

METAmorphoses v0.2.1: mapping and template language documentation

Contents

1	Overview	3
1.1	Running example	3
1.2	Comment on syntax	4
2	Language semantics (informative)	6
2.1	Language semantics overview	6
2.1.1	Schema mapping language	6
2.1.2	Template language	9
3	Language reference (normative)	13
3.1	Schema mapping language syntax	13
3.1.1	General elements	13
3.1.2	Concept elements	14
3.1.3	Relationship elements	16
3.2	Template language syntax	17
3.2.1	General elements	17
3.2.2	Production elements	18
4	Schema mapping and template document samples	20
4.1	Schema mapping document	20
4.2	Template documents	22
4.3	Resulting RDF	22

Chapter 1

Overview

To enable the schema mapping and data transformation described in the previous chapter, we developed two XML languages - one for schema mapping documents and another for template documents.

This document contains a discussion on the design of the languages followed by running examples. Later on we provide a normative syntax of both languages.

1.1 Running example

Here we provide a base to the running example what will illustrate concepts described in the further text. The complete mapping document for both schemas is in the appendix 4.

Relational database schema

When referring database schema in this chapter, we use common terms as table, column and row. In the running example we suppose a small database schema, consisting of four tables (primary keys are marked by #):

```
PERSON(#id, id_department, username, first_name, family_name)
PROJECT(#id, web)
PERSON_PROJECT(person_id, project_id)
DEPARTMENT(#id, name)
```

The table DEPARTMENT is in 1:N relationship with table PERSON, the foreign key is PERSON.id_department. The tables PERSON and PROJECT have an M:N relationship, the table PERSON_PROJECT is to join them.

Relational data

RDFS ontology

The RDFS ontology that specifies a vocabulary for resulting RDF in our running example is listed in 1.1.

Listing 1.1: RDF-S ontology sample

```
<rdfs:Class rdf:ID="Person">
  <rdfs:label>Person</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="surname">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#
    string" />
  <rdfs:domain rdf:resource="#Person"/>
</rdf:Property>

<rdf:Property rdf:ID="hasDepartment">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#
    string" />
  <rdfs:domain rdf:resource="#Person"/>
</rdf:Property>

<rdf:Property rdf:ID="currentProject">
  <rdfs:range rdf:resource="#Project"/>
  <rdfs:domain rdf:resource="#Person"/>
</rdf:Property>

<rdfs:Class rdf:ID="Project">
  <rdfs:label>Project</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="participants">
  <rdfs:domain rdf:resource="#Project"/>
  <rdfs:range rdf:resource="#Person"/>
</rdf:Property>

<rdf:Property rdf:ID="homepage">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#
    string" />
</rdf:Property>
```

The ontology concepts will be without a namespace prefix in the following examples.

1.2 Comment on syntax

The syntax of our mapping languages is XML. In the following list we provide some reasons to choose an XML language:

- XML was developed to represent this kind of documents and it is very common in the web community (and also in the semantic web community).

- An XML syntax can be analysed and processed using the Document Object Model (DOM), which can be parsed and manipulated by a number of free and commercial libraries, providing a good base for a mapping framework implementation.
- An XML syntax is human readable – a human user can edit an XML documents manually.
- RDF serialisation format is also XML. If a template document is a template for an RDF document, it is intuitive for a human user to build it as a XML tree since RDF can be imagined also as a tree.

Chapter 2

Language semantics (informative)

2.1 Language semantics overview

2.1.1 Schema mapping language

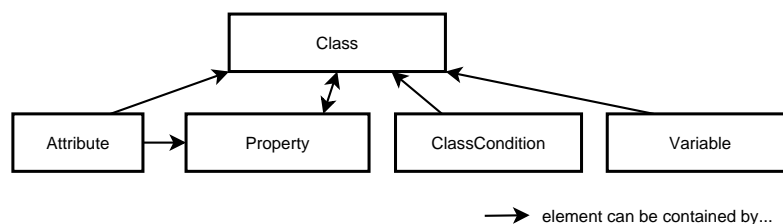
Schema mapping language is responsible for the mapping between a database schema and given ontology. It can describe elements, which map parts of a database schema to classes and properties of an ontology. These mapping elements are later used in a template layer. In addition to the concept mapping elements, there are control structures as conditions and variables, which define relations between concepts. These relation elements are also used in a template layer and they control an RDF generation. Moreover, the language contains support elements for the RDF creation. Their semantic is straightforward so that they are not detailed here but only in the section about the language syntax (3.1).

In short, we can say that the mapping document language is designed to:

- describe a mapping between a database schema and structure of a given ontology,
- enable control over these fragments,
- create framework for making a complete RDF document,
- establish connection with a specific relational database.

The mapping language elements reflecting schema mapping class, property, condition and attribute are **Class**, **Property**, **Condition** and **Attribute**. In addition to these, there is element **Variable**. The hierarchy of these elements is illustrated on the figure 2.1, their description follows.

Figure 2.1: The schema mapping language elements



Element Class (mapping class)

The mapping class is a basic element of the language to express a relationship between a relational database subset and corresponding ontology class. An ontology class is referenced by a full class name with a namespace, relational database schema concept are referenced by an SQL query. Using SQL we can address one or more database tables as one concept. The second reason for choosing SQL query on this level is because a template layer can use it for fetching data from a particular relational database.

The mapping between the ontology class `Person` with the relevant concept from relational schema can be expressed by the following mapping class:

Listing 2.1: Mapping class example

```

<Class templateName="person" rdfLabel="Person" sql="SELECT *
  FROM person p, department d WHERE d.id = p.id_department"
/>
  
```

Attribute `templateName` defines a unique ID for the element, `rdfLabel` refers an RDFS ontology class and `sql` specifies SQL query that addresses a relational concept corresponding to the ontology class. In this particular SQL query a *join* operation is used to interconnect more tables from a relational database creating one concept (a relationship between these tables must be 1:1).

Element Property (mapping property)

The `Property` element always belong to a `Class` element and joins an ontology `Property` with a corresponding database column that belongs to table referenced by the enveloping `Class`.

Listing 2.2: Datatype mapping property example

```

<Class templateName="person" rdfLabel="Person" sql="SELECT *
  FROM person p, department d WHERE d.id = p.id_department">
  <Property templateName="surname" rdfLabel="surname" sqlName
    ="family_name" />
</Class>
  
```

The listing 2.2 describes a datatype mapping property with ID surname (in the attribute `templateName`), which interconnects an RDFS property surname (in the attribute `rdflabel`) with a database column `family_name` (in the attribute `sqlName`).

An object mapping property can be described in very similar fashion by omitting an attribute `sqlName` (listing 2.3), since object mapping properties does not refer relational concepts.

Listing 2.3: Object mapping property example

```
<Class templateName="person" rdflabel="Person" sql="SELECT *
  FROM person p, department d WHERE d.id = p.id_department">
  <Property templateName="currentProject" rdflabel="
    currentProject" />
</Class>
```

Element Attribute (mapping attribute)

The element `Attribute` can be contained by `Class` or `Property` and adds RDF attributes to them. An `Attribute` element must contain attribute `rdflabel` that denotes a name of RDF attribute. The value of an RDF attribute can be specified by attributes `prefix` and `suffix` that are literals and `sqlName` that refers a database column similiary as in `Property` element.

Listing 2.4: Mapping attribute example

```
<Class templateName="person" rdflabel="Person" sql="SELECT *
  FROM person p, department d WHERE d.id = p.id_department">
  <Attribute rdflabel="rdf:about"
    prefix="http://webing.felk.cvut.cz/people/" sqlName="
      username"/>
  <Property templateName="surname" rdflabel="surname"
    sqlName="family_name">
    <Attribute rdflabel="rdf:datatype"
      prefix="http://www.w3.org/2001/XMLSchema#string"/>
  </Property>
</Class>
```

The listing 2.4 shows two `Attribute` elements. The first one, contained directly by `Class` element, defines an RDF attribute `rdf:about` as URI reference for each RDF instance of this mapping class. The URI (attribute value) will consist of `prefix` and value selected from `username` database column. The other one specifies datatype of the property by adding `rdf:datatype` attribute to it.

Element ClassCondition (mapping condition)

A mapping element `ClassCondition` is to describe a relationship between two mapping classes. A mapping condition always belongs to some mapping

class and it can denote that its parent mapping class (further as M_c^1) is in N:1 relation with some other mapping class (further as M_c^2).

In our running example, there are two concepts `Person` and `Project` where a person can work on several projects. This is practically modelled by relationship between tables `PERSON` and `PERSON_PROJECT` in a database schema.

Listing 2.5: Mapping condition example

```
<Class templateName="person" rdfLabel="Person" sql="SELECT *
  FROM person p, department d WHERE d.id = p.id_department">
  <ClassCondition templateName="projectId" whereString="p.id
    = pp.person_id and pp.project_id =" tableString="
    person_project pp" />
  <ClassCondition templateName="username" whereString="p.
    username =" />
</Class>
```

The first element `ClassCondition` in the listing 2.5 models this relationship. Using attribute `tableString` it adds table `PERSON_PROJECT` to parent mapping class `SELECT` query then adds a conditions to the query by attribute `whereString`. The condition connects tables from both classes using a primary key of the M_c^1 with a foreign key from M_c^2 and creates an opened condition, which can be completed later in the template layer by `Variable` element from the class M_c^2 . Applying a such mapping condition on the M_c^1 will result in selecting tuples that relates to the M_c^2 .

Another way of using of a `ClassCondition` is when an attribute `tableString` is omitted. In this case the condition compares a primary key of M_c^1 with a literal. This way a template layer can drive an instance production.

Element Variable

The `Variable` element is to expose a single value from an enveloping mapping class. This is used later in the template document in order to make a join between instances from different mapping classes. The sample of `Variable` element is in the listing 2.6, its further semantics and usage is discussed in section 2.1.2.

Listing 2.6: Mapping variable example

```
<Class templateName="person" rdfLabel="Person" sql="SELECT *
  FROM person p, department d WHERE d.id = p.id_department">
  <Variable templateName="idVariable" sqlName="p.id"/>
</Class>
```

2.1.2 Template language

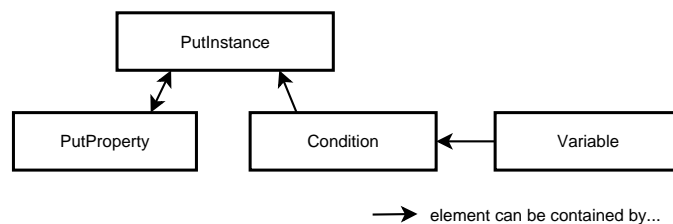
To serialise a template document we developed our own *template document language*, based on XML.

The template document language is based on a small set of elements: `PutInstance`, `PutProperty`, `Condition` and `Variable` (their hierarchy is depicted on the figure 2.2).

First two elements can be composed to a tree-based template document in order to create a template for an RDF document.

`Condition` and `Variable` can be added to a `PutInstance` node and they drive a production of RDF instances.

Figure 2.2: The template language elements



All these elements reference corresponding elements from a particular schema mapping document using attribute `name`, which links with a `templateName` attribute in schema mapping elements. Using SQL queries from mapping elements, a template document can be considered for an RDF view of a part of a source relational database.

Informal semantics of the elements is described with examples in the following paragraphs, their normative syntax in the section 3.2. All following examples are based on the mapping document from the appendix 4.

Element `PutInstance` (template instance)

A `PutInstance` element refers to a particular mapping class. When used in a template document, it selects data from a relational database according to a SQL query from a mapping class. Each tuple (a row) of a returned relation will be transformed into one RDF instance. An RDF instance is generated with all its attributes defined in a corresponding mapping class.

Listing 2.7: Template instance example

```
<PutInstance name="person" />
```

The code fragment from listing 2.7 writes all persons from a table `PERSON` to an RDF document and each produced RDF resource will contain RDF attribute `rdf:about`.

Element `PutProperty` (template property)

A `PutProperty` element refers to a particular mapping property. It is

always embedded in a `PutInstance` element. When used in a template document, it puts an RDF property with RDF attributes into produced RDF resources. In a case a `PutProperty` refers to a datatype mapping property, a value of the property is selected from a relational database (see the listing 2.8). If the element refers to an object mapping property, it must contain another `PutInstance` (this is detailed further) and produced RDF resource(s) will form a value(s).

Listing 2.8: Template property example

```
<putInstance name="person">
  <putProperty name="surname" />
</putInstance>
```

The example in the listing 2.8 will add RDF property `surname` with a corresponding value from a database to each produced instance of the ontology class `Person`.

Element Condition (template condition)

A `Condition` element refers to a particular mapping condition and it is always embedded in a template instance. From a template layer point of view, a template condition restricts tuples of a relation returned by the template instance. Only those tuples are selected (and transformed into RDF instances) that conform the mapping condition.

A template condition element in a template document must have a value, which can be a literal or a `Variable` element, which is a reference to relating template instance. The sample of the former case is the listing 2.9, the latter one is detailed in the `Variable` section.

Listing 2.9: Template condition example

```
<PutInstance name="person">
  <Condition name="username">svihlm1</Condition>
  <PutProperty name="surname"/>
</PutInstance>
```

The condition in the listing 2.9 contains a literal and reduces selected persons to one with the username *svihlm1*.

Element Variable (template variable)

A `Variable` element refers to a particular mapping variable and it is always embedded in a `Condition`. It puts a variable value to a condition. Since a mapping variable exposes a value from an enveloping mapping class, a referring template variable is able to make a join between instances from two corresponding mapping classes.

Listing 2.10: Complex template document example

```

<PutInstance name="person" id="1">
  <Condition name="username">svihlm1</Condition>
  <PutProperty name="surname"/>
  <PutProperty name="currentProject" ContainerNodeId="
    projectBagId">
    <PutInstance name="project">
      <Condition name="personId">
        <Variable id="1" name="personId"/>
      </Condition>
      <PutProperty name="projectHomepage"/>
    </PutInstance>
  </PutProperty>
</PutInstance>

```

The example in the listing 2.10) is most complex and it shows all features of the template language. The sample is based on the previous running example listings, but also on the schema mapping document 4.1 in the appendix 4. The template listing shows a relation between one person and his or her projects. Used variable has an id attribute that refers to the corresponding instance, the instance of the class Person. The result of this template is on the listing 2.11.

Listing 2.11: Resulting RDF

```

<Person rdf:about="http://webing.felk.cvut.cz/people/svihlm1">
  <surname rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Svihla
</surname>
  <currentProject>
    <rdf:Bag rdf:nodeID="projectBagId">
      <rdf:li>
        <Project rdf:about="http://webing.felk.cvut.cz/projects/123">
          <homepage>http://webing.felk.cvut.cz/~svihlm1/metamorphoses/
          </homepage>
        </Project>
      </rdf:li>
    </rdf:Bag>
  </currentProject>
</Person>

```

Chapter 3

Language reference (normative)

3.1 Schema mapping language syntax

3.1.1 General elements

Document prolog

Since the schema mapping language is XML-based, documents must begin with the appropriate XML prolog:

```
<?xml version="1.0"?>
```

Root element

The root element of a mapping document is `Mapping`. All mapping elements described in the following text must be embedded in this root, which has following form.

```
<Mapping xmlns="http://webing.felk.cvut.cz/metamorphoses/mapping/"> content </Mapping>
```

Header and footer of an RDF document

There are two mandatory elements in the mapping document language that refers directly to a produced RDF documents: header element and footer element. They have the following form:

```
<DocumentHeader><![CDATA[content]]></DocumentHeader>
```

resp.

```
<DocumentFooter><![CDATA[content]]></DocumentFooter>
```

content The content of these mandatory elements is put to the beginning (respectively to the end) of every RDF document based on a particular mapping document. The content of these elements is `CDATA`, which stands for character data and means that XML tags can be included.

Database connection

The `DatabaseConnection` element is to establish a connection with a particular RDBMS. The syntax of this mandatory element does not depend on a mapping model design but rather on a framework implementation. When using JDBC driver to connect a database, the syntax of this element can be following:

```
<DatabaseConnection jdbcURL = "jdbcURL "  
                    jdbcDriver = "jdbcDriver "  
                    username = "username "  
                    password="password" />
```

The attributes of the element are self-describing.

3.1.2 Concept elements

Class

The `Class` element is a basic element of the language to express a relation between one or more database tables and corresponding ontology class.

```
<Class templateName="templateName "  
      rdfLabel="rdfLabel "  
      sql="sqlQuery ">  
  Content  
</Class>
```

templateName (mandatory) Specifies a unique identifier of a fragment within a mapping. A mapping fragment is referenced from a template document by this identifier. Identifiers are case-sensitive.

rdfLabel (mandatory) Specifies a name of a relevant ontology class that is referenced by this fragment. A full class name with namespace of a relevant ontology is required.

sqlQuery (mandatory) The attribute contains an SQL query that returns a subset of a database to be mapped by this fragment. The query can be `SELECT`, `VIEW` etc. and it is RDBMS implementation specific.

content Any of fragment or relationship elements can appear in the content of the element `Mapping`. This includes elements `Property`, `Attribute`, `ClassCondition` or `Variable`.

Property

The **Property** element represents an ontology property. It always corresponds with an enveloping mapping class.

In a case of datatype mapping property (property for which the value is a literal) it joins it with a particular database column from a relation returned by a query from an enveloping mapping class. In a case of object mapping property, it only links an ontology property to the template layer.

```
<Property templateName="templateName"
          rdfLabel="rdflabel"
          [sqlName="sqlName"]
          [containerType="BAG | ALT | SEQ | COL"] >
  Content
</Property>
```

templateName (mandatory) Specifies a unique identifier of a fragment within a mapping. A mapping fragment is referenced from a template document by this identifier. Identifiers are case-sensitive.

rdflabel (mandatory) Specifies a name of a relevant ontology property that is referenced by this fragment. A full property name with namespace of a relevant ontology is required.

sqlName (optional) If a mapped ontology property is a datatype property, a value of this attribute is a name of a column from a database result set, returned by SQL query from an enveloping mapping class. If a mapped ontology property is an object property, this attribute is not specified.

containerType (optional) If a mapped ontology property is a object property that can have more values (ie. it is not functional property), this attribute specifies what kind of RDF container will be used for holding the property values. Legal attribute values are **BAG**, **ALT** and **SEQ** that stands for bag, alternative and sequence container type, or **COL** that stands for collection from RDF specification.

content An element **Attribute** can appear in the content of this element.

Attribute

The **Attribute** element is to specify an RDF attribute of an enveloping mapping class or property.

```
<Attribute rdfLabel="rdflabel"
           [prefix="prefix"]
```

```
[sqlName="sqlName"]  
[suffix="suffix"] />
```

rdflabel (mandatory) Specifies a name of an RDF attribute.

prefix (optional) Puts the specified literal at the beginning of an RDF attribute value.

suffix (optional) Puts the specified literal at the end of an RDF attribute value.

sqlName (optional) If specified, it joins this fragment with a column from a database result set, returned by SQL query from an enveloping mapping class. The value returned from a database is put into an RDF attribute value between *prefix* and *suffix* values.

3.1.3 Relationship elements

Condition

The `ClassCondition` element is to drive a use of mapping class elements from a template document. It can serve two purposes:

- It can restrict rows in a result set returned by SQL query from an enveloping mapping class (when *tableString* is not specified).
- It can add more database tables to query (by specifying their list in *tableString* attribute). This means by this element more mapping classes can be joined in a template document.

This is discussed more later, when we describe the template document language.

The syntax of the mapping condition element is following.

```
<ClassCondition templateName="templateName"  
                 whereString="whereString"  
                 [tableString="tableString"] />
```

templateName (mandatory) Specifies a unique identifier of a condition within a mapping. A condition is referenced from a template document by this identifier. Identifiers are case-sensitive.

whereString (mandatory) A value is added to a `WHERE` part of an SQL query from an enveloping mapping class.

tableString (optional) A list of database tables that should be added to a `FROM` part of an SQL query from an enveloping mapping class.

Variable

The **Variable** element is to expose a single value from an enveloping mapping class in order to make a join with other mapping classes in a template document. This is detailed in the section 2.1).

The syntax of the mapping variable element is following.

```
<Variable templateName="templateName"
          sqlName="sqlName" />
```

templateName (mandatory) Specifies a variable name – the unique identifier of a mapping variable within a mapping. A mapping variable is referenced from a template document by this identifier. Identifiers are case-sensitive.

sqlName (mandatory) A value of this attribute is a name of a column from a database result set, returned by SQL query from an enveloping mapping class. A literal returned from a database is a value of a variable.

3.2 Template language syntax

The template document language is designed to compose templates from which RDF documents are generated.

3.2.1 General elements

Document prolog

Since the template language is XML-based, documents must begin with the appropriate XML prolog:

```
<?xml version="1.0"?>
```

Root element

The root element of a template document is **Template**. Element **Mapping** must be embedded in this root. Another possible child element in the root is only **PutInstance**.

```
<Template xmlns="http://webing.felk.cvut.cz/metamorphoses/template/"> content
```

Reference to a mapping document

A template document is always based on one particular mapping document. This element is to join a template document with its mapping document. The syntax is following:

`<Mapping url="url" />`

url (mandatory) Specifies a location of a corresponding mapping document.

3.2.2 Production elements

PutInstance

Element puts instance(s) of a specified mapping class into a generated RDF document. An RDF instance is generated with all its attributes defined in a corresponding mapping class.

`<PutInstance name="name" [id="id"]> content </PutInstance>`

name (mandatory) Specifies a corresponding mapping class from a mapping document. The attribute corresponds with a fragment attribute *templateName* in a mapping document.

id (optional) In case a variable of the corresponding mapping class is used somewhere in a template document, this specifies identifier for such a variable.

nodeId (optional) In case the instance is a blank node, it is possible to specify a `rdf:nodeId` attribute for this node.

content (optional) Elements `PutProperty` and `Condition` can appear in the content of this element.

PutProperty

Element puts property of an enveloping instance into a generated RDF document. An RDF property is generated with all its attributes defined in a corresponding mapping property.

`<PutProperty name="name"> content </PutProperty>`

name (mandatory) Specifies a corresponding mapping property from a mapping document. The attribute corresponds with a fragment attribute *templateName* in a mapping document.

containerNodeId (optional) In case a referred mapping property is a container, this attribute sets a `rdf:nodeId` attribute for a produced RDF container.

content (optional) The content can be empty in case of a datatype property. In case of a object property `PutInstance` property must be embedded.

Condition

Element puts property of an enveloping instance into a generated RDF document. An RDF property is generated with all its attributes defined in a corresponding mapping property.

```
<Condition name="name"> content </Condition>
```

name (mandatory) Specifies a corresponding mapping condition from a mapping document. The attribute corresponds with a fragment attribute *templateName* in a mapping document.

content (mandatory) The content can be a literal or a *Variable* element.

Variable

This element puts a variable value to a condition.

```
<Variable name="name" id="id" />
```

name (mandatory) Specifies a corresponding mapping variable from a mapping document. The attribute corresponds with a fragment attribute *templateName* in a mapping document.

id (mandatory) Specifies a particular *PutInstance* element, from which the variable comes. The *Variable* element must be an ancestor in a subtree of a corresponding *PutInstance*.

Chapter 4

Schema mapping and template document samples

This chapter contains a complete mapping document, template document example based on the mapping document and resulting RDF document.

4.1 Schema mapping document

The mapping document from listing 4.1 maps the ontology and database schema from running example described in the section 1.1.

Listing 4.1: Mapping document listing

```
<?xml version="1.0" encoding="iso-8859-2"?>
<Mapping xmlns="http://webing.felk.cvut.cz/metamorphoses/
  mapping">

<DatabaseConnection jdbcURL = "jdbc:mysql://localhost/
  mm_sample"
  jdbcDriver = "com.mysql.jdbc.Driver" username = "mm_user"
  password="mm_pass">
</DatabaseConnection>

<DocumentHeader>
  <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
  <rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    xmlns="http://www.sample.org/my/sample/ontology/">
  ]]>
</DocumentHeader>

<DocumentFoot>
  <![CDATA[</rdf:RDF>]]>
</DocumentFoot>
```

```

<Class templateName="person" rdfLabel="Person" sql="SELECT *
  FROM person p, department d WHERE d.id = p.id_department">
  <Attribute rdfLabel="rdf:about"
    prefix="http://webing.felk.cvut.cz/people/" sqlName="
      username"/>
  <ClassCondition templateName="projectId" whereString="p.id
    =
      pp.person_id and pp.project_id =" tableString="
        person_project pp" />
  <ClassCondition templateName="username" whereString="p.
    username =" />
  <Variable templateName="idVariable" sqlName="p.id"/>

  <Property templateName="surname" rdfLabel="surname"
    sqlName="family_name">
    <Attribute rdfLabel="rdf:datatype"
      prefix="http://www.w3.org/2001/XMLSchema#string"/>
  </Property>
  <Property templateName="department" rdfLabel="hasDepartment"
    "
    sqlName="d.name">
    <Attribute rdfLabel="rdf:datatype"
      prefix="http://www.w3.org/2001/XMLSchema#string"/>
  </Property>
  <Property templateName="currentProject" rdfLabel="
    currentProject"
    containerType="Bag" />
</Class>

<Class templateName="project" rdfLabel="Project" sql="SELECT
  * FROM projects pro">
  <Attribute rdfLabel="rdf:about" prefix="http://webing.felk.
    cvut.cz/projects/" sqlName="pro.id"/>
  <ClassCondition templateName="personId" whereString="pro.id
    =
      pp.project_id AND pp.person_id ="
    tableString="person_project pp">
  </ClassCondition>

  <Property templateName="participants" rdfLabel="
    participants"
    containerType="Bag" />
  <Property templateName="homepage" rdfLabel="homepage"
    sqlName="pro.web">
    <Attribute rdfLabel="rdf:datatype"
      prefix="http://www.w3.org/2001/XMLSchema#string" />
  </Property>
</Class>

</Mapping>

```

4.2 Template documents

The following template document is based on the mapping document from the listing 4.1.

Listing 4.2: Template document listing for a particular person

```
<?xml version="1.0"?>
<Template xmlns:mmt="http://webing.felk.cvut.cz/metamorphoses
/template">
<Mapping url="/url/to/mapping/" />

<PutInstance name="person" id="1">
  <Condition name="username">svihlm1</Condition>
  <PutProperty name="surname" />
  <PutProperty name="currentProject" nodeId="projectBagId">
    <PutInstance name="project">
      <Condition name="personId">
        <Variable id="1" name="personId" />
      </Condition>
      <PutProperty name="projectHomepage" />
    </PutInstance>
  </PutProperty>
</PutInstance>

</Template>
```

4.3 Resulting RDF

The RDF listing is the result of the template documents from the previous section.

Listing 4.3: RDF result for the template document 4.2

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-
ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns="http://www.sample.org/my/sample/ontology/">
<Person rdf:about="http://webing.felk.cvut.cz/people/svihlm1
">
  <surname rdf:datatype="http://www.w3.org/2001/XMLSchema#
string">Svihla
</surname>
<currentProject>
  <rdf:Bag rdf:nodeID="projectBagId">
    <rdf:li>
      <Project rdf:about="http://webing.felk.cvut.cz/projects
/123">
        <homepage>http://webing.felk.cvut.cz/~svihlm1/
metamorphoses/
```

```
        </homepage>
    </Project>
</rdf:li>
</rdf:Bag>
</currentProject>
</Person>
</rdf:RDF>
```
