billTo>
    <comment>Hurry, my lawn is going wild!</comment>
    <items>
        <item partNum="872-AA">
            <productName>Lawnmower</productName>
            <quantity>1</quantity>
            <USPrice>148.95</USPrice>
```

```
            <comment>Confirm this is electric</comment>
        </item>
        <item partNum="926-AA">
            <productName>Baby Monitor</productName>
            <quantity>1</quantity>
            <USPrice>39.98</USPrice>
            <shipDate>1999-05-21</shipDate>
        </item>
    </items>
</purchaseOrder>
```

This has an XML Schema as follows. Again this is as in the Primer, but with the addition of a namespace.

The Purchase Order Schema, **po.xsd**

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.w3c.rl.ac.uk/xml/PO"
        xmlns="http://www.w3c.rl.ac.uk/xml/PO"
        elementFormDefault="qualified"
>
 <xsd:annotation>
  <xsd:documentation xml:lang="en">
   Purchase order schema for Example.com.
   Copyright 2000 Example.com. All rights reserved.
  </xsd:documentation>
 </xsd:annotation>
```

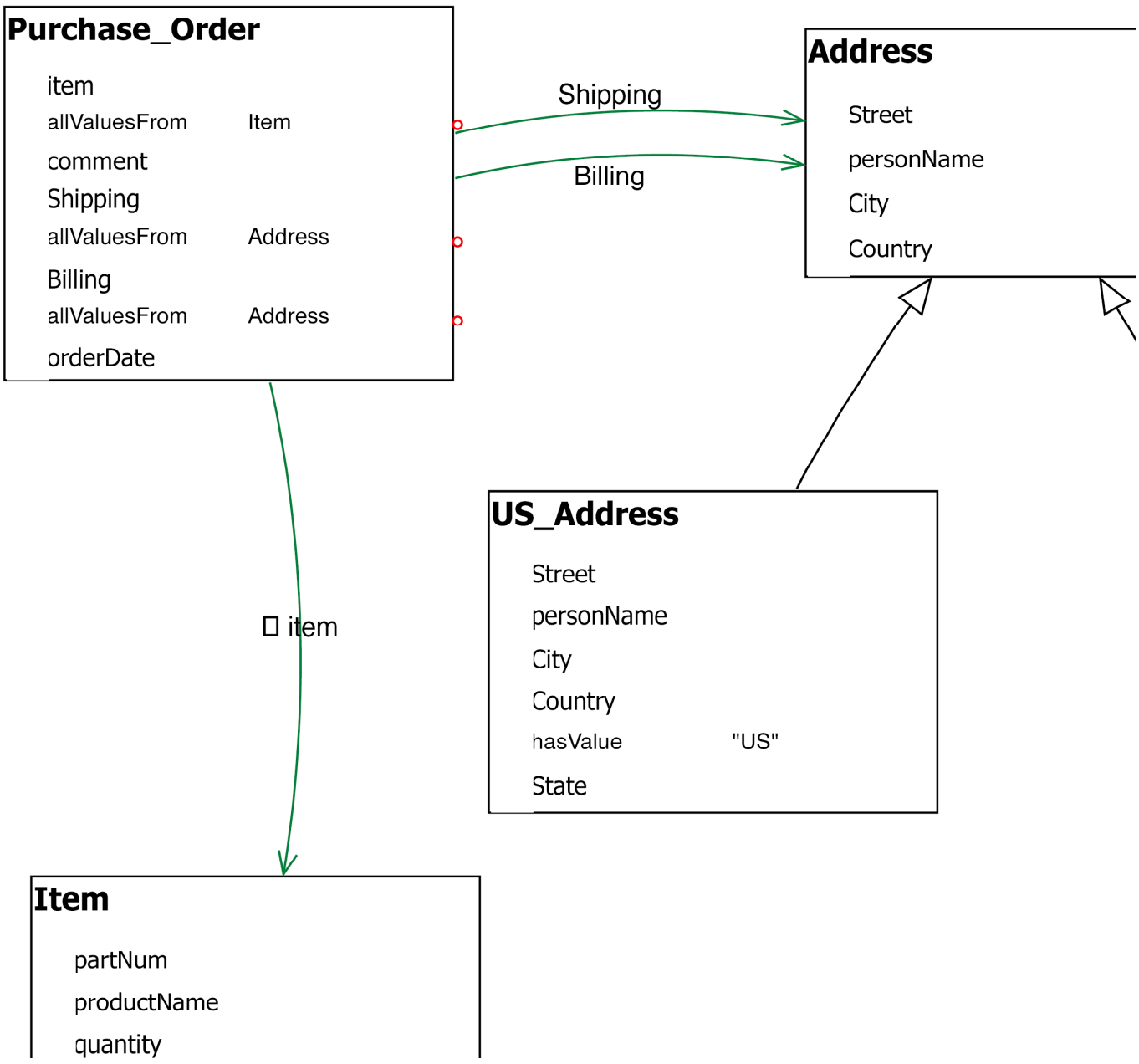
```
<xsd:element name="purchaseOrder" type="PurchaseOrderType"/>
<xsd:element name="comment" type="xsd:string"/>
<xsd:complexType name="PurchaseOrderType">
 <xsd:sequence>
  <xsd:element name="shipTo" type="USAddress"/>
  <xsd:element name="billTo" type="USAddress"/>
  <xsd:element ref="comment" minOccurs="0"/>
  <xsd:element name="items"  type="Items"/>
 </xsd:sequence>
 <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>
<xsd:complexType name="USAddress">
 <xsd:sequence>
  <xsd:element name="name"   type="xsd:string"/>
  <xsd:element name="street" type="xsd:string"/>
  <xsd:element name="city"   type="xsd:string"/>
  <xsd:element name="state"  type="xsd:string"/>
  <xsd:element name="zip"    type="xsd:decimal"/>
 </xsd:sequence>
 <xsd:attribute name="country" type="xsd:NMTOKEN"
    fixed="US"/>
</xsd:complexType>
<xsd:complexType name="items">
 <xsd:sequence>
```

```
  <xsd:element name="item" minOccurs="0" maxOccurs="unbounde
   <xsd:complexType>
    <xsd:sequence>
     <xsd:element name="productName" type="xsd:string"/>
     <xsd:element name="quantity">
      <xsd:simpleType>
       <xsd:restriction base="xsd:positiveInteger">
        <xsd:maxExclusive value="100"/>
       </xsd:restriction>
      </xsd:simpleType>
     </xsd:element>
     <xsd:element name="USPrice"  type="xsd:decimal"/>
     <xsd:element ref="comment"   minOccurs="0"/>
     <xsd:element name="shipDate" type="xsd:date" minOccurs=
    </xsd:sequence>
    <xsd:attribute name="partNum" type="SKU" use="required",
   </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>
<!-- Stock Keeping Unit, a code for identifying products -->
<xsd:simpleType name="SKU">
 <xsd:restriction base="xsd:string">
  <xsd:pattern value="\d{3}-[A-Z]{2}"/>
```

```
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

Of course this XML format defined under this schema is just one of many which are possible candidates XML format for Purchase Orders which are available.

We can then give an Ontology which tries to capture the essential information. This is represented as a diagram in Figure 4.



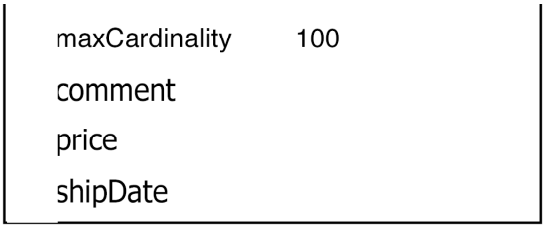


Figure 4: *The Purchase Order Ontology.*

Note that this includes a UKAddress class which is not in the above schema. That is reasonable to demonstrate the mapping, as the Ontology we are mapping to does not have to have exactly the same information as the XML Schema. The mapping may only be partial, enough to extract the appropriate piece of information.

OWL Ontology for Purchase Orders **po.owl**. Note the xmlns attribute; we need a namespace for this 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#"
    xmlns="http://www.w3c.rl.ac.uk/owl/PO"
>
  <owl:Ontology rdf:about="">
    <owl:imports rdf:resource="http://protege.stanford.edu/p
  </owl:Ontology>
  <owl:Class rdf:ID="Item">
  </owl:Class>
  <owl:Class rdf:ID="Address"/>
  <owl:Class rdf:ID="UK_Address">
    <rdfs:subClassOf rdf:resource="#Address"/>
```

```
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:DatatypeProperty rdf:about="#Country"/>
      </owl:onProperty>
      <owl:hasValue rdf:datatype="http://www.w3.org/2001/XM
      >UK</owl:hasValue>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="US_Address">
  <rdfs:subClassOf rdf:resource="#Address"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:DatatypeProperty rdf:about="#Country"/>
      </owl:onProperty>
      <owl:hasValue rdf:datatype="http://www.w3.org/2001/XM
      >US</owl:hasValue>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Purchase_Order">
  <rdfs:subClassOf>
```

```
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#Shipping"/>
        </owl:onProperty>
        <owl:allValuesFrom rdf:resource="#Address"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#Billing"/>
        </owl:onProperty>
        <owl:allValuesFrom rdf:resource="#Address"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#item"/>
        </owl:onProperty>
        <owl:allValuesFrom rdf:resource="#Item"/>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
```

```
<owl:ObjectProperty rdf:ID="item">
  <rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#(
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Purchase_Order"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="Billing"
   rdf:type="http://www.w3.org/2002/07/owl#FunctionalPrope
  <rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#(
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="Shipping"
   rdf:type="http://www.w3.org/2002/07/owl#FunctionalPrope
  <rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#(
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="partNum"
   rdf:type="http://www.w3.org/2002/07/owl#FunctionalPrope
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSche
  <rdfs:domain rdf:resource="#Item"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="productName">
```

```
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchem
  <rdfs:domain rdf:resource="#Item"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="State"
   rdf:type="http://www.w3.org/2002/07/owl#FunctionalPrope
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchem
  <rdfs:domain rdf:resource="#US_Address"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="County"
   rdf:type="http://www.w3.org/2002/07/owl#FunctionalPrope
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchem
  <rdfs:domain rdf:resource="#UK_Address"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="City"
   rdf:type="http://www.w3.org/2002/07/owl#FunctionalPrope
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchem
  <rdfs:domain rdf:resource="#Address"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="quantity"
   rdf:type="http://www.w3.org/2002/07/owl#FunctionalPrope
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchem
  <rdfs:domain rdf:resource="#Item"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="personName"
```

```
    rdf:type="http://www.w3.org/2002/07/owl#FunctionalPrope
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSche
  <rdfs:domain rdf:resource="#Address"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Country"
    rdf:type="http://www.w3.org/2002/07/owl#FunctionalPrope
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSche
  <rdfs:domain rdf:resource="#Address"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="shipDate"
    rdf:type="http://www.w3.org/2002/07/owl#FunctionalPrope
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSche
  <rdfs:domain rdf:resource="#Item"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Region"
    rdf:type="http://www.w3.org/2002/07/owl#FunctionalPrope
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSche
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Zip"
    rdf:type="http://www.w3.org/2002/07/owl#FunctionalPrope
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSche
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="UKPostcode"
    rdf:type="http://www.w3.org/2002/07/owl#FunctionalPrope
```

```
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchem
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Postcode"
   rdf:type="http://www.w3.org/2002/07/owl#FunctionalProper
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchem
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Street"
   rdf:type="http://www.w3.org/2002/07/owl#FunctionalProper
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchem
  <rdfs:domain rdf:resource="#Address"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="orderDate"
   rdf:type="http://www.w3.org/2002/07/owl#FunctionalProper
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchem
  <rdfs:domain rdf:resource="#Purchase_Order"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="comment"
   rdf:type="http://www.w3.org/2002/07/owl#FunctionalProper
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchem
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="price"
   rdf:type="http://www.w3.org/2002/07/owl#FunctionalProper
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchem
  <rdfs:domain rdf:resource="#Item"/>
```

```
    </owl:DatatypeProperty>

    <rdf:List rdf:about="http://www.w3.org/1999/02/22-rdf-synta

</rdf:RDF>
```

What we would like to do is derive a set of RDF triples from the original XML document. In the next section we discuss how to derive the mapping for this example.

Mapping from Ontologies to Schemas

We have decided to map from Ontologies to Schemas as it makes the mapping more straightforward; a single class can then correspond to some combination of components of the XML Schema as needed.

Mapping the Classes

We first map the classes within the Ontology.

We want to say that the class `Purchase_Order` in the **po.owl** is represented by the element: `purchaseOrder` in **po.xsd**. Of course this is a "design choice" in its own right already; we equally chould say that it the `PurchaseOrderType` which maps to `Purchase_Order`. Certainly any element of type `PurchaseOrderType` should be mapped to the class `Purchase_Order`. But for the time being, let us assume that we shall map the class to the element.

In this case, we are doing simple mapping of a class to an element. We want to then generate a RDF description of the maplet:

```
Purchase_Order   ------>  purchaseOrder
```

This is in itself is a short-hand; we would need to add more about the context of the mapping in form of XPaths into the XML Schema, and also we may map to attributes or (simple or

complex) type definitions as well as elements. Thus a fully expanded notation would be:

```
Purchase_Order    ------>  xsd:element[name="purchaseOrder"]
```

However, we (almost) always map to a filter expression which queries the name, so in general an acceptable shorthand for `[name="X"]` would be:

```
class(Purchase_Order)    ------>  element(purchaseOrder)
```

Mapping the class `US_Address` is reasonably clear; we map to the *complexType* `USAddress`:

```
class(US_Address)    ------>  complexType(USAddress)
```

Thus any *content model* of this type will correspond to the class `US_Address`; this mapping to a type map result in the creation of BNodes in a subsequent triple generation.

The class `UK_Address` has no analogue in the Schema; they can never occur in a valid instance of the XML Schema, so we can say that this schema does not have the expressive power to represent this class, and instances of `UK_Address` cannot be mapped into it. So no mapping for this class is given.

The class `Address` however, is partially representable in the XML Schema; that is, it is representable if it is a US Address. So we need a *conditional* mapping rule.

```
X:class(Address)    ------>  X:complexType(USAddress)

      if X:class(US_Address)
```

Note that in this definition of the map we have introduced a local instance variable `X`; such a variable is implied, but omitted as redundant in the previous maps. This would mean in general that we could generate instances of the class `Address` from instances of the complexType, `USAddress`, but in order to use this representation for the ontology, we would have to check in advance that the class instance was a US address.

Finally class `Item` maps to the local element `item` within the context of the `items` element.

```
X:class(Item)    ------>  element(items)/sequence/element(ite
```

In this case we have included the full path to the `item` element; as this element is local to the `items`, only those `item` elements which are within the scope of an `items` element truly represent a member of the class `Item`.

Mapping the Properties

We now consider the properties similarly. In the case of properties, in order to accurately provide the triple generation from the mapping, it may be necessary to include information on the domain and range instances of the property.

```
objectProperty(Billing)    ------>

        complexType(PurchaseOrderType)/sequence/element(billTo

      Domain ../purchaseOrder


      Range ./*



objectProperty(Shipping)    ------>

        complexType(PurchaseOrderType)/sequence/element(shipsT

       Domain ../purchaseOrder

       Range ./*
```

Thus to map from the Ontology to the XML Schema, we would produce a set of these mappings.

```
Purchase_Order    ---mapsToElement--->  purchaseOrder

Item       ---mapsToElement--->  item

Billing    ---mapsToElement--->  billTo

Shipping    ---mapsToElement--->  shipTo
```

```
US_Address  ---mapsToComplexType---> USAddress

Address ---mapsToComplexType---> USAddress
```

Note that in this example, there is no ambiguity about component of the ontology maps to which maps to which component of the Schema; in general this is not the case, and we would need to add path information give the context of the mapping.

Different components of the XML schema are mapped on - e.g both elements and types are mapped above.

A Mapping Schema

The mapping itself is rendered as an XML format. Here we give the mapping for the Purchase Order example in that format.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<map:mapping

  xmlns:map="http://www.w3c.rl.ac.uk/xml/SchemaMap"

>


<map:sourceNamespace value="http://a.com/ontology#"/>

<map:targetNamespace value="http://www.w3c.rl.ac.uk/orders"/>



<map:classMap>

  <map:source class="http://a.com/ontology#Customer"/>

  <map:target
path='/xsd:schema/xsd:element[@name="customer"]'/>

</map:classMap>
```

```
<map:propertyMap>

  <map:source property="http://a.com/ontology#customerName"/>

  <map:target
path='/xsd:schema/xsd:complexType[@name="orderType"]/xsd:att

  <map:domain
path='/xsd:schema/xsd:element[@name="customer"]'/>

  <map:range
path='/xsd:schema/xsd:complexType[@name="customerType"]/xsd:a

</map:propertyMap>



</map:mapping>
```

Another Example

We give another example together with a walkthrough of the Mapping development tool.

The Orders Ontology

The example is a simple example, which is representing an ordering control system for a company. Thus it has concepts for orders, parts, customers and manufacturers who produce parts. This is represented in the Ontology below.

```
<?xml version="1.0"?>

<rdf:RDF

    xmlns="http://a.com/ontology#"

    xmlns:protege="http://protege.stanford.edu/plugins/owl/p

    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://a.com/ontology">
<owl:Ontology rdf:about="">
  <owl:imports rdf:resource="http://protege.stanford.edu/p
</owl:Ontology>
<owl:Class rdf:ID="Customer">
  <rdfs:comment>Represents a customer object</rdfs:comment
</owl:Class>
<owl:Class rdf:ID="Manufacturer">
  <rdfs:comment>Represents a manufacturer</rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="OrderStatus">
  <rdfs:comment>The current status of an Order</rdfs:comme
</owl:Class>
<owl:Class rdf:ID="Part">
  <rdfs:comment>The class of Parts</rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="Order">
  <rdfs:comment>The class of orders.</rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="OrderPart">
  <rdfs:comment>Represents the "line" in an order</rdfs:co
</owl:Class>
<owl:ObjectProperty rdf:ID="orderStatus">
```

```
    <protege:allowedParent rdf:resource="#OrderStatus"/>
    <rdfs:domain rdf:resource="#Order"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Fu
    <rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:ID="shippingCity">
    <rdfs:domain rdf:resource="#Customer"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Fu
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSche
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="partDescription">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSche
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Fu
    <rdfs:domain rdf:resource="#Part"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="customerName">
    <rdfs:domain rdf:resource="#Customer"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSche
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Fu
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="orderDate">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSche
    <rdfs:domain rdf:resource="#Order"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Fu
```

```
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="orderStatusType">
  <rdfs:domain rdf:resource="#Order"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchem
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Fun
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="shippingAddress">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchem
  <rdfs:domain rdf:resource="#Customer"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Fun
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="manufacturerPhone">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Fun
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchem
  <rdfs:domain rdf:resource="#Manufacturer"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="orderStatusDescription">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchem
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Fun
  <rdfs:domain rdf:resource="#OrderStatus"/>
</owl:DatatypeProperty>
<owl:FunctionalProperty rdf:ID="part">
  <rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#C
  <rdfs:domain rdf:resource="#OrderPart"/>
```

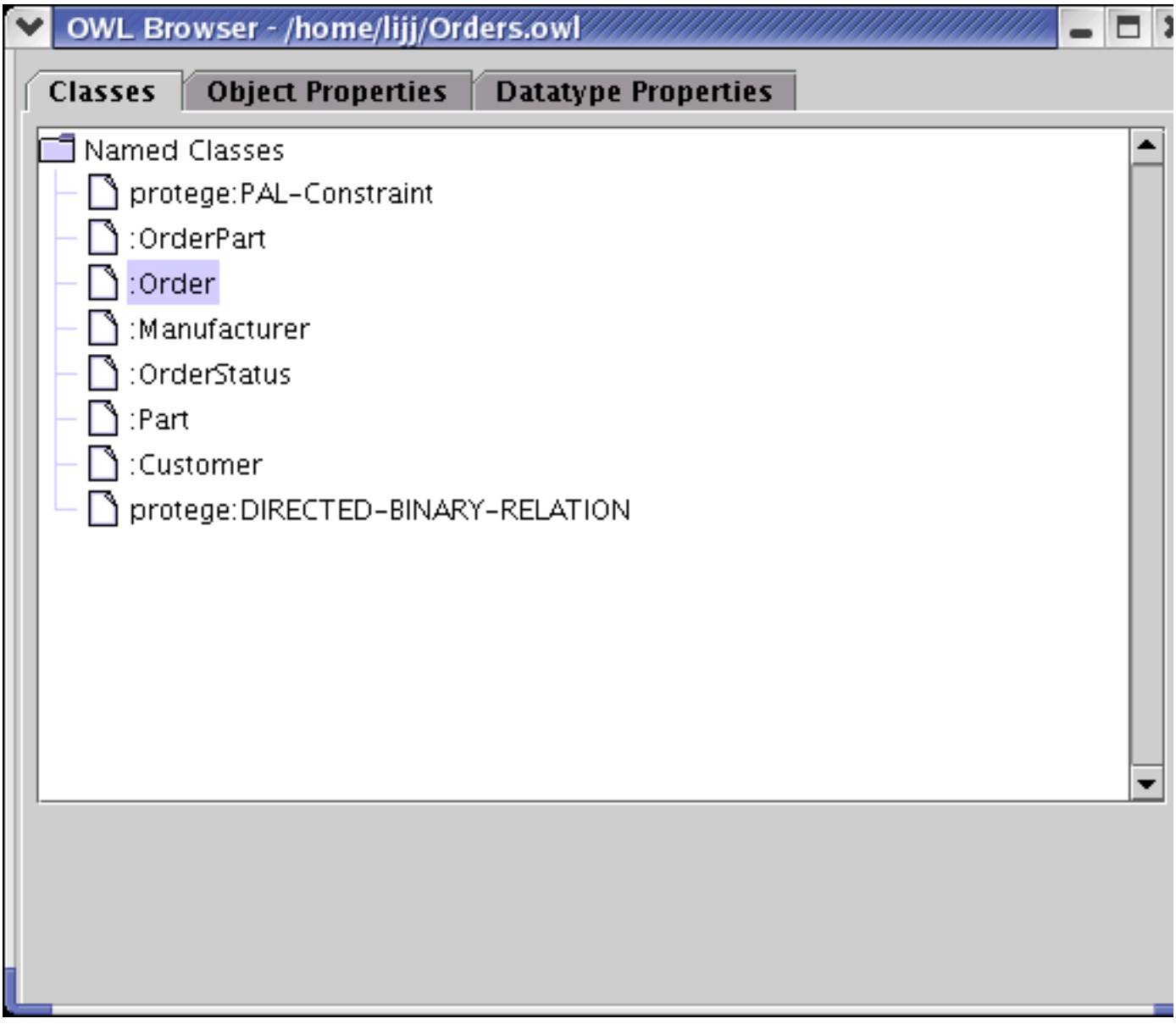
```
    <protege:allowedParent rdf:resource="#Part"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Obj
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="customer">
    <rdfs:domain rdf:resource="#Order"/>
    <rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#C
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Obj
    <protege:allowedParent rdf:resource="#Customer"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="shippingPostCode">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchem
    <rdfs:domain rdf:resource="#Customer"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Dat
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="manufacturerName">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Dat
    <rdfs:domain rdf:resource="#Manufacturer"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchem
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="price">
    <rdfs:domain rdf:resource="#OrderPart"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchem
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Dat
  </owl:FunctionalProperty>
```

```
  <owl:FunctionalProperty rdf:ID="order">
    <protege:allowedParent rdf:resource="#Order"/>
    <rdfs:domain rdf:resource="#OrderPart"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Obj
    <rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#C
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="customerPhone">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchem
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Dat
    <rdfs:domain rdf:resource="#Customer"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="quantity">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchem
    <rdfs:domain rdf:resource="#OrderPart"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Dat
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="madeBy">
    <rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#C
    <protege:allowedParent rdf:resource="#Manufacturer"/>
    <rdfs:domain rdf:resource="#Part"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Obj
  </owl:FunctionalProperty>
</rdf:RDF>
```

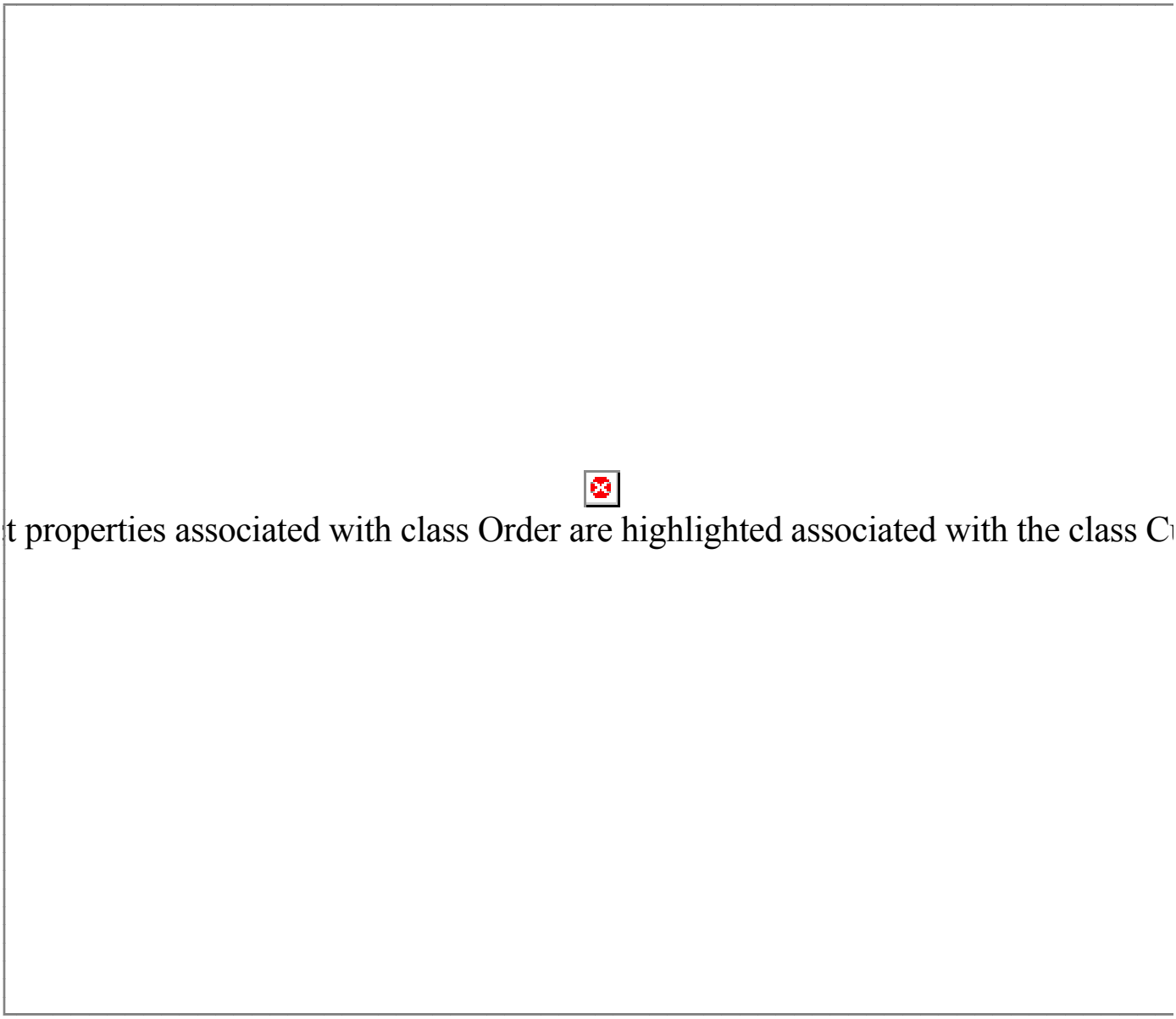
A walkthrough

We give some screen shots showing the stages via which the ontology is displayed in the tool.

The ontology browser shows the class hierarchy in the ontology in a simple tree representation:



Switching to another tabbed window shows the object properties associated with the selected class, Order:



Similarly, the datatype properties associated with class Order are shown in the third tab:

Datatype properties associated with class Order are highlighted

The Orders Schema

A schema representing an order document, orders1.xsd. This represents a view on the order ontology; it does not represent the whole of the information contained in the ontology, but a partial view for presenting a "customer report" giving the state of orders for a particular customer.

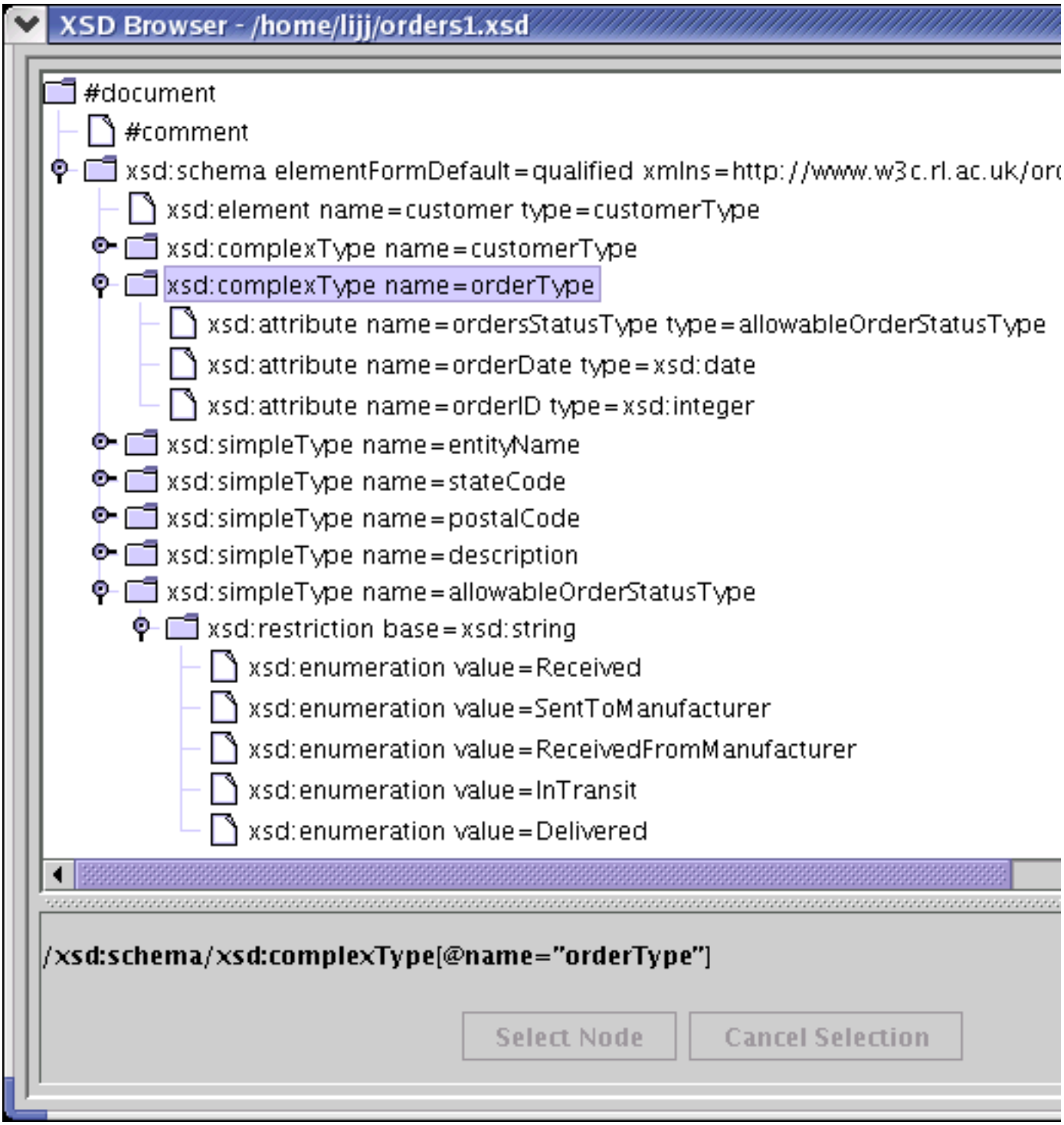
```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema

   xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
```

```
xmlns="http://www.w3c.rl.ac.uk/orders"
elementFormDefault="qualified">

<xsd:element name="customer" type="customerType"/>

<xsd:complexType name="customerType">
  <xsd:sequence>
    <xsd:element name="order" type="orderType" maxOccurs="unbou
  </xsd:sequence>
  <xsd:attribute name="shippingPostalCode" type="postalCode"/>
  <xsd:attribute name="shippingState" type="stateCode"/>
  <xsd:attribute name="shippingAddress" type="description"/>
  <xsd:attribute name="customerName" type="entityName"/>
</xsd:complexType>

<xsd:complexType name="orderType">
  <xsd:attribute name="ordersStatusType" type="allowableOrderSt
  <xsd:attribute name="orderDate" type="xsd:date"/>
  <xsd:attribute name="orderID" type="xsd:integer"/>
```

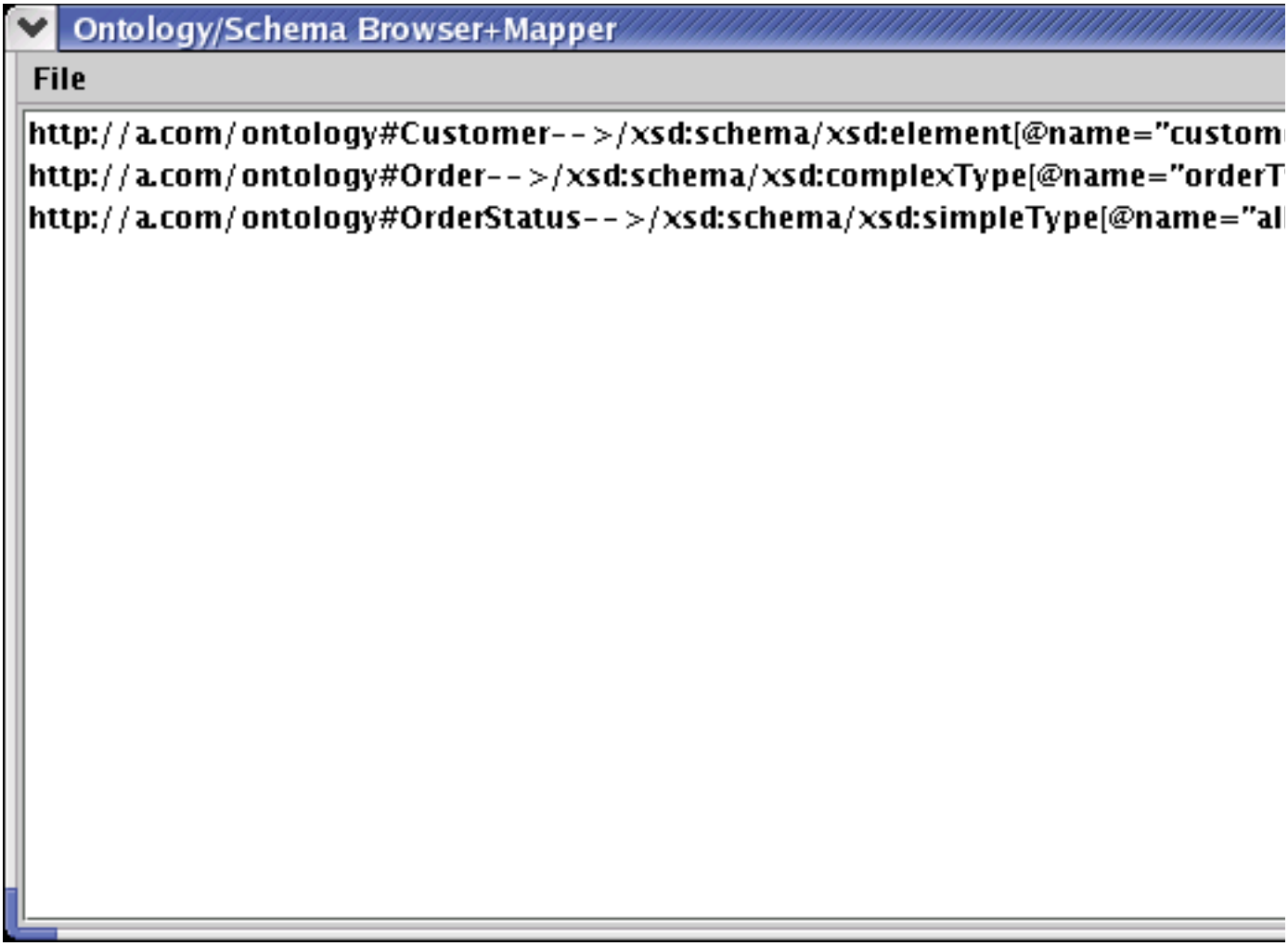
```
</xsd:complexType>

<xsd:simpleType name="entityName">

  <xsd:restriction base="xsd:string">

    <xsd:maxLength value="50" fixed="false"/>

  </xsd:restriction>

</xsd:simpleType>

<xsd:simpleType name="stateCode">

  <xsd:restriction base="xsd:string">

    <xsd:maxLength value="2"/>

  </xsd:restriction>

</xsd:simpleType>

<xsd:simpleType name="postalCode">

  <xsd:restriction base="xsd:string">

    <xsd:maxLength value="10"/>

  </xsd:restriction>

</xsd:simpleType>
```

```
  <xsd:simpleType name="description">
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="100"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="allowableOrderStatusType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Received"/>
      <xsd:enumeration value="SentToManufacturer"/>
      <xsd:enumeration value="ReceivedFromManufacturer"/>
      <xsd:enumeration value="InTransit"/>
      <xsd:enumeration value="Delivered"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

The schema browser presents this as shown below:



The mapping tool represents the mappings from the ontology to the schema as below. Here we are generating mappings from the classes in the ontology to the components of the schema. Note the we preserve the context information via an XPath through the XML Schema definition.



A similar process is undertaken to generated mappings from properties, using a selection method. This is part of a process as the user also has to enter domain and range components via a dialogue.

The mapping tool creates a file containing mappings such as these in the format below:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<map:mapping
  xmlns:map="http://www.w3c.rl.ac.uk/xml/SchemaMap"
>

<map:sourceNamespace value="http://a.com/ontology#"/>
<map:targetNamespace value="http://www.w3c.rl.ac.uk/orders"/

<map:classMap>
```

```
  <map:source class="http://a.com/ontology#Customer"/>
  <map:target path='/xsd:schema/xsd:element[@name="customer"]
</map:classMap>

<map:classMap>
  <map:source class="http://a.com/ontology#Order"/>
  <map:target path='/xsd:schema/xsd:complexType[@name="order
</map:classMap>

<map:classMap>
  <map:source class="http://a.com/ontology#OrderStatus"/>
  <map:target path='/xsd:schema/xsd:simpleType[@name="allowa
</map:classMap>

<map:propertyMap>
  <map:source property="http://a.com/ontology#customerName"/
  <map:target path='/xsd:schema/xsd:complexType[@name="order
  <map:domain path='/xsd:schema/xsd:element[@name="customer"]
  <map:range  path='/xsd:schema/xsd:complexType[@name="custo
</map:propertyMap>

<map:propertyMap>
  <map:source property="http://a.com/ontology#shippingPostCo
  <map:target path='/xsd:schema/xsd:complexType[@name="custo
  <map:domain path='/xsd:schema/xsd:element[@name="customer"]
  <map:range  path='/xsd:schema/xsd:simpleType[@name="postal
</map:propertyMap>

<map:propertyMap>
  <map:source property="http://a.com/ontology#shippingAddres
  <map:target path='/xsd:schema/xsd:complexType[@name="custo
  <map:domain path='/xsd:schema/xsd:element[@name="customer"]
  <map:range  path='/xsd:schema/xsd:simpleType[@name="descri
</map:propertyMap>

<map:propertyMap>
  <map:source property="http://a.com/ontology#orderStatus"/>
  <map:target path='/xsd:schema/xsd:complexType[@name="order
  <map:domain path='/xsd:schema/xsd:complexType[@name="custo
  <map:range  path='/xsd:schema/xsd:complexType[@name="order
</map:propertyMap>

<map:propertyMap>
  <map:source property="http://a.com/ontology#customer"/>
  <map:target path='/xsd:schema/xsd:complexType[@name="custo
  <map:domain path='/xsd:schema/xsd:complexType[@name="order
```

```
    <map:range  path='/xsd:schema/xsd:complexType[@name="custor
  </map:propertyMap>

  <map:propertyMap>
    <map:source property="http://a.com/ontology#orderStatusDes
    <map:target path='/xsd:schema/xsd:simpleType[@name="allowa
    <map:domain path='/xsd:schema/xsd:simpleType[@name="allowa
    <map:range  path='/xsd:schema/xsd:complexType[@name="custor
  </map:propertyMap>

  <map:propertyMap>
    <map:source property="http://a.com/ontology#customerName"/
    <map:target path='/xsd:schema/xsd:complexType[@name="custor
    <map:domain path='/xsd:schema/xsd:element[@name="customer"
    <map:range  path='/xsd:schema/xsd:complexType[@name="custor
  </map:propertyMap>


  </map:mapping>
```

Conclusions

The work described in this document represents the beginning of developing a tool for using mappings in an ontology to manage semantic relationships between XML Schema. This could leverage existing XML formats into the semantic web.

References

[SWAD5.2] SWAD-Europe: WP5.2 Extracting Semantics from XML Structure, Stephen Buswell.

[SCHT] Schematronhttp://www.ascc.net/xml/resource/schematron/

[XSLT] XSLT http://www.w3.org/Style/XSL/

[XNF]XML Normal Forms http://www.ltg.ed.ac.uk/~ht/normalForms.html , Henry Thompson.