Gloze Reference Manual

Generated by Doxygen 1.5.0

Thu Jun 7 17:34:29 2007

Contents

0.1 Gloze Main Page

gloze : to make explanatory notes or glosses on a text

Gloze is a tool for mapping between XML and RDF; describing the *content* of an XML document. It may be used to:

- generate an RDF description of an XML document (using the XML schema)
- generate an XML document from its RDF description (using the XML schema)
- generate an OWL ontology from the XML schema.

Gloze provides an alternative to hand-crafted XSLTs for translating XML into RDF/XML. Furthermore, the Gloze mapping is reversible so that an XML document may be mapped into RDF then back into XML with minimal loss of information. Key to this mapping is the XML schema, which provides additional type and compositional information not available in the source XML.

The concept behind Gloze is to make this mapping as unsurprising as possible, while avoiding the introduction of new vocabulary. Put simply, XML schema types, both simple and complex, map to OWL classes, while elements and their attributes map to properties. Any given *instance* of an XML attribute or element maps to an RDF statement. The content of the element or attribute is the object of the statement, which may be a literal or an RDF resource with its own properties.

The translation of schema into OWL is simplified by the fact that OWL uses the majority of XML schemas predefined data-types. The RDF semantics recommendation identifies a subset of these data-types suitable for use in RDF typed literals. Exceptions include: anySimpleType, duration, ENTITY, ENTITIES, ID, IDREF, IDREFS, NMTOKENS, NOTATION, QName. The remaining data-types are translated as-is. The tree structure of the XML translates almost directly into RDF. Intermediate nodes are bnodes unless they represent the document element (with the document base as its URI) or if they have an xs:ID identifier.

Qualified attributes and elements are declared in the target namespace of the schema, or in the default namespace if unqualified. All types are defined in the target namespace of the schema, or in the case of

a no-namespace schema we must give a default namespace. An element, attribute or type is unqualified if no target namespace is defined, or if it is defined locally and its form is unqualified. Unqualified attributes/elements are not defined in the target namespace, but the translation requires an absolute URI; an additional default namespace must be supplied. In xml schema, attributes and elements have their own symbol spaces, distinct from each other and the types. If there is a namespace clash between these symbol spaces, it is advisable to introduce extra symbolic prefixes, appended to the target namespace, to keep them distinct. We have to be careful to distinguish the name of the type from the identity of the schema component that defines it. If we need to refer to the schema component, to use its additional definitional machinery, we may use its id (relative to the schema base), or a schema component identifier.

The names of locally defined attributes & elements can be recycled in different type definitions (different particles), each time with a different type. From the perspective of XML schema they are different properties, but rather than trying to construct an elaborate and obscure naming scheme that keeps them apart, we take the view that these are different uses of the same property. This means that a property may refer to a data-type in one type, but to an object in another. There are therefore no guarantees that the translation will be in OWL DL; a given property could be both an object and a data-type property. If OWL DL is a desirable outcome then it is up to the schema author to come up with a clean design (union types raise similar design challenges). We can't assume that the schema as a whole prescribes all uses of that name because it can always be included in another unidentified schema that recycles the name again. We assume that globally defined attributes & elements are not recycled in this way (name clashes notwithstanding); and that all occurrences conform to this global declaration. This enables us to derive property ranges directly from global attribute & element declarations.

Where the lexical ordering of the children is significant, this is captured in an RDF sequence of reified statements. This is simply overlaid over the existing tree structure, so you can take it or leave it. Queries to the RDF that aren't interested in ordering can simply ignore it. As for OWL modelling, sequencing is regarded as a data-structuring issue rather than one of ontologogical significance.

Element *groups* and *attribute groups* are treated as syntactic sugar rather than fully fledges classes, so are flattened out of the OWL mapping. As indeed are *sequence*, *choice* and *all* compositors which are only used to calculate the cardinalities of their respective content. They don't add structure.

There are, of course, a few wrinkles caused by mixed and nillable content, and we will see how the non-recommended datatypes map into RDF. These issues are covered in the relevant documentation.

0.2 Gloze Namespace Index

0.2.1 Gloze Package List

Here are the packages with brief descriptions (if available):

0.3 Gloze Class Index

0.3.1 Gloze Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

0.4 Gloze Page Index

0.4.1 Gloze Related Pages

Here is a list of all related documentation pages:

0.5 Gloze Namespace Documentation

0.5.1 Package com.hp.gloze

Classes

- class Gloze

Packages

- package www_w3_org_2001_XMLSchema

0.5.1.1 Detailed Description

Author:

steve.battle@hp.com

0.5.2 Package com.hp.gloze.www_w3_org_2001_XMLSchema

0.5.2.1 Detailed Description

Author:

steve.battle@hp.com

0.6 Gloze Class Documentation

0.6.1 com.hp.gloze.Gloze Class Reference

Public Member Functions

- Gloze (boolean silent)
- Gloze (URL schema, URI targetNS) throws Exception
- Gloze (URL[ ] schema, URI[ ] targets) throws Exception
- void xml_to_rdf (File source, File target, URI base, Model model) throws Exception
- Document rdf_to_xml (File source, File target, URI base, Model model) throws Exception
- OntModel xsd_to_owl (File source, String base) throws Exception
- Model lift (Document xml, URL url, URI base) throws Exception
- boolean lift (Document xml, URL url, URI base, Model model) throws Exception
- Document drop (Model model, URI uri) throws Exception
- Document drop (Resource rez) throws Exception

Static Public Member Functions

- static schema loadSchema (URL url) throws Exception
- static void clearCache ()
- static schema initSchema (URI namespace, URL url, Map< URL, schema > schemaMap) throws Exception
- static schema initSchemaXSI (Element e, URL location, String defaultNS, Map< URL, schema > schemaMap) throws Exception
- static void main (String args[ ])

0.6.1.1 Detailed Description

The programmatic interface for Gloze.

0.6.1.2 Constructor & Destructor Documentation

com.hp.gloze.Gloze.Gloze (boolean *silent*)

Silent constructor doesn't output to console

Parameters:

silent

com.hp.gloze.Gloze.Gloze (URL *schema*, URI *targetNS*) throws Exception

Construct gloze for schema with target namespace.

Parameters:

schema URL
targetNS URI

Exceptions:

Exception

com.hp.gloze.Gloze.Gloze (URL[ ] *schema*, URI[ ] *targets*) throws Exception

Construct gloze with multiple schema and target namespaces

Parameters:

schema array of URLs
targets array of URIs

Exceptions:

Exception

0.6.1.3 Member Function Documentation

static schema com.hp.gloze.Gloze.loadSchema (URL *url*) throws Exception `[static]`

Load schema with URL into a static cache. The schema is not associated with a gloze instance.

Parameters:

url of the source schema

Returns:

schema instance

static void com.hp.gloze.Gloze.clearCache () `[static]`

clear the static cache

static schema com.hp.gloze.Gloze.initSchema (URI *namespace*, URL *url*, Map< URL, schema > *schemaMap*) throws Exception `[static]`

Initialise a gloze instance with a schema given its URL and target namespace.

Parameters:

namespace URI
url of schema

Exceptions:

Exception

static schema com.hp.gloze.Gloze.initSchemaXSI (Element *e*, URL *location*, String *defaultNS*, Map< URL, schema > *schemaMap*) throws Exception `[static]`

Initialise schema from location(s) defined in an xml instance.

Parameters:

e element potentially containing a schemaLocation
location URL of this document.

Exceptions:

Exception

void com.hp.gloze.Gloze.xml_to_rdf (File *source*, File *target*, URI *base*, Model *model*) throws Exception

Map XML to RDF

Parameters:

source document

target file

base of the XML document

model to add the RDF to

Document com.hp.gloze.Gloze.rdf_to_xml (File *source*, File *target*, URI *base*, Model *model*) throws Exception

Drop RDF to XML.

Parameters:

source RDF

target file

base for XML

model containing a prefix map

OntModel com.hp.gloze.Gloze.xsd_to_owl (File *source*, String *base*) throws Exception

lift XML schema to OWL

Parameters:

source schema

base

Model com.hp.gloze.Gloze.lift (Document *xml*, URL *url*, URI *base*) throws Exception

Lift XML data into RDF

Parameters:

xml input document to lift

url location of the input document (required for relative schema location)

base input document URI

Returns:

model

boolean com.hp.gloze.Gloze.lift (Document *xml*, URL *url*, URI *base*, Model *model*) throws Exception

Lift XML data into RDF (using existing model)

Parameters:

xml input document to lift

url location of the input document (required for relative schema location)

base input document base

model output model

Exceptions:

Exception

Document com.hp.gloze.Gloze.drop (Model *model*, URI *uri*) throws Exception

Drop the RDF meta-data into XML.

Parameters:

model containing the RDF meta-data

uri the (named) root of the XML output

Returns:

XML Document

Document com.hp.gloze.Gloze.drop (Resource *rez*) throws Exception

Drop the RDF meta-data into XML.

Parameters:

rez resource representing the root of the XML output

Returns:

XML Document

static void com.hp.gloze.Gloze.main (String *args*[ ]) `[static]`

Gloze main program. The main input parameter is XML to lift or RDF to drop. This is followed by an (optional) sequence of targetNamespace schemaLocation pairs, followed by a single (optional) no-namespace schema location.

0.7 Gloze Page Documentation

0.7.1 License

0.7.2 Getting Started

Gloze may be invoked from the command line using one of the following incantations:

1. `java [-options] com.hp.gloze.Gloze xmlfile (namespaceURI schema-URL)* [nonamespaceschemaURL]`
2. `java [-options] com.hp.gloze.Gloze rdffile (namespaceURI schema-URL)* [nonamespaceschemaURL]`
3. `java [-options] com.hp.gloze.Gloze xsdfile (namespaceURI schema-URL)*`
4. `java [-options] com.hp.gloze.Gloze directory`

In all cases the java classpath must include the Jena libraries (typically under JENA_HOME/lib), gloze.jar, and of course the java runtime.

The first option takes an `xmlfile` and maps it to RDF. To produce a good mapping, Gloze requires the XML schema that describes the XML document. The schema can be referenced explicitly from within the XML using the XML Schema Instance (XSI) schemaLocation or noNamespaceSchemaLocation attributes on the document element. Any schema not already associated with the instance in this way can be added to the command line prefixed by their namespace URI. The final schema may be (optionally) a no-namespace schema in which case no namespace need be supplied. The order of the schema is significant; schema are loaded left to right, so later schema may depend on earlier schema but not vice-versa.

The second form takes an `rdffile` and maps it to XML. The RDF will have no XML schema associated with it so the schema must be explicitly supplied as above. Note that in either case, if the XML schema is not supplied, Gloze will make a best attempt to perform a schema-less mapping.

The third form takes an XML schema `xsdfile` and maps it to OWL, the Web Ontology Language. Any included, imported or redefined schema are recursively mapped.

The final form allows you to lift the contents of an entire directory at once. This may contain both XML and xml schema documents.

By default, the output is written to the console. By supplying a target file or directory (see options below), the output can be saved.

0.7.2.1 Options

`-Dgloze.attribute=SYMBOL`

All attributes, elements and types are assigned a URI combining their local name and the target (or default) namespace. However, XML schema also states that attributes, elements and types define separate symbol spaces such that if their names were the same they would not become confused. While it is good practice to assign unique names to attributes, elements and types, this may be unavoidable when using an existing schema. If this occurs Gloze will warn you of the *potential* URI clash, and the remedy is to insert a special symbolic prefix ahead of attribute or element names to disambiguate them. This option defines a symbolic prefix for attributes, which by default is empty. A recommended attribute prefix is the '@' character.

`-Dgloze.base=URI`

The base URI defines the root of the XML document which by default is the URL of the XML source document. When mapping XML to RDF, the base is the URI of a resource that forms the root of an RDF 'tree'. When mapping from RDF to XML, there must be a resource with this base URI in the RDF model. The base is also required to expand relative URIs appearing as QNames in the XML, and for XML IDs which represent fragment identifiers relative to this base. This option may be used to supply a different base URI; typically an improvement over the 'file:' scheme of most input files. In either case, an explicit xml:base declaration in the document element will take precedence.

When lifting a whole directory in one batch, the bases can be differentiated by supplying a base terminated by a stroke '/'; gloze appends the relevant filename. This will be $*$.xml for a lifted XML file, or $*$.owl for a lifted schema.

`-Dgloze.class=subClassOf|intersectionOf`

OWL classes may be defined either as sub-classes or intersections of other classes. The intersection style is stronger, in that *any* individual consistent with the class description is a member by definition. This style is required for reasoning about schema extensions, but is much more expensive to compute. The default for this option is 'subClassOf'. In this case, extensions are not mapped to sub-class relationships (restrictions are unaffected).

`-Dgloze.closed=true|false`

Where an XML schema uses a 'russian doll' style with each type embedded in the definition of its parent element, and so on, there are no global, named complex types to map into classes. Furthermore, because elements define their types locally, different occurrences of the element may have different types. It is not even correct to take the union of these different types, as the schema may be included in another, where the same element is re-used with additional types. One solution to this problem, which may not be valid in all cases, is to take the closure over the globally defined attributes and elements, using this as their definition. Other local uses of these attributes and elements must be consistent with their global definition. This option is true by default, but may be disabled if it results in an invalid OWL mapping.

`-Dgloze.element=SYMBOL`

Just as attributes may be assigned a prefix to distinguish their symbol space, elements may also be assigned a symbolic prefix. The default is empty. Recommended values include '$\sim$'. Note that types may not be assigned a prefix.

`-Dgloze.fixed=true|false`

Add fixed values when dropping into XML. Fixed values must either be undefined or must match the fixed value declared in the XML schema. When lifting into RDF, the fixed value is always added.

`-Dgloze.lang=N3|RDF/XML|RDF/XML-ABBREV`

This option defines the RDF output format, the default is 'RDF/XML-ABBREV' which is about as pretty as it gets while still using XML. This option applies to both lifted RDF and to OWL. Many people prefer the sleek simplicity of N3, though user beware the lack of xml:base in the generation of ontologies in N3.

`-Dgloze.order=no|seq`

An XML tree is ordered in that the lexical ordering of children is significant. An RDF model is a graph, so a naive mapping will lose this ordering. Gloze can record this additional ordering information by adding the reified statements to an RDF sequence. Gloze will automatically avoid adding this overhead if the ordering can be reconstructed unambiguously from the schema. However, even where the ordering is ambiguous, it may not matter at an application level. In this case, sequencing can be globally disabled.

`-Dgloze.overwrite=true|false`

When true, (the default) Gloze will overwrite existing output files which is required if the source has changed. However, if it is necessary to interrupt a long run with multiple nested inclusions and imports, the user may opt to recycle the earlier output. In this case Gloze can be more or less restarted where it was interrupted.

`-Dgloze.report=true|false`

When working with large schema, it is often hard to find where a particular attribute, element or type is defined. By opting to generate a report, Gloze will list all the generated attribute, element, and type URIs and their sources.

`-Dgloze.roundtrip=true|false`

Used mainly for testing, it is sometimes useful to lift an XML document into RDF and then immediately drop this back into XML so the original and final versions can be compared. By default this is disabled.

`-Dgloze.schemaLocation=URI|dir`

This option allows the schemaLocation to be inserted into the dropped XML.

`-Dgloze.space=default|preserve`

Whitespace processing, modelled after xml:space; using this parameter is equivalent to setting xml:space on the document element. It has two settings, 'default' and 'preserve', with the latter preserving whitespace. The default setting is 'default', performing whitespace collapse and removal. It is not possible to *relax* whitespace processing of datatypes *other* than xs:string and xs:normalizedString which are already fully whitespace replaced and collapsed. The space setting will therefore only effect string types and other mixed text content.

`-Dgloze.target=file`

By default output is written to the console. By defining a target file or directory the output will be saved.

`-Dgloze.trace=true|false`

This option is only useful for low-level debugging of inference. When enabled it produces a trace of rule firings.

`-Dgloze.verbose=true|false`

When enabled, information (disabled by default) and warnings are logged to the console.

`-Dgloze.xmlns=URI`

Unqualified references to schema components are resolved against the default xml namespace. Adding this option is equivalent to defining an xmlns on the document element of the schema. It also provides a substitute target namespace for unqualified components, or more generally for no-namespace schema. The default value for this is the URL of the schema.

0.7.2.2 Examples

The following examples demonstrate a number of Gloze invocations using different combinations of these options. All examples assume the classpath has been initialised to point to the java runtime; gloze.jar; and

the libraries in JENA-HOME/lib.

This example lifts 'example.xml' into RDF using a schema 'schema.xsd' with base "http://example.org/". The output is written to 'example.rdf' and the target is the current directory. The base URI of the XML is "http://example.org/example.xml" and this named resource is the root of the RDF mapping.

```
java -Dgloze.target=.  -Dgloze.base=http://example.org/example.xml
com.hp.gloze.Gloze example.xml http://example.org/ schema.xsd
```

This example lifts example.xml using a pair of schema with namespaces. No base is provided as we assume the instance defines its own xml:base. No target is defined, so the output is written to the console.

```
java com.hp.gloze.Gloze example.xml http://www.example.org/
schema1.xsd http://www.example.com/ schema2.xsd
```

This example lifts example.xml using a no-namespace schema. Additionally, the resulting RDF is ordered.

```
java -Dgloze.order=seq -Dgloze.xmlns=http://example.org/
-Dgloze.base=http://example.org/example.xml com.hp.gloze.Gloze
example.xml schema.xsd
```

The example lifts 'example.xml', but supplies no schema because this is defined in the XML instance using xsi:schemaLocation. The target language is N3, so the output file is 'example.n3' in the current directory. Finally, the RDF is round-tripped back into XML so it may be compared with the XML input.

```
java -Dgloze.target=.  -Dgloze.roundtrip=true -Dgloze.base=http://example.org/example
-Dgloze.lang=N3 com.hp.gloze.Gloze example.xml
```

The example below drops example.rdf into XML, using "http://example.org/example.xml" as the root resource, and the schema 'schema.xsd'.

```
java -Dgloze.base=http://example.org/example.xml com.hp.gloze.Gloze
example.rdf http://www.example.org/ schema.xsd
```

This example lifts the schema 'schema.xsd' into OWL, using xml:base "http://example.org/schema.xsd". The output is written to 'schema.owl' in the current target directory.

```
java -Dgloze.target=.  -Dgloze.base=http://example.org/schema.xsd
com.hp.gloze.Gloze schema.xsd
```

This example lifts the same schema into OWL but uses N3 as the target language. The schema may import or include other schema which are also lifted into OWL. The output is written to the console.

```
java -Dgloze.lang=N3 com.hp.gloze.Gloze schema.xsd
```

The following example lifts a pair of schema into OWL. the first 'schema1.xsd' imports 'schema2.xsd' but the schemalocation is missing, hence the need to supply it as a user defined parameter.

```
java -Dgloze.target=.  -Dgloze.base=http://example.org/schema1.owl
com.hp.gloze.Gloze schema1.xsd http://www.example.com/ schema2.xsd
```

The following invocation is used to generate all the examples used in this documentation. They are contained in a single *examples* directory. Note that few of the examples contain an explicit reference to their schema. In the absence of a schema reference on the command line or in the XML instance, Gloze looks for a schema in the same directory with the same name (with an 'xsd' extsnsion). Because the supplied base is terminated by a stroke '/', the relevant file name is appended for each lifted file.

```
java -Dgloze.xmlns=http://example.org/def/ -Dgloze.base=http://example.org/
-Dgloze.target=examples -Dgloze.lang=N3 -Dgloze.verbose=true
com.hp.gloze.Gloze examples
```

The following invocation lifts example.xml using a combination of schema obtained via a web proxy, and one locally defined schema (xml.xsd - without a DTD). A schemaLocation would be defined within

example.xml

```
java -Dgloze.base=http://example.org/ -Dhttp.proxyHost=myproxy.com
-Dhttp.proxyPort=8080 com.hp.gloze.Gloze example.xml
http://www.w3.org/XML/1998/namespace xml.xsd
```

0.7.3 all

The 'all' compositor allows all combinations of its child elements. The ordering of these elements may be significant. Child elements may be optional (with minimum occurrences of 0) as in the example below with an element 'bar', but a *missing* 'foo' element..

```
<?xml version="1.0"?>
<foobar xmlns="http://example.org/">
        <bar>foobar</bar>
</foobar>
```

```
# Base: http://example.org/all.xml
@prefix ns1:     <http://example.org/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/def/> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .

<>    ns1:foobar
              [ ns1:bar "foobar"^^xs_:string
              ] .
```

This example demonstrates that compositors like 'all' don't explicitly appear in the RDF model nor in the OWL ontology. In the schema below, 'foo' and 'bar' are defined locally so are not included in the (global elements and attributes) closure. They are described as datatype properties with unknown range. The types that appear in the schema are translated as 'allValuesFrom' restrictions in the context of a class definition.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://example.org/" xmlns="http://example.org/"
        elementFormDefault="qualified">

        <xs:element name="foobar">
                <xs:complexType mixed="true">
                        <xs:all>
                                <xs:element name="foo" type="xs:string" minOccurs="0"/>
                                <xs:element name="bar" type="xs:string"/>
                        </xs:all>
                </xs:complexType>
        </xs:element>

</xs:schema>
```

Because 'foo' is optional, it has a maximum cardinality of 1 but an implied minimum cardinality of 0.

```
# Base: http://example.org/all.owl
@prefix ns1:     <http://example.org/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/def/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
```

```
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

<>    a       owl:Ontology .

ns1:bar
      a       owl:DatatypeProperty , rdf:Property .

ns1:foobar
      a       owl:ObjectProperty ;
      rdfs:range
              [ a       owl:Class ;
                rdfs:subClassOf
                        [ a       owl:Restriction ;
                          owl:allValuesFrom xs_:string ;
                          owl:onProperty ns1:foo
                        ] ;
                rdfs:subClassOf
                        [ a       owl:Restriction ;
                          owl:maxCardinality "1"^^xs_:int ;
                          owl:onProperty ns1:foo
                        ] ;
                rdfs:subClassOf
                        [ a       owl:Restriction ;
                          owl:allValuesFrom xs_:string ;
                          owl:onProperty ns1:bar
                        ] ;
                rdfs:subClassOf
                        [ a       owl:Restriction ;
                          owl:cardinality "1"^^xs_:int ;
                          owl:onProperty ns1:bar
                        ]
              ] .

ns1:foo
      a       owl:DatatypeProperty , rdf:Property .
```

0.7.3.1 Child components

- element

0.7.4 annotation

A documentation annotation on a schema, element, attribute or complex type is recorded as rdfs:comment. See documentation below for examples.

0.7.4.1 Child components

- Documentation

0.7.5 any

The use of xs:any makes it necessary to map content for which we have no schema.

The example below includes two marked-up 'parts', and the same content represented as an HTML table.

See also:

http://www.w3.org/MarkUp/Group/

```
<report xmlns="http://example.org/">
  <part>foo</part>
  <part>bar</part>

  <html xmlns="http://www.w3.org/1999/xhtml" xml:id="html">
    <table>
      <tr><td xml:lang="en">foo</td></tr>
      <tr><td>bar</td></tr>
    </table>
  </html>

</report>
```

The HTML content is here represented by an xs:any element from the 'xhtml' namespace, which may include attributes and arbitrarily nested elements.

```
<xs:schema targetNamespace="http://example.org/"
        xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">

        <xs:element name="report">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name="part" type="xs:string" maxOccurs="unbounded"/>
                                <xs:any namespace="http://www.w3.org/1999/xhtml" processContents="skip"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>

</xs:schema>
```

Not having the schema to hand, Gloze regards any content as ambiguously ordered so adds all content to an rdf:Seq. This helps to distinguish attributes from elements; any properties not added to this sequence are therefore attributes. Elements with a single value (and no attributes) are added as literal properties (see 'tr').

The example also includes an xml:id defined to be of type xs:ID, regardless of the schema. This identifies the html resource, and is translated as a fragment identifier relative to the document base.

```
# Base: http://example.org/any.xml
@prefix ns1:     <http://example.org/def/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .

:html
      a       rdf:Seq ;
      rdf:_1  [ a       rdf:Statement ;
                rdf:object _:b1 ;
                rdf:predicate <http://www.w3.org/1999/xhtml#table> ;
                rdf:subject :html
              ] ;
      <http://www.w3.org/1999/xhtml#table>
              _:b1 .

_:b1  a       rdf:Seq ;
      rdf:_1  [ a       rdf:Statement ;
                rdf:object _:b2 ;
```

```
                rdf:predicate <http://www.w3.org/1999/xhtml#tr> ;
                rdf:subject _:b1
              ] ;
      rdf:_2  [ a       rdf:Statement ;
                rdf:object _:b3 ;
                rdf:predicate <http://www.w3.org/1999/xhtml#tr> ;
                rdf:subject _:b1
              ] ;
      <http://www.w3.org/1999/xhtml#tr>
              _:b3 , _:b2 .

_:b3  a       rdf:Seq ;
      rdf:_1  [ a       rdf:Statement ;
                rdf:object "bar" ;
                rdf:predicate <http://www.w3.org/1999/xhtml#td> ;
                rdf:subject _:b3
              ] ;
      <http://www.w3.org/1999/xhtml#td>
              "bar" .

<>    ns2:report
              [ ns2:part "foo"^^xs_:string , "bar"^^xs_:string ;
                <http://www.w3.org/1999/xhtml#html>
                        :html
              ] .

_:b4  rdf:value "foo"@en .

_:b2  a       rdf:Seq ;
      rdf:_1  [ a       rdf:Statement ;
                rdf:object _:b4 ;
                rdf:predicate <http://www.w3.org/1999/xhtml#td> ;
                rdf:subject _:b2
              ] ;
      <http://www.w3.org/1999/xhtml#td>
              _:b4 .
```

If a schema is available for the wild-card element, then it may be referenced from within the XML instance document using an xsi:schemaLocation or xsi:noNamespaceSchemaLocation attribute. This can appear in the document element or on the wild-card element itself.

0.7.6 anyAttribute

Sometimes it is useful to create a slot that would match a range of attributes. One use of this is to allow predefined xml attributes on an element, such as xml:lang; xml:id; xml:space without having to include the XML schema. XML schema is not savvy to these new features, so they must be explicitly added to the schema.

See also:

http://www.w3.org/XML/1998/namespace

The example below includes two marked-up 'parts'. We use xml:id to add an identifier to 'foo', and xml:lang to define the language in which 'bar' is expressed.

```
<report xmlns="http://example.org/">
  <part xml:id="foo">foo</part>
  <part xml:lang="en">bar</part>
</report>
```

The schema for the XML namespace, and hence the xml:id attribute, is not imported, so the interpretation of anyAttribute is non-strict.

```
<xs:schema targetNamespace="http://example.org/"
        xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">

        <xs:element name="report">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name="part" maxOccurs="unbounded">
                                        <xs:complexType>
                                                <xs:simpleContent>
                                                        <xs:extension base="xs:string">
                                                        <xs:anyAttribute namespace="http://www.w3.org/XML/
                                                         processContents="lax" />
                                                        </xs:extension>
                                                </xs:simpleContent>
                                        </xs:complexType>
                                </xs:element>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>

</xs:schema>
```

Note that even without the XML namespace schema to hand, Gloze treats xml:id and xml:lang correctly.

```
# Base: http://example.org/anyAttribute.xml
@prefix ns1:     <http://example.org/def/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .

:foo  rdf:value "foo"^^xs_:string .

<>    ns2:report
                [ ns2:part :foo ;
                  ns2:part
                          [ rdf:value "bar"@en
                          ]
                ] .
```

0.7.7 xml:id

Gloze interprets xml:id attributes as being of type xs:ID, regardless of whether or not a schema definition is available. The schema author must still design the schema so that the xml:id is a valid attribute. This may be done by adding it explicitly as a reference to the (imported) XML namespace schema; as xs:anyAttribute; or within xs:any.

xml:id example

See also:

http://www.w3.org/XML/1998/namespace

0.7.8 attribute

Attributes map to RDF properties. An instance of an attribute maps to an RDF statement. A qualified attribute is defined in the target namespace of the schema. The target namespace is the namespace of the corresponding RDF property.

e.g. The following schema declares a global and therefore qualified attribute 'foo' defined in the target namespace.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://example.org/" xmlns="http://example.org/">
        <xs:attribute name="foo" />
</xs:schema>
```

By default, we take the closure over global attributes, assuming that *all* uses of the property conform to this global declaration. This means that 'foo' globally ranges over rdfs:Literal (the datatype corresponding to xs:anySimpleType). If this assumption is false, attribute closure may be disabled (closed=false), and global ranges are not asserted.

```
# Base: http://example.org/attribute1.owl
@prefix ns1:     <http://example.org/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/def/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

<>    a       owl:Ontology .

ns1:foo
      a       owl:DatatypeProperty , rdf:Property ;
      rdfs:range rdfs:Literal .
```

Like elements, if the target namespace ends with an alpha-numeric a fragment separator '#' is introduced.

Unqualified attributes are not defined in the target namespace. Unqualified attributes occur if no target namespace is defined, or where the attribute is defined locally and its form is unqualified. The example below defines the attribute 'foo' locally within an attributeGroup; it's form is unqualified by default.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://example.org/">
        <xs:attributeGroup name="myGroup">
                <xs:attribute name="foo"/>
        </xs:attributeGroup>
</xs:schema>
```

All we can say about the attribute is that it is a property, for the same name may be re-used where its range is a different datatype or even an object type. Closure doesn't apply to local attributes.

The following OWL was produced with lang=N3. The unqualified attribute is defined in the user-supplied default namespace, with xmlns=http://example.org/def/ .

```
# Base: http://example.org/attribute2.owl
@prefix ns1:     <http://example.org/def/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .
```

```
ns1:foo
      a       owl:DatatypeProperty , rdf:Property .

<>    a       owl:Ontology .
```

In xml schema, attributes have their own symbol space, distinct from other components such as elements and types. If there are overlaps between these symbol spaces, it is advisable to introduce a symbolic prefix to keep them distinct.

e.g. the attribute named 'foo' and type named 'foo' in the target namespace http://example.org/ will clash. Introducing a symbolic prefix '@' (at the command line) for attributes resolves the clash giving us an RDF property name http://example.org/@foo.

The OWL mapping below was generated from the first schema above, but with attribute=@

```
# Base: http://example.org/attribute3.owl
@prefix ns1:     <http://example.org/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/def/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

<http://example.org/@foo>
      a       owl:DatatypeProperty , rdf:Property ;
      rdfs:range rdfs:Literal .

<>    a       owl:Ontology .
```

0.7.8.1 Child components

- simpleType
- annotation

0.7.9 Example: xs:anySimpleType

This example shows how an untyped attribute 'bar' has a default type of xs:anySimpleType. All simple types are derived from this, so it occupies a similar place to rdfs:Literal.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://example.org/" >
        <xs:element name="foo">
                <xs:complexType>
                        <xs:attribute name="bar"/>
                </xs:complexType>
        </xs:element>
</xs:schema>
```

An XMl instance of this schema is as follows:

```
<?xml version="1.0"?>
<foo xmlns="http://example.org/" bar="foobar"/>
```

The value 'foobar' is mapped to a literal of type rdfs:Literal.

```
# Base: http://example.org/attributeAnySimpleType.xml
@prefix ns1:     <http://example.org/def/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .

<>    ns2:foo [ ns1:bar "foobar"
              ] .
```

0.7.10 Example: xs:NOTATION

This example demonstrates the use of notations. These are essentially QNames which must be expanded to give them global scope. The only restriction is that any notation should be predefined in the schema. The example includes a base64Binary embedded image which is notated as belonging to a particular predefined mime type.

```
<?xml version="1.0"?>
<!-- example adapted from http://www.w3.org/TR/2004/WD-xml-media-types-20041102/ -->
<notation xmlns="http://www.iana.org/assignments/media-types/image/"
        mimeType="png">/aWKKapGGyQ=</notation>
```

The xml schema defines the mime options is as follows. The only mime type options available are 'jpeg', 'gif' and 'png'.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.iana.org/assignments/media-types/image/"
        xmlns="http://www.iana.org/assignments/media-types/image/">

        <xs:notation name="jpeg" public="image/jpeg"/>
        <xs:notation name="gif" public="image/gif" />
        <xs:notation name="png" public="image/png" />

        <xs:element name="notation">
                <xs:complexType>
                        <xs:complexContent>
                                <xs:extension base="xs:base64Binary">
                                        <xs:attribute name="mimeType">
                                                <xs:simpleType >
                                                        <xs:restriction base="xs:NOTATION">
                                                                <xs:enumeration value="jpeg"/>
                                                                <xs:enumeration value="gif"/>
                                                                <xs:enumeration value="png"/>
                                                        </xs:restriction>
                                                </xs:simpleType>
                                        </xs:attribute>
                                </xs:extension>
                        </xs:complexContent>
                </xs:complexType>
        </xs:element>

</xs:schema>
```

In the resulting RDF mapping, the xs:NOTATION datatype does not appear, but the expanded QName names the global mime type resource.

```
# Base: http://example.org/attributeNotation.xml
@prefix ns1:     <http://www.iana.org/assignments/media-types/image/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
```

```
@prefix ns2:     <http://example.org/def/> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .

<>    ns1:notation
              [ rdf:value "/aWKKapGGyQ="^^xs_:base64Binary ;
                ns2:mimeType ns1:png
              ] .
```

0.7.10.1 Child components

- annotation
- simpleType
- simpleContent

0.7.11 Attribute Identity

The following example demonstrates the use of xs:ID, xs:IDREF, and xs:IDREFS. The element 'foobar' includes an 'id' attribute that identifies it. The bar element references it via its 'href' or 'hrefs' attributes, of type xs:IDREF or xs:IDREFS, respectively.

```
<?xml version="1.0" encoding="UTF-8"?>
<identity xmlns="http://example.org/" >
        <foobar id="foobar"/>
        <bar idref="foobar"/>
        <bar idrefs="foobar foobar" />
</identity>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://example.org/" xmlns="http://example.org/"
        elementFormDefault="qualified">

        <xs:element name="identity">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name="foobar">
                                        <xs:complexType>
                                                <xs:attribute name="id" type="xs:ID"/>
                                        </xs:complexType>
                                </xs:element>
                                <xs:element name="bar" maxOccurs="unbounded">
                                        <xs:complexType>
                                                <xs:attribute name="idref" type="xs:IDREF"/>
                                        <xs:attribute name="idrefs" type="xs:IDREFS"/>
                                        </xs:complexType>
                                </xs:element>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>

</xs:schema>
```

Observe that no resources of types xs:ID, xs:IDREF, or xs:IDREFS appear in the mapped RDF. These are all translated into resource URIs, and in the case of xs:IDREFS to an RDF:List of URIs.

```
# Base: http://example.org/attributeID.xml
@prefix ns1:     <http://example.org/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/def/> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix :        <#> .

<>    ns1:identity
              [ ns1:bar [ ns2:idref :foobar
                        ] ;
                ns1:bar [ ns2:idrefs (:foobar :foobar)
                        ] ;
                ns1:foobar :foobar
              ] .
```

0.7.12 xml:lang

The appearance of an xml:lang attribute states that the element content is expressed in the given language. The same lang value can be set on an RDF literal but not an RDF datatype. Language settings take precedence over datatyping.

```
<?xml version="1.0" encoding="UTF-8"?>
<lang xmlns="http://example.org/" xml:lang="en">language</lang>
```

This conforms to the following schema. Note that xml:lang is defined in a standard schema obtainable from http://www.w3.org/XML/1998/namespace. The attribute group 'specialAttrs' includes xml:base, xml:lang and xml:space.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://example.org/" xmlns="http://example.org/" xmlns:xs="http://www.w3.org/2
        <xs:import namespace="http://www.w3.org/XML/1998/namespace" schemaLocation="xml.xsd" />
        <xs:element name="lang">
                <xs:complexType>
                        <xs:simpleContent>
                                <xs:extension base="xs:string">
                                        <xs:attributeGroup ref="xml:specialAttrs"/>
                                </xs:extension>
                        </xs:simpleContent>
                </xs:complexType>
        </xs:element>
</xs:schema>
```

The resulting RDF below includes the literal 'language' in the english language.

```
# Base: http://example.org/attributeLang.xml
@prefix ns1:     <http://example.org/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/def/> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .

<>    ns1:lang
              [ rdf:value "language"@en
              ] .
```

0.7.13 attributeGroup

Attribute groups allow the schema designer to collect together common groups of attributes. The example below is based on the XML Linking standard *XLink* which defines groups of attributes that define a number

of link properties including xlink:href and xlink:type that we will see here.

```
<?xml version="1.0" encoding="UTF-8"?>
<link xmlns="http://example.org/" xmlns:xlink="http://www.w3.org/1999/xlink"
        xlink:href="foo.xml" />
```

The xlink:href attribute is of type xs:anyURI, and this is resolved against the document base to provide an absolute URI that retains its meaning in RDF. However, it remains a literal of type xs:anyURI.

```
# Base: http://example.org/link.xml
@prefix xlink_:  <http://www.w3.org/1999/xlink#> .
@prefix ns1:     <http://example.org/def/> .
@prefix xlink:   <http://www.w3.org/1999/xlink> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .

<>    ns2:link
              [ xlink_:href "http://example.org/foo.xml"^^xs_:anyURI ;
                xlink_:type "simple"^^xs_:string
              ] .
```

For brevity only the attributeGroup reference is included here. The simpleLink attribute group also defines a *fixed* xlink:type attribute that has an implied value of 'simple' which is added to the RDF model even though it is not explicit in the XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://example.org/" xmlns:xlink="http://www.w3.org/1999/xlink">
        <xs:import namespace="http://www.w3.org/1999/xlink" schemaLocation="xlink.xsd" />

        <xs:element name="link">
                <xs:complexType>
                        <xs:attributeGroup ref="xlink:simpleLink"/>
                </xs:complexType>
        </xs:element>

</xs:schema>
```

See also:

http://www.w3.org/1999/xlink

0.7.13.1 Child components

- attribute
- attributeGroup
- anyAttribute

0.7.14 choice

The choice compositor determines how property cardinalities are derived. Because only one choice may occur, each has a minimum cardinality of 0. The maximum is set by maxOccurs. These values are multiplied by the cardinalities of nested components to produce the derived cardinailities.

In the following example, the minimum cardinailites of both 'foo' and 'bar' are multipled by 0, so there is no minimum restriction on either property. In addition, 'foo' has no maximum limit so foo is unconstrained in the range of 'foobar'. Property 'bar' retains it's default cardinality of 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://example.org/" xmlns="http://example.org/"
        elementFormDefault="qualified">

        <xs:element name="foobar">
                <xs:complexType>
                        <xs:choice>
                                <xs:element name="foo" maxOccurs="unbounded"/>
                                <xs:element name="bar" />
                        </xs:choice>
                </xs:complexType>
        </xs:element>

</xs:schema>
```

The minimum cardinality of 'foo' is 0∗1. The maximum cardinality of 'foo' is 1∗unbounded. This has the interesting, and perhaps counter-intuitive, effect that no restrictions on 'foo' are defined in the class definition.

The minimum cardinality of 'bar' is 0∗1. The maximum cardinality of 'bar' is 1∗1.

```
# Base: http://example.org/choice.owl
@prefix ns1:     <http://example.org/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/def/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

ns1:bar
      a       rdf:Property .

ns1:foobar
      a       owl:ObjectProperty ;
      rdfs:range
              [ a       owl:Class ;
                rdfs:subClassOf
                        [ a       owl:Restriction ;
                          owl:maxCardinality "1"^^xs_:int ;
                          owl:onProperty ns1:bar
                        ]
              ] .

ns1:foo
      a       rdf:Property .

<>    a       owl:Ontology .
```

0.7.14.1 Child components

- element
- sequence
- choice

- group
- any

0.7.14.2 Child components

- restriction
- extension

0.7.15 complexType

The only complex type predefined in XML schema is anyType, the default type for elements, and a superclass of any user-defined type. The nearest analogue is RDF Resource. User defined complex types correspond to OWL classes which may be anonymous, globally qualified by the target namespace, or locally named and defined in a default namespace. For complex types with mixed or simple content, we model these as restrictions on its RDF value.

Global, named complex types are defined in the target namespace. The following schema defines a complex type, <http://example.org/Foo>.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://example.org/" >
        <xs:complexType name="Foo" />
</xs:schema>
```

```
# Base: http://example.org/complexType1.owl
@prefix ns1:     <http://example.org/def/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

<>    a       owl:Ontology .

ns2:Foo
      a       owl:Class .
```

If the target namespace ends with an alpha-numeric, a fragment separator '#' is introduced.

For locally defined elements we derive value (OWL allValuesFrom) restrictions from their type. A complex type defines a content model (particle) for element content. For global elements the type is set by the property range. The Russian Doll style of schema mirrors the structure of the XML instance document. Short on references to global definitions, we see attributes & elements defined in-situ (locally), where names are easily recycled. We might foresee a problem with different appearances of an element within the particle having different types. However, the Element Declarations Consistent constraint limits elements within the same particle to be of the same type. This permits us to construct an allValuesFrom restriction based only on the first appearance of an element within a particle.

This example also shows the definition of a property range based on an anonymous complex type. It also shows how simple content is captured in the form of an rdf:value. In this case the element value can appear at most once. If the element is empty the value isn't added.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://example.org/" >
        <xs:element name="foo">
                <xs:complexType>
                        <xs:simpleContent>
                                <xs:restriction base="xs:string"/>
                        </xs:simpleContent>
                </xs:complexType>
        </xs:element>
</xs:schema>
```

```
# Base: http://example.org/complexType2.owl
@prefix ns1:     <http://example.org/def/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

rdf:value
      a       owl:DatatypeProperty , rdf:Property .

<>    a       owl:Ontology .

ns2:foo
      a       owl:ObjectProperty ;
      rdfs:range
              [ a       owl:Class ;
                rdfs:subClassOf
                        [ a       owl:Restriction ;
                          owl:cardinality "1"^^xs_:int ;
                          owl:onProperty rdf:value
                        ] ;
                rdfs:subClassOf
                        [ a       owl:Restriction ;
                          owl:allValuesFrom xs_:string ;
                          owl:onProperty rdf:value
                        ]
              ] .
```

Instances of this in both XML and RDF are as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<foo xmlns="http://example.org/">foobar</foo>
```

```
# Base: http://example.org/complexType2.xml
@prefix ns1:     <http://example.org/def/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .

<>    ns2:foo [ rdf:value "foobar"^^xs_:string
              ] .
```

We may add attributes to a complex type either directly as in the example below (or as part of an attribute group), or within nested restrictions or extensions.

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

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://example.org/" >
        <xs:complexType name="Foo">
                <xs:attribute name="bar" type="xs:string" />
        </xs:complexType>
</xs:schema>
```

The attribute 'bar' is unqualified, so the corresponding property is defined in the default namespace (xmlns=http://example.org/def/).

```
# Base: http://example.org/complexType3.owl
@prefix ns1:     <http://example.org/def/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

ns1:bar
      a       owl:DatatypeProperty , rdf:Property .

<>    a       owl:Ontology .

ns2:Foo
      a       owl:Class ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:allValuesFrom xs_:string ;
                owl:onProperty ns1:bar
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:maxCardinality "1"^^xs_:int ;
                owl:onProperty ns1:bar
              ] .
```

We can add structured content to a complex type, using the compositors, 'all', 'choice', 'sequence', and sometimes 'group'. These don't add any nested structure to the class itself but are used to determine the cardinalities of any added properties.

0.7.15.1 Child components

- simpleContent
- complexContent
- group
- all
- choice
- sequence
- attribute
- attributeGroup
- anyAttribute
- annotation

0.7.16 Cardinality Restrictions

By looking at attribute usage and element occurrence we can derive the appropriate OWL cardinality restrictions. Cardinality restrictions are derived from occurrence constraints on particles. For straightforward content there is almost a direct mapping from occurrences to cardinality. Elements occur once by default, and attributes are optional. We can also change minOccurs and maxOccurs on an element and this will be directly reflected in the OWL cardinality constraint. However, this belies a significant difference between occurrence constraints and cardinality restrictions. An occurrence constraint in XML schema refers to the occurrence of content at a given lexical position in the document. Similar content may occur at different positions within the same particle. In contrast, Cardinality restrictions limit the total number of appearances of a property in the context of a given class.

The following example shows how cardinality restrictions are derived from occurrence constraints on complex content. The content of the 'all' compositor and sub-element 'foo' occur once by default. The maximum number of occurrences of 'bar' is unbounded. By default, the use of attribute 'baz' is optional.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://example.org/" xmlns="http:/
        <xs:element name="foo" type="xs:string"/>
        <xs:element name="bar" type="xs:string"/>

        <xs:complexType name="Foobar">
                <xs:all>
                        <xs:element ref="foo"/>
                        <xs:element ref="bar" maxOccurs="unbounded"/>
                </xs:all>
                <xs:attribute name="baz" type="xs:string"/>
        </xs:complexType>

</xs:schema>
```

Cardinality restrictions only constrain the number of occurences of a property in the context of a given class. They don't affect global property definitions. Property 'foo' is restricted to a cardinality of 1, while property 'bar' has a *minimum* cardinality of 1 (the maximum is unlimited). The optional attribute 'baz' has a maximum cardinality of 1 - occurring at most once. Note also, that because 'baz' is a locally defined attribute, it alone has an additional value constraint. The type of 'baz' is local to the class, whereas the ranges of 'foo' and 'bar' are globally defined.

```
# Base: http://example.org/cardinality1.owl
@prefix ns1:     <http://example.org/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/def/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

ns1:Foobar
      a       owl:Class ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:maxCardinality "1"^^xs_:int ;
                owl:onProperty ns2:baz
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:minCardinality "1"^^xs_:int ;
                owl:onProperty ns1:bar
              ] ;
```

```
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:cardinality "1"^^xs_:int ;
                owl:onProperty ns1:foo
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:allValuesFrom xs_:string ;
                owl:onProperty ns2:baz
              ] .

ns1:bar
      a       owl:DatatypeProperty , rdf:Property ;
      rdfs:range xs_:string .

ns1:foo
      a       owl:DatatypeProperty , rdf:Property ;
      rdfs:range xs_:string .

ns2:baz
      a       owl:DatatypeProperty , rdf:Property .

<>    a       owl:Ontology .
```

An occurrence constraint in XML schema refers to the occurrence of content at a particular lexical position in the document. Similar content may occur at a different position *within the same particle*. This is demonstrated in the following example, where a single element 'foo' may be followed (in sequence) by yet another element 'foo'. Because these are both properties of the same resource we calculate the cardinality of 'foo' by summing repeated occurrences.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://example.org/" xmlns="http:/
        <xs:element name="foo" type="xs:string"/>

        <xs:element name="foofoo">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="foo" />
                                <xs:element ref="foo" />
                        </xs:sequence>
                </xs:complexType>
        </xs:element>

</xs:schema>
```

Summing the occurrences of 'foo' makes for a cardinality of 2, right? Wrong. The OWL below has a *maximum* cardinality of 2, but a minimum cardinality of 1.

```
# Base: http://example.org/cardinality2.owl
@prefix ns1:     <http://example.org/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/def/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

<>    a       owl:Ontology .

ns1:foofoo
      a       owl:ObjectProperty ;
      rdfs:range
```

```
              [ a       owl:Class ;
                rdfs:subClassOf
                        [ a       owl:Restriction ;
                          owl:maxCardinality "2"^^xs_:int ;
                          owl:onProperty ns1:foo
                        ] ;
                rdfs:subClassOf
                        [ a       owl:Restriction ;
                          owl:minCardinality "1"^^xs_:int ;
                          owl:onProperty ns1:foo
                        ]
              ] .

ns1:foo
      a       owl:DatatypeProperty , rdf:Property ;
      rdfs:range xs_:string .
```

There is an edge case where there may be two occurrences of 'foo' but only one *distinct* property/value pair. This happens if both occurrences have the same literal value, as demonstrated below. Both occurrences map to the same statement [ns1:foo "foo"^^xs_:string] (with the same subject). Because these are *logical* statements it makes no difference how many times they are asserted, it amounts to saying the same thing twice. The actual cardinality of 'foo' in this case is 1. Because of this, the minimum cardinality of any datatype property will never exceed 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<foofoo xmlns="http://example.org/">
        <foo>foo</foo>
        <foo>foo</foo>
</foofoo>
```

The two elements above amount to saying the same thing twice.

```
# Base: http://example.org/cardinality2.xml
@prefix ns1:     <http://example.org/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/def/> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .

<>    ns1:foofoo
              [ ns1:foo "foo"^^xs_:string
              ] .
```

Does this mean we lose information and can't achieve the reverse mapping back into RDF? If the number of times something was asserted is significant we have to record the sequence in which they occur. The example below shows how each occurrence of the element has been *reified*, making each an objectified resource in its own right, with its own identity. Each reified 'foo' is added to the object (of 'foofoo') as an rdf:Seq in their correct lexical order. With this additional metadata we see that there are two identical statements. This sequencing information is viewed as a data-structuring issue and is not modelled ontologically.

```
# Base: http://example.org/cardinality2a.xml
@prefix ns1:     <http://example.org/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/def/> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .

<>    ns1:foofoo _:b1 .
```

```
_:b1  a       rdf:Seq ;
      rdf:_1  [ a       rdf:Statement ;
                rdf:object "foo"^^xs_:string ;
                rdf:predicate ns1:foo ;
                rdf:subject _:b1
              ] ;
      rdf:_2  [ a       rdf:Statement ;
                rdf:object "foo"^^xs_:string ;
                rdf:predicate ns1:foo ;
                rdf:subject _:b1
              ] ;
      ns1:foo "foo"^^xs_:string .
```

Compositors like sequence, choice, all (and group references) are not directly represented in OWL because they are concerned with the lexical form of a document. However, their effect is to modulate the occurrences of elements within a particle. Like elements the maximum is set by maxOccurs, and the minimum by minOccurs. With nested compositors, the cardinality is calculated by multiplying nested elements by the occurrence constraints on the compositor. By default, the sequence compositor has a minimum and maximum factor of 1. The choice compositor has a maximum factor of 1, but an overriding minimum factor of 0 because all but one of its elements will not occur at all.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://example.org/" xmlns="http://example.org/" xmlns:xs="http://www.w3.org/2
        <xs:element name="foo" type="xs:string"/>
        <xs:element name="bar" type="xs:string"/>
        <!-- minimum cardinality of 0 for each choice -->
        <xs:complexType name="Foobar" mixed="true">
                <xs:sequence maxOccurs="2">
                        <xs:choice maxOccurs="2">
                                <xs:element ref="foo"/>
                                <xs:element ref="bar" maxOccurs="unbounded"/>
                        </xs:choice>
                </xs:sequence>
        </xs:complexType>
</xs:schema>
```

The minimum cardinality of both 'foo' and 'bar' is $1*0*1 = 0$. The maximum cardinality of both 'foo' and 'bar' is $2*2*1 = 4$.

```
# Base: http://example.org/cardinality3.owl
@prefix ns1:     <http://example.org/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/def/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

<>    a       owl:Ontology .

ns1:Foobar
      a       owl:Class ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:maxCardinality "4"^^xs_:int ;
                owl:onProperty ns1:foo
              ] .

ns1:bar
      a       owl:DatatypeProperty , rdf:Property ;
      rdfs:range xs_:string .
```

```
ns1:foo
      a       owl:DatatypeProperty , rdf:Property ;
      rdfs:range xs_:string .
```

Another feature that will affect cardinality is the appearance of an xs:any wild-card. We have to count how many of these could match other elements in the particle and increment their cardinalities accordingly. The schema below includes an element 'foo' followed by a selection of xs:any elements that may, in principle, match yet another occurrence of 'foo'. We count how many of these may match 'foo' and increment its min and max cardinalities. The last two xs:any cannot match 'foo'. The 'foo' element is defined in the target namespace, but '#other' will only match an element in another namespace, while '#local' will only match a local, unqualified element.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://example.org/" xmlns="http://example.org/">

        <xs:element name="foo" type="xs:string"/>

        <!-- element foo has max cardinality 4 -->
        <xs:complexType name="Any">
                <xs:sequence>
                        <xs:element ref="foo" />
                        <xs:any namespace="http://example.org"/>
                        <xs:any namespace="##any"/>
                        <xs:any namespace="##targetNamespace"/>
                        <xs:any namespace="##other"/>
                        <xs:any namespace="##local"/>
                </xs:sequence>
        </xs:complexType>

</xs:schema>
```

As for other properties that may potentially match xs:any, there are no specific restrictions that can be derived from the schema. In an open-world we may pass over them in silence.

```
# Base: http://example.org/cardinality4.owl
@prefix ns1:     <http://example.org/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/def/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

ns1:foo
      a       owl:DatatypeProperty , rdf:Property ;
      rdfs:range xs_:string .

ns1:Any
      a       owl:Class ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:minCardinality "1"^^xs_:int ;
                owl:onProperty ns1:foo
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:maxCardinality "4"^^xs_:int ;
                owl:onProperty ns1:foo
              ] .
```

```
<>    a       owl:Ontology .
```

The final feature relevant to cardinality is the use of substitution groups.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://example.org/" xmlns="http://example.org/" >

        <xs:element name="foo" />
        <xs:element name="bar" substitutionGroup="foo" />
        <xs:element name="foobar" type="Foobar" />

        <xs:complexType name="Foobar">
                <xs:sequence>
                        <xs:element ref="foo" />
                        <xs:element ref="bar" />
                </xs:sequence>
        </xs:complexType>

</xs:schema>
```

The member element 'bar' can substitute for the head element 'foo', so the content may include up to 2 'bar's (we assume there may be other potential sub-properties that could also substitute for 'foo'). Conversely, as 'bar' is a sub-property of 'foo' each 'bar' statement implies a 'foo' statement. There are therefore up to 2 'foo's. Ordinarily, we might conclude that exactly two 'foo's are implied, but there is again the possibility that we might have two identical literal values.

```
# Base: http://example.org/substitution.owl
@prefix ns1:     <http://example.org/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/def/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

ns1:Foobar
      a       owl:Class ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:minCardinality "1"^^xs_:int ;
                owl:onProperty ns1:foo
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:minCardinality "1"^^xs_:int ;
                owl:onProperty ns1:bar
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:maxCardinality "2"^^xs_:int ;
                owl:onProperty ns1:foo
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:maxCardinality "2"^^xs_:int ;
                owl:onProperty ns1:bar
              ] .

<>    a       owl:Ontology .
```

```
ns1:bar
      a       rdf:Property ;
      rdfs:subPropertyOf ns1:foo .

ns1:foobar
      a       owl:ObjectProperty ;
      rdfs:range ns1:Foobar .

ns1:foo
      a       rdf:Property .
```

0.7.17 Documentation

Documentation on a schema, element, attribute or complex type is recorded as rdfs:comment. Documentation for the schema is added to the owl:Ontology header. Documentation for a complex type is added to the corresponding class. Documentation for an element or attribute is added to the corresponding property.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://example.org/" xmlns="http://example.org/"
        elementFormDefault="qualified">

        <xs:annotation>
                <xs:documentation>annotated schema</xs:documentation>
        </xs:annotation>

        <xs:attribute name="attribute" >
                <xs:annotation>
                        <xs:documentation xml:lang="en">annotated attribute</xs:documentation>
                </xs:annotation>
        </xs:attribute>

        <xs:element name="element" type="ComplexType">
                <xs:annotation>
                        <xs:documentation xml:lang="en">annotated element</xs:documentation>
                </xs:annotation>
        </xs:element>

        <xs:complexType name="ComplexType">
                <xs:annotation>
                        <xs:documentation xml:lang="en">annotated type</xs:documentation>
                </xs:annotation>
        </xs:complexType>

</xs:schema>
```

The translation into OWL is as follows:

```
# Base: http://example.org/documentation.owl
@prefix ns1:     <http://example.org/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/def/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

ns1:attribute
      a       owl:DatatypeProperty , rdf:Property ;
      rdfs:comment "annotated attribute"@en ;
      rdfs:range rdfs:Literal .
```

```
ns1:element
      a       owl:ObjectProperty ;
      rdfs:comment "annotated element"@en ;
      rdfs:range ns1:ComplexType .

<>    a       owl:Ontology ;
      rdfs:comment "annotated schema" .

ns1:ComplexType
      a       owl:Class ;
      rdfs:comment "annotated type"@en .
```

0.7.18 element

Elements map to RDF properties. An instance of an element maps to an RDF statement. A qualified element is defined in the target namespace of the schema, as is the corresponding RDF property. Where an element has a complex type we define a corresponding OWL ObjectProperty. Where an attribute or element has an RDF recommended simple type we define a corresponding OWL DatatypeProperty. Attribute and elements of non-recommended simple types form a grey zone with literal types (anySimpleType and ENTITY) still requiring a DatatypeProperty, but the URI (QName, NOTATION, IDREF) and structured types (duration, ENTITIES, IDREFS, NMTOKENS) switch; demanding an ObjectProperty. Where content has a named type (or if there is an explicit xsi:type) this is represented by an rdf:type statement on the node.

e.g. for target namespace http://example.org/ and element named 'foo', the RDF property name is http://example.org/foo

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://example.org/">
        <xs:element name="foo" type="xs:string" />
</xs:schema>
```

The OWL mapping was produced with lang=N3. This shows the property definition for 'foo'. It is a datatype property ranging over xs:string.

```
# Base: http://example.org/element1.owl
@prefix ns1:     <http://example.org/def/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

<>    a       owl:Ontology .

ns2:foo
      a       owl:DatatypeProperty , rdf:Property ;
      rdfs:range xs_:string .
```

Where the target namespace ends with an alphanumeric character, a fragment separator is introduced. e.g. for target namespace http://example.org and element named 'foo', the RDF property name is http://example.org#foo

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://example.org">
        <xs:element name="foo" type="xs:string" />
</xs:schema>
```

This OWL was produced with lang=N3

```
# Base: http://example.org/element2.owl
@prefix ns1:     <http://example.org/def/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org#> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

<>    a       owl:Ontology .

ns2:foo
      a       owl:DatatypeProperty , rdf:Property ;
      rdfs:range xs_:string .
```

Unqualified elements are not defined in the target namespace, but the mapping to RDF requires an absolute URI. Unqualified elements occur if no target namespace is defined, or if an element is declared locally and its form is unqualified. The properties corresponding to unqualified elements will be declared in the user-defined default namespace (xmlns) of the schema. This is a command line parameter (not the xmlns defined on the document element).

e.g. for default namespace http://example.com/def/ and unqualified element named 'foo', the RDF property name is http://example.org/def/foo

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:element name="foo" type="xs:string" />
</xs:schema>
```

The OWL mapping was produced with lang=N3 and xmlns=http://example.com/def/

```
# Base: http://example.org/element3.owl
@prefix ns1:     <http://example.org/def/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

ns1:foo
      a       owl:DatatypeProperty , rdf:Property ;
      rdfs:range xs_:string .

<>    a       owl:Ontology .
```

In xml schema, elements have their own symbol space, distinct from other components such as attributes and types. If there are overlaps between these symbol spaces, it is advisable to introduce a symbolic prefix to keep them distinct.

e.g. the element named 'foo' and type named 'foo' in the target namespace http://example.org/ will clash. Introducing a symbolic prefix '∼' (at the command line) for elements resolves the clash giving us an RDF property name http://example.org/∼foo.

The OWL mapping below was generated from the first schema above, but with element=∼

```
# Base: http://example.org/element4.owl
@prefix ns1:     <http://example.org/def/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

<http://example.org/~foo>
      a       owl:DatatypeProperty , rdf:Property ;
      rdfs:range xs_:string .

<>    a       owl:Ontology .
```

0.7.18.1 Type

An element may have a simple or complex type. On the whole, simple typed elements map to OWL datatype properties (exceptions include most of the datatypes that don't have a clean mapping into RDF); while complex typed elements map to object properties.

The schema below defines a simple typed property 'foo' and a complex type property 'bar'.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://example.org/" xmlns="http://example.org/">

        <xs:element name="foo" type="xs:string" />

        <xs:element name="bar">
                <xs:complexType mixed="true" />
        </xs:element>

</xs:schema>
```

With a corresponding OWL mapping:

```
# Base: http://example.org/element5.owl
@prefix ns1:     <http://example.org/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/def/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

ns1:bar
      a       owl:ObjectProperty .

ns1:foo
      a       owl:DatatypeProperty , rdf:Property ;
      rdfs:range xs_:string .

<>    a       owl:Ontology .
```

An element may reference or directly include a simple type declaration.

0.7.18.2 Child components

- simpleType
- complexType
- annotation

0.7.19 Datatype example xs:string

The 'string' element contains an xs:string "foobar" as its content.

```
<?xml version="1.0" encoding="UTF-8"?>
<string xmlns="http://example.org/">foobar</string>
```

This is defined in its schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://example.org/">
        <xs:element name="string" type="xs:string" />
</xs:schema>
```

The N3 translation declares a namespace prefix 'xs_' that allows the full URI <http://www.w3.org/2001/XMLSchema#string> to be abbreviated to xs_:string. Note that the RDF mapping preserves namespaces declared in the XML (so they can be recovered in the reverse mapping), the namespace 'xs' defined as <http://www.w3.org/2001/XMLSchema> is already taken, so Gloze defines an *extended* version with a trailing '#' and adds the underscore to the name. The XML base is http://example.org/

```
# Base: http://example.org/elementString.xml
@prefix ns1:     <http://example.org/def/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .

<>    ns2:string "foobar"^^xs_:string .
```

0.7.20 Example xs:ID, xs:IDREF

The xs:ID and xs:IDREF types have document scope so are not recommended for use in RDF. Take a fragment of XML with an attribute 'id' of type xs:ID. Instead of adding the 'id' attribute as a property of a resource, we use it to derive the URI of that resource. Given a base http://example.org/base the RDF mapping below includes a statement with property 'foo' and object http://example.org/base#foobar. The 'bar' element is an IDREF, and the object of this statement is the resource identified as 'foobar'.

```
<?xml version="1.0" encoding="UTF-8"?>
<identity xmlns="http://example.org/">
        <foo id="foobar" />
        <bar>foobar</bar>
</identity>
```

The xml schema for this is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://example.org/" elementFormDefault="qualified">

        <xs:element name="identity">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name="foo">
                                        <xs:complexType>
                                                <xs:simpleContent>
                                                        <xs:extension base="xs:anySimpleType">
                                                                <xs:attribute name="id" type="xs:ID" />
                                                        </xs:extension>
                                                </xs:simpleContent>
                                        </xs:complexType>
                                </xs:element>
                                <xs:element name="bar" type="xs:IDREF" />
                        </xs:sequence>
                </xs:complexType>
        </xs:element>

</xs:schema>
```

This XML maps to the following RDF. Note how the 'id' attribute has been dropped in the RDF.

```
# Base: http://example.org/elementIdentity.xml
@prefix ns1:     <http://example.org/def/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix :        <#> .

<>    ns2:identity
              [ ns2:bar :foobar ;
                ns2:foo :foobar
              ] .
```

0.7.21 Example xs:duration

The duration "P2M26DT14H18M" is split into two separate values "P2M" and "P26DT14H18M". Notice the missing year and seconds components, any component is optional. Whereas the value space of duration is partially ordered, the spaces of these year/month and day/time types are totally ordered.

```
<?xml version="1.0" encoding="UTF-8"?>
<duration xmlns="http://example.org/">P2M26DT14H18M</duration>
```

The xml schema for this is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://example.org/">
        <xs:element name="duration" type="xs:duration" />
</xs:schema>
```

The object of the 'duration' statement is now a bnode with a pair of rdf:values, representing the two components of the original duration. These values may be distinguished by their type. The base is http://example.org/base.

```
# Base: http://example.org/elementDuration.xml
@prefix ns1:     <http://example.org/def/> .
```

```
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .

<>    ns2:duration
              [ rdf:value "P26DT14H18M"^^xs_:dayTimeDuration , "P2M"^^xs_:yearMonthDuration
              ] .
```

Mapping back to XML, we make use of the fact that these year/month and day/time types are sub-classes of duration. Durations are added component-wise, so we needn't be concerned with (indeterminate) carry from days to months.

0.7.22 Example xs:ENTITY

Entities allow common blocks of text to be substituted in place, reducing duplication and errors. Entities are not just any plain text, but may also contain *balanced* markup. This fits the bill of the rdf:XMLLiteral datatype. Entity references are usually identified by surrounding them with '&' and ';', allowing the XML parser to expand them, but this is not required for the schema entity type.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://example.org/">
        <xs:element name="entity" type="xs:ENTITY" />
</xs:schema>
```

An XML instance that defines and uses an entity 'eg' is as follows:

```
<?xml version="1.0"?>
<!DOCTYPE example [
        <!ELEMENT entity (#PCDATA)>
        <!ATTLIST entity xmlns CDATA #IMPLIED xmlns:xsi CDATA #IMPLIED xsi:schemaLocation CDATA #IMPLIED>
        <!ENTITY eg "http://example.com/">
]>
<entity xmlns="http://example.org/">eg</entity>
```

The rdf:XMLLiteral type is used in conjunction with the 'Literal' parseType of the rdf/xml serialization. The base is http://example.org/base.

```
<?xml version="1.0" encoding="windows-1252"?>
<rdf:RDF
    xmlns:ns1="http://example.org/def/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:ns2="http://example.org/"
    xmlns:xs_="http://www.w3.org/2001/XMLSchema#"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <rdf:Description rdf:about="">
    <ns2:entity rdf:parseType="Literal">http://example.com/</ns2:entity>
  </rdf:Description>
</rdf:RDF>
```

0.7.23 Example xs:QName

This example defines a qname 'eg:foobar', with a prefix 'eg' defined as http://example.com#.

```
<?xml version="1.0" encoding="UTF-8"?>
<qname xmlns="http://example.org/" xmlns:eg="http://example.com#">eg:foobar</qname>
```

The corresponding xml schema is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://example.org/" >
        <xs:element name="qname" type="xs:QName" />
</xs:schema>
```

The resulting RDF shows that the offending QName type has been dropped, and the expanded URI is a resource name.

```
# Base: http://example.org/elementQName.xml
@prefix ns1:     <http://example.org/def/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix eg:      <http://example.com#> .

<>    ns2:qname eg:foobar .
```

0.7.24 Example xs:IDREFS

In this example we define an element 'foo' whose content type is an xs:ID so the foo element itself is implicated in the naming. The 'bar' element is of type xs:IDREFS; a list of IDREFs. The IDREF type is not used directly as a datatype (because this has document scope), but becomes a global resource URI.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://example.org/" elementFormDefault="qualified">

        <xs:element name="list">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name="foo" type="xs:ID" />
                                <xs:element name="bar" type="xs:IDREFS" />
                        </xs:sequence>
                </xs:complexType>
        </xs:element>

</xs:schema>
```

The 'bar' element refers twice to the same ID.

```
<?xml version="1.0" encoding="UTF-8"?>
<list xmlns="http://example.org/">
        <foo>foobar</foo>
        <bar>foobar foobar</bar>
</list>
```

In N3 the content of an RDF list is contained in brackets.

```
# Base: http://example.org/elementIDREFS.xml
@prefix ns1:     <http://example.org/def/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
```

```
<>    ns2:list
              [ ns2:bar (:foobar :foobar) ;
                ns2:foo :foobar
              ] .
```

0.7.25 Element Identity

This example includes an element 'foobar' with xs:ID content. This identifies the immediately containing element , 'foobar' itself.

```
<?xml version="1.0" encoding="UTF-8"?>
<identity xmlns="http://example.org/">
        <foo>foobar</foo>
        <bar>foobar</bar>
</identity>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://example.org/" elementFormDefault="qualified">

        <xs:element name="identity">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name="foo" type="xs:ID" />
                                <xs:element name="bar" type="xs:IDREF" />
                        </xs:sequence>
                </xs:complexType>
        </xs:element>

</xs:schema>
```

```
# Base: http://example.org/elementIdentity1.xml
@prefix ns1:     <http://example.org/def/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .

<>    ns2:identity
              [ ns2:bar :foobar ;
                ns2:foo :foobar
              ] .
```

0.7.26 xsi:nil

XML differentiates between empty and null data. Such *nillable* elements can be described in the XML schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://example.org/">
        <xs:element name="foobar" type="xs:String" nillable="true"/>
</xs:schema>
```

An example of a nil element is shown in the example below. A null value is represented in RDF using rdf:nil, the empty list.

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

```
<foobar xmlns="http://example.org/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:nil="true"/>
```

```
# Base: http://example.org/nil.xml
@prefix ns1:     <http://example.org/def/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .

<>    ns2:foobar () .
```

Compare this with the same element simply left empty.

```
<?xml version="1.0" encoding="UTF-8"?>
<foobar xmlns="http://example.org/" />
```

```
# Base: http://example.org/nil1.xml
@prefix ns1:     <http://example.org/def/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .

<>    ns2:foobar "" .
```

0.7.27 SubProperty relationships (substitution groups)

There is a hierarchy among elements defined by substitution groups. These substitution groups define sub-property relationships between properties. A substitution group is defined by a head element, and member elements that substitute for the head. The property corresponding to the member is a sub-property of that corresponding to the head.

In the example below, the *head* element 'foo' defines the substitution group, of which element 'bar' is a member. This means that in the XML instance, 'bar' may be subsituted for 'foo'. Logically, any RDF statement of 'bar' implies a corresponding statement of 'foo'.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://example.org/" xmlns="http://example.org/" >

        <xs:element name="foo" type="xs:string" />
        <xs:element name="bar" substitutionGroup="foo" type="xs:string" />

</xs:schema>
```

```
# Base: http://example.org/substitution1.owl
@prefix ns1:     <http://example.org/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/def/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .
```

```
ns1:bar
      a       owl:DatatypeProperty , rdf:Property ;
      rdfs:range xs_:string ;
      rdfs:subPropertyOf ns1:foo .

ns1:foo
      a       owl:DatatypeProperty , rdf:Property ;
      rdfs:range xs_:string .

<>    a       owl:Ontology .
```

0.7.28 enumeration

XML schema also allows a simple type restriction to be defined by enumeration. OWL Enumerations allow classes to be defined extensionally, in terms of their membership. For data-types, the class of values is specified as an OWL DataRange, with OWL oneOf listing the permitted range of values.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://example.org/" xmlns="http://example.org/"
        elementFormDefault="qualified">

        <xs:element name="data">
                <xs:simpleType>
                        <xs:restriction base="xs:int">
                                <xs:enumeration value="0"/>
                                <xs:enumeration value="1"/>
                        </xs:restriction>
                </xs:simpleType>
        </xs:element>

        <xs:element name="objects">
                <xs:simpleType>
                        <xs:restriction base="xs:QName">
                                <xs:enumeration value="foo"/>
                                <xs:enumeration value="bar"/>
                        </xs:restriction>
                </xs:simpleType>
        </xs:element>

</xs:schema>
```

Because xs:QNames are not recommended for use in RDF (they depend on locally defined prefixes) they are fully expanded to form absolute URIs. The QNames that appear in the schema are unprefixed and so are defined in the default XML namespace.

```
# Base: http://example.org/enumeration.owl
@prefix ns1:     <http://example.org/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/def/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

ns1:objects
      a       owl:ObjectProperty , rdf:Property ;
      rdfs:range
              [ a       owl:Class ;
                owl:oneOf (ns1:foo ns1:bar)
```

```
              ] .

ns1:data
      a       owl:DatatypeProperty , rdf:Property ;
      rdfs:range
              [ a       owl:DataRange ;
                owl:oneOf ("0"^^xs_:int "1"^^xs_:int)
              ] .

<>    a       owl:Ontology .
```

0.7.29 extension

The term 'extension' as used in XML schema is not used in its formal, mathematical sense. Rather than defining a superset of valid instances, an extension describes a different, possibly intersecting set of instances that structurally extend the base type. In many cases an extension will describe a valid sub-class, but there are cases where this does not hold. In the valid cases we assert a subClassOf relationship. If we use extensions simply to add new elements and attributes to an existing type we have nothing to worry about. The problems arise by adding additional occurrences of existing elements.

- Extension of Complex Content

0.7.30 Extension of Complex Content

Rather than defining a superset of valid instances an extension describes a different, possibly intersecting set of instances that structurally extend the base type. In many cases the extension will describe a valid subclass of the parent, and in these cases Gloze will assert a subClass relationship.

The schema below defines a class 'Foo' with a single element 'foo', and two extensions of it. If we use extensions simply to add new elements and attributes we have nothing to worry about. The first type 'FooBar' adds a new element 'bar'. The latter type 'FooFoo' extends the base type by adding another occurrence of 'foo'.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://example.org/" xmlns="http://example.org/"
        elementFormDefault="qualified">

        <xs:element name="foo">
                <xs:complexType />
        </xs:element>

        <xs:element name="bar">
                <xs:complexType />
        </xs:element>

        <xs:complexType name="Foo">
                <xs:sequence>
                        <xs:element ref="foo"/>
                </xs:sequence>
        </xs:complexType>

        <xs:complexType name="FooBar">
                <xs:complexContent>
                        <xs:extension base="Foo">
                                <xs:sequence>
                                        <xs:element ref="bar"/>
                                </xs:sequence>
                        </xs:extension>
```

```
                </xs:complexContent>
        </xs:complexType>

        <xs:complexType name="FooFoo">
                <xs:complexContent>
                        <xs:extension base="Foo">
                                <xs:sequence>
                                        <xs:element ref="foo"/>
                                </xs:sequence>
                        </xs:extension>
                </xs:complexContent>
        </xs:complexType>

</xs:schema>
```

The first extension 'FooBar' results in a valid subClass relationship, with each element having cardinality 1. The latter class 'FooFoo' is not a valid subClass, the cardinality of foo is 2, so the minimum cardinality (2) from the child is greater than the maximum cardinality (1) inherited from the parent. Because this interval is empty, the class 'FooFoo' is unsatisfiable.

```
# Base: http://example.org/extension.owl
@prefix ns1:     <http://example.org/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/def/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

<>    a       owl:Ontology .

ns1:FooBar
      a       owl:Class ;
      rdfs:subClassOf ns1:Foo ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:cardinality "1"^^xs_:int ;
                owl:onProperty ns1:bar
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:cardinality "1"^^xs_:int ;
                owl:onProperty ns1:foo
              ] .

ns1:bar
      a       owl:ObjectProperty , rdf:Property .

ns1:foo
      a       owl:ObjectProperty , rdf:Property .

ns1:FooFoo
      a       owl:Class ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:cardinality "2"^^xs_:int ;
                owl:onProperty ns1:foo
              ] .

ns1:Foo
      a       owl:Class ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:cardinality "1"^^xs_:int ;
```

```
    owl:onProperty ns1:foo
  ] .
```

Invalid subClass relationships are detected logically, by asserting the hypothetical relationship and seeing if it results in an inconsistency. If it does - the subclass relationship is retracted. In the next section we will see why schema that use extension in particular ways must be defined in terms of their necessary and sufficient conditions.

0.7.30.1 When extensions go bad

Unfortunately, a minor change to this example turns this violation into something far nastier. In the example below, we change the type of element 'foo' from an object to a datatype property (an xs:string). A valid instance of 'FooFoo' will include two occurrences of 'foo' with *identical* values. When this is mapped into RDF, these count as a single logical statement, so a valid instance may have a cardinality for 'foo' of 1. Of course we still have the typical case where each occurrence has a different value, with a cardinality of foo of 2. So the cardinality of 'foo' in 'FooFoo' lies in the interval [1,2]. The cardinality of 'foo' in class 'Foo' is 1, as before.

The previous line of reasoning now fails, as the minimum cardinality of the child (1) no longer crosses the maximum cardinality of the parent (1). Nor is it valid to argue that the maximum cardinality of the child (2) is greater than the maximum cardinality of the child (1), in effect widening the definition. The child is defined only in terms of its necessary conditions, and taking the intersection with its parent we still end up with a satisfiable class with a cardinality for 'foo' of 1.

It turns out that 'FooFoo' is a valid subclass of 'Foo' only because of those errant instances where 'foo' has identical values. It's unlikely that this is the meaning of 'FooFoo' intended by the schema author, especially as this particular implication remains implicit. One solution to this problem is to define classes in terms of their necessary and *sufficient* conditions, meaning that any instance satisfying the conditions is a member by definition. We express necessary and sufficient conditions in OWL by defining classes as an intersectionOf a set of restrictions.

We reason *semantically* as follows. Because to be a member of 'FooFoo' it is sufficient to have a cardinality of 'foo' of no more than 2, there must be a valid instance of 'FooFoo' with exactly 2 'foo' properties. This individual can't be a model of 'Foo' because this has a maximum cardinality for 'foo' of 1. If 'FooFoo' were a subClass of 'Foo' then every model of 'FooFoo' must also be a model of 'Foo'.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://example.org/" xmlns="http://example.org/"
        elementFormDefault="qualified">

        <xs:element name="foo" type="xs:string" />

        <xs:complexType name="Foo">
                <xs:sequence>
                        <xs:element ref="foo" />
                </xs:sequence>
        </xs:complexType>

        <xs:complexType name="FooFoo">
                <xs:complexContent>
                        <xs:extension base="Foo">
                                <xs:sequence>
                                        <xs:element ref="foo" />
                                </xs:sequence>
                        </xs:extension>
                </xs:complexContent>
        </xs:complexType>

</xs:schema>
```

The gloze mapping is invoked with class=intersectionOf. The logical inconsistency is detected and the subClass relationship between 'FooFoo' and 'Foo' is correctly retracted.

```
# Base: http://example.org/extension1.owl
@prefix ns1:     <http://example.org/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/def/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

<>    a       owl:Ontology .

ns1:foo
      a       owl:DatatypeProperty , rdf:Property ;
      rdfs:range xs_:string .

ns1:FooFoo
      a       owl:Class ;
      owl:intersectionOf ([ a       owl:Restriction ;
                  owl:minCardinality "1"^^xs_:int ;
                  owl:onProperty ns1:foo
                ] [ a       owl:Restriction ;
                  owl:maxCardinality "2"^^xs_:int ;
                  owl:onProperty ns1:foo
                ]) .

ns1:Foo
      a       owl:Class ;
      owl:intersectionOf ([ a       owl:Restriction ;
                  owl:cardinality "1"^^xs_:int ;
                  owl:onProperty ns1:foo
                ]) .
```

0.7.31 group

The group component allows groups of elements to be combined into reusable groups. From a modelling perspective we regard them as syntactic sugar with no counterpart in OWL. However, group references act like compositors in that they allow the schema designer to indicate how many times the group may occur.

The minimum and maximum occurrences of the group reference are multiplied by the cardinalities of the group members. In the example below, the element 'barfoo' references a group that has no occurrences.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://example.org/" xmlns="http://example.org/"
        elementFormDefault="qualified">

        <xs:group name="myGroup">
                <xs:all>
                        <xs:element name="foo" />
                </xs:all>
        </xs:group>

        <xs:element name="barfoo">
                <xs:complexType>
                        <xs:group ref="myGroup" minOccurs="0" maxOccurs="0" />
                </xs:complexType>
        </xs:element>

</xs:schema>
```

```
# Base: http://example.org/group.owl
@prefix ns1:     <http://example.org/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/def/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

<>    a       owl:Ontology .

ns1:foo
      a       rdf:Property .

ns1:barfoo
      a       owl:ObjectProperty ;
      rdfs:range
              [ a       owl:Class ;
                rdfs:subClassOf
                        [ a       owl:Restriction ;
                          owl:cardinality "0"^^xs_:int ;
                          owl:onProperty ns1:foo
                        ]
              ] .
```

0.7.31.1 Child components

- sequence
- choice
- all

0.7.32 Import

If a schema imports a schema then that schema is automatically loaded. When lifting a schema to OWL, a corresponding owl:imports is generated. The URI used in the import is base relative.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://example.org/">
        <xs:import namespace="http://example.com/" schemaLocation="example.xsd" />
</xs:schema>
```

```
# Base: http://example.org/import.owl
@prefix ns1:     <http://example.org/def/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

<>    a       owl:Ontology ;
      owl:imports <http://example.org/example.owl> .
```

0.7.33 Include

If a schema includes a schema then that schema is automatically loaded. When lifting a schema to OWL, a corresponding owl:imports is generated. The URI used in the import is base relative.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://example.org/" >
        <xs:include schemaLocation="example.xsd" />
</xs:schema>
```

```
# Base: http://example.org/include.owl
@prefix ns1:     <http://example.org/def/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

<>    a       owl:Ontology ;
      owl:imports <http://example.org/example.owl> .
```

0.7.34 list

An xs:list is mapped into an rdf:List. Lists assume whitespace separated content in the XML instance. In the case of lists of xs:string, the string value is separated into separate string tokens.

```
<?xml version="1.0" encoding="UTF-8"?>
<list xmlns="http://example.org/">
        foo
        bar
</list>
```

```
# Base: http://example.org/list.xml
@prefix ns1:     <http://example.org/def/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .

<>    ns2:list ("foo"^^xs_:string "bar"^^xs_:string) .
```

The mapping into OWL does not currently define lists for specific datatypes.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://example.org/" >

        <xs:element name="list">
                <xs:simpleType>
                        <xs:list itemType="xs:string"/>
                </xs:simpleType>
        </xs:element>

</xs:schema>
```

```
# Base: http://example.org/list.owl
```

```
@prefix ns1:     <http://example.org/def/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

ns2:list
      a       owl:ObjectProperty , rdf:Property ;
      rdfs:range rdf:List .

<>    a       owl:Ontology .
```

0.7.34.1 Child components

- annotation
- simpleType

0.7.35 redefine

This component is similar to include but also allows types and groups to be completely redefined. There is no comparable feature in OWL, so we simply include the new definitions as they appear in the redefinition. Because we can't selectively import things that are not redefined, anything unaffected by the redefinition is defined directly within the redefining ontology, rather than being imported. In effect, the redefinitions shadow only the affected schema components.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://example.org/" xmlns="http://example.org/"
        elementFormDefault="qualified">

        <xs:redefine schemaLocation="redefined.xsd">
                <xs:simpleType name="Foo">
                        <xs:restriction base="xs:string"/>
                </xs:simpleType>
        </xs:redefine>

        <xs:element name="foo" type="Bar" />

</xs:schema>
```

This schema includes a type 'Foo' that is redefined, becoming a string; and a type 'Bar' that is unchanged.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://example.org/"
        xmlns="http://example.org/" elementFormDefault="qualified">

        <xs:simpleType name="Foo">
                <xs:restriction base="xs:anySimpleType" />
        </xs:simpleType>

        <xs:complexType name="Bar">
                <xs:simpleContent>
                        <xs:restriction base="Foo" />
                </xs:simpleContent>
        </xs:complexType>

</xs:schema>
```

```
# Base: http://example.org/redefine.owl
@prefix ns1:     <http://example.org/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/def/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

ns1:foo
      a       owl:ObjectProperty ;
      rdfs:range ns1:Bar .

<>    a       owl:Ontology .

ns1:Foo
      a       owl:Class ;
      rdfs:subClassOf xs_:string .

ns1:Bar
      a       owl:Class .
```

0.7.35.1 Child components

- simpleType
- complexType
- group
- attributeGroup

New simple types can be derived by restriction. If a complex type is a restriction of another type we can derive a subclass relationship between them. Restrictions are easier to translate into OWL than extensions because they only add new constraints, thereby reducing the set of valid instances. The occurrences of elements and attributes may be reduced (though not below the minimum of the parent class) and their types may be narrowed. A restriction defines a subset of instances of its base type. OWL may be used to declare restrictions of simple types but is not able to define constraints on the new value space other than by enumeration.

- Restriction of Complex Content

0.7.36 Restriction of complex content

Restrictions are easier to map because they only add new constraints, thereby reducing the set of valid instances. The occurrences of elements and attributes may be reduced (though not below the minimum of the parent class) and their types may be narrowed.

In the example below, the complex type 'Bar' restricts 'Foo' by reducing the number of occurrences of element 'foo' from unbounded to 1; by narrowing the type from xs:anySimpleType to xs:string; and by disallowing mixed content.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://example.org/" xmlns="http://example.org/"
        elementFormDefault="qualified">
```

```
        <xs:complexType name="Bar" mixed="true">
                <xs:sequence maxOccurs="unbounded">
                        <xs:element name="foobar" type="xs:anySimpleType" />
                </xs:sequence>
        </xs:complexType>

        <xs:complexType name="Foo">
                <xs:complexContent>
                        <xs:restriction base="Bar">
                                <xs:sequence>
                                        <xs:element name="foobar" type="xs:string" />
                                </xs:sequence>
                        </xs:restriction>
                </xs:complexContent>
        </xs:complexType>

</xs:schema>
```

Gloze assumes correctness of the schema and asserts the subclass relationship without further checks.

```
# Base: http://example.org/restriction.owl
@prefix ns1:     <http://example.org/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/def/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

<>    a       owl:Ontology .

ns1:foobar
      a       owl:DatatypeProperty , rdf:Property .

ns1:Bar
      a       owl:Class ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:minCardinality "1"^^xs_:int ;
                owl:onProperty ns1:foobar
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:allValuesFrom rdfs:Literal ;
                owl:onProperty ns1:foobar
              ] .

ns1:Foo
      a       owl:Class ;
      rdfs:subClassOf ns1:Bar ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:cardinality "1"^^xs_:int ;
                owl:onProperty ns1:foobar
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:allValuesFrom xs_:string ;
                owl:onProperty ns1:foobar
              ] .
```

0.7.37 Schema Components

Gloze will process the following schema components. Excluded components are xs:field, xs:key, xs:keyref, xs:unique which enable XML content to be identified by XPath expressions.

- all
- annotation
- any
- anyAttribute
- attribute
- attributeGroup
- choice
- complexContent
- complexType
- documentation
- element
- enumeration
- extension
- fractionDigits
- group
- import
- include
- length
- list
- maxExclusive
- maxInclusive
- maxLength
- minExclusive
- minInclusive
- maxLength
- pattern

- redefine
- restriction
- sequence
- simpleContent
- simpleType
- totalDigits
- union
- whiteSpace

0.7.38 xsi:schemaLocation

The XML Schema Instance namespace defines two attributes that declare schema location hints that can be used by an XML processor to locate the schema.

Gloze may be be supplied with user-defined namespace/schema-location hints (from the command line or through the API), or it looks for xsi:schemaLocation or xsi:noNamespaceSchemaLocation on the document element.

The following simple XML may be lifted into RDF either by supplying the schema location on the command line (eg. "schemaLocation.xml http://example.org/ mySchema.xsd"), or as shown in this case, using an explicit xsi:schemaLocation. Note that the xsi namespace must be declared.

```
<?xml version="1.0" encoding="UTF-8"?>
<foobar xmlns="http://example.org/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://example.org/ mySchema.xsd" />
```

When dropping a document into XML, a schema location can be added by supplying the full (base) path of the schema using the schemaLocation parameter (e.g. "-Dgloze.schemaLocation=file:/C:/myExamples/mySchema.xsd"). The schema used in the mapping are relativized against this base and added to the xsi:schemaLocation or xsi:noNamespaceSchemaLocation attribute on the document element.

0.7.39 No-Schema Mapping

The use of xs:any makes it necessary to map content for which we have no schema.

- any example

The schema-less mapping can be used even where we have no schema at all (this is the default mode if no match can be found for the document element).

See also the use of xsi:type for returning from a No-Schema mapping back into a schema.

- xsi:type

0.7.40 Datatypes

The RDF semantics recommendation identifies a subset of XML schema datatypes that are suitable for use in RDF. The following XML datatypes may be used in RDF typed literals. For example, an xs:string "foobar", would be represented in RDF (N3) as "foobar"^^<http://www.w3.org/2001/XMLSchema#string>.

- xs:string
- xs:boolean
- xs:decimal
- xs:float
- xs:double
- xs:dateTime
- xs:time
- xs:date
- xs:gYearMonth
- xs:gYear
- xs:gMonthDay
- xs:gDay
- xs:gMonth
- xs:hexBinary
- xs:base64Binary
- xs:anyURI
- xs:normalizedString
- xs:token
- xs:language
- xs:NMTOKEN
- xs:Name
- xs:NCName
- xs:integer
- xs:nonPositiveInteger
- xs:negativeInteger
- xs:long
- xs:int
- xs:short

- xs:byte
- xs:nonNegativeInteger
- xs:unsignedLong
- xs:unsignedInt
- xs:unsignedShort
- xs:unsignedByte
- xs:positiveInteger

datatype example

The exceptions include:

- xs:anySimpleType
- xs:duration
- xs:ENTITY
- xs:ENTITIES
- xs:ID
- xs:IDREF
- xs:IDREFS
- xs:NMTOKENS
- xs:NOTATION
- xs:QName

See also:

http://www.w3.org/TR/2004/REC-rdf-mt-20040210/ http://www.w3.org/TR/xpath-function
http://www.w3.org/TR/swbp-xsch-datatypes/

The following sections explore work-arounds for all of these datatypes.

0.7.40.1 The mother of all simple types (xs:anySimpleType)

This type is the base of all simple types with an unconstrained lexical space. User defined restrictions of xs:anySimpleType are not allowed. Indeed, users are generally advised to steer clear of it altogether.

Yet, both elements and attributes may be defined to be of type xs:anySimpleType (it's also the default type for attributes). Also, anySimpleType may be used as the base of a simpleContent extension.

Thinking about an RDF savvy mapping, it occupies a similar place in the pantheon of classes as rdfs:Literal, the superclass of all literals including datatypes. Thus any XML content of type xs:anySimpleType is mapped to an rdfs:Literal.

anySimpleType example

See also:

http://www.w3.org/2001/05/xmlschema-rec-comments#pfiS4SanySimple-Type http://lists.w3.org/Archives/Member/w3c-xml-schema-ig/2002Jan/0065.html

0.7.40.2 Duration (xs:duration)

The problem with xs:duration is that there's no well-defined total ordering over it's value space (durations are partially ordered). The problem stems from there being an indeterminate number of days in a month. The recommended solution used by Gloze is to distill a single period such as "P7Y2M26DT14H18M10S" (years, months, days, hours, minutes, seconds) into separate xs:yearMonthDuration "P7Y2M" (years, months) and xs:dayTimeDuration "P26DT14H18M10S" (days, hours, minutes, seconds) datatypes. The reverse process, is equivalent to adding the these values, both of which are subclasses of duration. When adding two durations, each component is added independently, ignoring - in particular - any carry from days to months.

duration example

0.7.40.3 Entities (xs:ENTITY)

An XML schema ENTITY allows the substitution of common text values or balanced mark-up defined as XML entities. ENTITY values must match an entity name declared in the DTD of the instance document. The value space of unexpanded entities is scoped to the instance document it appears in. For the XML to RDF mapping, internally defined entities are therefore expanded. As they may include balanced mark-up, an expanded entity can be described as an RDF XMLLiteral. There is currently no reverse mapping due to technical issues in editing document type declarations in level 2 DOM.

entity example

See also:

`http://jena.sourceforge.net/how-to/typedLiterals.html#xsd`

0.7.40.4 Identity datatypes (xs:ID, xs:IDREF)

An element is considered to have an ID if it has an attribute of type ID, or if the type of the element itself is an ID.

IDs have no distinguishing features looking at the XML alone, they look like ordinary content. We look to the XML schema which will identify the datatype as xml schema ID. The ID is associated with the enclosing element, and that element can have at most one ID.

XML IDs are defined to have document scope, such that a given ID must be unique within a single document and that each ID reference should have a corresponding ID within the same document. One advantage of the mapping into RDF is that a single RDF model may contain descriptions of multiple documents. We have ensure that we preserve the global uniqueness of identifiers, and do not lose the correlation between IDs and their references when moving to this global context. An identifier of type ID can be transformed into a URI by treating it as a fragment identifier relative to the document base.

For example, a base `http://example.org/base` an XML ID "foobar" combine to give the URI, `http://example.org/base#foobar` .

Properties of type ID will disappear, as these simply define the URI of the identified resource. A corresponding reference to this resource with an IDREF is similarly expanded into a URI reference.

identity example

0.7.40.5 Notation (xs:NOTATION)

NOTATIONs are restricted to QNames declared in the schema. For the purposes of RDF mapping they are subject to the same rules as QNames. The target namespace and notation name are expanded to give an

absolute URI for the notation resource.

notation example

0.7.40.6 Qualified Names (xs:QName)

QNames define the space of (optionally) qualified local names. The scope of an XML namespace prefix includes the element it is defined in and its children (subject to shadowing). This lexical scoping doesn't translate directly into RDF where everything has global scope. However, the expanded QName is a URI, so it may be translated into an object reference, though typically we have no knowledge of the type of object referred to. This URI becomes associated with a resource.

For example, given a namespace prefix 'eg' defined as "http://example.org" the QName "eg:foobar" would be expanded to give the URI, `http://example.org#foobar` .

QName example

0.7.40.7 List types (xs:IDREFS, xs:ENTITIES, xs:NMTOKENS)

Although list types are treated as simple in XML schema, they are not recommended for use in RDF. Instead, we construct an rdf:list of the corresponding non-list type (xs:IDREF, xs:ENTITY, xs:NMTOKEN).

IDREFS example

0.7.41 SubClass Relationships (complex type derivation)

Sub-class relationships may be derived from extensions and restrictions of complex content.

- restriction of complex content
- extension of complex content

0.7.42 Ordered Content

An XML document has a tree structure where the children of each element are lexically ordered. Sometimes this ordering is significant, sometimes it is not. Gloze is designed from the point of view of retaining sufficient information to reconstruct the original document from the RDF, making it possible to round-trip from XML to RDF and back again. This doesn't mean recording the order in all cases. In many cases the XML schema contains enough ordering information to reconstruct the original sequence. Ambiguity is created by multiple occurrences of the same element, by mixed content, or by the appearance of the xs:any wild-card. Sequences of singly occurring xs:sequence are entirely unambigous, as are choices where only one of the choices can occur (so long as this occurs once). The compositor 'all', where all orderings are valid typically require sequencing (unless in the degenerate case where they only have a single element). The ordering of attributes is not significant.

All the examples in this section were generated with order=seq (-Dgloze.order=seq).

The following schema describes an unambiguous sequence containing an unambiguous choice.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://example.org/" xmlns="http://example.org/" elementFormDefault="qualified">

        <xs:element name="foobar">
                <xs:complexType>
```

```
                        <xs:sequence>
                                <xs:element name="foo" type="xs:string" minOccurs="0" />
                                <xs:choice>
                                        <xs:element name="bar" type="xs:string"/>
                                        <xs:element name="baz" type="xs:string"/>
                                </xs:choice>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>

</xs:schema>
```

The XML below requires no additional ordering information.

```
<?xml version="1.0" encoding="UTF-8"?>
<foobar xmlns="http://example.org/">
        <foo>foobar</foo>
        <bar>foobar</bar>
</foobar>
```

```
# Base: http://example.org/ordering.xml
@prefix ns1:     <http://example.org/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/def/> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .

<>    ns1:foobar
              [ ns1:bar "foobar"^^xs_:string ;
                ns1:foo "foobar"^^xs_:string
              ] .
```

Where there is ambiguity in the schema for a given element content this is captured as an RDF sequence. We need to record the order in which particular statements appear. This is achieved by reifying the statement, giving us an object that can be added to an rdf:Seq.

The schema below permits multiple occurrences of the element 'bar', so ordering information needs to be recorded if it is significant.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://example.org/" xmlns="http://example.org/" elementFormDefault="qualified">

        <xs:element name="foobar">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name="foo" type="xs:string"/>
                                <xs:element name="bar" type="xs:string" maxOccurs="unbounded"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>

</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<foobar xmlns="http://example.org/">
        <foo>foobar</foo>
        <bar>foobar</bar>
        <bar>foobar</bar>
</foobar>
```

The RDF mapping shows how ordering information is recorded alongside the standard RDF mapping. The standard mapping involves adding the property 'foo' with value "foobar". The subject of this statement is also an RDF sequence, and the first member of this sequence is the reification with rdf:predicate 'foo' and rdf:object "foobar". The two occurrences of 'bar'with identical values, "foobar" result in the assertion of a single property/value. Despite this, two reified statements are added to the sequence, one for each occurrence.

```
# Base: http://example.org/ordering1.xml
@prefix ns1:     <http://example.org/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/def/> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .

<>    ns1:foobar _:b1 .

_:b1  a       rdf:Seq ;
      rdf:_1  [ a       rdf:Statement ;
                rdf:object "foobar"^^xs_:string ;
                rdf:predicate ns1:foo ;
                rdf:subject _:b1
              ] ;
      rdf:_2  [ a       rdf:Statement ;
                rdf:object "foobar"^^xs_:string ;
                rdf:predicate ns1:bar ;
                rdf:subject _:b1
              ] ;
      rdf:_3  [ a       rdf:Statement ;
                rdf:object "foobar"^^xs_:string ;
                rdf:predicate ns1:bar ;
                rdf:subject _:b1
              ] ;
      ns1:bar "foobar"^^xs_:string ;
      ns1:foo "foobar"^^xs_:string .
```

Ordering is transparent from an ontological perspective. One of the design goals was to allow users of the RDF mapping to ignore the sequence if it is not relevant. Ordering is treated as a data-structuring issue rather than an ontological one. The addition of a property to a resource is treated independently of adding it to the sequence; sequencing meta-data is overlaid on top of the existing unordered information model.

0.7.43 Mixed Content

Mixed content, that is text interleaved with markup, also requires ordering. The following example includes combined text and markup. Note that the text content of an element appears as an rdf:value.

```
<?xml version="1.0"?>
<letterBody>
<salutation>Dear Mr.<name>Robert Smith</name>.</salutation>
Your order of <quantity>1</quantity> <productName>Baby
Monitor</productName> shipped from our warehouse on
<shipDate>1999-05-21</shipDate>. ....
</letterBody>
```

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="letterBody">
  <xsd:complexType mixed="true">
```

```
    <xsd:sequence>
      <xsd:element name="salutation">
        <xsd:complexType mixed="true">
          <xsd:sequence>
            <xsd:element name="name" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="quantity"    type="xsd:positiveInteger"/>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:element name="shipDate"    type="xsd:date" minOccurs="0"/>
      <!-- etc. -->
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

</schema>


# Base: http://example.org/mix.xml
@prefix xsd_:    <http://www.w3.org/2001/XMLSchema#> .
@prefix ns1:     <http://example.org/def/> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix :        <#> .

<>    ns1:letterBody _:b1 .

_:b2  a       rdf:Seq ;
      rdf:_1  [ a       rdf:Statement ;
                rdf:object "Dear Mr." ;
                rdf:predicate rdf:value ;
                rdf:subject _:b2
              ] ;
      rdf:_2  [ a       rdf:Statement ;
                rdf:object "Robert Smith"^^xsd_:string ;
                rdf:predicate ns1:name ;
                rdf:subject _:b2
              ] ;
      rdf:_3  [ a       rdf:Statement ;
                rdf:object "." ;
                rdf:predicate rdf:value ;
                rdf:subject _:b2
              ] ;
      rdf:value "Dear Mr." , "." ;
      ns1:name "Robert Smith"^^xsd_:string .

_:b1  a       rdf:Seq ;
      rdf:_1  [ a       rdf:Statement ;
                rdf:object _:b2 ;
                rdf:predicate ns1:salutation ;
                rdf:subject _:b1
              ] ;
      rdf:_2  [ a       rdf:Statement ;
                rdf:object "Your order of" ;
                rdf:predicate rdf:value ;
                rdf:subject _:b1
              ] ;
      rdf:_3  [ a       rdf:Statement ;
                rdf:object "1"^^xsd_:positiveInteger ;
                rdf:predicate ns1:quantity ;
                rdf:subject _:b1
              ] ;
      rdf:_4  [ a       rdf:Statement ;
                rdf:object "Baby Monitor"^^xsd_:string ;
                rdf:predicate ns1:productName ;
                rdf:subject _:b1
              ] ;
```

```
      rdf:_5  [ a       rdf:Statement ;
                rdf:object "shipped from our warehouse on" ;
                rdf:predicate rdf:value ;
                rdf:subject _:b1
              ] ;
      rdf:_6  [ a       rdf:Statement ;
                rdf:object "1999-05-21"^^xsd_:date ;
                rdf:predicate ns1:shipDate ;
                rdf:subject _:b1
              ] ;
      rdf:_7  [ a       rdf:Statement ;
                rdf:object ". ...." ;
                rdf:predicate rdf:value ;
                rdf:subject _:b1
              ] ;
      rdf:value "Your order of" , ". ...." , "shipped from our warehouse on" ;
      ns1:productName "Baby Monitor"^^xsd_:string ;
      ns1:quantity "1"^^xsd_:positiveInteger ;
      ns1:salutation _:b2 ;
      ns1:shipDate "1999-05-21"^^xsd_:date .
```

0.7.44 Identity

XML is not just a tree, but a tree with pointers. An xs:IDREF points to an element with an xs:ID within the same document. An xs:ID identifies the immediately containing element. Typically this is an xs:ID attribute on the element, though it may also be xs:ID simple content.

In RDF we identify the element *content*, by assigning it a URI. This URI is the object of the statement representing the element occurrence.

- attribute identity
- element identity

0.7.45 xsi:type

The XML schema instance namespace defines xsi:type allowing elements be be explicitly annotated with type information. This can be used as an alternative mechanism to substitution groups, but also provides a way to jump back into a schema from within xs:any content. An example of this is shown below using an XML document that has no schema.

The document element 'myLink' is not defined in the schema, so is processed as if it were subject to xs:any and a default no-schema mapping is employed. This unidentified element is treated as unqualified and is defined in the default namespace set by the xmlns parameter.

The link schema doesn't define the 'myLink' element, but it does define the 'SimpleLink' content. The XML instance refers to this using an xsi:type attribute. The RDF mapping for this generates a corresponding rdf:type statement. The content of 'myLink' is then processed according to the content model of 'SimpleLink', so we have escaped from the xs:any no-schema mapping.

```
<?xml version="1.0" encoding="UTF-8"?>
<myLink xmlns:eg="http://example.org/" xmlns:xlink="http://www.w3.org/1999/xlink"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="eg:SimpleLink"
        xlink:href="foo.xml" />
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://example.org/" xmlns:xlink="http://www.w3.org/1999/xlink">
```

```
        <xs:import namespace="http://www.w3.org/1999/xlink" schemaLocation="xlink.xsd" />

        <xs:complexType name="SimpleLink">
                <xs:attributeGroup ref="xlink:simpleLink"/>
        </xs:complexType>

</xs:schema>
```

The use of the 'SimpleLink' content model is evidenced by the correct datatyping of the xlink:href and the insertion of the xlink:type implied by the xlink attribute group.

```
# Base: http://example.org/linkType.xml
@prefix xlink_:  <http://www.w3.org/1999/xlink#> .
@prefix ns1:     <http://example.org/def/> .
@prefix xlink:   <http://www.w3.org/1999/xlink> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix eg:      <http://example.org/> .

<>    ns1:myLink
              [ a       ns2:SimpleLink ;
                xlink_:href "http://example.org/foo.xml"^^xs_:anyURI ;
                xlink_:type "simple"^^xs_:string
              ] .
```

See also:

http://www.w3.org/2001/XMLSchema-instance

0.7.46 sequence

Compositors like xs:sequence are not represented in OWL because they are primarily concerned with the lexical form of a document. However, they are used to derive restrictions on the cardinality of individual properties appearing within a class. Cardinalities involving xs:sequence are derived by multiplying all nested cardinalities by the minimum and maximum number of occurences of that sequence. By default the minimum and maximum are 1, leaving the nested cardinalities unchanged.

The example below demonstrates this multiplication at work. Element 'foo' has the default cardinality of 1. It's containing sequence has a minimum occurence of 0, so we derive a minimum cardinality of $1*0=0$ on 'foo'. From the default maximum (1) we derive a maximum cardinality on 'foo' of $1*1=1$. A minimum of 0 is no constraint at all, so only the maximum cardinality restriction appears in the OWL.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://example.org/" xmlns="http://example.org/"
        elementFormDefault="qualified">

        <xs:element name="foo" />
        <xs:element name="bar" />

        <xs:element name="foobar">
                <xs:complexType>
                        <xs:sequence minOccurs="0">
                                <xs:element ref="foo" />
                                <xs:element ref="bar" maxOccurs="unbounded"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
```

```
</xs:schema>
```

The element 'bar' is a little more interesting, having an unbounded number of occurrences and a default minimum cardinality of 1. The derived maximum cardinality is 1∗unbounded=unbounded, and in effect unrestricted. Similarly, the derived minimum cardinality is 0∗1=0, also unrestricted. There are therefore no cardinality restrictions on element 'bar'. The, possibly counter-intuitive, result is that 'bar' does not appear in the class definition.

```
# Base: http://example.org/sequence.owl
@prefix ns1:     <http://example.org/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/def/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

<>    a       owl:Ontology .

ns1:bar
      a       rdf:Property .

ns1:foobar
      a       owl:ObjectProperty ;
      rdfs:range
              [ a       owl:Class ;
                rdfs:subClassOf
                        [ a       owl:Restriction ;
                          owl:maxCardinality "1"^^xs_:int ;
                          owl:onProperty ns1:foo
                        ]
              ] .

ns1:foo
      a       rdf:Property .
```

Note that the default type of elements 'foo' and 'bar' is xs:anyType, which is effectively unconstrained hence there are no ranges defined for either property. Also, because xs:anyType is a super-class of xs:anySimpleType, it is unknown whether or not 'foo' and 'bar' are object or datatype properties (or both).

0.7.46.1 Child components

- element
- choice
- sequence
- any
- group

0.7.47 simpleContent

Simple content lets us add attributes to an element that otherwise has content representing a single value.

```
<?xml version="1.0" encoding="UTF-8"?>
<foo xmlns="http://example.org/" bar="bar">foo</foo>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://example.org/">

        <xs:element name="foo">
                <xs:complexType>
                        <xs:simpleContent>
                                <xs:extension base="xs:string"/>
                        </xs:simpleContent>
                        <xs:attribute name="bar" />
                </xs:complexType>
        </xs:element>

</xs:schema>
```

Note how the content and named properties derived from attributes share the same subject. The content is differentiated from attributes by the use of the rdf:value property.

```
# Base: http://example.org/simpleContent.xml
@prefix ns1:     <http://example.org/def/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .

<>    ns2:foo [ rdf:value "foo"^^xs_:string ;
                ns1:bar "bar"
              ] .
```

0.7.47.1 Child components

- annotation
- restriction
- extension

0.7.48 simpleType

New simple types can be derived by restriction. OWL can define new sub-classes of datatypes but is not able to define constraints on the new value space other than by enumeration. Simple types are therefore declared in OWL, but not defined to the same extent as in XML schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://example.org/" xmlns="http://example.org/" >

        <xs:simpleType name="mySimpleType">
                <xs:restriction base="xs:string" />
        </xs:simpleType>

        <xs:element name="foo" type="mySimpleType" />

</xs:schema>
```

```
# Base: http://example.org/simpleType1.owl
@prefix ns1:     <http://example.org/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/def/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

<>    a       owl:Ontology .

ns1:foo
      a       owl:DatatypeProperty , rdf:Property ;
      rdfs:range ns1:mySimpleType .

ns1:mySimpleType
      a       owl:Class ;
      rdfs:subClassOf xs_:string .
```

Gloze avoids using simple types in typed literals by instead using the datatype it is derived from. In the XML instance and below the lifted value 'bar' has type xs:string rather than the user-defined simple type 'mySimpleType'.

```
<?xml version="1.0" encoding="UTF-8"?>
<foo xmlns="http://example.org/">bar</foo>
```

```
# Base: http://example.org/simpleType1.xml
@prefix ns1:     <http://example.org/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/def/> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .

<>    ns1:foo "bar"^^xs_:string .
```

0.7.48.1 Child components

- restriction
- list
- union

0.7.49 union

Union datatypes merge the lexical spaces of several existing types to create another. Because OWL does not currently support user-defined datatypes, Gloze uses only the union member types to define datatyped literals.

```
<?xml version="1.0" encoding="UTF-8"?>
<union xmlns="http://example.org/">foobar</union>
```

This XML conforms to the schema below. The content is either an xs:int or an xs:string.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://example.org/">

        <xs:element name="union">
                <xs:simpleType>
                        <xs:union memberTypes="xs:int xs:string" />
                </xs:simpleType>
        </xs:element>

</xs:schema>
```

The XML content is validated against both xs:int and xs:string to determine its type. In this case it is an xs:string.

```
# Base: http://example.org/union.xml
@prefix ns1:     <http://example.org/def/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .

<>    ns2:union "foobar" .
```

There's little to say about the range of the property. A simple type may sometimes map to an object type (e.g. QNames), so a union is not necessarily a datatype property. In this case, both member types map to datatype properties so 'union' is a datatype property.

```
# Base: http://example.org/union.owl
@prefix ns1:     <http://example.org/def/> .
@prefix xs:      <http://www.w3.org/2001/XMLSchema> .
@prefix ns2:     <http://example.org/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xs_:     <http://www.w3.org/2001/XMLSchema#> .
@prefix :        <#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .

ns2:union
      a       owl:DatatypeProperty , rdf:Property .

<>    a       owl:Ontology .

(xs_:int xs_:string) .
```

0.7.49.1 Child components

- annotation
- simpleType

0.7.50 whiteSpace

WhiteSpace processing is a contentious area of XML and is one of the main reasons why round-tripping XML produces an output that is (semantically) equivalent rather than (lexically) identical to the original. The following example demonstrates three ways to control whitespace processing. It inlcudes an element 'foobar' with liberally spaced mixed content interspersed with sub-elements 'foo' and 'bar' both with content containing leading (tabbed) indentation.

The whitespace processing of element 'foo' is determined by the xml:space attribute which indicates that whitespace should be collapsed; trimming the leading whitespace. The whitespace processing of element 'bar' is determined by the 'whiteSpace' restriction in the schema, which is also set to collapse whitespace.

```
<?xml version="1.0" encoding="UTF-8"?>
<foobar xmlns="http://example.org/">
        foo
        bar
        <foo xml:space="collapse">
        foo
        </foo>

        <bar>
        bar
        </bar>
        foo             bar
</foobar>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://example.org/" xmlns="http://example.org/">

        <xs:element name="foobar">
        <xs:complexType mixed="true">
                <xs:sequence>
                        <xs:element ref="foo" />
                        <xs:element ref="bar" />
                </xs:sequence>
        </xs:complexType>
        </xs:element>

        <xs:element name="foo" type="xs:string" />

        <xs:element name="bar">
                <xs:simpleType>
                        <xs:restriction base="xs:string">
                                <xs:whiteSpace value="collapse" />
                        </xs:restriction>
                </xs:simpleType>
        </xs:element>

</xs:schema>
```

Finally, the gloze parameter space may be set to 'preserve' or 'default', equivalent to setting xml:space in the document element. Mapping to RDF with space=default, then round-tripping back into XML we get the following (equivalenmt but not identical to the original).

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:foobar xmlns:ns1="http://example.org/">foo bar<ns1:foo>foo</ns1:foo>
    <ns1:bar>bar</ns1:bar>foo bar</ns1:foobar>
```

Index