Cohere Data Model: examples and mappings to AIF

....

Knowledge Media Institute, The Open University,
Walton Hall, Milton Keynes, MK62PL, United Kingdom
{m.pasin, }@open.ac.uk

Abstract. I've put together a data model in rdf/owl for the entities dealt with in Cohere. The main source of inspiration was, of course, the existing data model in the Cohere database, which has been 'purified' from things like IDs and connection tables. This document is meant to be read while browsing the ontology using a tool like Protégé. For the moment, in this document I am just commenting the figures (hopefully quite self-explanatory) that sum up the modeling of the ontology. A more detailed description of the Cohere ontology will be produced after discussing it with the relevant people. Please note that the final section about the AIF mappings is still in a very early phase. Finally, sorry for using this paper-like format – it's the first that came up when I opened ms-word this morning, and I couldn't be bothered to change it!

1 Intro

The Cohere ontology is supposed to formally define the meaning of the entities dealt with in the Cohere application. In particular, it aims at facilitating data exchange and interoperability issues on the web. An automatically generated documentation of the ontology can be found online at:

- **http://coherewеb.net/ontology/docs1/** , for the ontology without the preliminary mappings to AIF (if you are not interested in them)
- **http://coherewеb.net/ontology/docs2/** , for the ontology with the preliminary mappings to AIF

The ontology is in its early days – please do not spare comments and suggestions. The ontology namespace is http://coherewеb.net/ontology/cohere.owl. Just point at that url with a browser or an ontology editor to see it. It's available in two versions:

- **http://coherewеb.net/ontology/cohere.owl** , has all the classes and also some example instances (see below)
- **http://coherewеb.net/ontology/cohere-aif.owl** , has in addition the classes for the AIF mappings.

2 The ontology

In this section I give more explanations about the figures appearing on the following pages. For a more detailed explanation of all the classes please check the Owl-Doc online or the ontology by using Protégé or another editor.

1. The first figure shows the modeling of the IDEA and PERSON classes, together with a few others for specifying the main features we want to associate with them (websites, images etc.). Notice how an ABSTRACT_IDEA is a complex entity, which subsumes SIMPLE IDEA, CONNECTION and SET_OF_CONNECTIONS. This is needed (see below) for guarantying formalizing arguments at various levels of abstractions (i.e. nested arguments, reifications etc.)
2. The second figure shows the modeling of the CONNECTION class, which is the only mean Cohere provides in order to create relations among ideas. The important thing to remember here is that *only* in the context of a connection an idea can play a role (e.g. premise, conclusion, example etc.). Therefore a connection class, beyond having the *from_node* and *to_node* properties, has also the *has_from_node_type* and *has_to_node_type* properties. By doing so, the role of a node in a connection is stored together with the connection, and not with the node itself. Further, the role is embodied in the *connection_link_type* class. Among the instances of it (that users can create) there are also the ones that come as default with Cohere (see website), which are divided into three types: neutral, positive and negative.
3. The third figure shows an example instantiation (the same instantiation can also be found in the OWL ontology online). I've tried to keep all the types at the bottom, and the instances at the top. Notice how the creation of the triple [chemistry – supports→ neurobiology] involves the instantiation of several other classes (here you can actually see only a small subset of them, pls check in Protégé for a more detailed description).
4. The fourth figure shows the same example described in figure 3, with the augmentation of a new connection obtained by *reifying* the previous connection and treating it as a simple idea (node). This is possible thanks to the hierarchical model of ideas seen in 1).
5. The fifth figure shows a simple AIF argument (bottom) and its transposition in the Cohere model (top). The key aspect is the transformation of an AIF S-node (Expert_opinion_1) into a Cohere *connection_link* with type the specific type of the S-node. Similarly, the roles of the I-nodes in the AIF schema are transformed in Cohere into *simple_idea* instances with type the relevant AIF schema-type. If you do not have a clue about what I am talking about, you better read first the AIF paper at http://portal.acm.org/citation.cfm?id=1285099.

Figure 1: IDEA and PERSON network

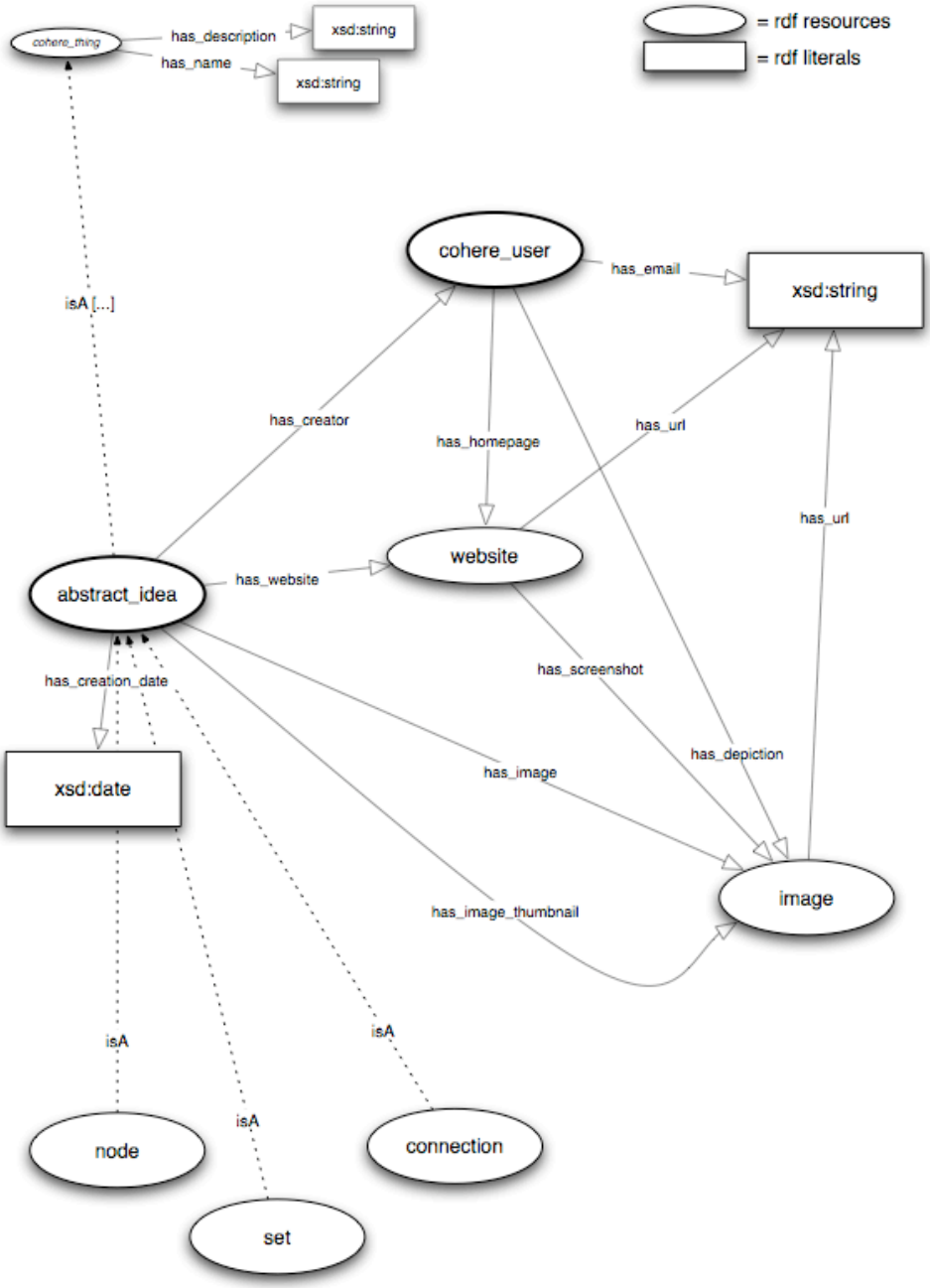


Figure 2: IDEA and CONNECTION networks

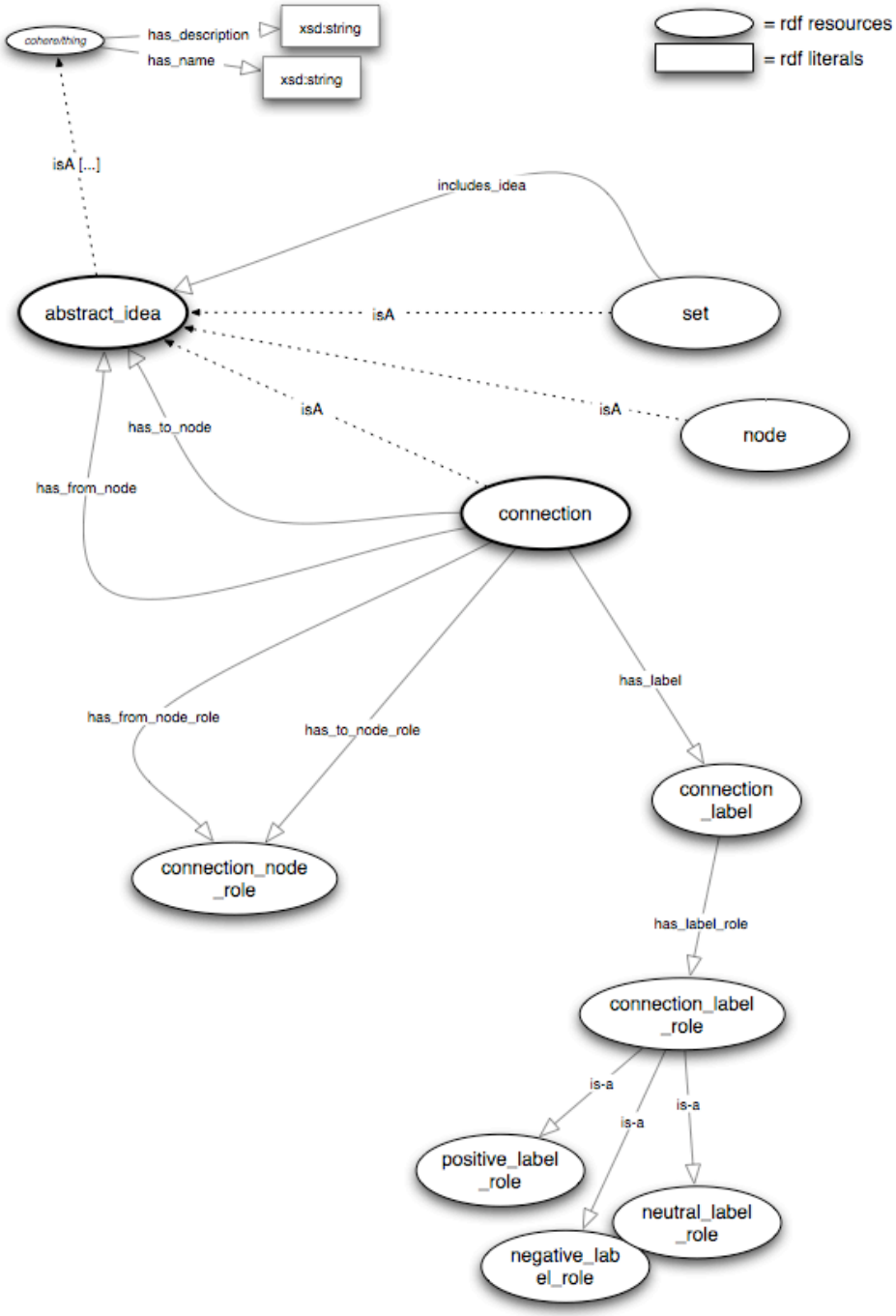


Fig. 3 Example instantiation (see also owlDoc)

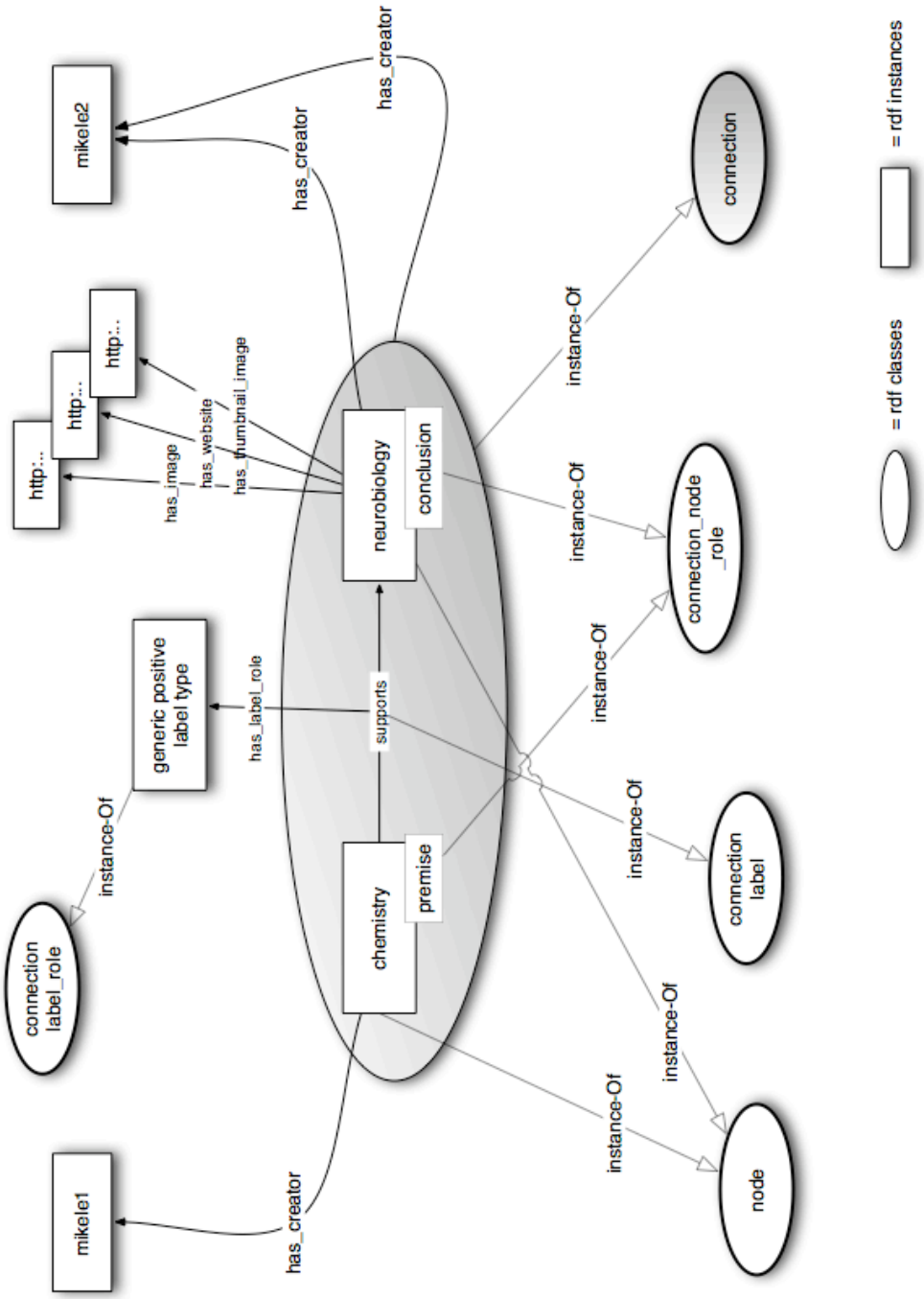


Fig. 4 Example instantiation with reification

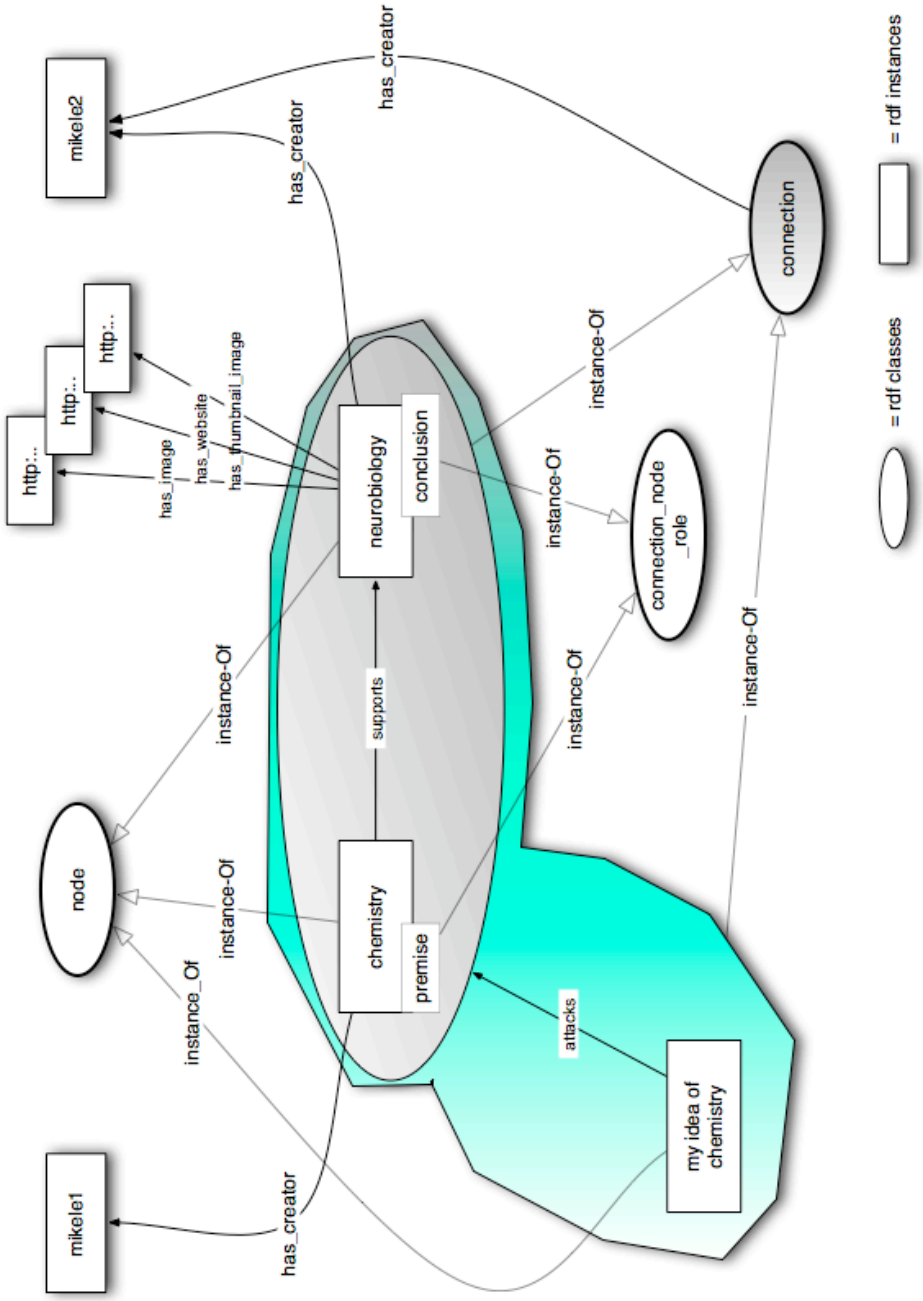
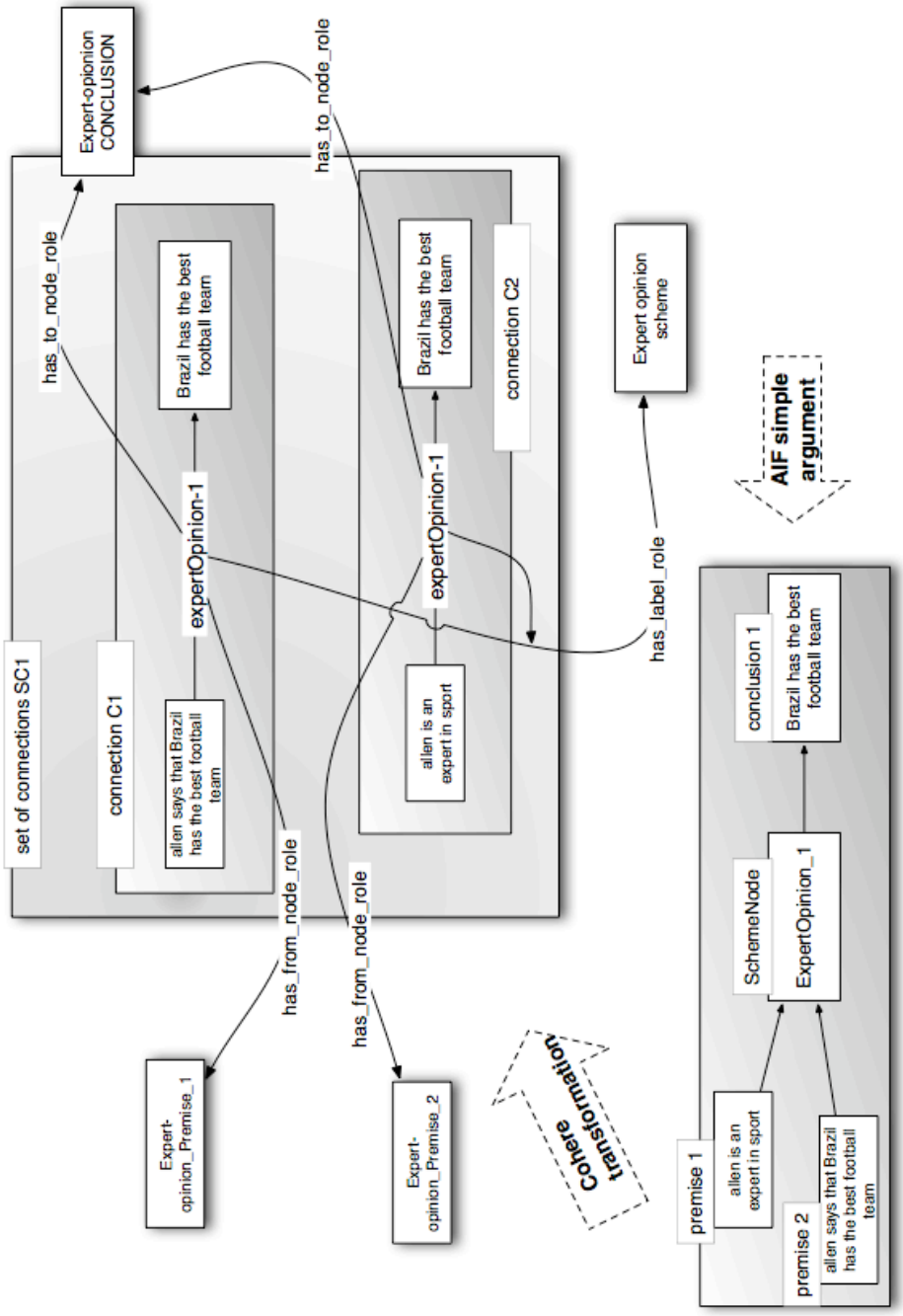


Fig. 5 AIF mappings: ‘explosion’ of an AIF argument in the Cohere model.



Finally, some RDF/XML code expressing the instantiation example on fig.3 (this code & the ontology can be found in the cohere.owl file).
The code, after being automatically generated by Protégé, was re-ordered by me for being more readable. Remember (if you're not familiar with the rdf family of languages) that in rdf the statements do not have to respect a fixed order: as a result, automatically generated code is quite difficult to read for humans…(e.g. some statements refer to objects which are created only at the end of the file). Also, resources can be created within other resources: that means that new resources do not all have to be created at the top-level. For example, we could have the `<person rdf:ID="person_mikele1">` created within the description of another resource, e.g. an instance of a connection link: `<connection_link rdf:ID="connection_link_chemistry_TO_neurobiology">` . This may appear very confusing, cause the person and the connection we are talking about are not strictly or exclusively related to each other. For a gentle and quick introduction to rdf, I suggest to start at http://www.xulplanet.com/tutorials/mozsdk/rdfsyntax.php or http://blog.gandrew.com/2006/06/rdf-in-nutshell.html. Enjoy!

In the code firstly all the websites are instantiated:

```
<rdf>

<website rdf:ID="website_49_chemistry">
   <has_description rdf:datatype="http://www.w3.org/2001/XMLSchema#string">my
favourite chemistry website</has_description>
   <has_url
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://chemistry_website.co
m</has_url>
   <has_name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">the website of
chemistry</has_name>
</website>


 <website rdf:ID="website_neurobiology">
  <has_name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">the website of
neurobiology</has_name>
  <has_description
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">neurobiology
website</has_description>
  <has_url
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://neurobiology_website
.com</has_url>
</website>


<website rdf:ID="website_mikele2">
  <has_name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">mikele2's
homepage</has_name>
  <has_url
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">www.mikele2.com</has_url>
</website>

<website rdf:ID="website_42_mikele1">
  <has_url
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">www.mikele1.com</has_url>
  <has_name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">mikele's
homepage</has_name>
</website>
```

The we instantiate the cohere_users which are mentioned in the example: *mikele1* and *mikele2*. Notice how the *has_homepage* property points at an already existing object, by using the *rdf:resource* property.

```
<cohere_user rdf:ID="person_mikele1">
  <has_homepage rdf:resource="#website_42_mikele1"/>
  <has_email_address
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">mikele1@open.ac.uk</has_ema
il_address>
  <has_description rdf:datatype="http://www.w3.org/2001/XMLSchema#string">one of
the aliases of this ontology's creator</has_description>
  <has_depiction rdf:resource="#image_43_mikele1"/>
  <has_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">mikele1</has_name>
</cohere_user>


<cohere_user rdf:ID="person_mikele2">
  <has_homepage rdf:resource="#website_mikele2"/>
  <has_email_address
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">mikele2@open.ac.uk</has_ema
il_address>
  <has_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">mikele2</has_name>
  <has_description rdf:datatype="http://www.w3.org/2001/XMLSchema#string">another
one of the aliases of this ontology's creator</has_description>
  <has_depiction rdf:resource="#image_mikele2"/>
</cohere_user>
```

Now we instantiate all the images which are related to persons and ideas:

```
<image rdf:ID="image_43_mikele1">
  <has_url
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://mikele1.tiff</has_ur
l>
</image>
<image rdf:ID="image_mikele2">
  <has_url
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://mikele2.tiff</has_ur
l>
</image>
<image rdf:ID="image_52_neurobiology">
  <has_url
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://neurobiologyimage.ti
ff</has_url>
</image>
<image rdf:ID="thumbnail_image_neurobiology">
  <has_url
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://thumbnail_neurobiolo
gy.tiff</has_url>
</image>
<image rdf:ID="image_47_chemistry">
  <has_url
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://chemistryimage.tiff<
/has_url>
</image>
<image rdf:ID="thumbnail_image_48_neurobiology">
  <has_url
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://thumbnail_neurobiolo
gy.tiff</has_url>
</image>
```

Now it's time to create the 'node' instances, that is, the two atomic ideas which are connected together, *neurobiology* and *chemistry*. Again, the already existing object are referenced to by using *rdf:resource=#...*

```
<node rdf:ID="simple_idea_51_neurobiology">
  <has_creator rdf:resource="#person_mikele2"/>
  <has_creation_date
rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2008-02-
19T14:34:49</has_creation_date>
  <has_image rdf:resource="#image_52_neurobiology"/>
  <has_website rdf:resource="#website_neurobiology"/>
  <has_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">neurobiology</has_name>
  <has_thumbnail_image rdf:resource="#thumbnail_image_neurobiology"/>
  <has_description rdf:datatype="http://www.w3.org/2001/XMLSchema#string">the idea
of neurobiology, the discipline</has_description>
</node>

<node rdf:ID="simple_idea_20_chemistry">
  <has_creation_date
rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2008-02-
19T14:34:49</has_creation_date>
  <has_creator rdf:resource="#person_mikele1"/>
  <has_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">chemistry</has_name>
  <has_description rdf:datatype="http://www.w3.org/2001/XMLSchema#string">the idea
of chemistry, the discipline</has_description>
  <has_image rdf:resource="#image_47_chemistry"/>
  <has_website rdf:resource="#website_49_chemistry"/>
  <has_thumbnail_image rdf:resource="#thumbnail_image_48_neurobiology"/>
</node>
```

At this point, we can create the connection object and the connection_label linking the two nodes:

```
<connection rdf:ID="connection_chemistry_to_neurobiology">
  <has_creator rdf:resource="#person_mikele1"/>
  <has_creation_date
rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2008-02-
19T14:41:50</has_creation_date>
  <has_to_node_role rdf:resource="#connection_node_type_conclusion"/>
  <has_to_node rdf:resource="#simple_idea_51_neurobiology"/>
  <has_from_node_role rdf:resource="#connection_node_type_premise"/>
  <has_from_node rdf:resource="#simple_idea_20_chemistry"/>
  <has_label rdf:resource="#connection_link_chemistry_TO_neurobiology"/>
</connection>

<connection_label rdf:ID="connection_link_chemistry_TO_neurobiology">
    <has_label_role rdf:resource="#positive_link_type_62_generic_support"/>
    <has_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">supports_idea</has_name>
</connection_label>
```

Finally, since we want to specify what *roles* the nodes and the label have in the connection, we also create some more individuals to express that. Notice that many of these roles are given *by default* by the Cohere tool, but the users could create new ones.

```
<positive_label_role rdf:ID="positive_link_type_62_generic_support">
  <has_description rdf:datatype="http://www.w3.org/2001/XMLSchema#string">just a
generic positive link, named supports</has_description>
  <has_creator rdf:resource="#person_mikele1"/>
```

```
  <has_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">supports</has_name>
</positive_label_role>


<connection_node_role rdf:ID="connection_node_type_conclusion">
  <has_creator rdf:resource="#person_mikele1"/>
  <has_description rdf:datatype="http://www.w3.org/2001/XMLSchema#string">a
conclusion in some argumentation schema</has_description>
  <has_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">conclusion</has_name>
</connection_node_role>


<connection_node_role rdf:ID="connection_node_type_premise">
  <has_creator rdf:resource="#person_mikele1"/>
  <has_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">premise</has_name>
  <has_description rdf:datatype="http://www.w3.org/2001/XMLSchema#string">a
premise in some argumentation schema</has_description>
</connection_node_role>


</rdf>
```