Project Number:	**215219**
Project Acronym:	**SOA4All**
Project Title:	**Service Oriented Architectures for All**
Instrument:	**Integrated Project**
Thematic Priority:	**Information and Communication Technologies**

D3.4.4. WSMO Data Grounding Component

Activity N:	Activity 2 – Core R&D Activities	
Work Package:	WP3 – Service Annotation and Reasoning	
Due Date:	M18	
Submission Date:	14/09/2008	
Start Date of Project:	01/03/2008	
Duration of Project:	36 Months	
Organisation Responsible of Deliverable:	Ontotext	
Revision:	0.6	
Author(s):	Alex Simov	ONTOTEXT
Reviewers:	Adrian Marte	UIBK

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	**X**

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	12.08.2009	Initial version	Alex Simov (ONTOTEXT)
0.2	14.08.2009	Main content added	Alex Simov (ONTOTEXT)
0.3	17.08.2009	Introduction parts added	Alex Simov (ONTOTEXT)
0.4	27.08.2009	Review comments	Adrian Marte (UIBK)
0.5	27.08.2009	Review changes applied	Alex Simov (ONTOTEXT)
0.6	14.09.2009	Final editing	Malena Donato (ATOS)

Table of Contents

List of Figures

Glossary of Acronyms

Acronym	Definition
D	Deliverable
EC	European Commission
WP	Work Package
REST	Representational State Transfer
RDF	Resource Description Framework
XML	Extensible Mark-up Language
OWL	Web Ontology Language
XSLT	Extensible Stylesheet Language Transformation
HTTP	Hypertext Transfer Protocol
WSDL	Web Service Description Language
XPath	XML Path Language

Executive summary

This document provides description of a tool-set for managing data transformations (grounding) between conceptual knowledge data and lower level (XML) data structures. The necessity for these activities (transformations) comes from the need of automated way of communication between semantic-aware agents and plain syntactic WSDL[1]-based web services. The latter use structural XML data for message exchange at execution time, defined by XML Schema[2]. During the communication of a semantic-level client and a syntactic-level web service, two directions of data transformations are necessary: the client's semantic data must be written in an XML form that can be sent as a request to the service, and the response data coming back from the service must be interpreted semantically by the client. The definition of these transformations is done in the form of mappings between semantic classes and XML Schema elements and types. These mappings, in general, cannot be created automatically, therefore a human agent assistance is required.

In this document, we describe a set of services, which facilitate the process of data mapping definitions and usage. The key functionalities supported are suggesting conceptualization of XML schema definitions and initial mappings; generation of two direction data transformations in the form of XSLT[3]; runtime transformation modules for data processing. Another major component, focused in this document, is a graphical end-user tool for definition and editing of mappings between ontology classes and XML Schema elements. The mapping editor is implemented as a lightweight web 2.0 application, which can be used with most of the contemporary web browsers without additional installation costs.

[1] http://www.w3.org/TR/wsdl

[2] http://www.w3.org/XML/Schema

[3] http://www.w3.org/TR/xslt

1. Introduction

1.1 Purpose and Scope

The process of defining proper mapping between conceptual and plain structural data involves a lot of manual work and requires familiarity with varied analytic and implementation technologies. This turns out to be a bottleneck in connecting the semantic and syntactical layers of dataflow. The purpose of this document and its derived work is to deliver means for optimizing the human labour by hiding the technological complexities and highlighting only the spots where domain expert's decision is necessary. What stands behind the users activities and consequences from them, is hidden behind neat and understandable graphical interfaces and in the internal logic of the implementing programming modules.

This document has two major purposes. On the one hand, it serves as technical documentation for the services intended to be utilized by external (machine) agents. On the other hand, it serves as a users guide for the graphical end-user tools. To certain extent, this determines the different description style in the different sections.

1.2 Structure of the document

The document structure is the following:

- Section 2 provides a brief overview and motivation of the problem
- Section 2.1 reveals a set of tools intended to be used at design time when grounding definitions are created. Each tool has a separate sub-section:
    - XML Schema to Ontology generation service – technical specification and usage description of the API of the service
    - Lowering and Lifting schema generation services – API description of a pair of services
    - The Grounding Editor – detailed description of the graphical interface and usage scenario of the mapping editor component
- Section 2.2 concentrates on the description of services, intended to be utilized at runtime, when the actual web service invocation takes place. The APIs of each service is described in a separate section with references to usage examples.

2. WSMO Data Grounding

Traditional (WSDL) web services use XML messages as inputs and outputs, described by XML Schemas as part of the WSDL definitions. This way, the agent communicating with the service has to be familiar with the specific message formats. On the other hand, semantic descriptions improve significantly the automatic discovery and execution process of the services. One important part of bringing together the semantic and syntactic (XML) descriptions of the web services is the availability of two directional data transformations from conceptual instances to XML data and vice versa. The mechanism for bridging the two representation layers is called *data grounding.*

2.1 Design-time Grounding Components

A central role in solving the problem with the mediation between the semantic model and invocation data model is the definition of proper mapping. This involves a lot of labour intensive work, which has to be done before the mapping can be used in practice. Here we describe a set of services and a visual end-user tool, which facilitate the mapping definition process. The result from the mapping definition is a pair of transformations, which can be applied automatically on the data at runtime.

2.1.1 XML Schema to Ontology Generation Service

This functionality of this service aims at facilitating the conceptualisation of XML data coming from (WSDL) web service descriptions. Fully automatic unsupervised ontology extraction from XML data is hard to be achieved for many reasons. Therefore, the service tries to extract as much information as possible and suggest an ontology prototype, which can be refined later by a human, i.e. an ontology expert. The ontology extraction algorithm is inspired by *Mapping XML to OWL Ontologies* [1] article.

The service API is realized as a RESTful[4] service available on the Web. The ontology generation operation can be addressed by a HTTP GET method (even from a web browser) offering a flexible set of input parameters specification. The input for the service is a XSD Schema definition, which can be provided as:

- a schema URL (query parameter *xsd*) in the request URL address
- a schema URL in header parameter *xsd-url*
- schema definition content in header parameter *xsd-data*

Additionally, a *format* header parameter may be used to specify the serialization format of the generated ontology. Valid values are *owl*, *rdf/xml*, *rdf/xml-abbrev*, *n-triple*, *n3*. If the parameter is omitted, the default value is *rdf/xml*.

The result ontology is contained in the body section of the service response message.

Resource/Operation	Method	Description
/genOntology	GET	Generates an ontology from XML Schema (XSD) **params:** *xsd* - the URL of the source XML Schema **header params:**

[4] http://en.wikipedia.org/wiki/Representational_State_Transfer

		xsd-url - URL of the XML Schema file (alternative to 'xsd' param) *format* - (optional) result ontology format, one of: *owl*, *rdf/xml*, *rdf/xml-abbrev*, *n-triple*, *n3*. Default is *rdf/xml*. *xsd-data* - XSD file content (alternative to 'xsd' and 'xsd-url') **result**: Ontology definition in the selected representation format

2.1.2 Lowering Schema Generation Service

The service generates an XSL transformation description, which can be used for transforming (lowering) semantic model instances into XML data. As input, the service receives a simple XML file containing references to the ontology and the XML schema used for mapping and a set of correspondences between the ontology classes and the XML schema elements. An example of such mapping description is available in the Appendix A of this document.

The service functionality is accessible through a RESTful API offering flexible input parameters specification. To invoke the transformation generation, the user must use a HTTP GET method. The input mapping descriptor can be provided as:

- a mapping URL (query parameter *mapping*) in the request URL address
- a mapping URL in header parameter *mapping*
- mapping definition content in header parameter *mapping-data*

The *format* header parameter is essential for the XSLT generation, specifying the input serialization format. Acceptable values are: *rdf* (stands for ‘rdf/xml’) and *owl*. Optionally, the user can provide ontology (*onto-url*) and XML schema (*xsd-url*) specification to overwrite the ones contained in the mapping descriptor.

The result lowering transformation is contained in the body section of the service response message.

Resource/Operation	Method	Description
/genLoweringSchema	GET	Generates a lowering schema between ontology and XML Schema instances. The mapping is provided as an input parameter in XML format. **params:** *mapping* – the URL of the XML mapping descriptor file **header params:** *mapping* - the URL of the XML mapping descriptor file (alternative to ' mapping ' query param) *format* - input ontology format: *owl* or *rdf*. Default is *rdf*.

		xsd-url (optional) – the URL of XML Schema file which should be used, instead of the one specified in the mapping file *onto-url* (optional) – the URL of ontology file which should be used, instead of the one specified in the mapping file *mapping-data* – XML mapping file content (alternative to 'mapping' parameters') **result:** Lowering XSL Transformation file

2.1.3 Lifting Schema Generation Service

The service generates an XSL transformation description, which can be used for transforming (lifting) XML data into semantic model instances. The result of the service invocation is the reverse transformation of the lowering transformation. The service has many features in common with the *Lowering schema generation service* so some descriptions will not be shorter here.

As input, it receives a mapping descriptor, containing references to the ontology, the XML schema, and a set of correspondences between the ontology classes and the XML schema elements.

Again, the service invocation is done by a (RESTful) HTTP GET method call. The input mapping descriptor can be provided as:

- a mapping URL (query parameter *mapping*) in the request URL address
- a mapping URL in header parameter *mapping*
- mapping definition content in header parameter *mapping-data*

Optionally, ontology (*onto-url*) and XML schema (*xsd-url*) specification can be provided to overwrite the ones contained in the mapping descriptor.

The result lifting transformation is contained in the body section of the service response message.

The lifting generation service provides an extra functionality, which includes the functionality available in the ontology generation service. The result of this is, on the base of input XML schema, generation of ontology, mapping descriptor and lifting XSL transformation. Description of the API is provided below (operation **/genLiftingMap**). The outcome of this operation might be a useful input for using *The Grounding Editor* component, providing the initial mapping prototype.

Resource/Operation	Method	Description
/genLiftingSchema	GET	Generates a lifting schema between XML Schema and ontology instances. The mapping is provided as an input parameter in XML format. **params:** *mapping* – the URL of the XML mapping descriptor file

		header params: *mapping* - the URL of the XML mapping descriptor file (alternative to ' mapping ' query param) *xsd-url* (optional) – the URL of XML Schema file which should be used, instead of the one specified in the mapping file *onto-url* (optional) – the URL of ontology file which should be used, instead of the one specified in the mapping file *mapping-data* – XML mapping file content (alternative to 'mapping' parameters') **result:** Lifting XSL Transformation file
/genLiftingMap	GET	Generates an ontology from XML Schema (XSD) and an XSL Transformation from XML data to OWL instances from the generated ontology **query param:** *xsd* – the URL of the XML Schema file **header params:** *xsd-url* – the URL of the XML Schema file (alternative to 'xsd' param) *format* - (optional) result ontology format, one of: *owl*, *rdf/xml*, *rdf/xml-abbrev*, *n-triple*, *n3*. Default is: *rdf/xml* *xsd-data* - XSD file content (alternative to 'xsd' and 'xsd-url') **result:** ZIP archive containing the ontology, XML mapping descriptor and the XSL Transformation

2.1.4 The Grounding Editor

The *Grounding Editor* is part of the Dashboard of SOA4All Studio, delivered by WP2. Its primary task is to allow the user to easily specify correspondences between XML elements defined as XSD schema and ontological data elements. On the base of such specification, corresponding lifting and lowering transformations are generated.

The editor component can be activated by opening the Dashboard main application[5] and selecting the **Grounding Editor** item from the main menu (Figure 1)

[5] Available online and regularly updated at: http://coconut.tie.nl:8080/SOA4All

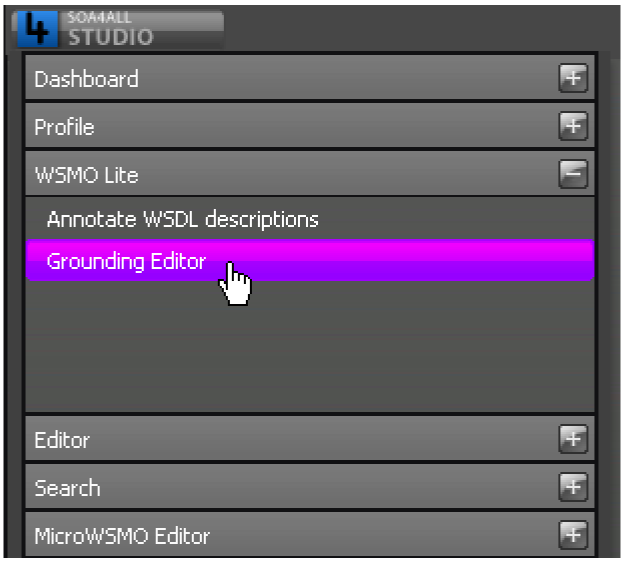


Figure 1 Dashboard main menu

This leads to opening the editor component on top of other applications in the web browser window. The editor layout consists of a number of important components, which will be described in the following paragraphs (Figure 2).

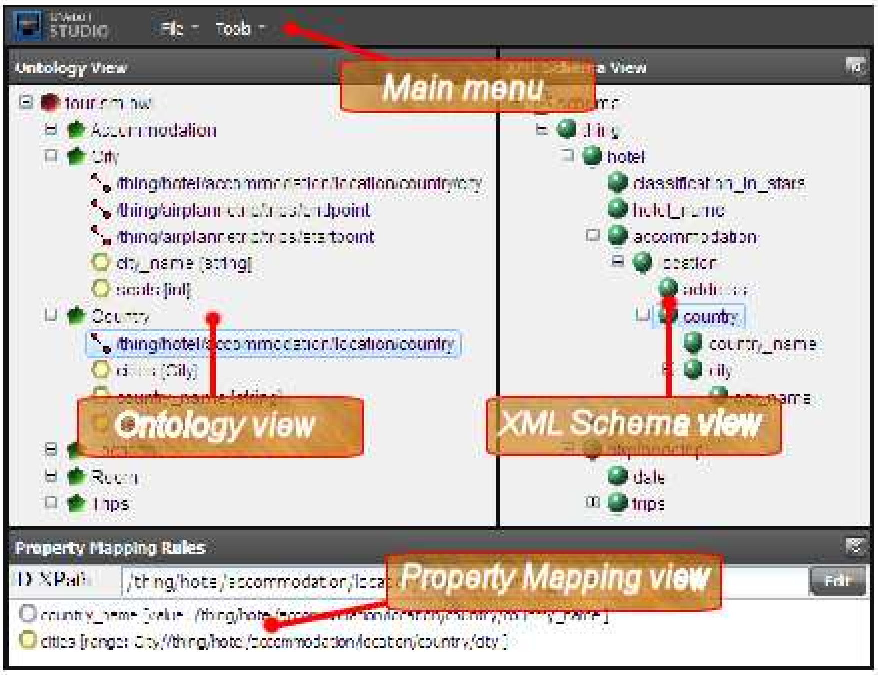


Figure 2 Grounding Editor main components

The most essential part of the editor is the ***Ontology View***, which reveals the hierarchical structure of an ontology, showing its classes, inheritance relations between them and belonging properties. Additionally, the tree structure might contain mapping information, which defines the correspondences between classes and XSD elements. Each class mapping definition consists of a class URI and an XPath[6] expression, locating the

[6] http://www.w3.org/TR/xpath

corresponding element in the XML data. These mapping are represented in the tree as leaf nodes attached to certain classes and labelled by the XPath expression value.

The ***XML Schema View*** is the second important component, which visualizes the elements and types definition structures from XSD schema. The representation is filtered from any technical particularities leaving only the relevant for the mapping specification parts. On the GUI layer, the user is offered a set of tree structures, generated on the base of *part-of* relations between elements (in contrast to the ontology tree structure).

The definition of correspondences between classes and elements by itself is not sufficient when transforming ontological instances into XML data and vice versa. Therefore, the mapping specification is extended by definition of class property correspondences. Such property mappings are handled by the ***Property Mapping View***. The property mappings are classified into two categories depending on the range type of the properties themselves, each having a set of characteristics. The *datatype property mappings* consist of pairs of property URI and XPath expression (pointing to the property value in the XML data). The *Object property mappings* are characterized by pairs of property URI and range class-mapping rule (containing range class URI and corresponding XPath reference).

Depending on the selection in the ontology view, the content of the view is updated dynamically allowing the user to explore or modify the set of mappings.

The ***Main menu*** supports the basic storage and retrieval operations for the editor. The *File* section (Figure 3) enables the user to:

- create a new mapping project between an ontology and XSD schema
- open an existing mapping in the editor
- save the current state of the mapping being edited
- view and modify mapping project properties (ontology or XSD schema URLs)
- close the editor’s content without storing the latest changes (if any)

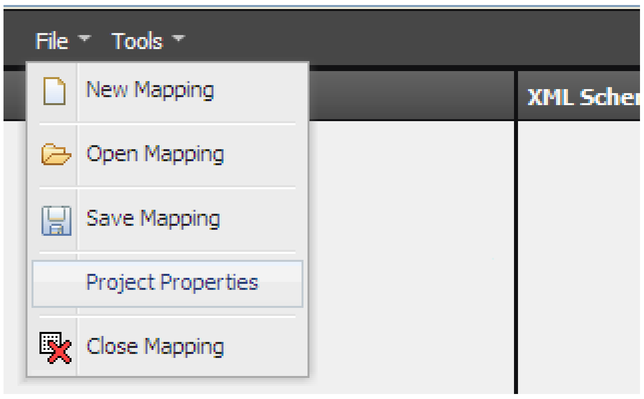


Figure 3 File menu

The *Tools* section (Figure 4) plays an important role for the deployment of the outcomes of the users work with the editor. The provided actions represent a visual front-end of the (lifting or lowering) transformation generation services. The input for these services is formed on the base of the editor’s content and on successful service invocation completion, the results are deployed at user provided URL, ready to be used by the runtime transformation services.

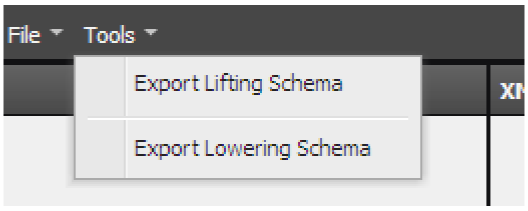


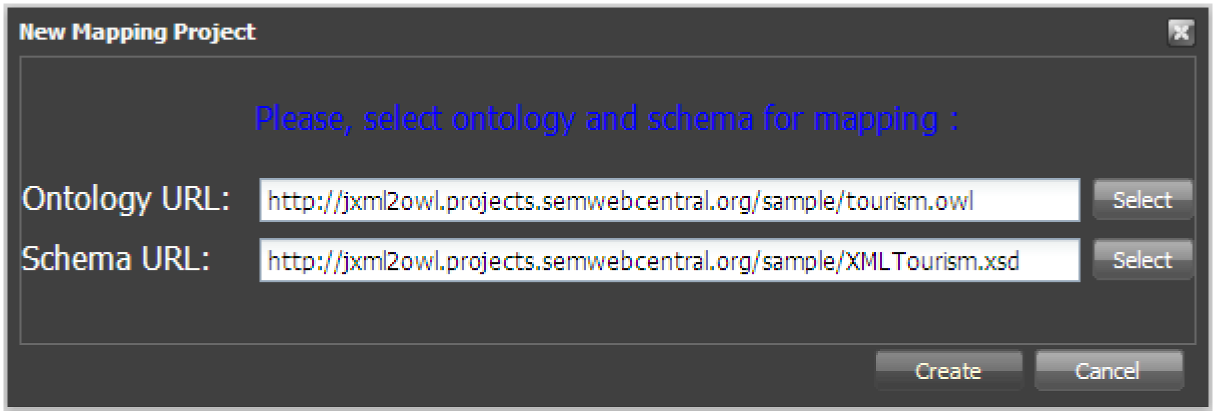
Figure 4 Tools menu

Usage of the *Grounding Editor*

The following section describes a common usage scenario of the editor presenting the full lifecycle of grounding creation. The final result will be a set of transformation definitions, capable to automatically translate XML data into ontological instances and vice versa.

Prerequisites for using the editor are clear idea of the data, which has to be mapped (ontology and xml schema) and its availability on the Web.

The first step in defining mapping is the creation of a *New Mapping* project in the editor. This requires the user to provide the two main ingredients – URLs to the ontology and the XSD definition (Figure 5).


New Mapping Project

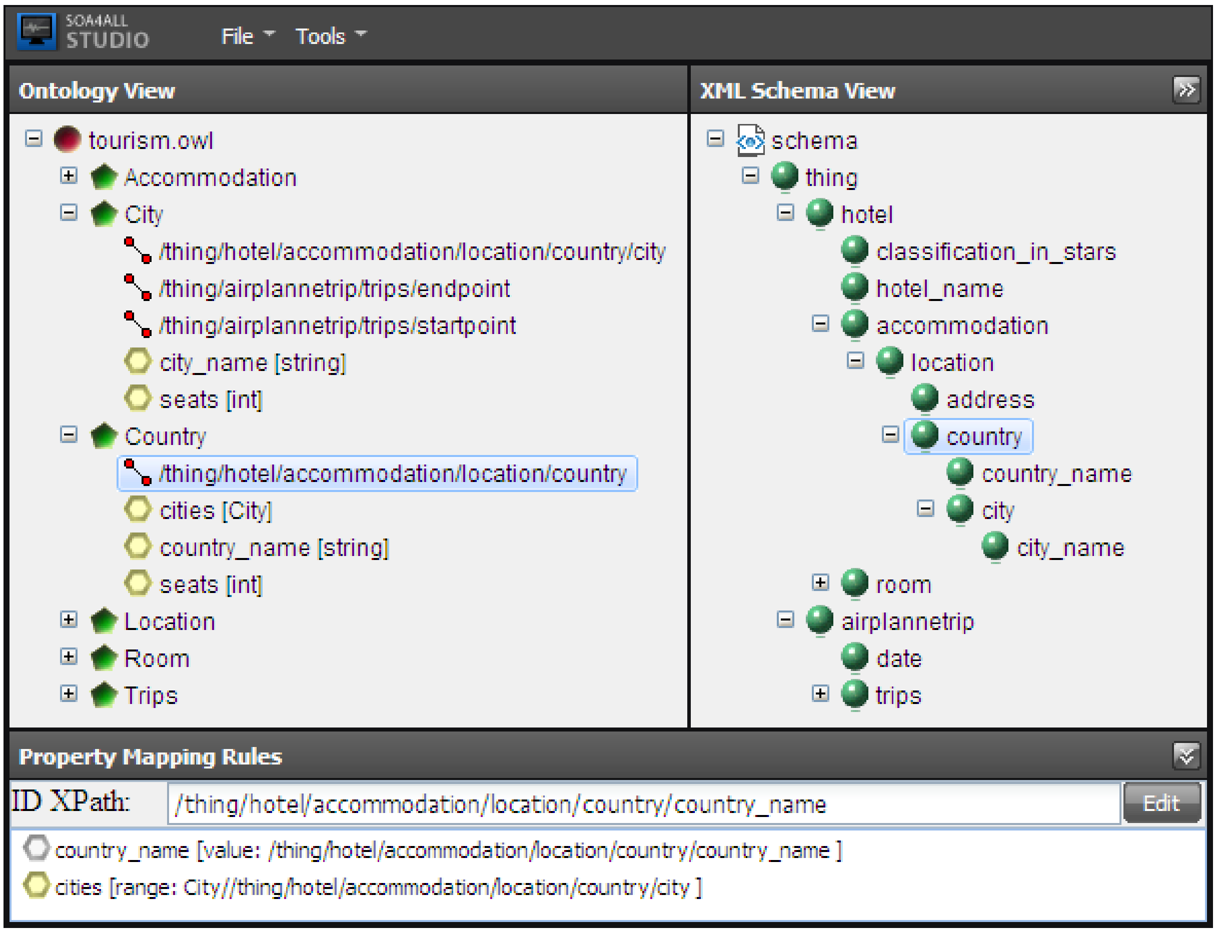
Please, select ontology and schema for mapping :

Ontology URL: http://jxml2owl.projects.semwebcentral.org/sample/tourism.owl

Schema URL: http://jxml2owl.projects.semwebcentral.org/sample/XMLTourism.xsd

Figure 5 New mapping project

Having confirmed the creation of the project, the two definitions are loaded and visualized in the editor (Figure 6) /for demonstrative purposes the figure already contains several mappings/.



```
tourism.owl
  Accommodation
  City
    /thing/hotel/accommodation/location/country/city
    /thing/airplannetrip/trips/endpoint
    /thing/airplannetrip/trips/startpoint
    city_name [string]
    seats [int]
  Country
    /thing/hotel/accommodation/location/country
    cities [City]
    country_name [string]
    seats [int]
  Location
  Room
  Trips
```

```
schema
  thing
    hotel
      classification_in_stars
      hotel_name
      accommodation
        location
          address
          country
            country_name
            city
              city_name
      room
    airplannetrip
      date
      trips
```

ID XPath: /thing/hotel/accommodation/location/country/country_name

country_name [value: /thing/hotel/accommodation/location/country/country_name]
cities [range: City//thing/hotel/accommodation/location/country/city]

Figure 6 Grounding Editor

The editor uses the following images for representing the various types of objects (detailed information is available as tooltip balloon messages on the visual nodes):

●	Ontology root node
⬟	Class node
⬡	Property node or object property mapping rule
⬡	Datatype property mapping rule
⟨⟩	XML Schema node
●	Schema element node
●	Schema attribute node
⤡	Class-to-element mapping rule

The creation of new class-to-element mappings is done by dragging an element from the

XML Schema View and dropping it on the corresponding ontology class. This results in a new node ( ) in the ontology tree, descendant of the selected class, labelled by automatically generated, on the base of the selected schema element, XPath expression.

A removal operation on such mappings can be applied by a context menu action in the tree.

Each class-to-element mapping might have one or more property mapping rules assigned to it. The management of the property mappings is done in a separate view (Figure 7) which updates its content on the base of the selection in the ontology tree. The property mappings do not have visual representation in the ontology tree in order to keep the representation more readable when bigger ontologies are used.

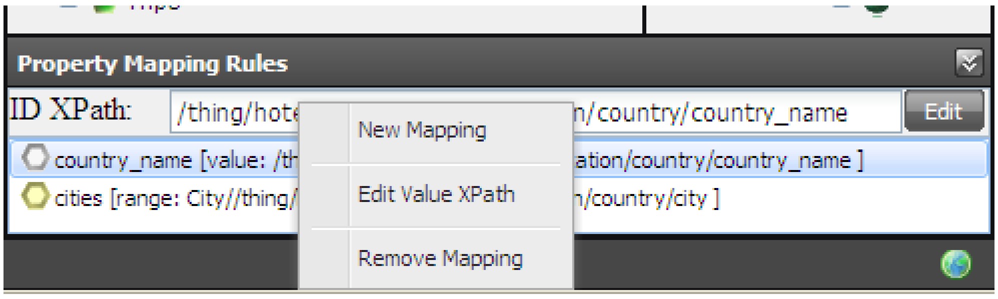


Figure 7 Property Mappings view

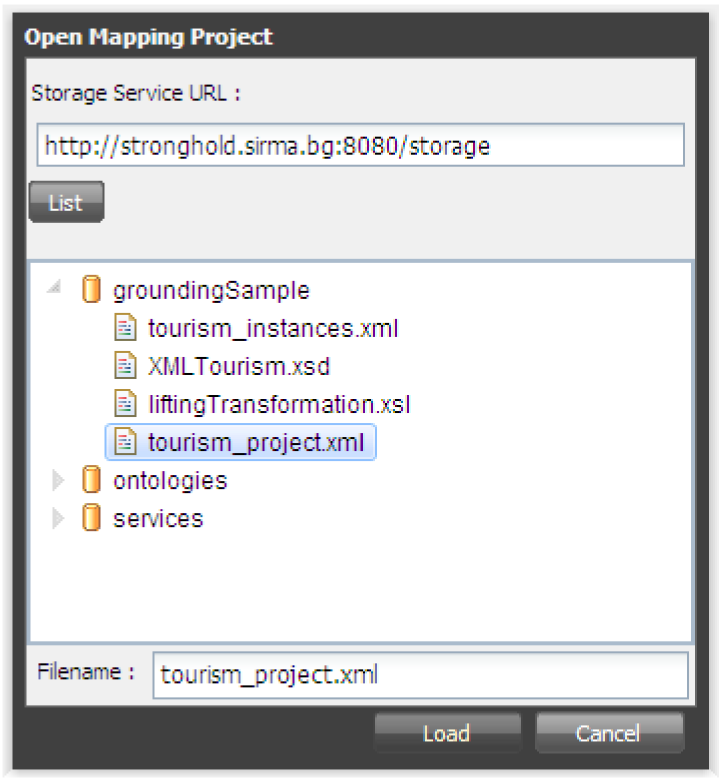
In the *Property Mapping* Rules view, the user can use context menu actions to:

- create new property mapping rules
- edit properties of existing rules
- remove a property mapping rule

The system assists the user by filtering the content displayed by using the conceptual knowledge from the ontology (for example, suggesting only defined for the class or inherited properties).

At any time, the user can save their work by using ***File > Save Mapping*** menu item, which stores the current mapping in a remote repository supported by the *Storage Services* delivered by WP2 (D2.4.2). The visual support (simple repository browser (Figure 8)) is provided by (reused from) the *WSMO-Lite Editor* (D2.1.3), which facilitates the user to organize his/hers storage data.

The work on a mapping project can be resumed by loading a mapping project from the repository (***File > Open Mapping***). This is supported by the (WSMO-Lite Editor's) repository browser as well (Figure 8).



Storage Service URL : http://stronghold.sirma.bg:8080/storage

groundingSample
tourism_instances.xml
XMLTourism.xsd
liftingTransformation.xsl
tourism_project.xml
ontologies
services

Filename : tourism_project.xml

Figure 8 Repository browser

Exporting Lowering and Lifting Transformations

When the specification of the mapping is completed, it is still not ready to be deployed for runtime usage. There is one final step left being performed, which is the generation of lifting and lowering transformation schemas derived from the mapping specification. The transformations themselves are expressed in XSLT[7], which is the most appropriate transformation language for XML data (the ontological instances are serialized in RDF/XML as well).

The generation of XSL Transformations can be initiated from the actions of the *Tools* menu (Figure 4 Tools menu). The user has to provide deployment URLs for the results, which can be reused later while creating SA-WSDL[8] annotations in the WSMO-Lite editor on web service descriptions. The specification of deployment locations is supported by the repository browser from Figure 8 Repository browser

2.2 Runtime Grounding Components

At service execution time, there is a need for intensive switching between semantic layer data and concrete service XML messages. The relations between those two types of data are defined in advance at design-time in the form of bidirectional transformations. These transformations are applied any time instances from a certain layer have to be transformed (lifted or lowered) in instances of the other one.

[7] http://www.w3.org/TR/xslt

[8] http://www.w3.org/2002/ws/sawsdl/

This functionality is supported by two services available on the Web. The APIs of the services are realized in RESTful style[9], which has plenty of advantages (performance, scalability, portability, simplicity and others). The following sections describe each service in details, providing in table form the API specification.

2.2.1 Data Lifting Service

This service is responsible for transforming XML data instances (coming from messages as a result of web service invocations) into ontology data instances.

The core functionality is supported by an XSLT processor, which executes the transformation procedure and the result is adapted according to the input parameters. The result can be represented as OWL or RDF depending on the needs.

The invocation of the service is done by a HTTP POST method call to the service deployment URL. The XML data, which has to be *lifted,* is contained in the body of the HTTP request. The request header must contain two important attributes: URL of the (XSL) transformation schema and output *format* for the generated instances (valid values are *owl*, *rdf/xml*, *rdf/xml-abbrev*, *n-triple*, *n3*).

Usage Java example of the lifting service (snippet) is included in Annex B of this document.

Resource/Operation	Method	Description
/doLifting	POST	Performs lifting from XML instances to OWL/RDF instances **header params**: *xslt-url* - the location of the lifting XSLT document *format* - the desired output format **result**: translated OWL or RDF instances

2.2.2 Data Lowering Service

The lowering is the reverse operation of lifting. The service supporting this functionality is responsible for transforming the ontological instances into XML data. One requirement of the service is that the input for the service has to be represented in XML based format (RDF/XML or OWL)

Similarly to the previous service, the core functionality is supported by an XSLT processor, which executes the transformation procedure. The result is an XML document containing the *lowered* instances.

The invocation of the service is done by a HTTP POST method call to the service deployment URL. The RDF or OWL data, which has to be *lowered*, is contained in the body of the HTTP request. The request header must contain the URL of the (XSL) transformation. The result XML data is returned in the body of the response message.

[9] http://en.wikipedia.org/wiki/Representational_State_Transfer

Resource/Operation	Method	Description
/doLowering	POST	Performs lowering from OWL/RDF instances to XML instances **header params**: *xslt-url* - the location of the lowering XSLT document **result**: translated XML instances

3. Conclusions

In this document, we provide description of a set of services, which cover many aspects of the WSMO data grounding. Implemented as RESTful services, they provide:

- means for generating conceptual models from XML schemas, lifting and lowering transformation schemas (in XSLT)
- capabilities for runtime transformations between XML data and conceptual intances in both directions

Additionally, we provide a graphical tool for editing and managing mapping descriptions – the *Grounding Editor*. Available as a web application it provides comprehensive data representation, filtering the technical particularities of the underlying technologies. The editor supports creating new mappings, editing existing ones, generation and deployment of lifting and lowering transformations.

In the following stages of the project, the services will be further refined on the base of usage feedback and also some effort will be invested in improvement of the usability aspects of the visual tools.

4. References

1. Bohring, H. Auer, S. *Mapping XML to OWL Ontologies*, LIT 2005 (21-23 September 2005), Leipzig

Annex A. Ontology to XML mapping descriptor (fragment)

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping>
    <xmlURI>http://jxml2owl.projects.semwebcentral.org/sample/XMLTourism.xsd</xmlURI>
    <owlURI>http://jxml2owl.projects.semwebcentral.org/sample/tourism.owl</owlURI>
    <owlPrefix>tourism</owlPrefix>
    <associations>
    <class>
      <owlClassName>
         http://jxml2owl.projects.semwebcentral.org/sample/tourism.owl#AirplaneTrip
      </owlClassName>
      <elementXPath>/thing/airplannetrip</elementXPath>
      <IdXPath>/thing/airplannetrip</IdXPath>
    </class>
    <class>
      <owlClassName>
        http://jxml2owl.projects.semwebcentral.org/sample/tourism.owl#Location
      </owlClassName>
      <elementXPath>/thing/airplannetrip/trips/startpoint</elementXPath>
      <IdXPath>/thing/airplannetrip/trips/startpoint</IdXPath>
    </class>
    …………
    <datatypeProperty>
      <owlPropertyName>
         http://jxml2owl.projects.semwebcentral.org/sample/tourism.owl#address
     </owlPropertyName>
     <owlDomainClassName>
        http://jxml2owl.projects.semwebcentral.org/sample/tourism.owl#Location
     </owlDomainClassName>
     <domainXPath>/thing/hotel/accommodation/location</domainXPath>
     <valueXPath>/thing/hotel/accommodation/location/address</valueXPath>
   </datatypeProperty>
    ……………
   <objectProperty>
     <owlPropertyName>
```

```
      http://jxml2owl.projects.semwebcentral.org/sample/tourism.owl#location
    </owlPropertyName>
    <owlDomainClassName>
      http://jxml2owl.projects.semwebcentral.org/sample/tourism.owl#Hotel
    </owlDomainClassName>
    <domainXPath>/thing/hotel</domainXPath>
    <owlRangeClassName>
       http://jxml2owl.projects.semwebcentral.org/sample/tourism.owl#Location
    </owlRangeClassName>
    <rangeXPath>/thing/hotel/accommodation/location</rangeXPath>
  </objectProperty>
  </associations>
</mapping>
```

Annex B. Example code snippet for lifting XML instances

```
public static void main(String[] args) throws Exception {
    …..
    String xsltURL = …
    String xmlDataURL = …
    liftXMLInstances(xsltURL, xmlDataUrl, "rdf/xml");
    …..
}

public static void liftXMLInstances(String xsltURL, String xmlDataUrl, String format) {

    String LIFTING_SERVICE_ENDPOINT = "http://localhost:8080/liftingService";
    String requestURL = LIFTING_SERVICE_ENDPOINT + "/doLifting";
    PostMethod postMtd = new PostMethod(requestURL);
    HttpClient httpclient = new HttpClient();
    try {
        postMtd.addRequestHeader("xslt-url", URLEncoder.encode(xsltURL, "UTF-8"));
        postMtd.addRequestHeader("format", URLEncoder.encode(format, "UTF-8"));
        postMtd.setRequestEntity(
                new ByteArrayRequestEntity(readXMLData(xmlUrl), "xml/application"));
        int result = httpclient.executeMethod(postMtd);
        if (result != 200) {
            System.out.println("Error message: ");
            System.out.println(postMtd.getResponseHeader("Error"));
        }
        else {
                System.out.println("Response OK");
                System.out.println(postMtd.getResponseBodyAsString());
        }
    } finally {
        postMtd.releaseConnection();
    }
}
```