University of New Brunswick

CS6795 - Semantic Web Techniques
Project Proposal

Nov. 7th, 2011

Normalizers for RuleML 1.0 in XSLT 2.0

Instructor: Dr. Harold Boley
Advisor: Dr. Tara Athan

Team 4:
Nada Alsalmi
Leah Bidlake
Ao Cheng
Thea Gegenberg
Emily Wilson

Introduction

RuleML is a markup language for sharing rules in XML. It is serialized as an XML tree whose elements alternate between representing classes or types (nodes), and representing methods or roles (edges). An example of such a serialization is in the main rule element <Implies> which can contain <if> and <then> edges, called 'stripes'. Because of XML's left-to-right ordering we can rely on the subelements to tell us the implied position of the <if> and <then> edges therefore we can remove them completely. Such a document with these elements removed is an example of 'stripe-skipped.' In the normalization of a RuleML document we want to transform 'stripe-skipped' serializations back into ones that are fully striped. We also want to assure that the document is in proper canonical order.

A partially built normalizer for RuleML 0.91 queries has been developed by Dr. Tara Athan using an XSLT stylesheet. The current normalizer uses some features from XSLT 2.0. There are several issues with the current normalizer that must be improved upon, as well as other tasks that the normalizer must be able to perform.

An XSLT stylesheet has been developed by Derek Smith for nomalizing the syntax used in RuleML 1.0 instances. Although normalizing RuleML 1.0 instances is also the objective of our normalizer, we will design our normalizer differently. Derek's normalizer is a catch-all template that runs for every tag. It then runs through a large choose statement and matches the tags to their names and checks if the proper role tags are in place. Therefore Derek's normalizer performs the normalization in one pass or phase. Our normalizer will be implemented in 3 phases that will use the previous phase's output as input for that particular phase (e.g. phase 2 will use the output from phase 1). Therefore the normalization will be performed in several passes.

Objectives

Our main objective is to further develop the current normalizer. We will upgrade the normalizer to be used with version 1.0 of RuleML. There are three phases of the transformation that our normalizer will perform. The first will be to fill in the missing edge stripes wherever they are needed in the XML. For example, any <if> and <then> edges that have been skipped. The second is to assure that the elements are in the correct canonical ordering. The third phase is to make all attributes that have default values explicit. We want to successfully implement each of these phases in the upgraded normalizer. We will use example RuleML documents for testing the transformations performed by our normalizer, and the output will be validated against the corresponding schemas. We will also examine the sequencing of our phases, and look at the possibility of implementing them in a different order. Also, a detailed comparison between the way that our normalizer is implemented, and the way that Derek's normalizer is implemented will be made.

Methodology

For phase 1 we will be checking the syntax for missing stripes and adding the missing role tags. We will be using XSLT to check for the missing edge stripes and adding them in where necessary. We will be checking some of the given examples using the normalizer, and then verifying the results with a given schema that defines normal RuleML 1.0.

For phase 2 we will be reordering elements so that their subelements are in canonical order (e.g., slotted arguments after positional arguments). We will make sure that the order of the elements in the outputted RuleML document follows the same order outlined in the normalizer schema. We will build canonically ordered content of each element. For example, we will put the contents of <Implies> into canonical ordering (oid, body, head, followed by all other subelements in the order that they appear). In addition, the order of comments will be preserved.

For phase 3 we will make all attributes that have default values explicit. Having to carefully map the hierarchy of an XML document in an XSLT style sheet may be inconvenient if the document does not follow a stable, predictable order like the periodic table. Hence, we need general rules that can find an element and apply templates to it regardless of where it appears in the source document. To make this process easier, we make the attributes with default values explicit in a (Deliberation, non-SWSL) RuleML instance. And copy their values into the output. The task of the phase 3:

1. makes @Material explicit
2. makes @mapMaterial explicit
3. Makes @direction explicit
4. Makes @mapDirection explicit
5. Makes @oriented explicit.
6. Makes @in explicit
7. Makes @val explicit
8. Copies everything else to the phase-3 output.
9. Copies everything to the transformation output

When each phase has been implemented in the XSLT stylesheet we will test the output of our normalizer. To do so we will develop 3 kinds of test cases. Ones that are completely normalized, ones that are partially normalized, and ones that are not normalized at all.

Project Tools

1. To test the output of our transformations we will use a validator for XML Schema at http://www.w3.org/2001/03/webdata/xsv

2. **XSLT** (Extensible Stylesheet Language) is a style sheet language for XML documents. XPath and XSLT are both parts of the XSL. For the preparation of this project, we need to know the basic knowledge XSLT, which includes the content of XSLT element like “apply-templates”, “for-each” and “value-of”.

3. **RuleML (**http://www.ruleml.org/) is a markup language for sharing rules in XML markup. Originally, it was specified by DTDs, but is now specified by XSD schemas.

References:

"XSL Transformations." *Cafe Con Leche XML News and Resources*. Web. 04 Nov. 2011. <http://www.cafeconleche.org/books/bible3/chapters/ch15.html>.

“MYNG." *RuleML Wiki*. Web. 05 Nov. 2011. <http://wiki.ruleml.org/index.php/MYNG>.