

Gloze: XML to RDF and back again

Steve Battle

HP Labs, Bristol, UK.

http://www.hpl.hp.com/personal/steve_battle

Abstract. Rather than thinking of XML and RDF as competing alternatives, we regard XML and RDF as complementary technologies. This paper discusses bidirectional mapping between XML and RDF, the problem of lift. Rather than inventing a new mapping language, we use information already available in the XML schema. The result is a tool, Gloze that works within the Jena framework. This is illustrated by examples from an application to Model Based Automation.

1 Introduction

XML and RDF are often presented as competing approaches to representation, but this confuses XML the text format (of which RDF/XML is an example) with the underlying data which can be modelled in RDF. XML is the primary data exchange format for Web-Services, by which we mean bespoke XML and associated schema rather than the schema-less RDF/XML. To be able to utilise RDF in a Web-Services setting we need to show that they are not incompatible formats, but are different, though connected views of the same system. The Gloze approach to this problem is to show how the content of this vanilla XML may be modelled in RDF, allowing XML to be mapped into RDF. Furthermore, the approach is non-lossy so that the same RDF model may be mapped back into XML.

Gloze is a set of tools that may be used with HP's Jena RDF framework[1]. The Gloze mapping allows us to directly interpret an XML document as an RDF model – without passing through RDF/XML. This is the so-called problem of *lift*, where different, possibly heterogeneous, data sources are mapped into a common representational framework for ease of processing. One might set about this problem by defining an XSLT that translates XML into RDF/XML. However, this is not an easy task to do by hand and in addition the reverse mapping would require a separate transform. It gets worse in

that the RDF/XML to XML mapping is difficult to construct because for any given model there is no canonical RDF/XML expression; the same triple may be expressed in many different ways in any position in the document. This makes it extremely difficult for XSLT to get a grip on RDF inputⁱ.

Seen as data-structures, a big difference between XML and RDF is that the former defines a tree while the latter defines a graph. A naive translation of XML into RDF has the side effect of losing the sequencing implicit in the XML tree structure. Patel-Schneider et al [2,3] make up for this deficit at a semantic level so that XML may be read directly as RDF. Gloze is more modest, using existing RDF modelling constructs – particularly the much maligned RDF sequence - in conjunction with a defined mapping based on the XML schema.

Gloze uses XML schema as the basis for describing how XML is mapped into RDF and back again. The benefit of this approach is that, unlike a procedural approach like XSLT, XML schema are declarative in that they are neutral with respect to the direction of the mapping. Compatibility with OWL, though not necessary, is desirable. While many features of XML schema can be mapped onto OWL, there are many cases where this can be misleading. XML Schema defines the syntactic structure of the document rather than the underlying model. For example, the (min or max) occurrence of an element describes what can appear at that lexical position and is rather different to an OWL cardinality constraint. Ideally, we would prefer the RDF model to have a simple OWL lite characterization, although this is not entirely straightforward. Gloze also includes an experimental XML schema to OWL mapping (an XSLT) but this is not described in detail here.

The mapping between XML and RDF must address a number of key questions:

- How do we represent elements and attributes?
- What namespaces does a schema define?
- How do we represent identity (ID) and reference (IDREF)?
- How do we preserve XML sequencing information?

ⁱ Unless one is using a tool such as TreeHugger <<http://rdfweb.org/people/damian/treehugger/>> that allows RDF to mimic a DOM.

2 Scenario

We illustrate the way that Gloze answers these questions with examples from a practical application of Gloze to Model Based Automationⁱⁱ. Service Oriented Architectures achieve a high degree of software modularity by defining an abstract service interface - focusing on the value delivered to the client rather than the implementation. Model Based Automation is the application of this idea to software system management, where the client describes their desired configuration (in terms of constraints) and the server responds with a model of the actual system configuration. A typical request might specify a number of logical subnets and server tiers containing virtualised servers and disks.

The semantic-web is a good fit to the high-level modelling requirements of this application. Different components of the model may be supported by different controllers, and these need to be broken apart and put back together again; RDF straightforwardly enables this integration of multiple models. A system controller must verify that the constraints are solvable and may need to capture policies on their solution; RDF provides a modular, logical representation in which these constraints are naturally expressed and solved. Gloze enters the story because RDF/XML is not the most convenient representation for the exchange of models between web-services. RDF/XML offers little in the way of validating message content as required by WS-I [4]. Gloze allows us to use bespoke XML syntax in conjunction with a high-level RDF model.

3 Elements and attributes

The core concept is that every element and attribute maps to an RDF property [5], viewing the XML structure as a relational model between parent nodes and their children. Alternative approaches have been proposed; Melnik [6] in particular has suggested using the element names to classify the content of the element, and only attributes would

ⁱⁱ The example has been adapted from the original to best illustrate the features of Gloze.

be identified as RDF properties. Indeed, this is quite often the intended meaning of the XML. However, Melnik is interested in a generic mapping between XML and RDF whereas we assume that an XML schema is available to guide us by providing additional semantic clues. In other words there is a choice between a more-or-less direct mapping of the DOM into RDF with generic ‘has-child’ relationships between parent and child, or a more informative approach where the relationship is appropriately named. Gloze assumes the latter approach.

Elements and attributes map to either object or data-type properties. The key differentiator is whether the XML schema describes them in terms of complex or simple types. Complex types are the closest that XML schema comes to defining a class. An element whose content is defined by a complex type maps to an object property. An element or attribute whose content is defined by a simple type is defined by a data-type property. This is not defined as such in any given mapping, but we seek to ensure that every (element or attribute) property is used consistently as an object or data-typed property so that they may be provided with a straightforward OWL lite characterization. This problem crops up because XML schema allows the same named property to be used in different contexts with different complex or simple types. Where such a conflict occurs the object-property characterization takes precedence and any simple typed values are *boxed* as a resource with an `rdf:value` (literal). Note also that simple content is also wrapped within its enclosing element using `rdf:value`.

Consider the following XML fragment extracted from a request for a virtual disk exceeding 90Gb of storage. It includes both an element ‘`disk:size`’ and an attribute ‘`greaterThan`’. This element & attribute may be defined by the XML schema fragment underneath, in terms of a complex and a simple type, and it is this that enables us to lift the fragment into RDF.

```
XML input:  
<disk:size greaterThan="90"/>
```

```
XML schema:  
<xs:element name="size">  
  <xs:complexType>  
    <xs:attribute name="greaterThan" type="xs:int"/>  
  </xs:complexType>  
</xs:element>
```

This XML fragment is lifted into RDF (see N3 below), with the ‘size’ (object) property referring to an anonymous resource with a ‘greaterThan’ (data-type) property. Note that the attribute value has acquired type information from the XML schema that was not in the original XML. The declarations of the namespaces will be discussed next.

```
[ ns1:size [
  ns2:greaterThan
  "90"^^<http://www.w3.org/2001/XMLSchema#int>
]].
```

4 Namespaces

For both elements and attributes we can include additional type information recovered from the schema. For named complex types we use this name to identify the `rdf:type` of a resource (anonymous complex types are ignored). For named simple types we construct a typed literal (for anonymous simple types we use plain literals). Also, rather than sticking with the basic set of predefined XML schema data-types we liberally make use of derived data-types even though OWL is currently unable to define them.

The RDF mapping now includes references to elements, attributes and types. Unfortunately, XML schema allows the same name to identify essentially different elements, attributes and types. To ensure that such namespace collisions don’t occur we partition the target namespace in a way that is modularⁱⁱⁱ and doesn’t add to name bloat. The (extended) name for a simple or complex type follows the convention of appending the local name to the namespace (inserting a ‘#’ if required); for example, a type defined as ‘foo’ in a target namespace ‘`http://example.org`’ would be identified as ‘`http://example.org#foo`’. The extended name for an element is distinguished by the use of the ‘/’ character so an element ‘foo’ defined in the same namespace is identified as ‘`http://example.org/foo`’. Finally (also inspired by xpath syntax), an attribute ‘foo’ defined in the same

ⁱⁱⁱ An alternative approach would be to use a conflict resolution strategy only where conflict occurs, however this would be non-modular because the name of an entity could only be determined from a global overview.

namespace is identified as 'http://example.org@foo', distinguishable by the use of the '@'.

We elaborate on our example, adding a named complex type (LogicalSanDiskType) that will be added to a different namespace partition besides our existing elements and attributes. The definition of the 'size' element is as above.

```
XML input:
<disk:logicalSanDisk>
  <disk:size greaterThan="90"/>
</disk:logicalSanDisk>
```

```
XML schema:
<xs:element name="logicalSanDisk"
  type="disk:LogicalSanDiskType"/>
<xs:complexType name="LogicalSanDiskType">
  <xs:sequence>
    <xs:element name="size">...</xs:element>
  </xs:sequence>
</xs:complexType>
```

Now we can define the namespace prefixes that were missing from the previous example, including one for the additional type definition. Note that the complex type information from the schema has been added to the RDF even though it was not present in the original XML.

```
@prefix ns1: <http://example.org/schema/>.
@prefix ns2: <http://example.org/schema@>.
@prefix ns3: <http://example.org/schema#>.

[ ns1:logicalSanDisk [ a ns3:LogicalSanDiskType ;
  ns1:size [
    ns2:greaterThan
    "90"^^<http://www.w3.org/2001/XMLSchema#int> ]
  ] ] .
```

5 Identity and reference

When a document is mapped into RDF it can still be thought of a tree-structure. The root of this tree is identified by the document base. The (single) document element is viewed as a property of this document base. However, XML is not strictly a tree, but a tree with pointers as introduced by IDs and IDREFs. Without IDs, the root, or

document base is the only named node in the tree, with all its descendents being anonymous. IDs introduce named nodes into the tree. Rather than preserving the `xs:ID` data-type in the RDF, we interpret an occurrence as a URI relative to the document base. There really is no option but to reconstruct the full name (document base + identifier) because once the mapping into RDF is complete, the XML document scope is all but lost.

What do ID's identify? Element or attribute IDs uniquely identify their enclosing element, corresponding to the subject of the ID property. Similarly, an IDREF property is cast from the `xs:IDREF` data-type onto a direct URI reference. IDREFs therefore become the only exception where a simple type does not refer to a literal value.

We continue our example, assuming that the service has received the request and allocated a suitable virtual disk. The response will be a model describing the current state of the system. We assume that the system controller has allocated a 100Gb drive identified as 'disk1'. This time we start with the RDF model. The response is communicated back to the client by serializing the RDF model into an XML format as outlined by the XML schema. We assume that the service has fleshed out the model by adding the actual allocated size (and modify the schema accordingly) as determined by the storage allocation policy.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

[ ns1:logicalSanDisk
  <http://example.org/instance#disk1> ].

<http://example.org/instance#disk1>
a ns3:LogicalSanDiskType ;
ns1:size [
  ns2:greaterThan
  "90"^^<http://www.w3.org/2001/XMLSchema#int>;
  rdf:value
  "100"^^<http://www.w3.org/2001/XMLSchema#int>
].
```

We modify the schema to account for the additional simple content (an `xs:int`) in 'size'. We also add the ID attribute to which the URI is assigned. The RDF is serialized to XML using this schema (given a document base of "`http://example.org/instance#`").

```

XML schema:
<xs:complexType name="LogicalSanDiskType">
  <xs:sequence>
    <xs:element name="size">
      <complexType>
        <simpleContent>
          <extension base="xs:int">
            <attribute name="greaterThan"
              type="xs:int"/>
          </extension>
        </simpleContent>
      </complexType>
    </xs:element>
  </xs:sequence>
  <attribute name="id" type="xs:ID"/>
</xs:complexType>

```

```

XML output:
<disk:logicalSanDisk id="disk1">
  <disk:size greaterThan="90">100</disk:size>
</disk:logicalSanDisk>

```

6 Sequencing

A central observation of Gloze is that sequencing is often necessary but is frequently redundant given the schema definition. Sequences only occur within complex types and are described using ‘sequence’, ‘all’ and ‘choice’ compositors. The ‘sequence’ compositor describes a particular admissible sequence of particles. For validation purposes the ‘all’ compositor admits all possible sequences of its particles, though the sequencing may nevertheless remain significant. The ‘choice’ compositor allows a choice of one among several particles, but sequencing again crops up where its particles may occur many times. XML schema also permits document style structures where markup is freely mixed with text, where again it is important to preserve the relative sequencing between markup and text.

There are many approaches one could take to sequencing. RDF lists were considered but because sequences can also have attributes, adding these to the empty list ‘rdf:nil’ is problematic. We adopt a simple solution based on RDF sequences which provide a low overhead solution (with only a few added triples). We must also determine exactly what it is that must be added to the sequence. Consider the fact

that every element maps to an RDF property (similarly, any mixed text values map to `rdf:value` properties). This implies that every member of the sequence is an occurrence (a reification) of an RDF property. In this way, sequencing is overlaid on the basic relational structure of elements and attributes. The advantage of this approach is that because sequencing doesn't disrupt the underlying relational structure, one is free to ignore sequencing if it is irrelevant.

Sequence definitions contain the seeds of their redundancy. In many cases, a sequence will list just those (optional) elements that can be included. In this case, even if we know only the values that should be listed we can fully reconstruct the sequence. If any of the particles could appear multiple times again, we need additional sequencing information to construct the XML.

Characterizing this sequencing in OWL is not easy. In particular, container membership properties fall foul of the need to discriminate between object and data-type properties in OWL DL; we face the same problems whether we select RDF containers or collections. We sidestep this issue by regarding sequencing as a data structuring, rather than an ontological, issue. The ontology captures the underlying relational structure but not the sequencing overlay.

Returning to the response of the previous section, we pan out one more element taking in the definition of the logical server tier which may include more than one virtual disk. The ordering of these may be significant so we represent this in the RDF model. Building on the RDF of the previous section, the N3 below simply says that the statement about the logical disk is the first in this tier.

```
_:a0 a rdf:Seq ; a ns3:logicalServerTier ;
ns1:logicalSanDisk <http://example.org/instance#disk1>;
rdf:_1 [ a rdf:Statement;
  rdf:subject _:a0 ;
  rdf:predicate ns1:logicalSanDisk ;
  rdf:object <http://example.org/instance#disk1>
].
```

The only additional XML schema machinery required is the definition of the logical server tier. The cue to use RDF sequencing is the possibility of an unbounded number of occurrences. We use this to serialize the accumulated RDF model to the XML below.

XML schema:

```
<xs:element name="logicalServerTier">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="disk:logicalSanDisk"
        maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

XML Output:

```
<disk:logicalServerTier>
  <disk:logicalSanDisk>
    <disk:size greaterThan="90">100</disk:size>
  </disk:logicalSanDisk>
</disk:logicalServerTier>
```

7 Conclusions

The Gloze tool explores simple bidirectional translation between XML and RDF based on the XML schema. Such an RDF mapping provides XML with a higher level logical interpretation over and above the Document Object Model. The use of a declarative XML schema provides a mapping that is equally at home in the context of an XML to RDF, or RDF to XML translations. One can argue that the raw XML to RDF translation produces results that are sometimes at odds with the intuitive model (not captured in the XML schema). In such cases, post-processing with rules appear to provide a suitable fix. Once we've mapped the data into the semantic web framework there are any number of tools one can use. The thesis is that there is no need to introduce a new and complex mapping language into the lift phase where existing schema will do. As to the argument, "why not just use XSLT?" there is no reason why the mapping described here cannot be compiled into XSLT. Either way, we still require separate XSLTs for lift and drop, and both can be compiled from the same XML schema.

References

- 1** Brian McBride, Jena: Implementing the RDF Model and Syntax Specification, <<http://www.hpl.hp.com/personal/bwm/papers/20001221-paper/>>.
- 2** Patel-Schneider, P. Simeon, J., Building the Semantic Web on XML, ISCW2002.
- 3** The Yin/Yang Web: A Unified Model for XML Syntax and RDF semantics, IEEE Transactions on Knowledge and Data Engineering, 15(3), July/Aug 2003, pp797-812.
- 4** Web Services Interoperability Organization, <<http://www.ws-i.org/>>
- 5** Trastour, D., Ferdinand, M. Zirpins, C., Pragmatic Reasoning-Support for Web-Engineering: Lifting XML-Schema to OWL, ICWE 2004, Munich, Germany.
- 6** Melnik, S, Bridging the Gap between RDF and XML, <http://www-db.stanford.edu/~melnik/rdf/fusion.html>