

*Looking for a low cost alternative to a commercial rules engine? If so, you may be interested in learning how you can use XML and XSLT to isolate your business rules and make your application more dynamic and flexible to change.*

### Why Isolate Business Rules?

Business policies and procedures (i.e. rules) help to define a company's competitive advantage. However, the ability to change these rules in response to revenue opportunities or environmental constraints is becoming key to sustaining that advantage. When rules are hard coded into application logic making policy changes often triggers a lengthy code-test-deploy cycle. Moving rules out of application code and into a separate repository introduces the possibility for isolated rule maintenance and testing. Technically savvy business people can be equipped with tools to update and test rule changes and, perhaps even deploy, the new rules into production. All of this can be accomplished without touching the existing compiled application.

### Why XSLT?

Commercial rules engine products are tremendously powerful tools for elegantly isolating business rules. However, they are not an option for some limited IT budgets. Also, there is a significant learning curve for IT resources unfamiliar with business rules technology. Open source rules engine products, like Drools (<http://drools.org>) or Mandarax (<http://www.mandarax.org>) help address the cost issue. However, they do not help overcome the rules engine learning curve. XML and XSLT related technologies, on the other hand, are becoming a regular part of the IT toolkit thanks to the surge of web publishing, integration and web services applications over the past few years. Dozens of books have been authored on these subjects and the utilities available for working with XML and XSLT are becoming ubiquitous.

The most compelling reason to consider XML and XSLT technology, however, is its availability as part of the standard Java J2SE SDK 1.4.x (or higher) distribution. J2SE 1.4 introduced JAXP (Java API for XML Parsing) and began packaging parsers and XSLT processors, Crimson and Xalan. Now highly mature, these components boast tremendous improvements in speed and accuracy since their first generations. JAXP also standardizes the means by which applications interact with XML, further progressing the overall level of acceptance.

### Our Sample Application

Let us examine a situation where externalizing business rules may help improve customer satisfaction for a hypothetical E-Commerce retailer. The retailer periodically offers promotions to online shoppers based on market conditions and the profile of the customer. The retailer considers it a significant competitive advantage to allow online managers the ability to modify these promotions in near real-time. Previously, promotions were coded into the application and changes would consistently require nightly change control cycles and involve one or more IT resources.

To address this need, the retailer uses an XSL style sheet to store its promotion rules. In this simple example there are only three promotion rules. However, XML allows for extensive nesting and linking of related documents. Therefore, a more robust solution can be envisioned where rules would be broken down into subject areas and aggregated by a master style sheet as necessary. The retailer's promotion rules style sheet is shown in Listing 1.

How does an XSLT processor interpret this document? An XSLT processor accepts an input "instance" XML document, such as one that describes a customer and their previous order in our example, along with an XSL style sheet document. Working from the top of the XSL style sheet, the processor attempts to match patterns in the instance document based upon the match attribute of the template element in the XSL document. Once a match is found, such as an instance of a customer element, the processor sets the "node" to be current and outputs the content in the template. Of course you also have the power of the XSLT scripting language to control what content is output as demonstrated by the conditional processing for the promotion rules. This language provides access to the values of the current node for programmatic evaluation as well as output. For example, in the shipping promotion rule, if the state equals 'PA' or 'NJ' an XML fragment is written with the eligible promotion information. Notice that the description element contains the customer's state in the description via the use of the "value-of" processing element.

Ultimately, the output of this transformation will be an XML document with a root node of <promotions> and zero or more <promotion> elements depending on the profile of the customer. Each of the eligible promotion elements contains child elements related to "description", "validUntil", and "promotionCode". We will examine how the customer XML document is built and passed to the transformation processor shortly.

**Listing 1.** PromotionRules.xsl. Rules for how to apply promotions.

```

<?xml version='1.0'?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <promotions >
    <xsl:apply-templates />
  </promotions>
</xsl:template>

<xsl:template match="/customer">
  <!-- valued customer promo (for items totaling more than $500) -->
  <xsl:if test="sum(orders/order/price) > 500">
    <promotion>
      <description>Receive a 15% discount on all purchases before January 1,
2005!</description>
      <validUntil>2005-01-01</validUntil>
      <promotionCode>VC0001</promotionCode>
    </promotion>
  </xsl:if>

  <!-- Spider Man DVD promotion (if less than 15 and ordered a Spider Man DVD' -->
  <xsl:variable name="spiderManDvdOrders" select="orders/order[productName = 'Spider Man DVD']"/>
  <xsl:if test="age < 21 and count($spiderManDvdOrders) > 0">
    <promotion>
      <description>Purchase a Spider Man mask from the Marvel Comics shop for just
$15!</description>
      <validUntil>2005-03-01</validUntil>
      <promotionCode>SM0001</promotionCode>
    </promotion>
  </xsl:if>

  <!-- Shipping promotion -->
  <xsl:if test="state = 'PA' or state = 'NJ'">
    <promotion>
      <description>Free shipping to <xsl:value-of select="state"/>
customers!</description>
      <validUntil>2005-02-01</validUntil>
      <promotionCode>SH0001</promotionCode>
    </promotion>
  </xsl:if>

</xsl:template>
</xsl:stylesheet>

```

The J2SE application begins in the main method of the XsltRules java class (Listing 2). A command line argument is required to identify the URI of the PromotionsRules.xsl style sheet. Next, some sample data is assembled. In this case a Customer object is created and populated with a series of Order objects. Order (not shown) is a simple class containing attributes for “partNum”, “quantity”, “price”, “shipDate”, and “productName”. Customer (not shown) has attributes for “firstName”, “lastName”, “age”, “state”, and a collection of Order objects (“orders”).

Next, an XsltPromotionsRulesEngine object is created and passed the style sheet URI parameter. Note that this class implements the PromotionsRulesEngine interface (Listing 3) that prescribes a single “firePromotionRules” method. This method accepts a Customer object and returns an array of Promotion objects. Promotion (not shown) is a simple class with attributes for “promotionCode”, “description”, and “validUntil”. Hiding the XSLT rules engine behind this interface allows us to plug in different Rules Engine implementations as necessary (or as our budgets grow). Finally, the test application simply loops through the available promotions printing their description to the console.

**Listing 2.** XsltRules.java. The sample application launch point.

```

package com.chariotsolutions.xsltRules;

import java.math.BigDecimal;
import java.util.Calendar;
import java.util.GregorianCalendar;

public class XsltRules {

    public static void main(String[] args) {
        if (args.length < 1) {
            System.out.println("U S A G E");
            System.out.println("=====");
            System.out.println("java XsltRules \"path/to/PromotionRulesStylesheet.xsl\"");
            System.out.println("");
            System.out.println("where \"path/to/PromotionRulesStylesheet.xsl\" is the promotion rules
stylesheet uri");
            System.out.println("");
            System.out.println("=====");
            System.exit(0);
        }

        // create test data
        Customer customer = new Customer();
        customer.setId(500);
        customer.setAge(41);
        customer.setFirstName("Dan");
        customer.setLastName("Hayes");
        customer.setState("PA");

        // add orders
        Order o1 = new Order();
        o1.setPartNum("00001");
        o1.setPrice(new BigDecimal(2500.00));
        o1.setProductName("Pioneer Big Screen TV");
        o1.setQuantity(1);
        Calendar cal1 = new GregorianCalendar(2004, GregorianCalendar.OCTOBER, 1);
        o1.setShipDate(cal1.getTime());
        customer.addOrder(o1);
        Order o2 = new Order();
        o2.setPartNum("00002");
        o2.setPrice(new BigDecimal(50.00));
        o2.setProductName("Component Video Cables");
        o2.setQuantity(2);
        Calendar cal2 = new GregorianCalendar(2004, GregorianCalendar.OCTOBER, 15);
        o2.setShipDate(cal2.getTime());
        customer.addOrder(o2);
        Order o3 = new Order();
        o3.setPartNum("00003");
        o3.setPrice(new BigDecimal(15.99));
        o3.setProductName("Spider Man DVD");
        o3.setQuantity(1);
        Calendar cal3 = new GregorianCalendar(2004, GregorianCalendar.OCTOBER, 15);
        o3.setShipDate(cal3.getTime());
        customer.addOrder(o3);

        long startTime = System.currentTimeMillis();
        PromotionsRuleEngine engine = new XsltPromotionsRuleEngine(args[0]);
        try {
            Promotion[] promotions = engine.firePromotionRules(customer);
            System.out.println("Eligible Promotions:");
            System.out.println("=====");
            for (int i = 0; i < promotions.length; i++) {
                Promotion promo = promotions[i];
                System.out.println(promo.getDescription());
            }
        }
    }
}

```

```

        System.out.println("=====");
    } catch (Exception e) {
        System.err.println(e.getMessage());
        System.exit(-1);
    }

    long endTime = System.currentTimeMillis();
    System.out.println("Processing complete [elapsed time: " + (endTime - startTime) + "
milliseconds]");
    System.exit(0);
}
}

```

**Listing 3.** PromotionsRuleEngine.java. The interface implemented by promotions rules engine components.

```

package com.chariotsolutions.xsltRules;

public interface PromotionsRuleEngine {

    Promotion[] firePromotionRules(Customer customer) throws Exception;

}

```

The implementation of the PromotionsRuleEngine interface is shown in Listing 4. The XsltPromotionsRuleEngine is instantiated with a style sheet URI. The constructor simply caches the javax.xml.transform.Transformer object that is created based on a javax.xml.transform.stream.StreamSource pointing to the rules XSL style sheet. Caching the Transformer object in this way provides a small amount of performance improvement when the rules engine is to be reused instead of destroyed for each promotion inquiry. In the firePromotionRules method, the customer object is converted to a w3c Document object using a helper class CustomerXmlDocumentWriter (Listing 5). This helper class utilizes JAXP and DOM to build an XML representation of the customer object. Next the XSLT transformation engine (in this case XALAN hidden behind the JAXP API) performs the translation of the customer document object and the rules style sheet to produce an XML document representing the promotions. This promotions XML stream is then parsed and translated into an array of Promotion objects that are returned to the client (Listing 6).

In this test case, the customer qualifies for two promotions: the valued customer promotion and the free shipping promotion. Since the age test in the Spider Man DVD promotion failed, the customer will not be shown the Spider Man Mask cross sell. Try changing the age of the test customer to something less than 21 and all three promotions should now become eligible. Presumably, in a real application setting the customer object will be activated from a permanent data store elsewhere in the program code. However, the rules may be modified as needed without changing a single line of code. You could easily envision a server side component such as a Java Stateless Session EJB that represents a promotion rules service in an E-Commerce application. The component could reference a rules style sheet at runtime in order to apply the most current rules. An online manager or buyer could be provided with authoring tools to modify the style sheet, test it, and even deploy it to the server hosting the rule service.

### Pros and Cons

This approach does assume some degree of familiarity with XML parsing and the XSLT style sheet language. However, considering the rapid acceptance of these technologies, betting on XML and XSLT should not represent a significant concern. Also, this approach does not propose to be as high performing as non-interpretive alternatives such as rules engines. Fortunately, the performance improvements of the major open source parsers and processors are making this differential less significant. Furthermore, the differences are likely to appear only after rule repositories become large and the rule conditions become extremely complicated. Perhaps a commercial rules engine is a more viable alternative in these circumstances. Finally, XSLT is not the most intuitive rule authoring language, especially compared to advanced grammars employed by most rules engines. In addition, the rules won't be portable to a commercial engine in all likelihood. However, collecting the rules in a centralized repository such as an XSL style sheet will go a long way toward smoothing the transition to a full-scale business rules management system down the road.

**Listing 4.** XsltPromotionsRuleEngine.java. The XSLT implementation of the PromotionsRuleEngine interface.

```

package com.chariotsolutions.xsltRules;

import org.apache.xpath.XPathAPI;
import org.w3c.dom.*;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;
import java.io.*;
import java.text.DateFormat;
import java.text.SimpleDateFormat;

public class XsltPromotionsRuleEngine implements PromotionsRuleEngine {
    private static final String DATE_FORMAT = "yyyy-MM-dd";
    private static DateFormat DATE_FORMATTER = new SimpleDateFormat(DATE_FORMAT);

    private Transformer transformer;

    public XsltPromotionsRuleEngine(String promoRulesXslPath) {
        StreamSource xsltSource = new StreamSource(promoRulesXslPath);
        TransformerFactory tFactory = TransformerFactory.newInstance();
        try {
            transformer = tFactory.newTransformer(xsltSource);
            System.out.println("Successfully created new transformer based on promotions rules xslt
file " + promoRulesXslPath);
        } catch (TransformerConfigurationException e) {
            System.err.println("Error creating transformer for promotions rules xslt [detail: " +
e.getMessage() + "]");
        }
    }

    public Promotion[] firePromotionRules(Customer customer) throws Exception {
        if (transformer == null) {
            throw new Exception("Transformer not initialized properly. Halting rule execution.");
        }
        CustomerXmlDocumentWriter customerXmlDocumentWriter = new CustomerXmlDocumentWriter(customer);
        Document customerDoc = customerXmlDocumentWriter.getDocument();
        ByteArrayOutputStream resultXml = new ByteArrayOutputStream();
        Source source = new DOMSource(customerDoc);
        transformer.transform(source, new StreamResult(resultXml));
        return parsePromotions(new ByteArrayInputStream(resultXml.toByteArray()));
    }

    private Promotion[] parsePromotions(InputStream promotionsXml) throws Exception {
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder parser = factory.newDocumentBuilder();
        Document promotionsDoc = parser.parse(promotionsXml);

        Element promotionsElem = promotionsDoc.getDocumentElement();
        NodeList promotions = XPathAPI.selectNodeList(promotionsElem, "promotion");
        Promotion[] result = new Promotion[promotions.getLength()];
        for (int i = 0; i < promotions.getLength(); i++) {
            Element promotion = (Element) promotions.item(i);
            Promotion p = new Promotion();
            Node description = XPathAPI.selectSingleNode(promotion, "description");
            p.setDescription(description.getFirstChild().getNodeValue());
            Node promotionCode = XPathAPI.selectSingleNode(promotion, "promotionCode");
            p.setPromotionCode(promotionCode.getFirstChild().getNodeValue());
            Node validUntil = XPathAPI.selectSingleNode(promotion, "validUntil");
            p.setValidUntil(DATE_FORMATTER.parse(validUntil.getFirstChild().getNodeValue()));
            result[i] = p;
        }
        return result;
    }
}

```

**Listing 5.** CustomerXmlDocumentWriter.java. A helper class that builds a w3c Document object from a Customer.

```

package com.chariotsolutions.xsltRules;

import org.w3c.dom.*;
import javax.xml.parsers.DocumentBuilderFactory;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Iterator;

public class CustomerXmlDocumentWriter {
    private static final String DATE_FORMAT = "yyyy-MM-dd";
    private static DateFormat DATE_FORMATTER = new SimpleDateFormat(DATE_FORMAT);
    private Document doc;

    public CustomerXmlDocumentWriter(Customer customer) {
        try {
            doc = DocumentBuilderFactory.newInstance().newDocumentBuilder().newDocument();
            Element customerElem = doc.createElement("customer");
            doc.appendChild(customerElem);
            customerElem.setAttribute("id", Integer.toString(customer.getId()));
            Element firstNameElem = doc.createElement("firstName");
            Text firstNameText = doc.createTextNode(customer.getFirstName());
            firstNameElem.appendChild(firstNameText);
            customerElem.appendChild(firstNameElem);
            Element lastNameElem = doc.createElement("lastName");
            Text lastNameText = doc.createTextNode(customer.getLastName());
            lastNameElem.appendChild(lastNameText);
            customerElem.appendChild(lastNameElem);
            Element ageElem = doc.createElement("age");
            Text ageText = doc.createTextNode(Integer.toString(customer.getAge()));
            ageElem.appendChild(ageText);
            customerElem.appendChild(ageElem);
            Element stateElem = doc.createElement("state");
            Text stateText = doc.createTextNode(customer.getState());
            stateElem.appendChild(stateText);
            customerElem.appendChild(stateElem);
            Element ordersElem = doc.createElement("orders");
            customerElem.appendChild(ordersElem);
            for (Iterator iter = customer.getOrders().iterator(); iter.hasNext(); ) {
                Order o = (Order) iter.next();
                Element orderElem = doc.createElement("order");
                orderElem.setAttribute("partNum", o.getPartNum());
                Element productNameElem = doc.createElement("productName");
                Text productText = doc.createTextNode(o.getProductName());
                productNameElem.appendChild(productText);
                orderElem.appendChild(productNameElem);
                Element quantityElem = doc.createElement("quantity");
                Text quantityText = doc.createTextNode(Integer.toString(o.getQuantity()));
                quantityElem.appendChild(quantityText);
                orderElem.appendChild(quantityElem);
                Element shipDateElem = doc.createElement("shipDate");
                Text shipDateText = doc.createTextNode(DATE_FORMATTER.format(o.getShipDate()));
                shipDateElem.appendChild(shipDateText);
                orderElem.appendChild(shipDateElem);
                Element priceElem = doc.createElement("price");
                Text priceText = doc.createTextNode(o.getPrice().toString());
                priceElem.appendChild(priceText);
                orderElem.appendChild(priceElem);
                ordersElem.appendChild(orderElem);
            }
        } catch (Exception e) {
            System.err.println("Error creating new document instance [detail: " + e.getMessage() +
                "]);
        }
    }

    public Document getDocument() {
        return doc;
    }
}

```

**Listing 6. Sample Console Output.**

```
Successfully created new transformer based on promotions rules xslt file
C:\\development\\projects\\xslt-rules\\rules\\PromotionRules.xsl
Eligible Promotions:
=====
Receive a 15% discount on all purchases before January 1, 2005!
Free shipping to PA customers!
=====
Processing complete [elapsed time: 160 milliseconds]

Process finished with exit code 0
```

**About the Author**

Dan Hayes is a technical consultant and rules engine specialist for Chariot Solutions, LLC located in Ft. Washington, PA. Dan has over 15 years experience in the insurance and financial services industry as an executive, entrepreneur, and technical consultant having worked with some of the largest insurance companies on enterprise technology projects including BRMS implementations. Dan has experience with commercial products from Fair Isaac (Blaze Advisor) and ILOG (JRules) as well as JESS and the open source Drools project. He is a Sun Certified J2EE Enterprise Architect and holds Master's Degrees from Columbia University (MBA) and Penn State University (M.S. Software Engineering).