DX.YvZZ RDFXSLT: XSLT-based Data Grounding for RDF

WSMO Working Draft 12 April 2007

This version:
http://www.wsmo.org/TR/d24/d24.2/v0.1/20070412/rdfxslt.html
Latest version:
no permanent link yet
Previous version:
none
Editors:
Jacek Kopecky

Table of contents

1. Introduction
2. Lifting using XSLT
3. Lowering with support of RDFXSLT
- **3.1 Difficulties with RDF/XML**
- **3.2 RDFXSLT "Flattened" Form**
- **3.3 RDFXSLT Support Functions**
- **3.4 Lowering example**

4. Attaching the XSLT stylesheets to SWS descriptions
5. Conclusions and Future Work

References
Acknowledgements

---

1. Introduction

The research area of Semantic Web Services (SWS) aims to increase the level of automation of some tasks commonly performed in service-oriented systems (e.g. discovering available services and composing them to provide more complex functionalities) using Semantic Web technologies, i.e. ontologies and reasoning. Automation will make Web service usage cheaper and free the human resources for higher-level work, but it can also enable proliferation of Web services on a much higher scale than currently feasible.

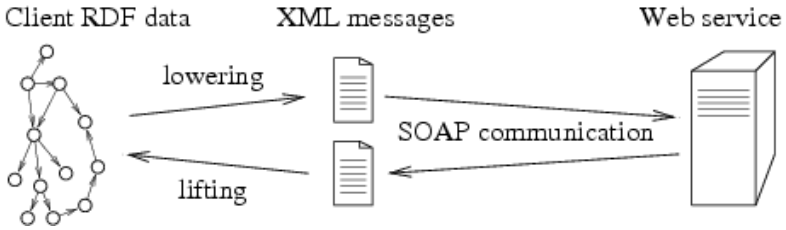
Semantic Web Service approaches are based on semantic descriptions of data, that is, on ontologies. An ontology language generally directly influences the form of data: an OWL ontology has data in RDF ([RDF]) graphs, a WSML ontology has data in sets of WSML instances. SWS framework implementations (e.g. WSMX) will also process data on the semantic level, i.e., the internal data format of a SWS implementation would be RDF or WSML etc. For the purpose of this deliverable we assume that the data is modeled as RDF graphs, or that the system can interface with the world using RDF.

Among the tasks that SWS aim to automate are service negotiation and invocation, both involving direct interaction with the Web service. While the SWS framework will internally use RDF data, Web services use XML for communication. We have here application-specific data in RDF, application-specific data format in XML, and we need to transform between the two forms in order for the SWS framework to be able to communicate with the Web services.

These data transformations are called *data grounding*. The data grounding transformations must go in both directions, from RDF to XML and back. Transforming from RDF data to XML is called *lowering*, because it goes from the (higher) semantic level to the lower level of XML. Transforming from XML back to RDF is accordingly called *lifting*. In the case of an automated SWS client that needs to use a Web service, lowering

creates the XML request message from the RDF data available to the client, and lifting converts the XML response message into RDF data that fits right back into the client system. Figure 1.1 illustrates the process.

Figure 1.1: RDF data grounding for WS communication



In this deliverable, we show that the XML transformation language XSLT, an off-the-shelf, standard and well-known technology, can be used to implement RDF data grounding, readily allowing the cooperation of semantic systems with Web services. In particular, we use XSLT version 2.0, which is a recent development (January 2007), but we expect that it will be adopted quickly, due to the improvements over version 1.0. XSLT 2.0 is implemented completely in the freely available Saxon processor.

The major issue addressed in our work is the problem with accessing RDF/XML data from XML processing tools using XPath. We provide a package called RDFXSLT that simplifies using XSLT to process RDF data tremendously. We are highly concerned with the adoption of SWS technologies, therefore our package aims to be easy to use for developers already skilled in XML technologies.

The readers of this deliverable are expected to know the RDF/XML exchange format for RDF (see [RDF/XML]) as well as N-triples (see N-Triples), and the XSLT transformation language ([XSLT]).

The deliverable is structured as follows: in Section 2 we talk about using XSLT for lifting, which is basically a straightforward application of XSLT. In Section 3 we describe the not-so-straightforward situation in lowering, especially detailing how we overcome the difficulties posed by RDF/XML. Section 4 specifies how data grounding XSLT stylesheets can be referenced from SWS descriptions, in order for the descriptions to be complete. And Section 5 presents conclusions about application of XSLT for data grounding, about possible alternative approaches and our planned future work.

2. Lifting using XSLT

Lifting is the part of data grounding where XML data from SOAP messages need to be translated to semantic data in RDF. The source data for lifting transformations is part of the public interface of a Web service, therefore we can assume that it is well-structured and therefore it should pose no problems to XPath addressing (see [XPath]), such as those described in Section 3.1 posed by RDF/XML.

Therefore the design of an XSLT transformation from the source XML data into RDF/XML should be a straightforward, with perhaps a single exception: the source XML data may, in some applications, contain cross-links within the data hierarchy. In XML, such cross-links generally use element IDs or matching data values, which may be context-sensitive. In RDF, context-sensitive identifiers are discouraged, and certainly URIs and explicit blank node IDs are not context sensitive. Creating RDF IDs for cross-links is illustrated in the following example.

Suppose we have an XML structure like the one in the following listing. Note that we omit any namespaces for brevity.

Listing 2.1: XML Snippet with cross-hierarchy links

```
<sales>
  <products>
    <product id="p1">Packing Boxes</product>
    <product id="p2">Packing Tape</product>
  </products>
  <sale-records>
    <customer num="C1001">
      <productsale idref="p1">100</productsale>
      <productsale idref="p2">200</productsale>
    </customer>
    <customer num="C1002">
      <productsale idref="p2">50</productsale>
    </customer>
  </sale-records>
</sales>
```

We see that product sale records reference product definitions by their IDs. In case the id attribute on a product is declared as an XML ID, it is guaranteed that its value will be unique among all the IDs in a given XML document. Thus this value can be translated into an identifier (node ID) in the RDF data, which would look like this:

Listing 2.2: RDF representation of the data above (in N-triples)

```
_:p1  rdf:type  Product  .
_:p1  name  "Packing Boxes"  .

_:p2  rdf:type  Product  .
_:p2  name  "Packing Tape"  .

_:genid_1  rdf:type  ProductSale  .
_:genid_1  customer  cust:C1001  .
_:genid_1  product  _:p1  .
_:genid_1  quantity  100  .

_:genid_2  rdf:type  ProductSale  .
_:genid_2  customer  cust:C1001  .
_:genid_2  product  _:p2  .
_:genid_2  quantity  200  .

_:genid_3  rdf:type  ProductSale  .
_:genid_3  customer  cust:C1002  .
_:genid_3  product  _:p2  .
_:genid_3  quantity  50  .
```

We use the identifiers _:genid_n to indicate node IDs that are generated by the serializer of the RDF graph, and not given in the data. In RDF/XML, the attribute rdf:nodeID is used to indicate a node ID, and an XSLT stylesheet would create the descriptions of the products and product sales by putting the id from the XML data in the rdf:nodeID attribute as shown in the listing below.

Listing 2.3: XSLT templates that transforms the sale records into RDF/XML

```
<xsl:template match="product">
  <Product rdf:nodeID="{@id}">
    <name><xsl:value-of select="."/></name>
  </Product>
</xsl:template>

<xsl:template match="productsale">
  <ProductSale>
    <customer rdf:reference="http://example.com/customer/{../@num}"/>
    <product rdf:nodeID="{@idref}"/>
    <quantity><xsl:value-of select="."/></quantity>
  </ProductSale>
</xsl:template>
```

In case when the id attribute in the original product sale data is not defined as an XML ID, it may be possible that other unrelated parts of the data use the same ID values, because in XML the association of the value with what it identifies is often context-sensitive. We might, for instance, have a product "p1" and a purchase record also identified as "p1". In RDF, identifiers are unambiguous (not context-sensitive), therefore we could not reuse the identifier values directly as node IDs, because the product node and the purchase node would clash and become one. In this case, the XSLT function generate-id() can be used like this:

Listing 2.4: XSLT templates that transforms the sale records into RDF/XML, using the function generate-id()

```
<xsl:template match="product">
  <Product rdf:nodeID="{generate-id(.)}">
    <name><xsl:value-of select="."/></name>
  </Product>
</xsl:template>

<xsl:template match="productsale">
  <ProductSale>
    <customer rdf:reference="http://example.com/customer/{../@num}"/>
    <product rdf:nodeID="{generate-id(//product[@id=current()/@idref])}"/>
    <quantity><xsl:value-of select="."/></quantity>
  </ProductSale>
</xsl:template>
```

Apart from the required care when generating cross-links in the resulting RDF data, using XSLT for lifting

XML data into RDF is not special.

3. Lowering with support of RDFXSLT

In contrast to lifting, using XSLT for lowering is complicated by the fact that the source data is in the RDF/XML format. XPath and XSLT were optimized to handle XML data with a simple and known hierarchical structure. RDF/XML provides significant flexibility in how RDF graphs can be serialized. This flexibility is very useful when creating RDF/XML data, or when presenting the data to users in its raw form (RDF/XML can be made almost as readable as good XML), but the processors that handle RDF/XML as XML data (not as a set of triples), such as XSLT, need to take the flexibility into account when looking for pieces of the data. We describe the problems presented by RDF/XML to XML processing tools in Section 3.1.

We have developed a set of XSLT stylesheets and functions (collectively called *RDFXSLT*) that mostly eliminate these problems, making it possible to transform RDF/XML with XSLT with little or no difficulty.

The first part of RDFXSLT is a stylesheet that **preprocesses** the incoming RDF/XML data so that the RDF statements are captured in a predictable way, independent of the structure of the original RDF data — in the so-called "flattened" form. We describe this preprocessing step and the "flattened" subset of the RDF/XML format in Section 3.2.

Even after the preprocessing, a few features of RDF/XML may still complicate XSLT processing. To assist the creators of lowering stylesheets, RDFXSLT provides three functions that hide said RDF/XML features. These support functions and their use are described in Section 3.3.

Finally, Section 3.4 shows a full example of an RDFXSLT-based lowering transformation.

3.1 Difficulties with RDF/XML

The difficulty with XPath processing of RDF/XML is best demonstrated on an example. The following listing contains two different RDF/XML serialization of the same RDF graph (adopted from TreeHugger Introduction):

Listing 3.1: Two different RDF/XML serialization of the same RDF graph

```
(1)

<foaf:Person>
  <foaf:name>Jacek Kopecky</foaf:name>
  <foaf:mbox rdf:resource="mailto:jacek.kopecky@deri.org"/>
  <foaf:knows>
    <foaf:Person>
      <foaf:name>Dieter Fensel</foaf:name>
    </foaf:Person>
  </foaf:knows>
</foaf:Person>

(2)

<rdf:Description foaf:name="Jacek Kopecky">
  <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
  <foaf:mbox rdf:resource="mailto:jacek.kopecky@deri.org"/>
  <foaf:knows rdf:nodeID="k49034"/>
</rdf:Description>

<foaf:Person rdf:nodeID="k49034">
  <foaf:name>Dieter Fensel</foaf:name>
</foaf:Person>
```

The two RDF/XML fragments look very different to XML processing tools, yet exactly the same to RDF tools. In either case we could create a simple XPath expression that finds the name of the person known to Jacek Kopecky, but a single expression that would work in both cases would get very ugly.

Here is a list of some particular offenders — features of the RDF data model and features of the RDF/XML syntax that complicate XPath processing:

- a property element can contain its value, or it can reference a value located elsewhere
- a string property (or the rdf:type property) can be represented as an element or as an attribute
- the URI of a resource can be in rdf:ID or in rdf:about, the latter being a relative URI or an absolute one
- also rdf:resource (a reference to a resource) and rdf:type can be relative or absolute
- the type of a resource can be indicated by rdf:type or with the element name of the resource description element

- RDF container membership may be expressed as rdf:li or rdf:_1, rdf:_2 etc.
- statements about the same subject resource need not be grouped in a single element

All these features and more make the XPaths, the basic building block of XML processing, very complicated, if they try to account for every possible alternative in how the data might be structured.

3.2 RDFXSLT "Flattened" Form

If we select a suitable subset of RDF/XML, we can avoid most of the complications, and simplify the XPaths needed to navigate the data structure. This section describes such a subset (*flattened RDF/XML*) and an XSLT stylesheet that turns an input RDF/XML document of any form into this perhaps seemingly ugly, but predictable form.

The following are the rules that define the RDF/XML subset that we use:

1. The root element is rdf:RDF.
2. The root element only contains rdf:Description children, one for each resource or nodeID mentioned in the model. Every rdf:Description element has either rdf:nodeID or rdf:about attribute.
    - This rule introduces empty rdf:Description elements that do not contain statements, and it also makes all bnodes explicit with nodeIDs. This is necessary for consistent handling of property values — when looking for the values of a given property, the result is all the appropriate rdf:Description elements.
    - This rule also means that all statements about one node are grouped under one rdf:Description element.
3. All properties are expressed as elements, including rdf:type and string properties.
    - This allows us to always use the property QName in XPath to select property values.
4. All property elements either have literal value, or rdf:resource or rdf:nodeID attributes to point to their value.
    - This means that parseType="Resource" is never used.
5. Lists and reification are in expanded form, i.e. parseType="Collection" and rdf:ID on properties are never used.
6. All container membership properties are expressed in the numerical form rdf:_n, and ordered with respect to one another, whereas rdf:li is never used.
    - This form preserves any duplicates and "holes" in the numbering.
    - RDFXSLT provides helper functions to select all of the rdf:_n properties, and the requirement on ordering means that the natural XPath document order will also match the order of the membership.
7. All RDF attributes (e.g. rdf:about, rdf:resource etc.) are namespace-qualified, even though RDF/XML technically allows them to be in the default namespace.
8. All URI values are absolute. rdf:ID is never used.
    - Using entity references to shorten URI is still allowed because they are resolved before XPath processing.

In RDFXSLT, we call this subset the "Flattened". To illustrate it, the following listing shows the flattened version of the foaf:Person example from Listing 3.1.

Listing 3.2: The flattened form of the data from Listing 3.1, with namespaces.

```
00  <rdf:RDF xmlns:foaf="http://xmlns.com/foaf/0.1/"
01           xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
02    <rdf:Description rdf:nodeID="d4e1">
03      <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
04      <foaf:name>Jacek Kopecky</foaf:name>
05      <foaf:mbox rdf:resource="mailto:jacek.kopecky@deri.org"/>
06      <foaf:knows rdf:nodeID="d4e10"/>
07    </rdf:Description>
08    <rdf:Description rdf:about="http://xmlns.com/foaf/0.1/Person"/>
09    <rdf:Description rdf:about="mailto:jacek.kopecky@deri.org"/>
10    <rdf:Description rdf:nodeID="d4e10">
11      <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
12      <foaf:name>Dieter Fensel</foaf:name>
13    </rdf:Description>
14  </rdf:RDF>
```

Note the two rdf:Description elements on lines 08 and 09, created due to rule 2, which allows us always to use rdf:Description to represent a resource (or a bnode, for that matter). Also note that the rdf:Description bnode on line 02 has a nodeID even though it is not referenced, this is for consistency, providing a unique identifier to each rdf:Description (either in rdf:about or in rdf:nodeID) which may be used to provide IDs in the XML serialization, if necessary.

We provide an XSLT stylesheet, rdfxslt-flatten.xslt, that takes as input any RDF/XML document and outputs

its flattened form. This stylesheet can be used as a preprocessor whose results are run through the actual lowering stylesheet, but the RDFXSLT flattening stylesheet can also be imported by the lowering stylesheet and used in a single invocation of an XSLT processor, thanks to the improvements in XSLT 2.0. This is shown below in Section 3.4.

As it is currently implemented, the rdfxslt-flatten.xslt stylesheet has several minor limitations:

1. If the processed file uses IDs or relative URIs and does not specify a base URI, a local filesystem base URI may be used by the XSLT processor, and various implementations on some systems differ in how many slashes they put after "file:". Then one may end up with URIs like "file:/home/jacek/something" and "file:////home/jacek/something" (created in one test by Saxon and Jena respectively), which can complicate data merging.
2. Namespace URIs and datatype URIs are not resolved against the base, so if the input data contains relative URIs as namespaces or datatypes, it may cause mismatches in XPath processing.
3. rdf:_XXX properties where XXX is not a normalized natural number may be bundled together with the real container membership properties in some processing.
4. XML literals in the source RDF/XML data are unchanged, i.e. they can be either expressed with parseType="Literal" (or any unknown parse type) or as a literal value with datatype rdf:XMLLiteral. In the second form, the XML data is canonicalized and escaped into a string. The lowering stylesheet may need to account for these differences.
5. The stylesheet may not be the optimal implementation of the transformation. So far this has not been a problem.

3.3 RDFXSLT Support Functions

The flattened RDF/XML produced by the rdfxslt-flatten.xslt stylesheet is much more amenable to XPath addressing and XSLT processing, but still there are a few features of RDF/XML or the RDF data model that can be annoying:

- an object value is identified with rdf:resource or with rdf:nodeID,
- the rdf:_n container memebership properties are a potentially infinite set of different names,
- properties with literal or resource values look like very similar elements.

To help the designer of the lowering stylesheet with these problems, we provide XSLT functions rx:references(), rx:is-literal-property() and rx:is-rdf-li(), as described below. The functions are defined in the rdfxslt-functions.xslt stylesheet that can be imported in the lowering stylesheet.

We intend to add more functions as need arises, and to investigate the possibility of using XSLT keys to further simplify accessing RDF/XML data in lowering XSLT stylesheets.

Before we proceed to the function definitions, we want to introduce two sample XPath queries to show how RDF data is accessed in the flattened RDF/XML form:

```
//rdf:Description[@rdf:about = 'http://example.com/thing']
//rdf:Description[foaf:name = 'Jacek Kopecky']
```

The first XPath expression basically finds a resource by name: it selects the rdf:Description node that may contain statements about the resource "http://example.com/thing". The second XPath expression finds a resource by a literal value, in this case it finds the resource that has the foaf:name "Jacek Kopecky". These XPath expressions are reliable in the RDFXSLT flattened form of RDF/XML.

The following are the definitions of the helper functions from rdfxslt-functions.xslt.

FUNCTION rx:references(property as element*, description as element) as xs:boolean

This function checks whether any of the given property elements references the given description element, in other words whether the given description is the target (value) of the property. The function returns true when the description is the target of any of the property elements.

The function is intended to be used to find the object value(s) of a property, or to find the subject(s) that have a property whose value is the current description element, i.e. traversing the property backwards.

To find the value(s) of a property eg:prop (when the current context node is rdf:Description), use an XPath of the following form:

```
//rdf:Description[rx:references(current()/eg:prop, .)]
key('rx:values', rx:refval(eg:prop))
```

To find the reverse value(s) of a property eg:prop (when the current context node is rdf:Description), use an

XPath of the following form:

```
//rdf:Description[eg:prop[rx:references(., current())]]
key('rx:inv-values', rx:refval(.))
```

FUNCTION rx:is-literal-property(property as element) as xs:boolean

This function checks whether the given property element has a literal value (which may be empty). The function is intended to be used to find the literal value(s) of a property.

To find the literal value(s) of a property eg:prop (when the current context node is rdf:Description), use an XPath of the following form:

```
eg:prop[rx:is-literal-property(.)]
```

One can also find all the subjects that have a given literal as the value of a given property (this will not work for XML Literals with rdf:parseType="Literal"):

```
//rdf:Description[string(eg:prop[rx:is-literal-property(.)]) = 'value']
```

FUNCTION rx:is-rdf-li(property as element) as xs:boolean

This function checks whether the given property element is rdf:li or rdf:_n. The function is intended to be used to select all rdf:li and rdf:_n properties on a node.

To find the rdf:li and rdf:_n properties of the current context node (presumably rdf:Description), use an XPath of the following form:

```
*[rx:is-rdf-li(.)]
```

3.4 Lowering example

This section presents a complete example scenario with a lowering XSLT stylesheet that uses the RDFXSLT flattening and functions support. This example scenario shows the lowering grounding transformation for the SWS Challenge Muller shipment service, in particular for the input message of the ShipmentOrder operation (see the WSDL description of the service). The actual lowering XSLT stylesheet is in Listing 3.7, the other listings are support material.

The following listing shows an example of the input XML data for the ShipmentOrder operation.

Listing 3.3: Example input data for Muller ShipmentOrder operation, also available as muller-input.xml.

```
<shipmentOrderRequest xmlns="http://www.example.org/muller/">
  <addressFrom>
    <firstname>Michael</firstname>
    <lastname>Moon</lastname>
    <address>Moon Road 13, Moon City</address>
    <location>
      <postalCode>1234</postalCode>
      <country>USA</country>
      <state>California</state>
    </location>
    <contactInformation>
      <phone>+1 424242</phone>
      <EMail>michael.moon@moon.ie</EMail>
      <fax>+1 424243</fax>
    </contactInformation>
  </addressFrom>
  <shipmentDate>
    <earliestPickupDate>2006-01-31T12:00:00</earliestPickupDate>
    <latestPickupDate>2006-01-31T14:00:00</latestPickupDate>
  </shipmentDate>
  <packageInformation>
    <quantity>1</quantity>
    <weight>1</weight>
    <length>5</length>
    <height>5</height>
    <width>5</width>
  </packageInformation>
  <packageInformation>
    <quantity>1</quantity>
    <weight>2</weight>
    <length>6</length>
    <height>6</height>
    <width>6</width>
  </packageInformation>
  <addressTo>
    <firstname>Jacek</firstname>
    <lastname>Kopecky</lastname>
    <address>Technikerstrasse 21a, Innsbruck</address>
    <location>
      <postalCode>6020</postalCode>
      <country>Austria</country>
      <state/>
    </location>
  </addressTo>
</shipmentOrderRequest>
```

We have created a simple ontological model for the data that a semantic client of the SWS Challenge shipment services would likely have. The following listing shows the model in a simple class-and-attribute syntax.

Listing 3.4: Our ontological model for the SWS Challenge shipment services.

```
class Order
   fromAddr: PickupAddress
   fromContact: Contact
   toAddr: Address
   toContact: Contact
   package*: Package
   pickupFrom: date
   pickupTo: date

class Address
   firstname: string
   middlename: string
   lastname: string
   company: string
   street: string
   city: string
   zip: string
   state: string
   country: string

class PickupAddress:Address
   specialInstructions: string

class Contact
   firstname: string
   middlename: string
   lastname: string
   phone: string
   fax: string
   email: string

class Package
   quantity: int
   weight: float
   width: float
   height: float
   length: float
```

This model has some things that the Muller service does not support. For instance, our model has fields for street address and city, whereas Muller combines these in the element <address>.

The following listing shows the RDF data with which the semantic client wants to invoke the shipment service. The data is in partially abbreviated RDF/XML.

Listing 3.5: Example data according to our ontological model, also available as data.rdf.

```
<rdf:RDF xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns="http://example.com/shipping#" >
<Order>
  <fromAddr>
    <PickupAddress>
      <firstName>Michael</firstName>
      <lastName>Moon</lastName>
      <street>Moon Road 13</street>
      <city>Moon City</city>
      <zip>1234</zip>
      <state>California</state>
      <country>USA</country>
    </PickupAddress>
  </fromAddr>
  <fromContact>
    <Contact>
      <firstName>Michael</firstName>
      <lastName>Moon</lastName>
      <phone>+1 424242</phone>
      <fax>+1 424243</fax>
      <email>michael.moon@moon.ie</email>
    </Contact>
  </fromContact>
  <toAddr>
    <Address>
      <firstName> Jacek</firstName>
      <lastName>Kopecky</lastName>
      <street>Technikerstrasse 21a</street>
      <city>Innsbruck</city>
      <zip>6020</zip>
      <country>Austria</country>
    </Address>
  </toAddr>
  <toContact>
    <Contact>
      <firstName> Jacek</firstName>
      <lastName>Kopecky</lastName>
      <phone>+435125076481</phone>
      <fax>+435125079872</fax>
      <email>jacek.kopecky@deri.org</email>
    </Contact>
  </toContact>
  <package>
    <Package>
      <quantity>1</quantity>
      <weight>1</weight>
      <width>5</width>
      <height>5</height>
      <length>5</length>
    </Package>
  </package>
  <package>
    <Package>
      <quantity>1</quantity>
      <weight>2</weight>
      <width>6</width>
      <height>6</height>
      <length>6</length>
    </Package>
  </package>
  <pickupFrom>2006-01-31T12:00:00</pickupFrom>
  <pickupTo>2006-01-31T14:00:00</pickupTo>
</Order>
</rdf:RDF>
```

For illustration, the following listing shows the flattened form of the above data.

Listing 3.6: Flattened form of data from listing 3.5.

```
<rdf:RDF xmlns="http://example.com/shipping#"
       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:nodeID="d4e5">
    <rdf:type rdf:resource="http://example.com/shipping#Order"/>
    <fromAddr rdf:nodeID="d4e9"/>
    <fromContact rdf:nodeID="d4e37"/>
    <toAddr rdf:nodeID="d4e58"/>
    <toContact rdf:nodeID="d4e83"/>
    <package rdf:nodeID="d4e104"/>
    <package rdf:nodeID="d4e126"/>
    <pickupFrom>2006-01-31T12:00:00</pickupFrom>
    <pickupTo>2006-01-31T14:00:00</pickupTo>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.com/shipping#Order"/>
  <rdf:Description rdf:nodeID="d4e9">
    <rdf:type rdf:resource="http://example.com/shipping#PickupAddress"/>
    <firstName>Michael</firstName>
    <lastName>Moon</lastName>
    <street>Moon Road 13</street>
    <city>Moon City</city>
    <zip>1234</zip>
    <state>California</state>
    <country>USA</country>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.com/shipping#PickupAddress"/>
  <rdf:Description rdf:nodeID="d4e37">
    <rdf:type rdf:resource="http://example.com/shipping#Contact"/>
    <firstName>Michael</firstName>
    <lastName>Moon</lastName>
    <phone>+1 424242</phone>
    <fax>+1 424243</fax>
    <email>michael.moon@moon.ie</email>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.com/shipping#Contact"/>
  <rdf:Description rdf:nodeID="d4e58">
    <rdf:type rdf:resource="http://example.com/shipping#Address"/>
    <firstName>Jacek</firstName>
    <lastName>Kopecky</lastName>
    <street>Technikerstrasse 21a</street>
    <city>Innsbruck</city>
    <zip>6020</zip>
    <country>Austria</country>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.com/shipping#Address"/>
  <rdf:Description rdf:nodeID="d4e83">
    <rdf:type rdf:resource="http://example.com/shipping#Contact"/>
    <firstName>Jacek</firstName>
    <lastName>Kopecky</lastName>
    <phone>+435125076481</phone>
    <fax>+435125079872</fax>
    <email>jacek.kopecky@deri.org</email>
  </rdf:Description>
  <rdf:Description rdf:nodeID="d4e104">
    <rdf:type rdf:resource="http://example.com/shipping#Package"/>
    <quantity>1</quantity>
    <weight>1</weight>
    <width>5</width>
    <height>5</height>
    <length>5</length>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.com/shipping#Package"/>
  <rdf:Description rdf:nodeID="d4e126">
    <rdf:type rdf:resource="http://example.com/shipping#Package"/>
    <quantity>1</quantity>
    <weight>2</weight>
    <width>6</width>
    <height>6</height>
    <length>6</length>
  </rdf:Description>
</rdf:RDF>
```

The actual lowering stylesheet is shown in the following listing. The stylesheet imports both the flattening RDFXSLT stylesheet and the supporting functions. The first template invokes RDFXSLT flattening and then gives the flattened results to the following template which, in our case, constructs the whole message. In this case it is simpler to construct the whole message in one place as opposed to splitting the template into smaller ones, especially because there is no recursion and very little other reuse in the result tree.

Listing 3.7: The lowering XSLT stylesheet using RDFXSLT, also available as muller.rdfxslt.

```
<xsl:stylesheet version="2.0"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:rx="http://www.wsmo.org/TR/d24/d24.2/v0.1/tmp/rdfxslt"
   xmlns:m="http://www.example.org/muller/"
   xmlns:d="http://example.com/shipping#"
   exclude-result-prefixes="#all" >

<xsl:import href="rdfxslt-functions.xslt"/>
<xsl:import href="rdfxslt-flatten.xslt"/>

<xsl:template match="/">
  <xsl:variable name="rdfxslt">
    <xsl:apply-templates select="/" mode="RDFXSLT"/>
  </xsl:variable>

  <xsl:apply-templates select="$rdfxslt//rdf:Description[rdf:type/@rdf:resource='http://example.com/shipping#Order']"/>
</xsl:template>

<xsl:template match="rdf:Description[rdf:type/@rdf:resource='http://example.com/shipping#Order']">
  <m:shipmentOrderRequest>
    <m:addressFrom>
      <xsl:for-each select="//rdf:Description[rx:references(current()/d:fromAddr, .)]">
        <m:firstname><xsl:value-of select="d:firstName"/></m:firstname>
        <m:lastname><xsl:value-of select="d:lastName"/></m:lastname>
        <m:address><xsl:value-of select="d:street"/>, <xsl:value-of select="d:city"/></m:address>
        <m:location>
          <m:postalCode><xsl:value-of select="d:zip"/></m:postalCode>
          <m:country><xsl:value-of select="d:country"/></m:country>
          <m:state><xsl:value-of select="d:state"/></m:state>
        </m:location>
      </xsl:for-each>
      <xsl:for-each select="//rdf:Description[rx:references(current()/d:fromContact, .)]">
        <m:contactInformation>
          <m:phone><xsl:value-of select="d:phone"/></m:phone>
          <m:EMail><xsl:value-of select="d:email"/></m:EMail>
          <m:fax><xsl:value-of select="d:fax"/></m:fax>
        </m:contactInformation>
      </xsl:for-each>
    </m:addressFrom>
    <m:shipmentDate>
      <m:earliestPickupDate><xsl:value-of select="d:pickupFrom"/></m:earliestPickupDate>
      <m:latestPickupDate><xsl:value-of select="d:pickupTo"/></m:latestPickupDate>
    </m:shipmentDate>
    <xsl:for-each select="//rdf:Description[rx:references(current()/d:package, .)]">
      <m:packageInformation>
        <m:quantity><xsl:value-of select="d:quantity"/></m:quantity>
        <m:weight><xsl:value-of select="d:weight"/></m:weight>
        <m:length><xsl:value-of select="d:length"/></m:length>
        <m:height><xsl:value-of select="d:height"/></m:height>
        <m:width><xsl:value-of select="d:width"/></m:width>
      </m:packageInformation>
    </xsl:for-each>
    <m:addressTo>
      <xsl:for-each select="//rdf:Description[rx:references(current()/d:toAddr, .)]">
        <m:firstname><xsl:value-of select="d:firstName"/></m:firstname>
        <m:lastname><xsl:value-of select="d:lastName"/></m:lastname>
        <m:address><xsl:value-of select="d:street"/>, <xsl:value-of select="d:city"/></m:address>
        <m:location>
          <m:postalCode><xsl:value-of select="d:zip"/></m:postalCode>
          <m:country><xsl:value-of select="d:country"/></m:country>
          <m:state><xsl:value-of select="d:state"/></m:state>
        </m:location>
      </xsl:for-each>
    </m:addressTo>
  </m:shipmentOrderRequest>
</xsl:template>

</xsl:stylesheet>
```

When the lowering stylesheet from Listing 3.7 is applied on the data from Listing 3.5, it effectively outputs the structure shown in Listing 3.3, the intended result.

Once a grounding designer gets used to accessing the RDF data using XPath expressions like "//rdf:Description[rx:references(current()/d:toAddr, .)]", creating lowering grounding XSLT stylesheets using the support of RDFXSLT is very simple, as it does not differ significantly from normal XSLT stylesheets on "well-behaved" XML data. RDFXSLT mostly shields the designer from the horrors of RDF/XML.

4. Attaching the XSLT stylesheets to SWS descriptions

This is a placeholder to be filled in a next version of this deliverable.

todo

5. Conclusions and Future Work

This deliverable describes how XSLT can be used to implement data grounding for RDF-based Semantic Web Service framework implementations. The direction of lifting, i.e. transforming XML data into RDF, poses little difficulty to skilled XSLT designers, whereas the direction of lowering, i.e. transforming RDF data to XML, is normally absolutely horrible due to the flexible nature of RDF/XML. We present RDFXSLT, an XSLT pre-processing stylesheet and a set of helper functions, to make it much easier to transform RDF/XML data using XSLT 2.0.

Frankly, we are afraid that this work reinvents the wheel, but we have not found a useful existing implementation of a similar idea, actually. There are a number of apparently abandoned projects that aim to make it easier to transform RDF data using XSLT; for instance Norman Walsh suggested RDF Twig, whose XSLT extension functions provide various useful views on the sub-trees of an RDF graph; Damian Steer created TreeHugger which makes it possible to navigate the graph structure of RDF using XPath-like expressions, disregarding the actual RDF/XML structure; Morten Frederiksen described rdf2r3x that uses a tool called Redland and an XSLT stylesheet to transform RDF data into something very similar to our Flattened form, but with special allowances for RSS; and Jeremy Carroll and Patrick Stickler take it one step further, putting RDF into a minimalist syntax called TriX, using XSLT as an extensibility mechanism to provide human-friendly macros (syntactic sugar) for this syntax.

All the mentioned approaches share one characteristic: they rely on non-standard extensions or tools, providing implementations in some particular programming language, tied to a particular version of some XSLT processor. The intent of our work is to use only XSLT code, enhancing the portability of the resulting package. RDFXSLT can readily be used on any standards-compliant XSLT 2.0 implementation with no extensions.

Even though our approach may be slower than the alternatives implemented in lower-level languages, we do not expect performance of data grounding to be a problem in SWS frameworks, at least initially. We also see a lot of space for simple and effective optimization, if and when that becomes necessary.

We are highly concerned with the adoption of SWS technologies, therefore RDFXSLT aims to be very familiar for developers already skilled in XML technologies. In this work, we are betting that portability of the approach initially outweighs its prettiness, and we admit that some of the mentioned alternatives, especially TreeHugger, have much prettier XPath expressions.

References

[RDF] Resource Description Framework (RDF) http://www.w3.org/RDF/.

[RDF/XML] D. Beckett, B. McBride (Eds.): *RDF/XML Syntax Specification*, W3C Recommendation 10 February 2004, available at http://www.w3.org/TR/rdf-syntax-grammar/.

[XSLT] J. Clark (editor): *XSL Transformations (XSLT) Version 2.0*, W3C Recommendation, 2007, available at http://www.w3.org/TR/2007/REC-xslt20-20070123

[XPath] A. Berglund et al. (editors): *XML Path Language (XPath) 2.0*, W3C Recommendation, 2007, available at http://www.w3.org/TR/2007/REC-xpath20-20070123/

Acknowledgements

The work is funded by the European Commission under the projects ASG, EASAIER, enIRaF, Knowledge Web, Musing, Salero, Seemp, SemanticGOV, Super, SWING and TripCom; by Science Foundation Ireland under the DERI-Lion Grant No.SFI/02/CE1/I13; by the FIT-IT (Forschung, Innovation, Technologie - Informationstechnologie) under the projects Grisino, RW², SemBiz, SeNSE and TSC.

The editors would like to thank to all the members of the WSMO working group for their advice and input to this document.

$Date: 2007/04/12 02:53:44 $

webmaster