David Dodds

http://www. open-meta.com

Open-Meta Computing Inc

Using Open Source Software to Process Geo-Semantic / Geo-Spatial Knowledge and Visuals

Meta-programmed eventListeners push activation records on the Call Stack and persistent AR list

Virtual linking and result ontology fragments are referenced via PCS

--What an XML schema is and some uses of GML schemas
(examples)
XML Schema. A language used to define the structure (and data types)
of specific XML languages.
--The kinds of errors that schemas can help detect and
correct (examples)

```
<?xml version="1.0"?>
<purchaseOrder orderDate="1999-10-20">
  <shipTo country="US">
    <name>Alice Smith</name>
    <street>123 Maple Street</street>
    <city>Mill Valley</city>
    <state>CA</state>
    <zip>90952</zip>
  </shipTo>
  <billTo country="US">
    <name>Robert Smith</name>
    <street>8 Oak Avenue</street>
    <city>Old Town</city>
    <state>gronk</state> PA but not AP
    <zip>gronk</zip> 95819
```

```
  </billTo>
  <comment>Hurry, my lawn is going wild!</comment>
  <items>
    <item partNum="872-AA">
      <productName>Lawnmower</productName>
      <quantity>1</quantity>
      <USPrice>148.95</USPrice>
      <comment>Confirm this is electric</comment>
    </item>
    <item partNum="926-AA">
      <productName>Baby Monitor</productName>
      <quantity>1</quantity>
      <USPrice>39.98</USPrice>
      <shipDate>1999-05-21</shipDate>
    </item>
  </items>
</purchaseOrder>



[(( <state>PA</state>  <zip>95819</zip>  gronk))]


<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:annotation>
    <xsd:documentation xml:lang="en">
     Purchase order schema for Example.com.
     Copyright 2000 Example.com. All rights reserved.
    </xsd:documentation>
  </xsd:annotation>

  <xsd:element name="purchaseOrder" type="PurchaseOrderType"/>

  <xsd:element name="comment" type="xsd:string"/>

  <xsd:complexType name="PurchaseOrderType">
    <xsd:sequence>
      <xsd:element name="shipTo" type="USAddress"/>
      <xsd:element name="billTo" type="USAddress"/>
      <xsd:element ref="comment" minOccurs="0"/>
      <xsd:element name="items"  type="Items"/>
    </xsd:sequence>
    <xsd:attribute name="orderDate" type="xsd:date"/>
  </xsd:complexType>

  <xsd:complexType name="USAddress">
    <xsd:sequence>
      <xsd:element name="name"   type="xsd:string"/>
```

```
    <xsd:element name="street" type="xsd:string"/>
    <xsd:element name="city"   type="xsd:string"/>
    <xsd:element name="state"  type="xsd:string"/>
    <xsd:element name="zip"    type="xsd:decimal"/>
   </xsd:sequence>
   <xsd:attribute name="country" type="xsd:NMTOKEN"
                  fixed="US"/>
  </xsd:complexType>

  <xsd:complexType name="Items">
   <xsd:sequence>
    <xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name="productName" type="xsd:string"/>
       <xsd:element name="quantity">
        <xsd:simpleType>
         <xsd:restriction base="xsd:positiveInteger">
          <xsd:maxExclusive value="100"/>
         </xsd:restriction>
        </xsd:simpleType>
       </xsd:element>
       <xsd:element name="USPrice"  type="xsd:decimal"/>
       <xsd:element ref="comment"   minOccurs="0"/>
       <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
      </xsd:sequence>
      <xsd:attribute name="partNum" type="SKU" use="required"/>
     </xsd:complexType>
    </xsd:element>
   </xsd:sequence>
  </xsd:complexType>

  <!-- Stock Keeping Unit, a code for identifying products -->
  <xsd:simpleType name="SKU">
   <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{3}-[A-Z]{2}"/>  ((regex)
   </xsd:restriction>
  </xsd:simpleType>

</xsd:schema>
```

Example
Using the Enumeration Facet

((explicit inclusion list, no exclusion))
((no regex))

```
<xsd:simpleType name="ColourName">
```

```
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="red"/>
    <xsd:enumeration value="orange"/>
    <xsd:enumeration value="yellow"/>
    <xsd:enumeration value="green"/>
    <xsd:enumeration value="blue"/>
    <!-- and so on ... -->
  </xsd:restriction>
</xsd:simpleType>


<xsd:simpleType name="USState">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="AK"/>
    <xsd:enumeration value="AL"/>
    <xsd:enumeration value="AR"/>
    <!-- and so on ... -->
  </xsd:restriction>
</xsd:simpleType>
```

some uses of GML schemas are covered
on the website open-meta.com

```
                    :
                    :
<blah>contentgoeshere</blah>
<DirectionString>gronk</DirectionString>            !validates,  needs semantic qualifier
                    :

<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.opengis.net/gml" xmlns:gml="http://www.opengis.net/gml"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:xlink="http://www.w3.org/1999/xlink" elementFormDefault="qualified"
       version="3.1.1">
          <annotation>
                    <appinfo source="urn:opengis:specification:gml:schema-
xsd:direction:3.1.1">direction.xsd</appinfo>
                    <documentation>This schema defines "direction" element and type.
   Copyright (c) 2002-2005 OGC, All Rights Reserved.
          For conditions, see OGC Software Notice
http://www.opengeospatial.org/about/?page=ipr</documentation>
          </annotation>
          <!-- ======================================================================
      includes and imports
          ====================================================================== -->
          <include schemaLocation="geometryBasic0d1d.xsd"/>
          <!-- ====================================================================== -->
          <!--
```

```
========================================================================== -->
	<element name="direction" type="gml:DirectionPropertyType"/>
	<!--
========================================================================== -->
	<complexType name="DirectionPropertyType">
		<annotation>
			<documentation/>
		</annotation>
		<choice>
			<element ref="gml:DirectionVector"/>
			<element ref="gml:CompassPoint"/>
			<element name="DirectionKeyword" type="gml:CodeType"/>
			<element name="DirectionString" type="gml:StringOrRefType"/>
		</choice>
		<attributeGroup ref="gml:AssociationAttributeGroup"/>
	</complexType>
	<!--
========================================================================== -->
	<element name="DirectionVector" type="gml:DirectionVectorType"/>
	<!--
========================================================================== -->
	<complexType name="DirectionVectorType">
		<annotation>
			<documentation>Direction expressed as a vector, either using components, or
using angles.</documentation>
		</annotation>
		<choice>
			<element ref="gml:vector"/>
			<sequence>
				<element name="horizontalAngle" type="gml:AngleType"/>
				<element name="verticalAngle" type="gml:AngleType"/>
			</sequence>
		</choice>
	</complexType>
	<!--
========================================================================== -->
	<element name="CompassPoint" type="gml:CompassPointEnumeration"/>
	<!--=space.owl relates 'N' string with numerics,and other spatial concepts***=  -->
	<!--=space.owl relates 'N' string with numerics,and other spatial concepts***=  -->
	<!--=*********************************************************=  -->
	<simpleType name="CompassPointEnumeration">
		<restriction base="string">
			<enumeration value="N"/>
			<enumeration value="NNE"/>
			<enumeration value="NE"/>
			<enumeration value="ENE"/>
			<enumeration value="E"/>
			<enumeration value="ESE"/>
			<enumeration value="SE"/>
```

```
                        <enumeration value="SSE"/>
                        <enumeration value="S"/>
                        <enumeration value="SSW"/>
                        <enumeration value="SW"/>
                        <enumeration value="WSW"/>
                        <enumeration value="W"/>
                        <enumeration value="WNW"/>
                        <enumeration value="NW"/>
                        <enumeration value="NNW"/>
                </restriction>
        </simpleType>
        <!--
================================================================================  -->
</schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.opengis.net/gml" xmlns:gml="http://www.opengis.net/gml"
xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" version="3.1.1">
        <annotation>
                <appinfo source="urn:opengis:specification:gml:schema-xsd:dynamicFeature:3.1.1"/>
                <documentation xml:lang="en">Basic support for tracking moving objects and objects
with changing state.
   Copyright (c) 2002-2005 OGC, All Rights Reserved.
        For conditions, see OGC Software Notice
http://www.opengeospatial.org/about/?page=ipr</documentation>
        </annotation>
        <!--
================================================================================ -->
        <include schemaLocation="feature.xsd"/>
        <include schemaLocation="direction.xsd"/>
        <!--
================================================================================ -->
        <element name="dataSource" type="gml:StringOrRefType"/>
        <element name="status" type="gml:StringOrRefType"/>
        <!--
================================================================================ -->
        <element name="_TimeSlice" type="gml:AbstractTimeSliceType" abstract="true"
substitutionGroup="gml:_GML"/>
        <!-- ====================================== -->
        <complexType name="AbstractTimeSliceType" abstract="true">
                <annotation>
                        <documentation xml:lang="en">A timeslice encapsulates the time-varying
properties of a dynamic feature--it
      must be extended to represent a timestamped projection of a feature. The dataSource
      property describes how the temporal data was acquired.</documentation>
                </annotation>
```

```
            <complexContent>
                <extension base="gml:AbstractGMLType">
                    <sequence>
                        <element ref="gml:validTime"/>
                        <element ref="gml:dataSource" minOccurs="0"/>
                    </sequence>
                </extension>
            </complexContent>
        </complexType>
        <!--
================================================================== -->
        <element name="MovingObjectStatus" type="gml:MovingObjectStatusType"
substitutionGroup="gml:_TimeSlice"/>
        <!-- ===================================== -->
        <complexType name="MovingObjectStatusType">
            <annotation>
                <documentation xml:lang="en">This type encapsulates various dynamic
properties of moving objects
         (points, lines, regions). It is useful for dealing with features whose
         geometry or topology changes over time.</documentation>
            </annotation>
            <complexContent>
            <!-- = ********************************************************* == -->
            <!-- = ********************************************************* == -->
            <!-- = ********************************************************* == -->
            <!-- = ((!!***time.owl classrdf:ID="Duration")) !!!!! *** AbstractTimeSlice  == -->
                <extension base="gml:AbstractTimeSliceType">
                    <sequence>
                        <element ref="gml:location"/>
                        <element name="speed" type="gml:MeasureType"
minOccurs="0"/>
                        <element name="bearing" type="gml:DirectionPropertyType"
minOccurs="0"/>
                        <element name="acceleration" type="gml:MeasureType"
minOccurs="0"/>
                        <element name="elevation" type="gml:MeasureType"
minOccurs="0"/>
                        <element ref="gml:status" minOccurs="0"/>
                    </sequence>
                </extension>
            </complexContent>
        </complexType>
        <!--
================================================================== -->
        <element name="history" type="gml:HistoryPropertyType"/>
        <!-- ===================================== -->
        <complexType name="HistoryPropertyType">
            <annotation>
                <documentation xml:lang="en">The history relationship associates a feature
```

```
with a sequence of TimeSlice instances.</documentation>
                </annotation>
                <sequence maxOccurs="unbounded">
                        <element ref="gml:_TimeSlice"/>
                </sequence>
        </complexType>
        <!--
========================================================================== -->
        <element name="track" type="gml:TrackType" substitutionGroup="gml:history"/>
        <!-- ======================================== -->
        <complexType name="TrackType">
                <annotation>
                        <documentation xml:lang="en">The track of a moving object is a sequence of
specialized timeslices       that indicate the status of the object.</documentation>
                </annotation>
                <complexContent>
                        <restriction base="gml:HistoryPropertyType">
                                <sequence maxOccurs="unbounded">
                                        <element ref="gml:MovingObjectStatus"/>
                                </sequence>
                        </restriction>
                </complexContent>
        </complexType>
        <!--
========================================================================== -->
        <group name="dynamicProperties">
                <sequence>
                        <element ref="gml:validTime" minOccurs="0"/>
                        <element ref="gml:history" minOccurs="0"/>
                        <element ref="gml:dataSource" minOccurs="0"/>
                </sequence>
        </group>
        <!--
========================================================================== -->
        <complexType name="DynamicFeatureType">
                <annotation>
                        <documentation>A dynamic feature may possess a history and/or a
timestamp.</documentation>
                </annotation>
                <complexContent>
                        <extension base="gml:AbstractFeatureType">
                                <group ref="gml:dynamicProperties"/>
                        </extension>
                </complexContent>
        </complexType>
        <!-- ======================================== -->
        <complexType name="DynamicFeatureCollectionType">
                <annotation>
                        <documentation>A dynamic feature collection may possess a history and/or a
```

```
timestamp.</documentation>
                </annotation>
                <complexContent>
                        <extension base="gml:FeatureCollectionType">
                                <group ref="gml:dynamicProperties"/>
                        </extension>
                </complexContent>
        </complexType>
        <!-- ========================================== -->
</schema>
```

XSLT stuff right after the XML Schema stuff

```
<xsl:stylesheet version="1.0"
xmlns:xml="http://www.w3.org/1999/XSL/Transform"
xmlns:xalan="http://xml.apache.org/xslt"
xmlns:count="http://www.ora.com/XSLTCookbook/extend/counter">

<xsl:output method="text"/>

<xalan:component prefix="count"
                        functions="counter nextCount resetCount makeCounter">
    <xalan:script lang="javascript">

        function counter(initValue)
        {
          this.value = initValue ;
        }

        function nextCount(ctr)
        {
          return ctr.value++ ;
        }

        function resetCount(ctr,value)
           {
                ctr.value = value ;
                   return "" ;
        }

        function makeCount(initValue)
        {
                   return new counter(initValue) ;
        }
    </xalan:script>
</xalan:component
```

```
<xsl:template match="/">
  <xsl:variable name="aCounter" select="count:makeCount(0)"/>
  Count: <xsl:value-of select="count:nextCount($aCounter)"/>
  Count: <xsl:value-of select="count:nextCount($aCounter)"/>
  Count: <xsl:value-of select="count:nextCount($aCounter)"/>
  Count: <xsl:value-of select="count:nextCount($aCounter)"/>
  <xsl:value-of select="count:resetCount($aCounter,0)"/>
  Count: <xsl:value-of select="count:nextCount(#aCounter)"/>
</xsl:template>
</xsl:stylsheet>
```

qqqqq

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xalan="http:////xml.apache.org/xslt
xmlns:regex="http://www.ora.com/XSLTCookbook/extend/regex">

<xsl:output method="text"/>

<xalan:component prefix="regex"
    functions="match leftContext rightContext getParenMatch makeRegExp">
  <xalan:script lang="javascript">

     function Matcher(pattern)
     {
          this.re = new RegExp(pattern) ;
          this.re.compile(pattern) ;
          this.result= "" ;
          this.left="" ;
          this.right="" ;
       }

         function match(matcher, input)
      {
                  matcher.result = matcher.re.exec(input) ;
                  matcher.left = RegExp.leftContext ;
                  matcher.right = RegExp.rightContext ;
                  return matcher.result[0] ;
         }

          function leftContext(matcher)
      {
            return matcher.left ;
      }
```

```
        function rightContext(matcher)
        {
                return matcher.right ;
        }

        function getParenMatch(matcher, which)
        {
                return matcher.result[which]  ;
        }

        function makeRegExp(pattern)
        {
                return new Matcher(pattern) ;
        }
  </xalan:script>
</xalan:component>

<xsl:template match="/">
  <xsl:variable name="dateParser"
  select="regex:makeRegExp('(\d\d?)[/-](\d\d?)[/-](\d{4}|\d{2})')"/>
  Match: <xsl:value-of
          select="regex:match($dateParser,
          'I was born on 04/11/1855 in London.')"/>
  Left: <xsl:value-of select="regex:leftContext($dateParser)"/>
  Right: <xsl:value-of select="regex:rightContext($dateParser)"/>
  Month: <xsl:value-of select="regex:getParenMatch($dateParser, 1)"/>
  Day: <xsl:value-of select="regex:getParenMatch($dateParser, 2)"/>
   Year: <xsl:value-of select="regex:getParenMatch($dateParser, 3)"/>
</xsl:template>
</xsl:stylsheet>
```

This code results in:

```
        Match: 04/11/1855
        Left: I was born on
        Right: in London.
        Month: 04
        Day: 11
        Year: 1855
```

qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
For example: SAXON XSLT

```
<xsl:value-of select="math:sqrt($arg)"
  xmlns:math="java:java.lang.Math"/>
```

This will invoke the static method java.lang.Math.sqrt(), applying it to the value of the variable $arg, and copying the value of the square root of $arg to the result tree.

from jclark.com XT documentation:

'The following stylesheet prints the date and time. This example is copied from the documentation of the xt product, and it works unchanged with SAXON, because SAXON does not care what the namespace URI for extension functions is, so long as it ends with the class name. (Extension functions are likely to be compatible between SAXON and xt provided they only use the data types string, number, and boolean).'

```
<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:date="http://www.jclark.com/xt/java/java.util.Date">

<xsl:template match="/">
  <html>
    <xsl:if test="function-available('date:to-string') and function-available('date:new')">
      <p><xsl:value-of select="date:to-string(date:new())"/></p>
    </xsl:if>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Note that Saxon's tree structure conforms to the DOM Core Level 2 interface. However, it is read-only: any attempt to modify the tree causes an exception. Saxon's trees can only be built using the Saxon subclasses of the com.icl.saxon.tree.Builder class, and they cannot be modified in situ.

Writing Java node handlers

A Java node handler can be used to process any node, in place of an XSL template. The handler is nominated by using a saxon:handler element with a handler attribute that names the node handler class. The handler itself is an implementation of com.icl.saxon.NodeHandler or one of its subclasses (the most usual being com.icl.saxon.ElementHandler). The saxon:handler element must be a top-level element, and must be empty. It takes the same attributes as xsl:template (match, mode, name, and priority) and is considered along with xsl:template elements to decide which template to execute when xsl:call-template or xsl:apply-templates is used.

Java node handlers have full access to the source document and the current processing context (for example, the values of parameters). The may also trigger processing of other nodes in the document by calling applyTemplates(): this works just like xsl:apply-templates, and the selected nodes may be processed either by XSL templates or by further Java node handlers.

A Java node handler may also be registered with a name, and may thus be invoked using xsl:call-template. There is no direct mechanism for a Java node handler to call a named XSLT template, but the effect can be achieved by using a mode that identifies the called template uniquely.

Writing input filters

SAXON takes its input from a SAX2 Parser reading from an InputSource. A very useful technique is to interpose a filter between the parser and SAXON. The filter will typically be an instance of the SAX2 XMLFilter class.

See the TrAX examples for hints on using a Saxon Transformer as part of a chain of SAX Filters.

((an error-correcting parser can be built by you coding the SAX eventHandlers adequately))

Note that SAXON relies on the application to supply a well-balanced sequence of SAX events; it doesn't need to be well-formed (the root node can have any number of element or text children), but if it isn't well-balanced, the consequences are unpredictable.

Writing output filters

The output of a SAXON stylesheet can be directed to a user-defined output filter. This filter can be defined either as a standard SAX1 DocumentHandler, a SAX2 ContentHandler, or as a subclass of the SAXON class com.icl.saxon.output.Emitter. The advantage of using an Emitter is that more information is available from the stylesheet, for example the attributes of the xsl:output element.

When a ContentHandler is used, Saxon will by default always supply a stream of events corresponding to a well-formed document. (The XSLT specification also allows the output to be an external general parsed entity.) If the result tree is not well-formed, Saxon will notify the content handler of the fact by sending a processing instruction with the name "saxon:warning" and the text "Output suppressed because it is not well-formed". If the content handler is happy to accept output that is not well-formed, it can respond to this processing instruction by throwing a SAXException whose message text is "continue"; in this case subsequent events will be notified whether or not they are well-formed.

The best way to see how to implement an extension element is by looking at the example, for SQL extension elements, provided in package com.icl.saxon.sql, and at the sample stylesheet books.sqlxsl which uses these extension elements.

Saxon
parse(string)

This function takes a single argument, a string containing the source text of a well-formed XML document. It returns the document node (root node) that results from parsing this text. It throws an error if the text is not well-formed XML. Applications should not rely on the identity of the returned document node (at present, if the function is called twice with the same arguments, it returns a new document node each time, but this may change in future).

This function is useful where one XML document is embedded inside another using CDATA, or as an alternative way of passing an XML document as a parameter to a stylesheet.

eval(stored-expression)

This returns the result of evaluating the supplied stored expression. A stored expression may be obtained as the result of calling the saxon:expression() function.

The stored expression is evaluated in the current context, that is, the context node is the current node, and the context position and context size are the same as the result of calling position() or last() respectively.

Example: saxon:eval(saxon:expression(concat(2, $op, 2)))

evaluate(string)

The supplied string must contain an XPath expression. The result of the function is the result of evaluating the XPath expression. This is useful where an expression needs to be constructed at run-time or passed to the stylesheet as a parameter, for example where the sort key is determined dynamically. The context for the expression (e.g. which variables and namespaces are available) is exactly the same as if the expression were written explicitly at this point in the stylesheet. The function saxon:evaluate(string) is shorthand for saxon:eval(saxon:expression(string)).

expression(string)

The supplied string must contain an XPath expression. The result of the function is a stored expression, which may be supplied as an argument to other extension functions such as saxon:eval(), saxon:sum() and saxon:distinct(). The result of the expression will usually depend on the current node. The expression may contain references to variables that are in scope at the point where saxon:expression() is called: these variables will be replaced in the stored expression with the values they take at the time saxon:expression() is called, not the values of the variables at the time the stored expression is evaluated. Similarly, if the expression contains namespace prefixes, these are interpreted in terms of the namespace declarations in scope at the point where the saxon:expression() function is called, not those in scope where the stored expression is evaluated.

Look at dir xsltMyExamples which is inside XSLTstuff

An extended example which converts Visio to SVG can be downloaded from http://sourceforge.net/projects/vdxtosvg/.

Look at dir KML examples

Look at SVG dir examples.

Look at MathML examples. Note Javascript parser and calculator engine in OntoClock.

--What an XML ontology is, and how it is different from schemas (examples)

Ontology. Languages used to define vocabularies and establish the usage of words and terms in the context of a specific vocabulary. RDF Schema is a framework for constructing ontologies and is used by many more advanced ontology frameworks. OWL is an ontology language designed for the Semantic Web.

Ontologies
The classic AI-oriented (Artificial Intelligence) view would be that by classifying the words and concepts in web resources into categories defined by a suitable ontology, the essential contents of the resources could be captured and matched to the concepts behind the categories. Failing that, the terms used in a query could at least be automatically related to known terms from the ontology.

Space ontology http://sweet.jpl.nasa.gov/ontology/space.owl

Space is essentially a multidimensional numerical scale with terminology specific to the spatial domain. We developed a space ontology in which the spatial extents and relations are special cases of numeric extents and relations, respectively. Spatial extents include: country, Antarctica, equator, inlet, etc. Spatial relations include: above, northOf, etc.

These ontologies constitute a concept space for Earth system science. The expression of a concept in words may differ from one person to another depending upon their scientific community, cultural background, language, and level of expertise. SWEET enables the same concept to be represented using various phrases to satisfy these needs. Rather than define a compound concept such as air temperature, we separated the physical property (temperature) from the element that the property applies to (air). This provides a more scalable solution to a growing knowledge base. In this case, knowledge of the independent concepts of the substance "air" and property "temperature" provide a complete understanding of "air temperature" without a need to create an explicit definition of the compound concept. Such a decomposition does not preclude term recompositions, but the compound terms are designated as synonymous with their integral parts. In other instances, the compound concepts contain more meaning than their component parts (e.g., static pressure) and are explicitly included in the ontology.

A deficiency of RDF and OWL is that the languages contain no direct support for numerical concepts, and must rely on a limited XSD (XML Schema Definition) specification of datatypes. This spec defines number types (e.g., floating point, unsigned integer) and methods to create derivations of these types (e.g. the closed interval between 0 and 1), but contains no operations or relations on these numbers. This is a deficiency, because many scientific concepts are defined through numeric concepts. For example, "brighter", "higher", "later", and "more northerly" are special cases of the "greater than" relation, applied in specific domains. In particular, spectral regions are defined in terms of wavelength (e.g., visible light is between 0.4 and 0.7 micrometers), atmospheric layers are defined by altitude (e.g., troposphere is between 0 and 15 km), etc. This specification also has no notion of a multidimensional space Rn. The Numerics ontology adds extensions needed to define scientific concepts and is used to define concepts in the spatial and temporal ontologies. Although other spatial and temporal ontologies

exist, none exploit the fact that space and time are numerical scales. Without such a connecting thread, many numerical concepts must be reinvented to create definitions of space and time.

((driver.owl.xml)) [compare with ((person.owl.xml)) below, MERGE, ALIGN]

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.owl-ontologies.com/unnamed.owl#"
  xml:base="http://www.owl-ontologies.com/unnamed.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Driver"/>
  <owl:Class rdf:ID="Pedestrian"/>
  <owl:Class rdf:ID="Car"/>
  <owl:ObjectProperty rdf:ID="drives">
    <rdfs:range rdf:resource="#Car"/>
    <rdfs:domain rdf:resource="#Driver"/>
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:ID="Model">
    <rdfs:domain rdf:resource="#Car"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  </owl:DatatypeProperty>
  <owl:FunctionalProperty rdf:ID="name">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:domain rdf:resource="#Pedestrian"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="DriverName">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:domain rdf:resource="#Driver"/>
  </owl:FunctionalProperty>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 2.2, Build 331)  http://protege.stanford.edu -->
```

((person.owl.xml))

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.owl-ontologies.com/unnamed.owl#"
```

```
  xml:base="http://www.owl-ontologies.com/unnamed.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Person"/>
  <owl:Class rdf:ID="Car"/>
  <owl:ObjectProperty rdf:ID="drives">
    <rdfs:range rdf:resource="#Car"/>
    <rdfs:domain rdf:resource="#Person"/>
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:ID="model">
    <rdfs:domain rdf:resource="#Car"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:FunctionalProperty rdf:ID="name">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Person"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>
  <Person rdf:ID="Bob">
    <drives>
      <Car rdf:ID="Honda">
        <model rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >Honda</model>
      </Car>
    </drives>
    <drives>
      <Car rdf:ID="Ford">
        <model rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >Ford</model>
      </Car>
    </drives>
    <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Bob</name>
  </Person>
  <Person rdf:ID="Cherie">
    <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Cherie</name>
  </Person>
  <Car rdf:ID="Chrysler">
    <model rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Chrysler</model>
  </Car>
  <Person rdf:ID="Alice">
    <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Alice</name>
    <drives rdf:resource="#Chrysler"/>
  </Person>
</rdf:RDF>
```

<!-- Created with Protege (with OWL Plugin 2.2, Build 331) http://protege.stanford.edu -->

((humans.owl.xml))

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.owl-ontologies.com/unnamed.owl#"
  xml:base="http://www.owl-ontologies.com/unnamed.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Human"/>
  <owl:DatatypeProperty rdf:ID="Name">
    <rdfs:domain rdf:resource="#Human"/>
  </owl:DatatypeProperty>
  <Human rdf:ID="Bob">
    <Name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Bob</Name>
  </Human>
  <Human rdf:ID="Dave">
    <Name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Dave</Name>
  </Human>
  <Human rdf:ID="Clarence">
    <Name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Clarence</Name>
  </Human>
  <Human rdf:ID="Alice">
    <Name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Alice</Name>
  </Human>
</rdf:RDF>
```

<!-- Created with Protege (with OWL Plugin 2.2, Build 324) http://protege.stanford.edu -->

(some item extracts from)
http://sweet.jpl.nasa.gov/ontology/material_thing.owl

```
<owl:Class rdf:ID="Equipment">
<rdfs:subClassOf rdf:resource="#Infrastructure"/>
</owl:Class>
<owl:Class rdf:ID="Instrument">
```

<rdfs:subClassOf rdf:resource="#Equipment"/>
</owl:Class>
<owl:Class rdf:ID="Facility">
<rdfs:subClassOf rdf:resource="#Infrastructure"/>
</owl:Class>
<owl:Class rdf:ID="SchoolDistrict">
<rdfs:subClassOf rdf:resource="#Infrastructure"/>
</owl:Class>
<owl:Class rdf:ID="Structure">
<rdfs:subClassOf rdf:resource="#Facility"/>
</owl:Class>
<owl:Class rdf:ID="Building">
<rdfs:subClassOf rdf:resource="#Structure"/>
</owl:Class>
<owl:Class rdf:ID="Computer">
<rdfs:subClassOf rdf:resource="#Equipment"/>
</owl:Class>

Class and subClassOf are the referents which signify levels / directions of hierarchy.

Directed graph structure is indicated by those two terms. Inferencing can use graph theory techniques on structures (such as ontologies) built from RDFfamily elements such as subClassOf.

For example, in the above ontology excerpt we see that a Computer and a Building are different kinds of Infrastructure. The Computer is an Equipment kind of Infrastructure and the Building is a Facility kind of Infrastructure.

<owl:Class rdf:ID="Barn">
<rdfs:subClassOf rdf:resource="#Building"/>
</owl:Class>
<owl:Class rdf:ID="School">
<rdfs:subClassOf rdf:resource="#Building"/>
</owl:Class>
<owl:Class rdf:ID="House">
<rdfs:subClassOf rdf:resource="#Building"/>
</owl:Class>
<owl:Class rdf:ID="GasStation">
<rdfs:subClassOf rdf:resource="#Building"/>
</owl:Class>

If our computer system determines that a School District has an Infrastructure then it can infer, by means of the previously mentioned inferencing techniques, that a School District may have Schools, Houses, and even Barns, and equipment such as computers. <SchoolDistrict> appears below in http://www.opengis.net/examples/schools.xml . (now ex-parrot)

There are 15 different ontologies in SWEET:
http://sweet.jpl.nasa.gov/ontology/ Earth Realm,

Physical Phenomena, Physical Process, Physical Property, Sun Realm, Biosphere, Data, Data Center, Human Activity, Material Thing, Numerics, Sensor, Space, Time, Units.

space.owl.xml (snippet)

<owl:Class rdf:ID="Downward">
<owl:equivalentClass rdf:resource="#Down"/>
</owl:Class>
<owl:Class rdf:ID="Base">
<owl:equivalentClass rdf:resource="#Bottom"/>
</owl:Class>
<owl:Class rdf:ID="Bottom">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#hasDirection"/>
<owl:allValuesFrom rdf:resource="#Down"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf
rdf:resource="http://sweet.jpl.nasa.gov/ontology/numerics.owl#Maximum"/>
</owl:Class>

expressed in OWL a class-name y is expressed to be a synonym of classname x by using the equivalentClass term, such as Y is a synonym of X, =
<owl:Class rdf:ID="Yyy">
<owl:equivalentClass rdf:resource="#Xxxx"/>

the snippet above says:
"base is the maximum downward direction and is a synonym for bottom", the base of a statue is therefore the maximum downward direction in/of the statue if foot is located at the bottom of x then foot is located at the maximum downward direction in/of x. Foot / feet located at 'bottom' of statue, as is the foot of a cliff, and the bottom of a cloud on high is at the maximum downward direction of the cloud structure. There's no one home in the computer and so we have to explain everything in code.

Look at OWL dir examples **********

(from CYC ontology)
<rdf:Property rdf:ID="above-Directly">
<rdfs:label xml:lang="en">above - directly</rdfs:label>
<rdfs:comment>(#$above-Directly ABOVE BELOW) means either that (1) the volumetric center of ABOVE is directly
above some point of BELOW, if ABOVE is smaller than BELOW; or that (2) some point of ABOVE is directly above the
volumetric center of BELOW, if ABOVE is larger than, or equal in size to, BELOW.</rdfs:comment>
<guid>bd58fbde-9c29-11b1-9dad-c379636f7270</guid>
<rdfs:subPropertyOf rdf:resource="#above-Generally"/>
<rdfs:domain rdf:resource="#SpatialThing-Localized"/>
<rdfs:range rdf:resource="#SpatialThing-Localized"/>
</rdf:Property>

Listing F.2.2.2 schools.xml
xsi:schemaLocation="http://www.opengis.net/examples/schools.xsd"

school district(s), from GML dataset.
Each has a stringname and has a named geographical region as a location indicator or a collection of points (coordinates) in
some spatial reference system. Since they are spatially disjoint they have locations in "direction" (like north, south,
northwest, etc) relative to / from each other. Regions have an area magnitude aka size. They have "content" (generally) such
as grass , dirt, water, trees, buildings, animals / people, roads, in various degrees at various sub-regions or #$Place's.
Therefore school district 5 may be 'north' of school district 7 and part of a school district's #$boundary may be #$Water (of
type #$Ocean, for example.) Maybe these two districts are contained in a super-region called 'Pacific NorthWest'. Pacific is
a reference to the ocean which partly borders this super-region. NorthWest is a direction (away) from a central point or
origin (aka '1030' on the clock face)
Examples of CYC ontology knowledge fragments:
#$SpatiallyDisjointRegionType region types whose instances are non-overlapping region types
..each of the regions in the collection is spatially disjoint with the other regions in the collection. Other instances of
#$SpatiallyDisjointRegionType include #$CanadianProvince, #$IndependentCountry, #$City, and #$Colony. A nonexample
is #$EcologicalRegion, since ecological regions can overlap.
guid: bd58e513-9c29-11b1-9dad-c379636f7270
direct instance of: #$SecondOrderCollection #$AtemporalNecessarilyEssentialCollectionType
direct specialization of: #$RegionType

#$bordersOn borders on
(#$bordersOn REGION1 REGION2) means that the #$GeographicalRegions REGION1 and REGION2 are physically
adjacent to each other and do not overlap, i.e. there is a border between them. Examples: (#$bordersOn
#$CentralUSATimeZone #$MountainUSATimeZone), (#$bordersOn (#$TerritoryFn #$Nepal)

(#$TerritoryFn #$Tibet).
guid: bd58e17a-9c29-11b1-9dad-c379636f7270
direct instance of: #$SymmetricBinaryPredicate #$IrreflexiveBinaryPredicate
#$InterExistingObjectPredicate
#$SpatialPredicate
direct specialization of: #$onSamePlanetSurfaceAs #$adjacentTo #$touchesDirectly-Apartanomic

```
<?xml version="1.0" standalone="yes" ?>
<svg xmlns = 'http://www.w3.org/2000/svg' id="metadata-GML-2006" width="450" height="450" >
<title id="test-title">metadata-GML-2006</title>
<desc id="test-desc">Verify that the SVG viewer handles
the presence of (GML) metadata and associated elements.</desc>
<metadata xmlns="http://www.opengis.net/examples" xmlns:gml="http://www.opengis.net/gml"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://www.opengis.net/examples/schools.xsd"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >
<State>
<gml:description>
Educational institutions with student populations exceeding 500.
</gml:description>
<gml:name>School districts in the North Region.</gml:name>
<gml:boundedBy>
<gml:Box srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
<gml:coordinates>0,0</gml:coordinates>
<gml:coordinates>50,50</gml:coordinates>
</gml:Box>
</gml:boundedBy>
<gml:featureMember>
<SchoolDistrict>
<gml:name>District 28</gml:name>
<gml:boundedBy>
<gml:Box srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
<gml:coordinates>0,0</gml:coordinates>
<gml:coordinates>50,40</gml:coordinates>
</gml:Box>
</gml:boundedBy>
<schoolMember>
<School>
<gml:name>Alpha</gml:name>
<address>100 Cypress Ave.</address>
<gml:location>
<gml:Point srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
<gml:coordinates>20.0,5.0</gml:coordinates>
```

```
</gml:Point>
</gml:location>
</School>
</schoolMember>
<schoolMember>
<School>
<gml:name>Beta</gml:name>
<address>1673 Balsam St.</address>
<gml:location>
<gml:Point srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
<gml:coordinates>40.0,5.0</gml:coordinates>
</gml:Point>
</gml:location>
</School>
</schoolMember>
<gml:extentOf>
<gml:Polygon srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
<gml:outerBoundaryIs>
<gml:LinearRing>
<gml:coordinates>0,0</gml:coordinates>
<gml:coordinates>50,0</gml:coordinates>
<gml:coordinates>50,40</gml:coordinates>
<gml:coordinates>0,0</gml:coordinates>
</gml:LinearRing>
</gml:outerBoundaryIs>
</gml:Polygon>
</gml:extentOf>
</SchoolDistrict>
</gml:featureMember>
<gml:featureMember>
<SchoolDistrict>
<gml:name>gronk</gml:name> District 32
<gml:boundedBy>
<gml:Box srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
<gml:coordinates>0,0</gml:coordinates>
<gml:coordinates>30,50</gml:coordinates>
</gml:Box>
</gml:boundedBy>
<schoolMember>
<School>
<gml:name>Gamma</gml:name>
<address>651 Sequoia Ave.</address>
<gml:location>
<gml:Point srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
<gml:coordinates>5.0,20.0</gml:coordinates>
</gml:Point>
</gml:location>
</School>
</schoolMember>
```

<schoolMember>
<College>
<gml:name>Delta</gml:name>
<address>260 University Blvd.</address>
<gml:pointProperty>
<gml:Point srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
<gml:coordinates>5.0,40.0</gml:coordinates>
</gml:Point>
</gml:pointProperty>
</College>
</schoolMember>
<schoolMember xlink:type="simple" xlink:title="Epsilon High School"
xlink:href="http:www.state.gov/schools/cgibin/
wfs?schoolID=hs736"
gml:remoteSchema="schools.xsd#xpointer(//complexType[@name='SchoolType'])"/>
<gml:extentOf>
<gml:Polygon srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
<gml:outerBoundaryIs>
<gml:LinearRing>
<gml:coordinates>0,0</gml:coordinates>
<gml:coordinates>40,50</gml:coordinates>
<gml:coordinates>50,50</gml:coordinates>
<gml:coordinates>0,0</gml:coordinates>
</gml:LinearRing>
</gml:outerBoundaryIs>
</gml:Polygon>
</gml:extentOf>
</SchoolDistrict>
</gml:featureMember>
<studentPopulation>392620</studentPopulation>
</State>
<rdf:RDF
<owl:Ontology rdf:about="">
<owl:Class rdf:ID="schoolMember">
<rdfs:subClassOf rdf:resource="#SchoolDistrict"/>
<owl:Class rdf:ID="School">
<rdfs:subClassOf rdf:resource="#schoolMember"/>
<owl:Class rdf:ID="College">
<rdfs:subClassOf rdf:resource="#schoolMember"/>
<rdf:Property rdf:ID="above-Directly">
<rdfs:label xml:lang="en">above - directly</rdfs:label>
<rdfs:comment>(#$above-Directly ABOVE BELOW) means either that (1) the volumetric center of ABOVE is directly
above some point of BELOW, if ABOVE is smaller than BELOW; or that (2) some point of ABOVE is directly above the
volumetric center of BELOW, if ABOVE is larger than, or equal in size to, BELOW.</rdfs:comment>
<guid>bd58fbde-9c29-11b1-9dad-c379636f7270</guid>
<rdfs:subPropertyOf rdf:resource="#above-Generally"/>
<rdfs:domain rdf:resource="#SpatialThing-Localized"/>

<rdfs:range rdf:resource="#SpatialThing-Localized"/>
</rdf:Property>
</rdf:RDF>
</metadata>
<rect id="box1" x="0" y="0" width="50" height="50" style="stroke:green; fill:none; stroke-width:1" />
<title> School districts in the North Region. </title>
<desc> Educational institutions with student populations exceeding 500. </desc>
<rect id="box2" x="0" y="0" width="50" height="40" style="stroke:blue; fill:none; stroke-width:1" />
<title>District 28</title>
<rect id="box3" x="0" y="0" width="30" height="50" style="stroke:brown; fill:none; stroke-width:1" />
and a bunch more SVG statements which draw the boxes, points and other geometric figures described in the
GML metadata above and place the associated text near by.
</svg>

findings by the system:
object1 named School districts in the North Region., isa area rectangle 2500 untyped units.
object2 named District 28, isa area rectangle 2000 untyped units.
object2 is instance1 childOf SchoolDistrict instance1.
object3 named Alpha, isa location point 20.0,5.0 untyped units, note address 100 Cypress Ave.
object4 is School instance2 hasChild named Beta, isa location point 40.0,5.0 untyped units, note address 1673 Balsam St.
object5 named District 32, isa area rectangle 1500 untyped units.
object5 is instance1 childOf SchoolDistrict instance2.
SchoolDistrict instance2 hasChild schoolMember School instance1 hasChild named Gamma, isa location point 5.0,20.0
untyped units.
SchoolMember instance2 hasChild instance1 College named Delta, isa location point 5.0,40.0 untyped units.
Contained in the area named School districts in the North Region is the area named District 28 which is the largest of the
two contained areas there.
(upper left coordinate “corner” = zero, zero; = “North” and “West”)
100 Cypress Ave is [qualitative amount] “left of” (west of?) 1673 Balsam St.
Delta College is [qualitative amount] “below” (south of?) both Beta School and Gamma School.

Merging ontologies
is a process that intends to join different ontologies
about overlapping domains into a new one and most of its
problems and techniques are related to the identification of
similar concepts through structure analysis (e.g. graph analysis,
path length, common nodes or/and edges and lexical

analysis). (see diagrams in the PDFs, passin_chp5.pdf   pdf page=15   bookpage=119)

'Engineering a complex ontology with time' (excerpt from)

"In particular, we (Jorge Santos, Steffen Staab) have
included the notion of Role as a core concept. While there
are concepts that give identity to their instances (i.e. they are
semantically rigid [Guarino andWelty, 2000]), e.g. while the
identity of a particular person depends on being an instance of
Person, the identity of the same person does not change when
it ends being a student and starts being a professor. Thus, the
notion of Role is important when connecting a temporal theory
with a concrete domain."
[Joe Blow is (always, until deceased) an instance of Person,
at time = 'before PHD' he is a student, afterward he is a professor.]

((right at the end of the file named=NE_States.svg (the SVG geo map SVG file)   ))

<g id="NE_Text" style="display:inline;fill-rule:evenodd">
<desc>Layer NE_Text</desc>
<text x="621301.0" y="-4914782.1" style="font-family:Arial;fill:rgb(80,80,80);font-size:36606.77;">
Montpelier
</text>
<text x="560317.7" y="-4730610.3" style="font-family:Arial;fill:rgb(80,80,80);font-size:36600.31;">
Albany
</text>
<text x="629004.0" y="-4632895.0" style="font-family:Arial;fill:rgb(80,80,80);font-size:36604.26;">
Hartford
</text>
<text x="857157.1" y="-4935912.0" style="font-family:Arial;fill:rgb(80,80,80);font-size:36635.43;">
Augusta
</text>
<text x="726639.7" y="-4796183.3" style="font-family:Arial;fill:rgb(80,80,80);font-size:36613.74;">
Concord
</text>
<text x="835919.0" y="-4685282.6" style="font-family:Arial;fill:rgb(80,80,80);font-size:36622.21;">
Boston
</text>
<text x="732885.5" y="-4599485.6" style="font-family:Arial;fill:rgb(80,80,80);font-size:36614.46;">
Providence
</text>
</g>
</svg>

XTM-D Topic Map to represent contexts and inter-ontology relationships ("alignment")
(see diagrams MERGE ALIGN pdfPage=2 noy99algorithm.pdf)

```
<?xml-stylesheet type="text/xsl" href="xgmmlContext1xsl.xsl"?>
   <!DOCTYPE graph SYSTEM "xgmml.dtd">
   <graph xmlns="http://www.cs.rpi.edu/XGMML" >
   <node id="1" label="timestamp" weight="0">
   <graphics type="circle" x="270" y="90" h="10" >
   </graphics>
   <att name="systemtimestamp" value="123455432112345"/> <!--EXSLT gets time and date
   from system puts it in node-->
   </node>
   <node id="2" label="AtRight" weight="0">
   <graphics type="circle" x="350" y="190" h="10" >
   </graphics>
   <att name="object" value="1"/>
   <att name="object" value="2"/>
   <att name="origin" value="upperleft"/>
   </node>
   <node id="3" label="object1" weight="0">
   <graphics type="circle" x="190" y="190" h="10" >
   </graphics>
   <att name="xcoord" value="110"/>
   <att name="ycoord" value="35"/>
   <att name="origin" value="upperleft"/>
   </node>
   <node id="4" label="object2" weight="0">
   <graphics type="circle" x="290" y="290" h="10" >
   </graphics>
   <att name="xcoord" value="120"/>
   <att name="ycoord" value="39"/>
   <att name="origin" value="upperleft"/>
   </node>
   <node id="5" label="object1" weight="0">
   <graphics type="circle" x="390" y="390" h="10" >
   </graphics>
   <att name="xcoord" value="210"/>
   <att name="ycoord" value="27"/>
   <att name="origin" value="upperleft"/>
   </node>
   <node id="7" label="object2" weight="0">
   <graphics type="circle" x="490" y="090" h="10" >
   </graphics>
   <att name="xcoord" value="220"/>
   <att name="ycoord" value="29"/>
```

<att name="origin" value="upperleft"/>
</node>
<edge source="2" target="4" weight="0" label="Edge from node AtRight to node object2" >
</edge>
<edge source="1" target="2" weight="0" label="Edge from node timestamp to node AtRight" >
</edge>
<edge source="2" target="3" weight="0" label="Edge from node AtRight to node object1" >
</edge>
</graph>

--The purpose of syntactic and semantic processing
(discussion - review)

--The kinds of errors that ontologies can help detect
and correct (gronk as colour even though string)
(detect inclusion in a parent class:: synonyms: city town hamlet burgh metropolis)
(in part using CYC / Wordnet type processing)
(conversion of RGB triples to colourname and vice-versa, equivalencing)

ADDENDA

```
#TimeInterval  #TemporalObjectType  #TimePoint  #TimeOfDay
<owl:Class rdf:ID="Date">
      <rdfs:label xml:lang="en">dates</rdfs:label>
      <rdfs:comment>A specialization of #$TimeInterval.  Each instance
         of #$Date is a temporally continuous instance of
         #$TimeInterval which can be defined purely by its location
         on a particular calendar.  Thus, an instance of #$Date could
         be a particular calendar day, calendar quarter, calendar
         month, or decade.</rdfs:comment>
      <guid>bd58ac59-9c29-11b1-9dad-c379636f7270</guid>
      <rdf:type rdf:resource="#PublicConstant-DefinitionalGAFsOK"/>
      <rdf:type rdf:resource="#PublicConstant-CommentOK"/>
      <rdf:type rdf:resource="#PublicConstant"/>
      <rdf:type rdf:resource="#TemporalObjectType"/>
      <rdfs:subClassOf rdf:resource="#TimeInterval"/>
      <rdfs:subClassOf rdf:resource="#Individual"/>
   </owl:Class>

    <owl:Class rdf:ID="DayOfWeekType">
        <rdfs:label xml:lang="en">days of the week</rdfs:label>
        <rdfs:comment>A collection of collections and a specialization
           of #$WeeklyTemporalObjectType. Each instance of
           #$DayOfWeekType is a collection of #$CalendarDays, all of
           whose instances correspond to the same particular  day of
           the week  in  the respective weeks in which they occur.  For
           example, #$Monday -- the  collection of all mondays -- is an
           instance of #$DayOfWeekType.</rdfs:comment>
        <guid>bd58d679-9c29-11b1-9dad-c379636f7270</guid>
        <rdf:type rdf:resource="#CyclicalIntervalGroupType"/>
        <rdf:type rdf:resource="#CoreConstant"/>
        <rdf:type rdf:resource="#CollectionType"/>
        <rdf:type rdf:resource="#SiblingDisjointCollectionType"/>
        <rdf:type rdf:resource="#SecondOrderCollection"/>
        <rdfs:subClassOf rdf:resource="#WeeklyTemporalObjectType"/>
        <rdfs:subClassOf rdf:resource="#ConventionallyClassifiedDisjointTimeIntervalType"/>
        <rdfs:subClassOf rdf:resource="#TemporalObjectType"/>
   </owl:Class>
```

XSLT Example

Sample of incoming XML document

```
<?xml version="1.0" ?>
<persons>
  <person username="JS1">
```

```
    <name>John</name>
    <family_name>Smith</family_name>
  </person>
  <person username="MI1">
    <name>Morka</name>
    <family_name>Ismincius</family_name>
  </person>
</persons>
```

[edit] Example 1 (transforming XML to XML)

This XSLT stylesheet provides templates to transform the XML document:

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="xml" indent="yes"/>

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

<xsl:template match="//person">
       <name username="{@username}">
          <xsl:value-of select="name" />
       </name>
</xsl:template>

</xsl:stylesheet>
```

Its evaluation results in a new XML document, having another structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
     <name username="JS1">John</name>
     <name username="MI1">Morka</name>
</root>
```

[edit] Example 2 (transforming XML to XHTML)

Example XSLT Stylesheet:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<xsl:template match="/persons">
       <html xmlns="http://www.w3.org/1999/xhtml">
       <head> <title>Testing XML Example</title> </head>
       <body>
            <h1>Persons</h1>
```

```
            <ul>
            <xsl:apply-templates select="person">
                  <xsl:sort select="family_name" />
            </xsl:apply-templates>
            </ul>
      </body>
      </html>
</xsl:template>

<xsl:template match="person">
      <li>
            <xsl:value-of select="family_name"/>,
            <xsl:value-of select="name"/>
      </li>
</xsl:template>

</xsl:stylesheet>
```

XHTML output that this would produce (whitespace has been adjusted here for clarity):

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
<head> <title>Testing XML Example</title> </head>
<body>
      <h1>Persons</h1>
      <ul>
        <li>Ismincius, Morka</li>
        <li>Smith, John</li>
      </ul>
</body>
</html>
```

Image:xslt_ex2.png