XRules Tutorial

Draft Version 0.43, November 7, 2005

Abstract

This document is an introduction to XRules along with examples and illustrations of use cases and applications. XRules is an XML rules language that expresses constraints, calculations, interdependencies, and properties that describe and exist among elements and attributes of an XML document.

Status of this document

This document is a work in progress. Software implementations and test tools of the XRules specification are available for download from www.xrules.org. These implementations are used as a proof of concept and are updated regularly as the XRules specification evolves.

If you like to participate in developing XRules or have feedback or questions, please contact the author at waleed_abdulla@xrules.org. All contributions are welcome and credit will be given where credit is due.

Table of Contents

1 Introduction

1.1 What is XRules?

XRules is an XML-based language for describing rules that govern other XML documents. More specifically, XRules describes constraints, calculations, properties, relations and interdependencies that exist between nodes (elements and attributes) of a *target XML document*. The target XML document could be a purchase order, a bank transaction, a web service message, an employee record, etc. XRules provides three mains categories of rules:

1. **Constraints**: These are restrictions on the content of an XML Document that must be satisfied for the document to be considered valid. For example:
    a. `OrderClosingDate` must be greater than `OrderCreationDate`.
    b. The attribute `@BookID` must be unique within the scope of a `<Library>` element, but can be repeated outside that scope.
2. **Calculations**: These are formulas that describe how values of XML nodes (elements and attributes) in an XML document are derived or calculated from the values of other nodes. For example:
    a. The value of the `Tax` element is equal to: `Total * SalesTaxPercent`.
    b. `SalesTaxPercent` is 0.08 if `State=CA`, 0.06 if `State=NJ`, etc.
3. **Properties**: XRules expands the XML Infoset by adding automatic and user-defined properties that are attached to XML elements and attributes. These properties can also participate in calculations and constraints. For example:
    a. An XRules document has a rule that specifies that the maximum valid value for the element `ItemQuantity` is 50. This rule generates an *Automatic XRules Property*, `[xr:max]`, which has the value 50.
    b. Using the `<xr:property>` rule, a user-defined property with the name `[IsIgnored]` can be attached to the `CreditCardNumber` element when the payment type is cash.

1.2 What is the Dynamic DOM?

The Dynamic DOM (DDOM) is an extension of the W3C DOM modified to enforce and execute the rules of XRules documents dynamically. First, the target XML document is loaded into the DDOM, and then one or more XRules documents are attached to it. Once attached, the DDOM starts enforcing these rules dynamically by:

1. Preventing invalid values from being assigned to nodes of the target XML document and generating appropriate error messages.
2. When a node is updated with a valid value, the DDOM automatically updates the values of all other nodes that are dependent on it based on Calculation Rules.
3. Providing a mechanism to expand the XML Infoset to add XRules automatic and user-defined properties to XML nodes.

1.3 Key Features of XRules

1. **Adds Dynamism to XML**. When used with a dynamic processor, such as the Dynamic DOM, XRules allows XML documents to behave like dynamic data objects rather than static data documents. Validations are done on the fly as node values are changed, and the values of dependent nodes are recalculated automatically based on provided formulas.
2. **XML Schema Friendly.** XRules Supports W3C XML Schema data types. And, XRules rules can be embedded in XML Schema documents and vice versa.
3. **Metadata Properties**: XRules provides a framework that makes it easy to attach metadata information to XML documents in the form of properties attached to elements and attributes and accessible using APIs or XPath expressions (for example, in XSLT transformations).
4. **Expresses complex rules and interdependencies**. Using specialized rules and XPath expressions, XRules can express a wide range of rules beyond what can be done with XML Schema.
5. **Extensible.** In addition to built in rules, XRules provides an extensible framework in which new rules can be created to provide additional constraints or calculations.
6. **Composable.** Multiple XRules documents can be attached at the same time to one XML document. The behavior and the resulting outputs are generated based on the combined effect of all XRules documents.

1.4 Three Modes of XRules Processing

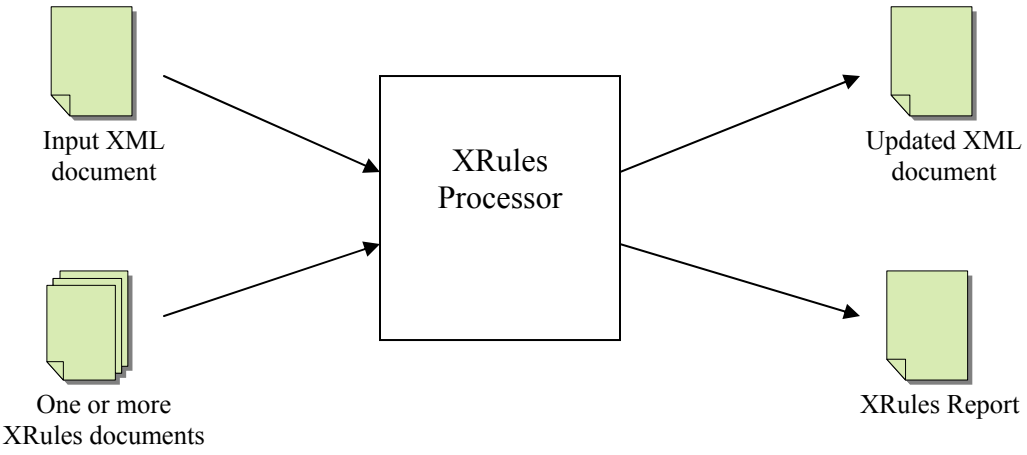
XRules is a declarative language, and therefore, can be processed in different ways based on requirements. Three modes of XRules processing are described below:

1.4.1 Dynamic Processing Mode

A dynamic XRules processor keeps an object model of the target XML document in memory and binds it to one or more XRules documents. Changes applied to the XML

document are validated against the appropriate rules before being committed. And, once a node is modified, dependent nodes that are affected by the change are updated dynamically at run time. The Dynamic DOM described earlier is an example of an XRules dynamic processor. See Section 3 for more details about available XRules tools.

1.4.2 Updating Mode



In this mode of processing, the input to the XRules processor is an XML document and one or more XRules documents. The output is an updated version of the XML document with values of calculated nodes updated with the results from applying the calculation rules. Another output is an XRules Report which contains a list of errors found in the XML document, if any.

The XRules command-line utility, `xrules.exe`, is an example of a processor that can perform this mode of processing. This utility can also perform an XSLT transformation on the updated XML document to generate a third output file. The benefit of performing the XSLT transformation with the XRules processing in one pass is it to allow the XSLT transformation to access and utilize the XRules properties attached to the updated XML document nodes since these properties are lost once the XML document is serialized to a file. See Section 3 for more details about available XRules tools.

1.4.3 Validation Mode

The validation mode is the simplest form of XRules processing. In this mode the XML document is validated against the rules in the XRules document(s) and an XRules Report is generated that contains a list of errors. The XRules utility can also perform this mode of XRules processing.

2 XRules Tutorial

The following sections describe the syntax and the different rules of the XRules language. A set of tools are available to run and experiment with the provided examples, and they are described in Section 3. And, you can download the tools and the examples from http://www.xrules.org.

2.1 Structure of XRules Documents

Here is how an XRules document looks like:

```
<xr:rules xmlns:xr="http://www.xrules.org/2003/11">

   <xr:ruleset context="...XPath expression to select nodes ...">
      <!-- ... rules are inserted here ... -->
      <xr:calculate target="..." value="..." />
      <xr:validate test="..." />
      <xr:bind target="..." type="..." />
   </xr:ruleset>

   <xr:ruleset context="...XPath expression to select nodes ...">
      <!-- ... rules are inserted here ... -->

      <!-- rulesets can be embedded inside other rulesets -->
      <xr:ruleset context="XPath expression to select nodes ">
         <!-- ... rules are inserted here ... -->
      </xr:ruleset>
   </xr:ruleset>

</xr:rules>
```

An XRules document starts with the `<rules>` element which is the top level container of the document. Under the `<rules>` element you can have one or more `<ruleset>` elements. A ruleset contains rules or other rulesets, and establishes a context of execution for all XPath expressions used in it and its child rules. The context nodes of a ruleset are selected using the `context` attribute, which is an XPath expression that selects a set of nodes from the target XML document.

Different types of rules can be used inside a ruleset. The `<validate>` rule validates a condition, the `<calculate>` rule specifies how the value of a target node is calculated, and the `<bind>` rule specifies different properties and constraints for a target node such as the data type, minimum and maximum values, and so forth. Other rules are used to specify other types of constraints, relations, or calculations.

2.2 The `validate` Rule

Validation rules are used to describe constraints that the target XML document must satisfy. The conditions are expressed using XPath expressions.

Example 1 Validation Rules

Purchase Order

```
<PurchaseOrder>
   <AuthorizedAmount>500</AuthorizedAmount>

   <Item>
      <Name>PC Comptuer</Name>
      <Quantity>1</Quantity>
      <UnitPrice>500</UnitPrice>
      <ItemTotal>500</ItemTotal>
   </Item>
   <Item>
      <Name>XML Book</Name>
      <Quantity>-4</Quantity>          <!-- error: negative quantity -->
      <UnitPrice>30</UnitPrice>
      <ItemTotal>120</ItemTotal>
   </Item>

   <SubTotal>620</SubTotal>
   <Tax>31</Tax>
   <GrandTotal>651</GrandTotal>  <!-- error: GrandTotal > AuthorizedAmount-->
</PurchaseOrder>
```

XRules Document

```
<xr:rules xmlns:xr="http://www.xrules.org/2003/11">

   <xr:ruleset context="/PurchaseOrder">
      <xr:validate test="AuthorizedAmount >= GrandTotal" />
   </xr:ruleset>

   <xr:ruleset context="/PurchaseOrder/Item">
      <xr:validate test="Quantity > 0" />
      <xr:validate test="UnitPrice > 0" />
   </xr:ruleset>

</xr:rules>
```

In this example we're validating a purchase order against a set of rules defined in an XRules document. The XRules document above has two `<ruleset>` elements. One applies to the `/PurchaseOrder` element, and one applies to all `/PurchaseOrder/Item` elements as the `context` attribute shows. These rulesets state the following rules:

- `GrandTotal` of the purchase order must not exceed the `AuthorizedAmount`.

- Item `Quantity` must be greater than zero.
- Item `UnitPrice` must be greater than zero.

When applied to this sample XML document, the XRules processor detects that two rules are not satisfied: First, the Grand Total amount is greater than the Authorized Amount. And, second, the quantity of the second item is negative.

Different XRules processors provide different ways for the application program to read the result of the processing. A simple command line processor, for example, generates and XML report file showing the results of the processing and validation. On the other hand, a dynamic processor, such as the DDOM, can provide events and properties that allow dynamic detection and discovery of the errors in addition to the ability to generate an XRules output report.

Here, we're going to use an XRules output report to show the results of processing the examples included in this document. XRules reports are the standard way of reporting the result of applying an XRules document to an XML document, and it can be generated by the Dynamic DOM, an XRules Validator, or an XRules Calculator.

Here is the output of applying the sample XRules document to the sample Purchase Order document listed above:

```
<xrr:report xmlns:xrr="http://www.xrules.org/2003/11/report">
   <xrr:errors>
      <xrr:error context="/PurchaseOrder">
         <xrr:message>Validation Failed: AuthorizedAmount &gt;= GrandTotal
         </xrr:message>
         <xrr:node path="/PurchaseOrder/AuthorizedAmount[1]" />
         <xrr:node path="/PurchaseOrder/GrandTotal[1]" />
      </xrr:error>
      <xrr:error context="/PurchaseOrder/Item">
         <xrr:message>Validation Failed: Quantity &gt; 0</xrr:message>
         <xrr:node path="/PurchaseOrder/Item[2]/Quantity[1]" />
      </xrr:error>
   </xrr:errors>
</xrr:report>
```

As you can see, the report shows two errors. And for each error, it displays the error message (default error messages in this case since no custom messages were defined). And, also, for each error a list of the XPath path expressions that identify the nodes that participated in the rule that fired the error.

2.3 The `calculate` Rule

Calculation rules differ from constraints rules in that they can change values of their target nodes in the XML document. However, the behavior of a calculation rule depends on the type of XRules processor as explained later in this section.

Example 2 Calculation Rules

Purchase Order:

```
<PurchaseOrder>
   <Item>
      <Name>PC Comptuer</Name>
      <Quantity>1</Quantity>
      <UnitPrice>500</UnitPrice>
      <ItemTotal>500</ItemTotal> <!-- error: exceeds max limit -->
   </Item>
   <Item>
      <Name>XML Book</Name>
      <Quantity>4</Quantity>
      <UnitPrice>30</UnitPrice>
      <ItemTotal>120</ItemTotal>
   </Item>
   <SubTotal>620</SubTotal>
   <Tax>31</Tax>
   <GrandTotal>653</GrandTotal>  <!-- error: wrong total here -->
</PurchaseOrder>
```

XRules Document:

```
<xr:rules xmlns:xr="http://www.xrules.org/2003/11">

   <xr:ruleset context="/PurchaseOrder">
      <xr:calculate target="SubTotal">
         <xr:value>sum(Item/ItemTotal)</xr:value>
      </xr:calculate>
      <xr:calculate target="Tax">
         <xr:value>SubTotal * 0.05</xr:value>
      </xr:calculate>
      <xr:calculate target="GrandTotal">
         <xr:value>SubTotal + Tax</xr:value>
      </xr:calculate>
   </xr:ruleset>

   <xr:ruleset context="/PurchaseOrder/Item">
      <xr:calculate target="ItemTotal">
         <xr:value>UnitPrice * Quantity</xr:value>
      </xr:calculate>
      <xr:validate test="Quantity > 0" />
      <xr:validate test="UnitPrice > 0" />
      <xr:validate test="ItemTotal &lt; 400" />
   </xr:ruleset>

</xr:rules>
```

In addition to validation rules, the XRules document above describes the expressions used to calculate the `ItemTotal` for each item, and the `SubTotal`, `Tax`, and `GrandTotal` of the purchase order.

The actual behavior of the `<calculate>` rule depends on the type of XRules processor.

2.3.1 Behavior in Dynamic or Updating Modes

In dynamic or updating processing the XPath expressions are evaluated to calculate the values of the nodes specified by the `target` attributes and the XML document is updated with the calculated values.

Here is the error report that the Dynamic DOM (an XRules Dynamic processor) generates if used on the example above:

XRules Report:

```
<xrr:report xmlns:xr="http://www.xrules.org/2003/11"
            xmlns:xrr="http://www.xrules.org/2003/11/report">
  <xrr:errors>
    <xrr:error context="/PurchaseOrder/Item">
      <xrr:message>Validation Failed: ItemTotal &lt; 400</xrr:message>
      <xrr:node path="/PurchaseOrder/Item[1]/ItemTotal[1]" />
    </xrr:error>
  </xrr:errors>
</xrr:report>
```

Notes:

- Notice that the error report shows only one error although the XML document has two. That's because the `<calculate>` rule on the `GrandTotal` element updated the XML document by replacing the wrong value with one calculated from the expression in the `value` attribute. The value of `GrandTotal` after the rules are applied is updated to 651.
- The context of execution of XPath expressions of the top level rulesets is the root of the XML document. And, the context of execution of the rules inside a ruleset is the context nodes selected by the ruleset using the `context` attribute.
- In the `<calculate>` element, the target attribute is an XPath expression that returns a node, and the value attribute is an XPath expression that returns a value (string, Boolean, or numeric). Both XPath expressions are evaluated from the context established by the ruleset context.

2.3.2 Behavior in Validation Mode

In the XRules Validation mode, on the other hand, the original XML document is not changed. The XRules processor calculates the `value` expressions and compares the result to the value of the `target` nodes in the XML document to determine if the existing values are correct. So, in validation mode, the calculate rule is used for validation not for calculation.

Here is the error report that is generated in validation mode if the example above is used:

XRules Report:

```
<report xmlns="http://www.xrules.org/2003/11/report">
  <errors>
    <error context="/PurchaseOrder">
      <message>Incorrect Value: GrandTotal is not equal to
               SubTotal + Tax. Correct Value is: 651.</message>
    </error>
    <error context="/PurchaseOrder/Item">
      <message>Failed Validation: ItemTotal &lt; 400.</message>
    </error>
  </errors>
</report>
```

Notes:

- Notice that the error report shows two errors compared to one only in the DDOM report. The additional error states that the value of `GrandTotal` is wrong and shows the correct value. The reason is that, when using the validation mode, the calculate rule does not update the XML document, but is used, instead, to validate that the value of the `GrandTotal` is the same as the result of evaluating the provided expression.

2.3.3 Conditional Calculated Values

The `calculate` rule allows you to specify multiple expressions for calculating the value of a node, and a criteria for choosing which expression to evaluate. This is done using the `when` attribute of the `value` element as shown below:

```
<xr:ruleset context="/PurchaseOrder">
   <xr:calculate target="Discount">
      <xr:value when="TotalQuantity &gt; 20">0.15</xr:value>
      <xr:value when="TotalQuantity &gt; 10">0.07</xr:value>
      <xr:value>0</xr:value>
   </xr:calculate>
</xr:ruleset>
```

This rule states that `Discount` should be 15% if `TotalQuantity` is greater than 20; and 7% if `TotalQuantity` is greater than 10. Otherwise, `Discount` should be 0. The rule is: The first `<xr:value>` with a successful `when` condition is used.

2.4 The `bind` Rule

The <bind> rule is the most complex rule so far. It selects a set of target nodes and specifies several different restrictions on those nodes. Here is an example showing the

use of the bind rule to specify the data types and minimum and maximum values for the target nodes.

Example 3 Bind Rules

Purchase Order:

```
<PurchaseOrder>
   <PaymentMethod>wire</PaymentMethod>  <!-- error: method not allowed -->
   <Item>
      <Name>PC Comptuer</Name>
      <Quantity>1</Quantity>
      <UnitPrice>500</UnitPrice>
      <ItemTotal>500</ItemTotal>       <!-- error: exceeds max limit -->
   </Item>
   <Item>
      <Name>XML Book</Name>
      <Quantity>four</Quantity>        <!-- error: wrong data type -->
      <UnitPrice>30</UnitPrice>
      <ItemTotal>120</ItemTotal>
   </Item>
   <SubTotal>620</SubTotal>
   <Tax>31</Tax>
   <GrandTotal>651</GrandTotal>
</PurchaseOrder>
```

XRules Document:

```
<xr:rules xmlns:xr="http://www.xrules.org/2003/11"
          xmlns:xs="http://www.w3.org/2001/XMLSchema">

   <xr:ruleset context="/PurchaseOrder/Item">
      <xr:bind target="Quantity" type="xs:positiveInteger" />
      <xr:bind target="UnitPrice" type="xs:decimal" />
      <xr:bind target="ItemTotal" type="xs:decimal" max="400" />
      <xr:calculate target="ItemTotal">
         <xr:value>UnitPrice * Quantity</xr:value>
      </xr:calculate>
   </xr:ruleset>

   <xr:ruleset context="/PurchaseOrder">
      <xr:bind target="PaymentMethod">
         <xr:valueset>
            <xr:enum value="cash" />
            <xr:enum value="credit" />
            <xr:enum value="check" />
         </xr:valueset>
      </xr:bind>
      <xr:calculate target="SubTotal">
         <xr:value>sum(Item/ItemTotal)</xr:value>
      </xr:calculate>
      <xr:calculate target="Tax">
         <xr:value>SubTotal * 0.05</xr:value>
      </xr:calculate>
      <xr:calculate target="GrandTotal">
         <xr:value>SubTotal + Tax</xr:value>
      </xr:calculate>
```

```
   </xr:ruleset>

</xr:rules>
```

In the XRules document above, the `bind` rules specify the followings:

1. Data types using the `type` attribute. XRules supports the W3C XML Schema data types and provides a mechanism to assign them to individual nodes. This is typically used if the XML document does not have an XML Schema. Otherwise, if an XML Schema exists, XRules uses the types specified by that schema.
2. Maximum values, as shown by specifying a maximum value of 400 for the `ItemTotal` element. In addition to that, the bind rule can specify minimum values, minimum and maximum lengths, and regular expressions.
3. Value enumeration through the use of the `<valueset>` element.

Here is the XRules report showing the errors detected in the above XML document:

XRules Report:

```
<xrr:report xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns:xr="http://www.xrules.org/2003/11"
            xmlns:xrr="http://www.xrules.org/2003/11/report">

<xrr:errors>

   <xrr:error context="/PurchaseOrder/Item">
     <xrr:message>
       'four' is an invalid value for the type
       'System.Xml.Schema.Datatype_positiveInteger'.
     </xrr:message>
     <xrr:node path="/PurchaseOrder/Item[2]/Quantity[1]" />
   </xrr:error>

   <xrr:error context="/PurchaseOrder/Item">
     <xrr:message>
       '500' is greater than the maximum acceptable value of '400'
       (XPath Expression = 400).
     </xrr:message>
     <xrr:node path="/PurchaseOrder/Item[1]/ItemTotal[1]" />
   </xrr:error>

   <xrr:error context="/PurchaseOrder/Item">
     <xrr:message>
       'NaN' is an invalid value for the type
       'System.Xml.Schema.Datatype_decimal'.
     </xrr:message>
     <xrr:node path="/PurchaseOrder/Item[2]/ItemTotal[1]" />
   </xrr:error>

   <xrr:error context="/PurchaseOrder">
     <xrr:message>
       'wire' is not in the list of acceptable values.
     </xrr:message>
```

```
      <xrr:node path="/PurchaseOrder/PaymentMethod[1]" />
    </xrr:error>

  </xrr:errors>
</xrr:report>
```

Note: The third error element above indicates that the `ItemTotal` element of the second item contains the invalid value NaN. This is because the report above was generated using the DDOM (a dynamic processor) which executes `calculate` rules and updates the nodes of the XML document accordingly. Since the `Quantity` value of the second item is 'four' which is not a valid number, this causes the `ItemTotal` to evaluate to NaN which triggers this error. This error will not be displayed when using an XRules Validator (or the DDOM in validation mode) since in that case the existing values of the XML nodes will not be updated.

2.5 XRules and Namespaces

XRules is a namespace aware language. This means that all XPath expressions in rules and rulesets can use namespace prefixes in the XRules document if the target XML document uses namespaces.

To show namespaces in action, here is the same example from Section 2.2 above but now the XML document uses the namespace `http://www.MyPurchaseOrder.org`:

Example 4 Purchase Order with a namespace

XML Document:

```
<PurchaseOrder xmlns="http://www.MyPurchaseOrder.org">
   <AuthorizedAmount>500</AuthorizedAmount>

   <Item>
      <Name>PC Comptuer</Name>
      <Quantity>1</Quantity>
      <UnitPrice>500</UnitPrice>
      <ItemTotal>500</ItemTotal>
   </Item>
   <Item>
      <Name>XML Book</Name>
      <Quantity>-4</Quantity>          <!-- error: negative quantity -->
      <UnitPrice>30</UnitPrice>
      <ItemTotal>120</ItemTotal>
   </Item>

   <SubTotal>620</SubTotal>
   <Tax>31</Tax>
   <GrandTotal>651</GrandTotal>  <!-- error: GrandTotal > AuthorizedAmount-->
</PurchaseOrder>
```

Here, the namespace is defined as the default namespace of the document and, therefore, all the nodes in the document belong to it.

And, here is the same XRules document from the example above, updated to use the new namespace.

XRules Document:

```
<xr:rules xmlns:xr="http://www.xrules.org/2003/11"
          xmlns:po="http://www.MyPurchaseOrder.org">

   <xr:ruleset context="/po:PurchaseOrder">
      <xr:validate test="po:AuthorizedAmount >= po:GrandTotal" />
   </xr:ruleset>

   <xr:ruleset context="/po:PurchaseOrder/po:Item">
      <xr:validate test="po:Quantity > 0" />
      <xr:validate test="po:UnitPrice > 0" />
   </xr:ruleset>

</xr:rules>
```

Notice that a namespace prefix, '`po`', is defined in the XRules document to represent the namespace "`http://www.MyPurchaseOrder.org`" and allow the XPath expressions to refer to it. And, as shown above, all the XPath expressions now use the `po` prefix to reference nodes that belong to that namespace. The result report file will now look as follows:

```
<xrr:report xmlns:po="http://www.MyPurchaseOrder.org"
            xmlns:xrr="http://www.xrules.org/2003/11/report">
  <xrr:errors>
    <xrr:error context="/po:PurchaseOrder">
      <xrr:message>Validation Failed: po:AuthorizedAmount &gt;= po:GrandTotal
      </xrr:message>
      <xrr:node path="/po:PurchaseOrder/po:AuthorizedAmount[1]" />
      <xrr:node path="/po:PurchaseOrder/po:GrandTotal[1]" />
    </xrr:error>
    <xrr:error context="/po:PurchaseOrder/po:Item">
      <xrr:message>Validation Failed: po:Quantity &gt; 0</xrr:message>
      <xrr:node path="/po:PurchaseOrder/po:Item[2]/po:Quantity[1]" />
    </xrr:error>
  </xrr:errors>
</xrr:report>
```

Implementation Note: In the current version of the DDOM, all namespace declarations in an XRules document must be in the root element, which is the most common approach anyway. Namespace declarations defined locally at the ruleset level or below might cause some errors. This will be remedied in a future version of the DDOM.

2.6 Hierarchical Rulesets

Rulesets can include other rulesets in a hierarchical fashion. This adds flexibility in how rules are grouped and organized, and allows the hierarchy of the XRules document to resemble that of the target XML document.

The example below is a modified version of the example from the previous section. In this example, the target XML document is the same, but the XRules document is modified by moving the `Item` ruleset inside the `PurchaseOrder` ruleset to better show the relation between the rules and their target nodes.

When a ruleset is embedded inside another ruleset, the context nodes of the parent rulesets are used as the XPath context of execution for the XPath expressions of the `context` attribute of the embedded ruleset. Therefore, in the example below, note that the value of the `context` attribute of the internal ruleset was changed from '`/po:PurchaseOrder/po:Item`' to '`po:Item`' to take into account the fact that the execution context for the internal ruleset starts from the context node of the parent ruleset.

Example 5 Using Hierarchical Rulesets

XML Document:

```
<PurchaseOrder xmlns="http://www.MyPurchaseOrder.org">
   <AuthorizedAmount>500</AuthorizedAmount>

   <Item>
      <Name>PC Comptuer</Name>
      <Quantity>1</Quantity>
      <UnitPrice>500</UnitPrice>
      <ItemTotal>500</ItemTotal>
   </Item>
   <Item>
      <Name>XML Book</Name>
      <Quantity>-4</Quantity>          <!-- error: negative quantity -->
      <UnitPrice>30</UnitPrice>
      <ItemTotal>120</ItemTotal>
   </Item>

   <SubTotal>620</SubTotal>
   <Tax>31</Tax>
   <GrandTotal>651</GrandTotal>  <!-- error: GrandTotal > AuthorizedAmount-->
</PurchaseOrder>
```

XRules Document:

```
<xr:rules xmlns:xr="http://www.xrules.org/2003/11"
          xmlns:po="http://www.MyPurchaseOrder.org">

   <xr:ruleset context="/po:PurchaseOrder">
      <xr:validate test="po:AuthorizedAmount >= po:GrandTotal" />

      <xr:ruleset context="po:Item">
         <xr:validate test="po:Quantity > 0" />
         <xr:validate test="po:UnitPrice > 0" />
```

```
      </xr:ruleset>
   </xr:ruleset>

</xr:rules>
```

The XML report generated from this modified XRules document is almost identical to the one generated by the original one, except for the value of the `context` attribute to match the modified value in the XRules document.

```
<xrr:report xmlns:po="http://www.MyPurchaseOrder.org"
            xmlns:xrr="http://www.xrules.org/2003/11/report">
  <xrr:errors>
    <xrr:error context="/po:PurchaseOrder">
      <xrr:message>Validation Failed: po:AuthorizedAmount &gt;= po:GrandTotal
      </xrr:message>
      <xrr:node path="/po:PurchaseOrder/po:AuthorizedAmount[1]" />
      <xrr:node path="/po:PurchaseOrder/po:GrandTotal[1]" />
    </xrr:error>
    <xrr:error context="po:Item">
      <xrr:message>Validation Failed: po:Quantity &gt; 0</xrr:message>
      <xrr:node path="/po:PurchaseOrder/po:Item[2]/po:Quantity[1]" />
    </xrr:error>
  </xrr:errors>
</xrr:report>
```

2.7 Controlling Ruleset and Node Occurrences

Sometimes it's required to make sure that a ruleset binds to a specific number of context nodes (or a range of numbers). For example, a ruleset to validate the Shipping Address of a purchase order and to make sure that only one shipping address element is provided, not more and not less.

To achieve this, the ruleset element has two attributes to make sure that a ruleset does bind and determine the minimum and maximum number of occurrences allowed for the context node.

minOccurs: Sets the minimum number of occurrences of the context node. If the ruleset doesn't bind to at least the specified number of nodes, an error is generated. This attribute accepts a positive integer number or zero. The default is 1 if the attribute is missing.

maxOccurs: Sets the maximum allowed number of occurrences of the context node. An error message is generated if this number is exceeded. This attribute accepts a positive integer or the word 'unbounded' to indicate that no maximum is specified. The default value if the attribute is omitted is 'unbounded'.

The following example shows a ruleset that binds to a `ShippingAddress` element and requires that the element exists once and only once.

```
<xr:ruleset context="/PurchaseOrder/ShippingAddress"
            minOccurs="1" maxOccurs="1">
   <!-- rules here -->
</xr:ruleset>
```

And, here is a sample error report that will be generated if the `po:PurchaseOrder` element is missing:

```
<xrr:report xmlns:xrr="http://www.xrules.org/2003/11/report">
  <xrr:errors>
    <xrr:error context="/PurchaseOrder/ShippingAddress">
      <xrr:message>The node '/PurchaseOrder/ShippingAddress' must occur
                   at least '1' times. (minOccurs = '1').</xrr:message>
    </xrr:error>
  </xrr:errors>
</xrr:report>
```

Note: for hierarchical rulesets `minOccurs` and `maxOccurs` of a child ruleset describe the occurrence limits of that ruleset for each occurrence of the parent ruleset as explained in the sample code below:

```
<xr:ruleset context="/po:PurchaseOrder" minOccurs="2">
   <xr:ruleset context="po:Item" minOccurs="3">
      <!-- This ruleset must occur 3 times for each occurrence
           of its parent. Total of 6 (2*3) -->
   </xr:ruleset>
</xr:ruleset>
```

The `bind` rule also supports the use of the `minOccurs` and `maxOccurs` attributes to determine the number of occurrences of the target node. Here is an example:

```
<xr:ruleset context="po:PurchaseOrder" minOccurs="2">
   <xr:bind target="po:PaymentMethod" minOccurs="1" maxOccurs="1">
      ...
   </xr:bind>
</xr:ruleset>
```

Here, again, the number of occurrences specified for the target node is per context node. So, in the example above, the target node, `po:PaymentMethod`, can only occur once for each occurrence of the `po:PurchaseOrder` node.

2.8 Error Messages

The XRules processor generates appropriate error messages when a rule fails. The standard error messages explain the failed condition and show which ruleset/rule had failed. It's also possible to override the standard error messages with custom messages tailored for the specific application using the `errorMessage` attribute. This attribute can be applied to all rule and ruleset elements (`ruleset`, `validate`, calculate, `bind` …etc.). For example,

```
<xr:ruleset context="po:Item" minOccurs="2"
       errorMessage="There should be at least two items.">

   <xr:validate test="po:Quantity > 0"
       errorMessage="Quantity must be positive." />

   <xr:calculate target="po:ItemTotal" value="po:UnitPrice * po:Quantity"
       errorMessage="ItemTotal is not calculated correctly." />

</xr:ruleset>
```

In the example above, the custom error messages defined by the `errorMessage` attributes will be used instead of the standard error messages if any of the rules fail. Note that the `calculate` rule will fail only when using a validating processor in which case the calculate rule is used to make sure the value of a node is calculated correctly rather than calculating and updating it's value.

It's also possible to embed XPath expressions in the error message text. These XPath expressions get evaluated at run-time when the error is generated. Expressions enclosed in curly braces {} are evaluated as XPath expressions using the context node of the ruleset as a context of execution. For example,

```
<xr:validate test="Quantity > 0"
   errorMessage="Invalid Quantity: {Quantity}. Value must be positive."
/>
```

Here, the `{po:Quantity}` is an XPath expression embedded in the error message. If the Quantity field has the value -2, this rule generates the following error:

```
<xrr:report xmlns:xrr="http://www.xrules.org/2003/11/report">
  <xrr:errors>
    <xrr:error context="/PurchaseOrder/Item">
      <xrr:message>Invalid Quantity: -2. Value must be positive.</xrr:message>
```

```
      <xrr:node path="/PurchaseOrder/Item[1]/Quantity[1]" />
    </xrr:error>
  </xrr:errors>
</xrr:report>
```

Note: To use curly braces inside the text of the message without having them interpreted as the start and the end of an XPath expressions, use double braces such as {{ and }}. Double braces are converted to single braces in the message output. For example, the error message "`Value must be equal to {{text}}`" will be displayed as "`Value must be equal to {text}`" and no XPath expression is executed. This also applies to XPath expressions that happen to have curly braces as part of the expression; they must be doubled to avoid confusing them with single braces that point the start and end of the XPath expression.

The `errorMessage` attribute is a catch-all message for all types of errors that a rule can generate. To specify more specific error messages for each type of error, you can use more specific attributes in the form `errorMessage.<type>`, where <type> is the type of error, and is in most of the cases, the same as the name of the attribute that provides the constraint that could trigger that error. For example, `errorMessage.minOccurs` allows specifying a custom error message that will be used in the `minOccurs` constraint is broken. Here is a more detailed example:

```
<xr:ruleset context="po:Item" minOccurs="2" maxOccurs="4"
   errorMessage.minOccurs = "Minimum is 2"
   errorMessage.maxOccurs = "Maximum is 4" >

   ...
      <xr:bind target="ItemTotal"
         type="xs:decimal"    errorMessage.type = "Wrong type. Use Decimal."
         max="400"            errorMessage.max = "Don't go over 400."
         required="true()"    errorMessage.required = "Item Total is required."
      />

</xr:ruleset>
```

When an error is generated, the XRules processor uses for the `errorMessage.`*`Type`* attribute that corresponds to that error. If that attribute is not found, the XRules processor uses the general `errorMessage` attribute. And, if that one is not found either, the XRules processor generates the standard error message corresponding to that error.

2.9 Configuration Sections

Configuration sections are static XML fragments included in an XRules document and referenced by the rules and expressions of that document. Configuration sections serve

the same purpose in XRules as constants and INI files in programming languages: they allow developers to avoid hard-coding constant values in expressions.

The example below is an updated version of the XRules document of Example 2 showing the use of a configuration section to store the tax percentage instead of hard-coding it in the tax expression.

Example 6 Using Configuration Sections

XRules Document:

```
<xr:rules xmlns:xr="http://www.xrules.org/2003/11">

   <xr:ruleset context="/PurchaseOrder">
      <xr:calculate target="SubTotal">
         <xr:value>sum(Item/ItemTotal)</xr:value>
      </xr:calculate>
      <xr:calculate target="Tax">
         <xr:value>SubTotal * config('Tax')/TaxPercent</xr:value>
      </xr:calculate>
      <xr:calculate target="GrandTotal">
         <xr:value>SubTotal + Tax</xr:value>
      </xr:calculate>
   </xr:ruleset>

   <xr:ruleset context="/PurchaseOrder/Item">
      <xr:calculate target="ItemTotal">
         <xr:value>UnitPrice * Quantity</xr:value>
      </xr:calculate>
      <xr:validate test="Quantity > 0" />
      <xr:validate test="UnitPrice > 0" />
      <xr:validate test="ItemTotal &lt; 400" />
   </xr:ruleset>

   <xr:config id="Tax">
      <TaxPercent>0.05</TaxPercent>
   </xr:config>

</xr:rules>
```

As you see in the example, a `<config>` section was added and given the id '`Tax`'. Inside this section, any XML document (or XML fragment) can be embedded and made accessible from XPath expressions using the `config()` function. The example above shows the XPath expression used to calculate the tax amount to:

```
po:SubTotal * config('Tax')/TaxPercent
```

`config()` is an XRules extension function to XPath, and it takes one parameter which is the id of the `<config>` section and returns a reference to the root of the configuration

section element (which is the `<config>` element). From that point, the regular XPath syntax is used to locate the exact node required in the expression.

2.10 XRules Properties

XRules properties extend the XML Infoset by attaching named values to XML nodes. There are two types of XRules properties:

2.10.1 XRules Automatic Properties

When an XRules document is bound to an XML document, rules and rulesets are bound to one or more nodes and attach some information or impose some constraints on these nodes. For example, a `bind` rule with a `type` attribute attaches an XML Schema type to its target node. And, a ruleset with a `minOccurs` attribute constraints the minimum occurrence of its context node to a specific value. XRules automatic properties are generated by the XRules processor to provide easy access to the information and constraints attached to XML nodes.

The following example illustrates the use of these properties.

```
<xr:ruleset context="/PurchaseOrder/Item">
   <xr:bind target="Quantity" type="xs:positiveInteger" />
</xr:ruleset>

<xr:ruleset context="/PurchaseOrder">
   <xr:bind target="PaymentMethod">
      <xr:valueset>
         <xr:enum value="cash" />
         <xr:enum value="credit" />
         <xr:enum value="check" />
      </xr:valueset>
   </xr:bind>
</xr:ruleset>
```

In the example above, the first `bind` rule attaches to the `/PurchaseOrder/Item` elements and specify the XML Schema data type for these nodes. The data type is exposed as an XRules automatic property with the name `xr:type` for each `Item` node. If you're using the Dynamic DOM, this is the C# syntax used to access this property.

```
string quantityType = quantityNode.Properties["xr:type"].Value;
```

Similarly, the second `bind` rule in the example above attaches a property with the name `xr:valueset` to the `PaymentMethod` node. When accessed using the DDOM implementation, this property returns an array that contains the values "cash", "credit", and "check".

```
string[] allowedPayMethods = paymentMethodNode.Properties["xr:valueset"].Value;
```

All XRules automatic properties start with the "`xr:`" prefix. This prefix is reserved for XRules usage.

2.10.2 XRules User-Defined Properties

These are properties defined using the `<property>` element of the `bind` rule, or using APIs of the XRules processor. User-defined properties can be accessed by the application program using the same syntax used to access XRules automatic properties.

The following example uses XRules properties to calculate the shipping fee for the order without having to update the XML document by adding new elements or attributes. The business rules in the example are: first item in the order always ships free, and the rest of the items ship free if their value is greater than $500, otherwise the shipping fee is $9.99 per item.

Example 7 XRules Properties

XML Document:

```
<PurchaseOrder>
   <Item>
      <Name>PC Computer</Name>
      <Quantity>1</Quantity>
      <UnitPrice>1300</UnitPrice>
      <ItemTotal></ItemTotal>
   </Item>
   <Item>
      <Name>XML Book</Name>
      <Quantity>4</Quantity>
      <UnitPrice>30</UnitPrice>
      <ItemTotal></ItemTotal>
   </Item>
   <SubTotal></SubTotal>
   <Tax></Tax>
   <GrandTotal></GrandTotal>
</PurchaseOrder>
```

XRules Document:

```
<xr:rules xmlns:xr="http://www.xrules.org/2003/11"
        xmlns:xs="http://www.w3.org/2001/XMLSchema">

   <xr:ruleset context="/PurchaseOrder">
      <!-- First item always ships free -->
      <xr:bind target="Item[1]">
         <xr:property name="ChargeForShipping" value="N" />
      </xr:bind>
```

```
        <!-- Other items ship free if the total is greater than $500 -->
        <xr:bind target="Item[position() &gt; 1]">
            <xr:property name="ChargeForShipping"
                         dvalue="iif(target()/ItemTotal &gt; 500, 'N', 'Y')" />
        </xr:bind>

        <!-- Shipping fee is $9.99 for each item that is not exempt -->
        <xr:bind target=".">
            <xr:property name="ShippingFee"
                dvalue="9.99 * count(Item[property('ChargeForShipping')='Y'])" />
        </xr:bind>
    </xr:ruleset>

    <xr:ruleset context="/PurchaseOrder/Item">
        <xr:calculate target="ItemTotal">
            <xr:value>UnitPrice * Quantity</xr:value>
        </xr:calculate>
    </xr:ruleset>

    <xr:ruleset context="/PurchaseOrder">
        <xr:calculate target="SubTotal">
            <xr:value>sum(Item/ItemTotal)</xr:value>
        </xr:calculate>
        <xr:calculate target="Tax">
            <xr:value>SubTotal * 0.05</xr:value>
        </xr:calculate>
        <xr:calculate target="GrandTotal">
            <xr:value>SubTotal + Tax</xr:value>
        </xr:calculate>
    </xr:ruleset>

</xr:rules>
```

The first rule in the example, repeated below, attaches a property with the name `ChargeForShipping` to the first `Item` element in the order. This is done be using a `<bind>` rule that selects the node `Item[1]` as its target node. Then the property rule attaches the specified property with a value of N.

```
        <!-- First item always ships free -->
        <xr:bind target="Item[1]">
            <xr:property name="ChargeForShipping" value="N" />
        </xr:bind>
```

The second rule applies to all `Item` elements except the first one, and attaches a property with the same name, `ChargeForShipping`, to them. The value of the property is N if the `ItemTotal` value is greater than $500 and Y otherwise.

Note here the use of two XPath extension functions that XRules provides:

- `iif(expression, trueValue, falseValue)` which provides the *Immediate If* functionality. The first argument is a Boolean expression. If it evaluates to True, the function returns the value of the second argument, otherwise, the function returns the value of the third argument.

- `target()` which returns the target node. It's used here to access the `ItemTotal` value under the target node (remember that the context node is always the node determined in the ruleset element, which in this case is `PurchaseOrder`).

```
<!-- Other items ship free if the total is greater than $500 -->
<xr:bind target="Item[position() &gt; 1]">
   <xr:property name="ChargeForShipping"
                dvalue="iif(target()/ItemTotal &gt; 500, 'N', 'Y')" />
</xr:bind>
```

Also note in the code above the use of the `dvalue` attribute which takes and evaluates an XPath expression, as opposed to the `value` attribute used in the first rule which takes a static value.

The third rule calculates the total shipping fee for the order and attaches the value to the `PurchaseOrder` node as a property named `ShippingFee`. The `<bind>` rule selects the current context node "." as its target (which in this case is the `PurchaseOrder` node as specified in the parent `ruleset`). Then, the property rule uses the `dvalue` attribute to dynamically calculate the value of the property.

```
<!-- Shipping fee is $9.99 for each item that is not exempt -->
<xr:bind target=".">
   <xr:property name="ShippingFee"
      dvalue="9.99 * count(Item[property('ChargeForShipping')='Y'])" />
</xr:bind>
```

Note in the code above the use of the `property()` XPath extension function to access the value of the `ChargeForShipping` property assigned by the first two rules. This function takes one of the following two forms:

- `property(propertyName)`: Returns the value of the specified property on the current context node of execution. In the example above, since the function is called inside the square brackets of the `Item` node, the context of execution is the `Item` node itself (per XPath rules). Note that the context of execution for the XPath expression as a whole is still the context node specified by the parent `ruleset` element.
- `property(propertyName, targetNode)`: Similar to the first form except that the property is read from the target node specified by the XPath expression in the second parameter.

Now that the properties are attached to the target nodes, they can be accessed by the application using the following syntax (for DDOM using C#):

```
double shippingFee = purchaseOrderNode.Properties["ShippingFee"].Value;
```

2.10.3 Accessing XRules Properties in XSLT

Once properties are attached to elements and attributes, they can be accessed using either application APIs as shown above, or in XPath expressions as shown in the XSLT example below. This example uses the previous XML and XRules documents then applies the XSLT transformation shown below to the resulting XML document. Note that after applying an XRules document, the XML Infoset is updated in memory to include the automatic and user-defined XRules properties. The `xrules.exe` tool provides the ability to apply XRules documents and then an XSLT transformation on the updated Infoset.

Example 8 Accessing XRules Properties in XSLT

XSLT Document:

```
<xsl:stylesheet version="1.0"
              xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
              xmlns:xre="http://www.xrules.org/2003/11/ext">

   <xsl:template match="/">
      <html>
         <head></head>
         <body>
            <h3>Items:</h3>
            <xsl:apply-templates select="/PurchaseOrder/Item" />
            <br/><br/>

            Total Shipping Fee: $<xsl:value-of
                  select="xre:property('ShippingFee',PurchaseOrderx)" />
         </body>
      </html>
   </xsl:template>

   <xsl:template match="Item">
      Name:                <xsl:value-of select="Name" /><br/>
      Item Total:         $<xsl:value-of select="ItemTotal" /><br/>
      Charge for Shipping: <xsl:value-of
                              select="xre:property('ChargeForShipping',.)"/>
      <br/><br/>
   </xsl:template>

</xsl:stylesheet>
```

Note that accessing XRules properties is done using the `xre:property()` extension function, which belongs to the namespace `http://www.xrules.org/2003/11/ext`.

The example above generates the following HTML output:

```
<html xmlns:xre="http://www.xrules.org/2003/11/ext">
   <head>
   </head>
   <body>
      <h3>Items:</h3>
      Name: PC Computer<br />
      Item Total: $1300<br />
      Charge For Shipping: N<br />
      <br />
      Name: XML Book<br />
      Item Total: $120<br />
      Charge For Shipping: Y<br />
      <br />
      <br />
      <br />
      Total Shipping Fee: $9.99
   </body>
</html>
```

2.11 XRules and W3C XML Schema

XRules works with XML Schema in two ways:

- Embedding XRules rules in XML Schema documents.
- Embedding XML Schema snippets in XRules documents.

XRules complements XML Schema by allowing it to carry complex rules that are not possible or easy to describe using XML Schema alone. The rules of XRules can be embedded in an XML Schema document inside `<appinfo>` elements, and therefore don't affect the way the XML Schema documents are processed by non-XRules-aware parsers.

2.11.1 Embedding XRules in XML Schema

XRules allows attaching rules to three XML Schema components: elements, attributes, and complex types. When rules are attached to an element or an attribute, these rules apply to the specified elements and attributes only. However, when rules are attached to a complex type, then the rules apply to every element of that complex type. This is an important feature that allows binding rules to XML elements in ways that XPath alone can't describe efficiently (at least, not until XPath 2.0 in which XML Schema types are supported).

Example 9 Embedding XRules in XML Schema documents

```
<xs:schema targetNamespace="http://www.MyPurchaseOrder.org"
       xmlns:po="http://www.MyPurchaseOrder.org"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:xr="http://www.xrules.org/2003/11"
       attributeFormDefault="qualified"
       elementFormDefault="qualified">
```

```
    <xs:element name="PurchaseOrder">
        <xs:complexType>
            <xs:sequence>

                <xs:element name="Item" type="po:ItemType"
                            minOccurs="0" maxOccurs="unbounded" />
                <xs:element name="SubTotal" minOccurs="0" type="xs:decimal">
                    <xs:annotation>
                        <xs:appinfo>
                            <!-- rules attached to an element -->
                            <xr:calculate target=".">
                                <xr:value>sum(/PurchaseOrder/Item/ItemTotal)</xr:value>
                            </xr:calculate>
                        </xs:appinfo>
                    </xs:annotation>
                </xs:element>

                <xs:element name="Tax" type="xs:decimal">
                    <xs:annotation>
                        <xs:appinfo>
                            <xr:calculate target=".">
                                <xr:value>../po:SubTotal * 0.08</xr:value>
                            </xr:calculate>
                        </xs:appinfo>
                    </xs:annotation>
                </xs:element>

                <xs:element name="GrandTotal" type="xs:decimal">
                    <xs:annotation>
                        <xs:appinfo>
                            <xr:calculate target=".">
                                <xr:value>../po:SubTotal + ../po:Tax</xr:value>
                            </xr:calculate>
                        </xs:appinfo>
                    </xs:annotation>
                </xs:element>

            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="ItemType">
        <xs:annotation>
            <xs:appinfo>
                <!-- rules attached to a complexType -->
                <xr:calculate target="ItemTotal">
                    <xr:value>UnitPrice * Quantity</xr:value>
                </xr:calculate>
            </xs:appinfo>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="Name" type="xs:string" minOccurs="0" />
            <xs:element name="Quantity" type="xs:positiveInteger" minOccurs="0" />
            <xs:element name="UnitPrice" type="xs:decimal" minOccurs="0" />
            <xs:element name="ItemTotal" type="xs:decimal" minOccurs="0" />
        </xs:sequence>
    </xs:complexType>

</xs:schema>
```

Implementation Note: Support for XML Schema as described in this section is not yet implemented in the current version of the DDOM. It's planned for a future version.

2.11.2 Embedding XML Schema in XRules

In certain situations, you might need to use XML Schema to validate a portion of an XML document. While you can use a stand-alone XML Schema document for this purpose, you can also embed your schema in an XRules documents using the `<schema>` rule. This is useful in situations like:

- You want to validate only the portion of an XML document that you're interested in. XML Schema is more suitable for validating an entire XML document.
- You want to validate certain elements against a schema if they satisfy certain conditions and against another schema in other cases. The example below shows the case of validating an `<item>` in a purchase order against a schema snippet only if the price of the item exceeds a preset limit.
- You want to utilize the easy access to node constraints that is made available through XRules properties of the DDOM. For example, to check the minimum valid value for a node in C# use: `myNode.Properties["xr:min"]`. This is much easier for many implementations than using the SOM (Schema Object Model) to parse the Schema document.

Example 10 Embedding XML Schema Snippets in XRules documents

```
<xr:rules xmlns:xr="http://www.xrules.org/2003/11"
          xmlns:po="http://www.MyPurchaseOrder.org"
          xmlns:xs="http://www.w3.org/2001/XMLSchema">

   <xr:ruleset select="/po:PurchaseOrder/po:Item">
      <!-- Schema rule to embed an XML Schema snippet for <item> elements -->
      <xs:schema targetNamespace="http://www.MyPurchaseOrder.org">
         <xs:element name="Item">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="Name" type="xs:string" />
                  <xs:element name="Quantity" type="xs:positiveInteger" />
                  <xs:element name="UnitPrice" type="xs:decimal" />
                  <xs:element name="ItemTotal" type="xs:decimal" />
               </xs:sequence>
            </xs:complexType>
         </xs:element>
      </xs:schema>

      <xr:calculate target="po:ItemTotal">
         <xr:value>po:UnitPrice * po:Quantity</xr:value>
      </xr:calculate>

   </xr:ruleset>

   <xr:ruleset select="/po:PurchaseOrder">
      <xr:calculate target="po:SubTotal">
         <xr:value>sum(po:Item/po:ItemTotal)</xr:value>
      </xr:calculate>
      <xr:calculate target="po:Tax">
         <xr:value>po:SubTotal * 0.08</xr:value>
      </xr:calculate>
      <xr:calculate target="po:GrandTotal">
         <xr:value>po:SubTotal + po:Tax</xr:value>
      </xr:calculate>
   </xr:ruleset>
```

```
</xr:rules>
```

Implementation Note: Support for XML Schema as described in this section is not yet implemented in the current version of the DDOM. It's planned for a future version.

3 XRules Tools

As of the writing of this tutorial, two XRules utilities and a test application are available to implement and test XRules documents. All three tools require that the .NET Framework 1.1 is installed on your computer.

3.1 XRules Command-Line Utility

This is a command-line executable, "`xrules.exe`", that applies one or more XRules documents to an XML target document in updating mode and writes out the updated XML document and the generated XRules report. It can also perform an XSLT transformation on the updated XML document to allow the XSLT to access XRules properties and use them in the transformation.

```
xrules.exe /xml:<file>.xml /xrules:<file>.xr [/out:<file>.xml]
           [/report:<file>.xml] [xslt:<file>.xslt]
```

Where,

`/xml:<file>.xml`
: The target XML file to which XRules will be applied.

`/xrules:<file>.xr`
: The XRules file name. This option can be repeated multiple times.

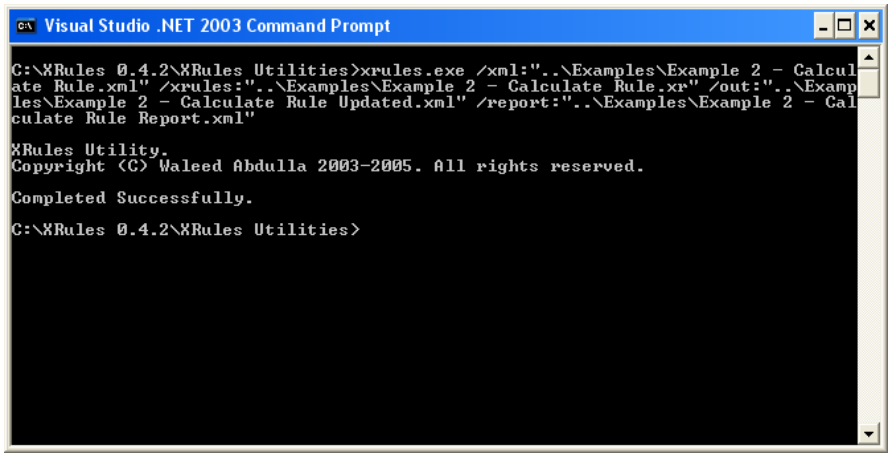
`/out:<file>.xml`
: The updated XML document after applying XRules (and optionally XSLT).

`/report:<file>.xml`
: The XRules report which lists errors resulted after applying XRules.

`/xslt:<file>.xslt`
: An XSLT file to apply to the target XML document after XRules are applied.

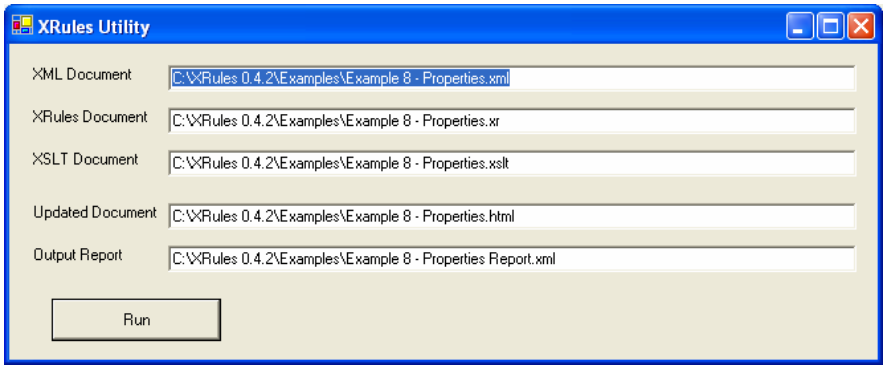
The following command line executes Example 2 above and writes the XRules report and the updated XML document to the Examples folder:

```
C:\XRules 0.43\XRules Utility>xrules.exe
 /xml:"..\Examples\Example 2 - Calculate Rule.xml"
 /xrules:"..\Examples\Example 2 - Calculate Rule.xr"
 /out:"..\Examples\Example 2 - Calculate Rule Updated.xml"
 /report:"..\Examples\Example 2 - Calculate Rule Report.xml"
```



3.2 XRules Windows Utility

The XRules Windows Utility, `winxrules.exe`, is similar in functionality to the command-line utility, but adds a simple graphical user interface as shown below.



3.3 Dynamic DOM Test Application

This test application allows you to see the Dynamic DOM in action. You can make changes to XML nodes and immediately see the effects propagate to other nodes and see the XML report change dynamically.

1. Load an XML document. Click “Load XML…” and select a file.

2. Load an XRules document. Click “Load XRules…” and select a file.
3. Bind the documents. Click the “Bind” button. Once clicked, it changes to “Unbind” to allow you to unbind the documents.
4. Once the XRules document is bound, the XML document is automatically updated based on the rules and an XRules report is generated.
5. Right-click on any node in the XML document tree on the left to edit, delete, or add nodes. You’ll see the effect of any change immediately reflected on other nodes and the XRules report.
6. Right-click on any node in the XML document tree and select “Properties” to see the XRules properties attached to the node.