SWAD–Europe Deliverable 6.3a Description of prototype implementation (documentation for deliverable 6.2)

Building knowledge objects from disparate, related resources

Project name:
Semantic Web Advanced Development for Europe (SWAD-Europe)
Project Number:
IST-2001-34732
Workpackage name:
6. XML and Semantic Web Integration Research Prototypes
Workpackage description:
☞http://www.w3.org/2001/sw/Europe/plan/workpackages/live/esw-wp-6.html
Deliverable title:
6.2 Description of prototype implementing the use of RDF and other XML syntaxes

Author:
☞Martin Pike, Stilo, UK.

Abstract:
This document describes the architecture and the techniques brought to the problem of unifying significantly different types of information encoded in different formats and residing in potentially different repositories. The particular domain is engineering information.
Status:
Completed report, 2004-06-06

Contents

1. Introduction

This document describes the architecture and the techniques brought to the problem of unifying related but different different types of information encoded in different formats and residing in potentially different repositories. The goal of the demonstrator is to present to the user a page of information built on-demand from independent information fragments The particular domain is engineering information.

A code base for this application is available[1]

The use case for this demonstrator is described in Work Package 6.2[2]

The demonstrator illustrates how RDF encoded data can be transformed into relatively simple XML, delivered to the desktop and processed there to provide a rich user interface. It shows how the user may enter data for RDF encoding. It also illustrates the exchange of XML with the server to provide instructions on which resources are required and how they are to be displayed.

In this demonstrator textual, graphical and mathematical information are pulled together for display. The demonstrator will:

- Present a user interface that incorporates a dynamic forms layout, allowing users to enter data in a form that can be encoded in RDF on the server.
- Use a relatively simple XML format as an interchange format between the browser and the server.
- Use an RDF-enabled web server, incorporating Apache Tomcat and JENA to store and serve

RDF-based information and the knowledge resources.

- Present updated RDF information via a template for display in a browser.
- Display formatted knowledge objects, containing the integrated data fragments and delivered through a presentation template.

2. The user interface

The prototype makes extensive use of XML syntaxes throughout its implementation, including the user interface. It was determined at an early stage that it was necessary to deliver XML content to the browser and render it on the client, rather than server-side generated HTML.

This decision allowed a rich, dynamic interface to be constructed. This solution also provides good cross-browser compatibility and potential delivery of information to different devices and other applications. An example can be seen in the screenshot(s) below.
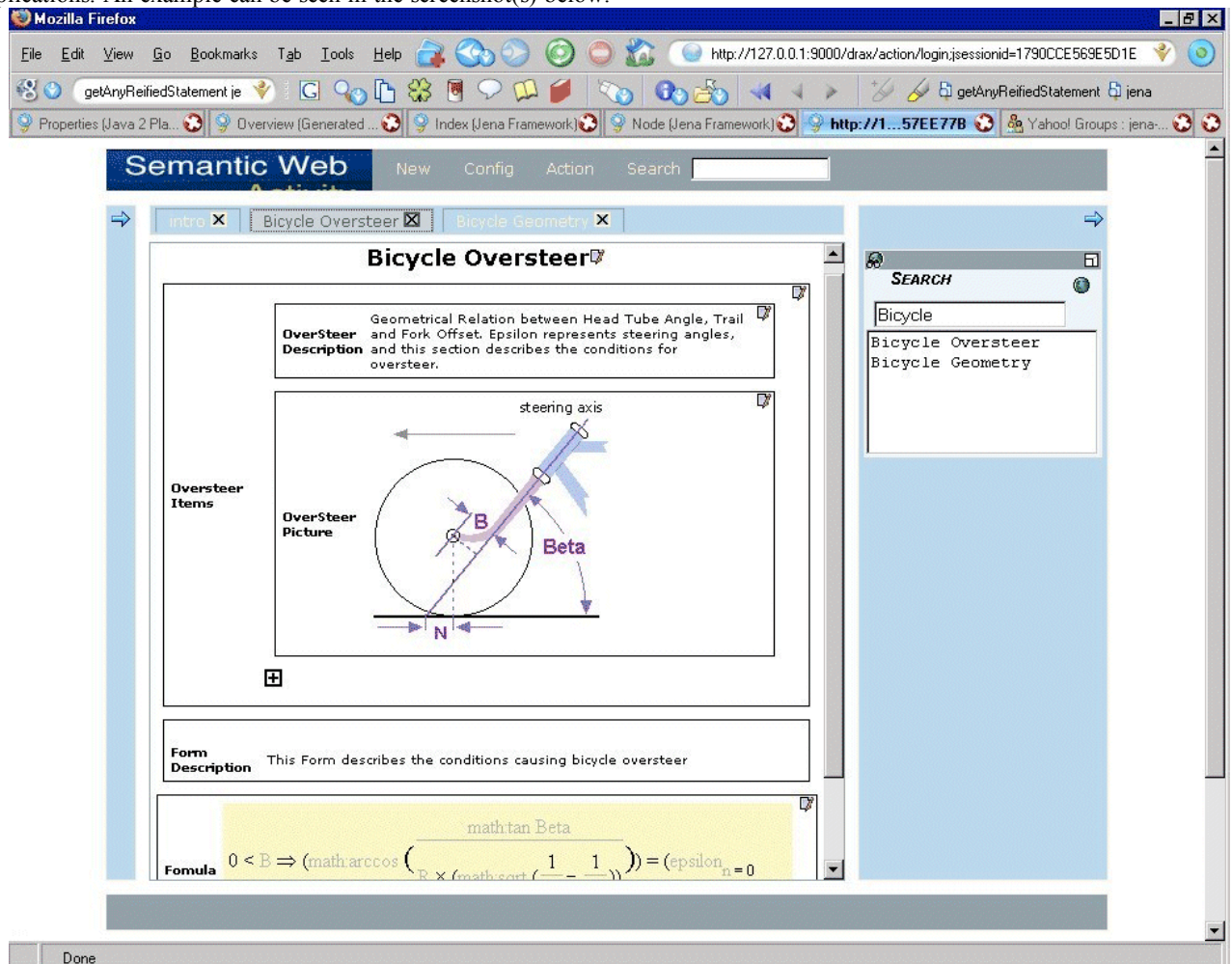

Figure 1. Screenshot of displayed knowledge object

The user is presented with a multi-faceted interface, incorporating frames and tabbed views. The intention of the interface is to provide a central control "panel" either for display or as a form for data entry. Other panels can then supply supplementary information, some of which could be incorporated into the data entry. For instance, a side panel may be used to show a list of images, or text pieces. Selection of one of these could cause that URL to become part of the input information for the form. Alternatively, Figure 1. shows one of the side panels being used to display a search pane.

The tabs may be used to provide different views on the information set. For example, while creating some instance data about a particular object the user may realise that the model they are working with does not allow a particular attribute to be captured. Without losing context, the user may change tabes (or add a new one) to display a class and/or property creation form. They may then augment the model, commit the change to the server and return to instance data entry. They may then take advantage of the extensions to the model and enter values for the new type of attribute.

The user interface is driven completely by XML content, XSLT stylesheets and DHTML/Javascript. A requirement of the development is simple deployment as part of an enterprise application. The use of Java applets was considered for the data manipulation and logic, but with organisations increasingly

'locking down' desktops, with only certain types of applications allowed to be installed, it was felt that the DHTML option would be more acceptable and could be used to achieve the same result.

3. XML Exchange Format

Although it is possible to render RDF as HTML directly, it was decided for a number of reasons to provide an intermediate format. The principal reason was to minimise the complexity of the client application. With only one well-defined format delivered to the client; one stylesheet is required to produce the interface.

Another criterion is that, although RDF is represented as XML and is becoming increasingly well-known and understood, the majority of developers in commercial organisations do not understand it and many are "frightened" by it. By keeping the RDF on the server and transforming it into a more familiar pattern of syntax, the encoded data may be made accessible to more applications developers and therefore easing and speeding application development. Relatively simple XSLT transforms are used on content encoded in this syntax to produce the forms and display interfaces.

The interchange format has been named "Linear Input Notation for Knowledge" (LINK). The schema is illustrated below:

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
        <xs:element name="class">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="subClassOf" minOccurs="0" maxOccurs="unbounded"/>
                                <xs:element ref="className"/>
                                <xs:element ref="properties" minOccurs="0"/>
                        </xs:sequence>
                        <xs:attribute name="uri" type="xs:string"/>
                </xs:complexType>
        </xs:element>
        <xs:element name="className" type="xs:string"/>
        <xs:element name="formula" type="xs:string"/>
        <xs:element name="label" type="xs:string"/>
        <xs:element name="literal" type="xs:string"/>
        <xs:element name="object">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="label" minOccurs="0"/>
                                <xs:element ref="objectName" minOccurs="0"/>
                                <xs:element ref="objectType"/>
                                <xs:element ref="properties" minOccurs="0"/>
                                <xs:element ref="rules" minOccurs="0"/>
                        </xs:sequence>
                        <xs:attribute name="uri" type="xs:string"/>
                </xs:complexType>
        </xs:element>
        <xs:element name="objectID" type="xs:string"/>
        <xs:element name="objectName" type="xs:string"/>
        <xs:element name="objectType">
                <xs:complexType>
                        <xs:simpleContent>
                                <xs:extension base="xs:string">
                                        <xs:attribute name="uri" type="xs:string"/>
                                </xs:extension>
                        </xs:simpleContent>
                </xs:complexType>
        </xs:element>
        <xs:element name="properties">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="property" maxOccurs="unbounded"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
        <xs:element name="property">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="propertyName"/>
                                <xs:element ref="propertyType" minOccurs="0"/>
                                <xs:element ref="propertyValue" maxOccurs="unbounded"/>
                        </xs:sequence>
                        <xs:attribute name="uri" type="xs:string"/>
                </xs:complexType>
        </xs:element>
        <xs:element name="propertyName" type="xs:string"/>
        <xs:element name="propertyType">
                <xs:complexType>
                        <xs:simpleContent>
                                <xs:extension base="xs:string">
```

```
                        </xs:extension>
                    </xs:simpleContent>
                </xs:complexType>
        </xs:element>
        <xs:element name="propertyValue">
                <xs:complexType>
                    <xs:choice>
                            <xs:element ref="literal"/>
                            <xs:element ref="formula"/>
                            <xs:element ref="object"/>
                            <xs:element ref="objectID"/>
                    </xs:choice>
                </xs:complexType>
        </xs:element>
        <xs:element name="rule">
                <xs:complexType>
                    <xs:sequence>
                            <xs:element ref="ruleType"/>
                            <xs:element ref="rulePart" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
        </xs:element>
        <xs:element name="rulePart">
                <xs:complexType>
                    <xs:sequence>
                            <xs:element ref="rulePartName"/>
                            <xs:element ref="rulePartValue"/>
                    </xs:sequence>
                </xs:complexType>
        </xs:element>
        <xs:element name="rulePartName" type="xs:string"/>
        <xs:element name="rulePartValue" type="xs:string"/>
        <xs:element name="ruleType" type="xs:string"/>
        <xs:element name="rules">
                <xs:complexType>
                    <xs:sequence>
                            <xs:element ref="rule" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
        </xs:element>
        <xs:element name="sophxdoc">
                <xs:complexType>
                    <xs:choice maxOccurs="unbounded">
                            <xs:element ref="object"/>
                            <xs:element ref="class"/>
                    </xs:choice>
                </xs:complexType>
        </xs:element>
        <xs:element name="subClassOf" type="xs:string"/>
</xs:schema>
```
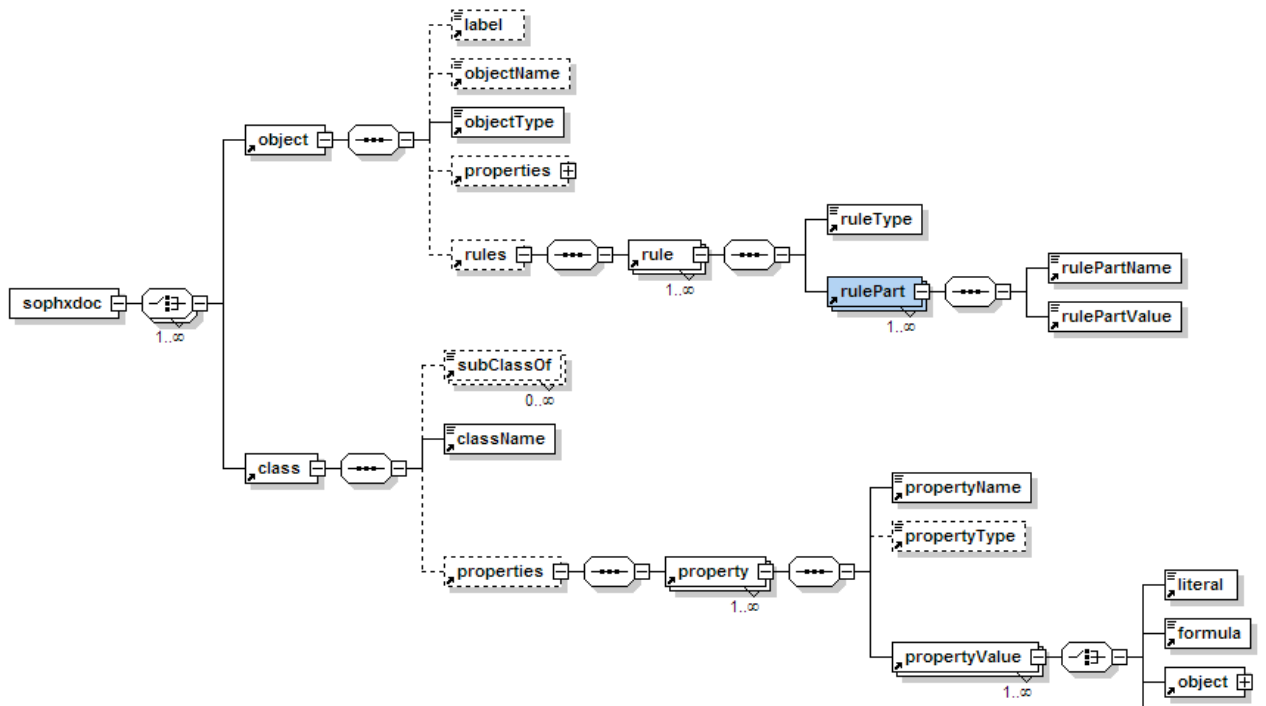
Figure 2. The LINK schema

objectID

Figure 3. Diagram of the LINK schema

The schema splits under the root into two parts object and class. This split allows forms to be built from documents conforming to this schema to be capturing instance data or class/property data.

XSLT stylesheets have been developed to display data of both types in the user interface.

4. RDF-enabled web server

The server for the prototype is constructed using Apache Tomcat, the JENA Semantic Web Framework in conjunction with the Apache Struts MVC Framework. The server accepts LINK documents and uses the JENA RDF API to convert and store the content as RDF.

Tomcat acts as a basic web server. The content used in the display of knowledge objects is accessed via Tomcat, with URIs derived from RDF encoded data processed via the JENA framework.

5. RDF information via a template for display in a browser.

The prototype is able to display hyperlinked LINK data, so that users may navigate the RDF model, with data being fed in real-time. At any point a page of information may be "switched" into edit mode for modification.

6. Display formatted knowledge objects

The goal of the prototype is to display an aggregated knowledge object with related information fragments. Templates are used to reference code for accessing the RDF store and then to process the RDF to identify information objects for display within the template.

References

[1]Code for Prototype Knowledge Object Construction
[2]Use case for Knowledge Object Construction
☞"http://www.w3.org/2001/sw/Europe/reports/WP6.2UseCases.htm#Part3
    [3] JENA - A Semantic Web Framework
☞http://www.hpl.hp.com/semweb/jena.htm
    [4] Apache Tomcat
☞http://jakarta.apache.org/tomcat/
    [5] Apache Jakarta Struts
☞http://jakarta.apache.org/struts/
    [6] XSLT ☞http://www.w3.org/TR/xslt