Implementing a Usecase, Part 1: Prerequisites

Table of contents

1 Introduction

In this tutorial we'll implement a simple user interaction scenario using the usecase framework (../../../../ docs/2_0_x/reference/usecase-framework/index.html) . The implementation will be based on the *person* resource type module described in the tutorial Creating a Resource Type (../../../../docs/2_0_x/tutorials/ resourcetype/part1.html) . We'll extend the resource type to include "knows" relationships. Our usecase will allow the connect a person to other people.

To accomplish this task, we will

- Extend the person schema to support `<foaf:knows/>` elements,
- Implement a wrapper class to manage person documents,
- Add a usecase handler class to provide the functionality to connect people,
- Add a JX template (http://cocoon.apache.org/2.1/userdocs/flow/jxtemplate.html) which acts as the view for the usecase,
- Add a menu item to trigger the usecase, and
- Specify who is allowed to invoke the usecase.

2 Add the "Knows" Relation to the Person Resource Type

First, we'll add support for the `<foaf:knows/>` element to the Relax NG schema, which is located at `$MODULE_HOME/resources/schemas/foaf.rng`. We allow to specify the known person using an `rdf:resource` attribute, which can hold an internal `lenya-document:` URI.

```
<grammar ...>
  <start>
    <element name="rdf:RDF">
      <element name="foaf:Person">
        ...

        <zeroOrMore>
          <element name="foaf:knows">
            <attribute name="rdf:resource">
              <data type="anyURI"/>
            </attribute>
          </element>
        </zeroOrMore>

      </element>
    </element>
  </start>
</grammar>
```

Now we need to extend our presentation layer to consider the "knows" relations. We use CInclude to lookup the person's name based on the `foaf:knows` element. This requires two steps: an XSLT preprocessing to add the CInclude element to the `foaf:knows` element, and the actual CInclude transformation. In the module sitemap, which is located at `$MODULE_HOME/sitemap.xmap`, add these steps to the presentation pipeline:

```
<!-- {format}.xml/{pubId}/{area}/{uuid}/{language} -->
<map:match pattern="*.*/*/*/*/*">
  <map:generate src="lenya-document:{5},lang={6}{link:rev}"/>
  <map:transform src="fallback://lenya/modules/person/xslt/knows2include.xsl"/>
  <map:transform type="cinclude"/>
  <map:transform src="fallback://lenya/modules/person/xslt/foaf2{1}.xsl">
  ...
```

```
</map:match>
```

Now we add the XSLT which adds the CInclude statements. It is located at `$MODULE_HOME/xslt/knows2include.xsl`. The `src` attribute of the `<ci:include/>` element is set to the `rdf:resource` attribute, which holds a `lenya-document:` URI.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:ci="http://apache.org/cocoon/include/1.0"
  >

  <xsl:template match="foaf:knows">
    <xsl:copy>
      <xsl:copy-of select="@*"/>
      <ci:include src="{@rdf:resource}"/>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="@*|node()" priority="-1">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

Finally, we have to include the known people in the output. We add a table row to the `$MODULE_HOME/xslt/foaf2xhtml.xsl` stylesheet:

```
<tr>
  <th>Knows:</th>
  <td>
    <xsl:for-each select="foaf:knows">
      <a href="{@rdf:resource}">
        <xsl:value-of select="rdf:RDF/foaf:Person/foaf:givenname"/>
        <xsl:text> </xsl:text>
        <xsl:value-of select="rdf:RDF/foaf:Person/foaf:family_name"/>
      </a>
      <br/>
    </xsl:for-each>
  </td>
</tr>
```

To test this functionality, you can add two person documents to your publication, edit one of them using the one-form editor, and add the `<foaf:knows/>` element, pointing to the other person document, e.g.:

```
<foaf:knows rdf:resource="lenya-document:5e13b150-6855-11dc-8b37-f5f12a4217db"/>
```

3 Implement the Person Document Wrapper

In Lenya projects, the wrapper pattern (also known as adapter pattern) has proven useful to simplify the handling of XML documents. A wrapper object provides access to a certain object, in our case a Lenya document, to client objects. It can be used to add a layer of abstraction. In our case we use the wrapper to abstract from the XML content and provide the functionality of adding "knows" relations between person documents.

The following code snippet contains some methods to illustrate the functionality of the Person class. For the full source code, check out the person module from the Subversion repository.

We pass an `org.apache.lenya.cms.publication.Document` object to the constructor which stores the XML content describing the person. The `getName()` function returns the full name, i.e. the concatenation of the given name and the family name. This method will be used to show the person's name on the usecase view page. The `load()` and `save()` methods act as the persistence facilities of the person object.

```
public class Person extends AbstractLogEnabled {

    ...

    public Person(Document doc) {
        ...
    }

    public Person[] getKnownPeople() {
        load();
        Collection persons = this.knownPersons.values();
        return (Person[]) persons.toArray(new Person[persons.size()]);
    }

    public void addKnownPerson(Person person) {
        load();
        this.knownPersons.put(person.getDocument().getUUID(), person);
        save();
    }

    public String getName() {
        load();
        return this.givenName + " " + this.familyName;
    }

    ...

}
```

Now we can go on with the actual usecase (../../../../docs/2_0_x/tutorials/usecase/part2.html) .