t1.1

An XML-driven data translation engine for XML

Don Murray, President, Safe Software Inc

Abstract

There are a number of XML based spatial data formats currently available (GML1, GML2, SOTF, LandXML, SVG, for example) and many more being developed (GML3 and new version of GDF) throughout the GIS industry.

In addition to the regular spatial data translation challenges XML represents new flexibility in the way in which people can exchange data. This flexibility presents an interesting challenge, as users are able to structure their XML formats in vastly different manners.

One approach to supporting these XML-based formats would be to develop different translation components for each format resulting in a large effort for each format.

The problem with this approach is that as the number of XML based formats for spatial information is growing the amount of development required would be large.

The approach described was to write a single configurable XML data translation engine that enables translation between XML based formats and other GIS formats.

The XML translation engine is configured using XML and is able to exploit the capabilities of our general-purpose data translation components. This enables new XML-based support to be added by anyone thereby enabling new formats to be configured in time measured in hours rather than weeks.

Once added the XML format can be used to move data from the XML based format to any supported GIS format or from any GIS format to the XML based format.

Additional benefits of using this approach are the XML-based format can then be viewed with the toolkit viewing capability and embedded into applications with a generic format API.

Paper

A markup language is a system for marking or tagging documents. The markup supplies the documents with a definite pattern of organization that identifies the document's structures and logical relationships between them. Given this general definition, a markup language can be defined in a variety of ways; but this variation potentially produces closed systems, where sharing markup documents with other systems becomes difficult. The effort required for writing and re-writing software that reads the new markup language is one part of the problem. This part of the problem is alleviated by XML.

The eXtensible Markup Language (XML) is a specification that provides a standard way to define markup languages for textual documents. Markup languages that are defined with XML follow certain lexical and syntactical constraints that make the production of generic XML parsing software possible. Many of these XML parsers are freely available; they can read any of the XML defined markup languages, which ranges from purchase orders to geographical information.

Historically, the task of moving geographic data from one format to another has been difficult. As a result, users with large data stores have been locked into a single vendor's format and have been restricted to using one vendor's analysis and decision support tools. Naturally, the GIS community sees XML as an opportunity to create open and accessible geographic data. There are a number of XML based spatial formats currently available (GML1, GML2, SOTF, LandXML, SVG, for example) and many more being developed (GML3 and a new version of GDF) throughout the GIS industry.

In addition to the regular spatial data translation challenges, XML represents new flexibility in the way in which people can represent data. This flexibility presents an interesting challenge, as users are able to structure their XML formats in vastly different manners.

One approach to supporting these XML-based formats would be to develop different translation components for each format. The problem with this approach is that as the number of XML based formats for spatial information is growing the amount of development required would be large. Although GIS translation components can leverage the use of freely available XML parsers for lexical and syntactical analysis, the semantic analysis -- the meaning of the XML elements -- is left to the GIS translation component resulting in a potentially large effort for each format.

Another approach is to write a single configurable XML data translation engine that enables translation between XML based formats and other GIS formats. The XML translation engine is configured by mapping rules (themselves, represented in XML); in other words, the translation engine interprets the meaning of the XML elements through the mapping rules. This enables new XML-based format support to be added with little effort when compared to traditional application development.

When the XML data translation engine is coupled to a data translation hub such as the Feature Manipulation Engine (FME), the XML-based formats can be translated to any of the large array of other formats supported by the hub. The FME data translation hub represents both spatial and non-spatial features in a format neutral manner -- a feature is the atomic data packet of FME and is considered to be a collection of attribute names and values with optional associated geometry. For these reasons, the XML data translation engine constructs, out of the XML elements, features that can be passed to the translation hub. The following sections describe how the XML data translation engine converts the XML-based formats elements into features.

The XML data translation engine is XML driven; this means, that it solves the requirement for interpreting XML-based formats not with complex programming code, but with declarative XML-based mapping rules that map from the elements in the XML-based format into the neutral features of the data translation hub.

In designing the XML data translation engine and its declarative mapping rules, a stream-based processing model was chosen for the input of the XML-based formats. The potential size of geographical data makes a tree-based processing model un-feasible.

The Processing Model

The XML data translation engine is event driven; it takes two input XML documents: the XML-based format document and another document that contains the declarative mapping rules (called an X-Map). The mapping rules react to the input stream of elements from the XML-based formats. The XML data translation engine may activate, execute, suspend, or de-activate mapping rules. There are several different types of mapping rules; for now, we'll only consider mapping rules that construct the hub's neutral geographic features; these mapping rules are called "feature mapping rules".

The XML data translation engine constructs one feature at a time. The first feature mapping rule that activates, builds a new feature; this feature may be worked upon by subsequent feature mapping rules that activate; the feature is considered complete only after the original feature mapping rule de-activates. After a geographic feature is completely constructed it is passed on to the data translation hub.

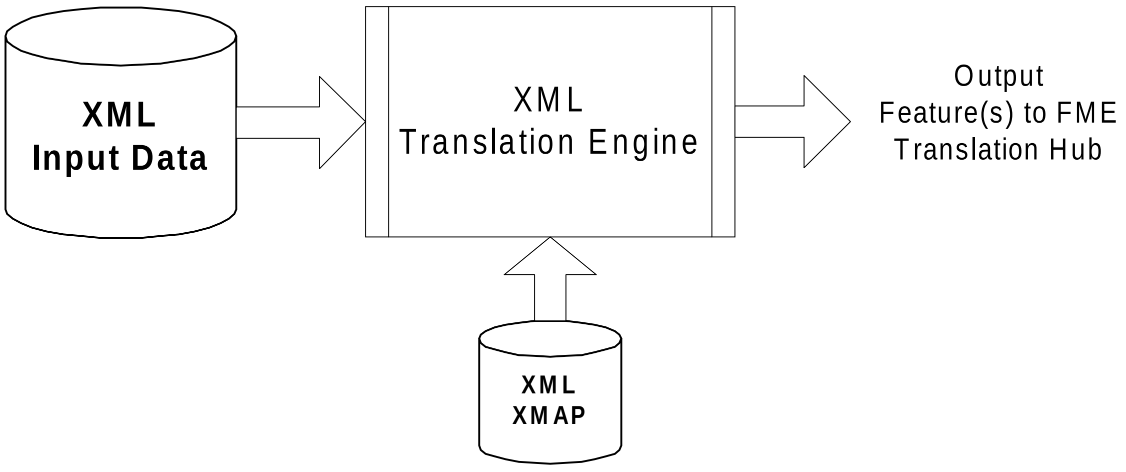

Figure 1: Data Processing Flow

Consider the following XML document fragment:

Fragment 1.

```
<building>
   <featureCode>1234</featureCode>
   <theme>City</theme>
   <location x=”10.0” y=”0.0”/>
</ building >
```

If we want a feature to be constructed on the “building” element for the above XML fragment; then, we must have a feature mapping rule that activates right when the XML translation engine reads the “building” element’s start tag. The feature is considered constructed after the activated feature mapping rule de-activates; that is, once the XML translation engine reads the “building” element’s end tag. The following mapping rule satisfies the requirements:

Fragment 2.

```
<mapping match=”building”>
</mapping>
```

The geographic feature that is constructed is a vacuous one; it has no feature-type, attributes nor geometry. The following is the textual representation of the vacuous feature (it is a log of the feature from the data translation hub):

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Feature Type: `'
Attribute: `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
============================================================================
```

The feature-type and attributes may be constructed by adding the ‘feature-type’ and ‘attributes’ elements to the above feature mapping rule:

Fragment 3.

```
<mapping match="building">
   <feature-type>
      <extract expr="./theme"/>
   </feature-type>
   <attributes>
      <attribute>
         <name>  <literal expr="featureCode"/> </name>
         <value>  <extract expr="./featureCode"/> </value>
      </attribute>
   </attributes>
</mapping>
```

The “extract” element allows extraction of information from the input XML stream. Since the input document is read in a streaming manner, the “extract” element in a mapping rule can locate and extract information only from the sub-tree whose root is the element that activated the mapping rule.

The feature type of the geographic feature is set by the “feature-type” element. In fragment 3, the value of the “extract” element’s “expr” attribute, that is, “./theme”, evaluates to “City”. This is because, in fragment 2, the “building” element caused the activation of the mapping rule; and the “building” element has a “theme” child element whose content is “City”.

The attributes of the geographic feature are set by the “attributes” element. Notice that each “attribute” element has a “name” and “value” element as content. The “literal” element, under the “name” element, sets the geographic feature’s attribute name to the literal value of its “expr” attribute, in this case it sets it to “featureCode”. While the “extract” element, under the “value” element, sets the value of geographic attribute to “1234”. The log below shows the constructed geographic feature with its feature-type and an attribute, but the constructed feature still lacks geometry:

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Feature Type: `Buildings'
Attribute: `featureCode' has value `1234'
Attribute: `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
===========================================================================
```

The geometry can be constructed by adding a ‘geometry’ element to the above mapping rule:

Fragment 4.

```
<mapping match="building">
   ...
   <geometry activate="xml-point">
      <data name="data-string">
          <extract expr="./location[@x]"/>
          <literal expr=","/>
          <extract expr="./location[@y]"/>
      </data>
   </geometry>
</mapping>
```

The following is the log of the geographic feature. Notice that a point geometry feature is constructed:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Feature Type: `Buildings'
Attribute: `featureCode' has value `1234'
Attribute: `fme_geometry' has value `fme_point'
Attribute: `xml_type' has value `xml_point'
Geometry Type: Point (1)
Number of Coordinates: 1 -- Coordinate Dimension: 2 -- Coordinate System: `'
(10,0)
============================================================================
```

In fragment 4, the “geometry” element in the feature mapping rule directs the XML translation engine to create the pre-defined xml-point geometry builder. The XML translation engine contains several pre-defined geometry builders that are non-format specific. These geometry builders are capable of constructing geometry for points, lines, areas, aggregates, and annotations. In addition, the XML translation engine can be easily extended with new geometry builders as need requires. Every geometry builder receives the information that it needs from the “geometry” element’s “data” elements. The names and values for the “data” elements are depended on the geometry builder. The xml-point geometry builder requires a “data” elements with the name “data-string” and the value must be the coordinate string to parse. The xml-point geometry builders accepts other “data” elements as well, but they are optional; these optional “data” elements can specify the coordinate dimension, the character(s) that separate each coordinate, the character(s) that separate each axis of the coordinate, the order of the axis of the coordinates (e.g., x, y, z or y, x, z, etc…), and the decimal character for each coordinate (e.g., “.” or “,”).

The “extract” and “literal” elements in the XML translation engine’s mapping rule terms are called “expression elements”. Several of the elements in the feature mapping rule illustrated above accept an “expression sequence” as its content (these include the “feature-type”, “name”, “value”, and “data” elements); as its name implies an “expression sequence” is a sequence of expression elements. The “data” element in fragment 4 above has an expression sequence that consists of an “extract”, a “literal” and “extract” element; when this expression sequence is evaluated its value is the appended value of each of the expression elements in the sequence; in this example it is: “10.0,0.0”. There are several other expression elements that are not illustrated in the above fragments, they are the “tclexpr”, “defnval”, “parmval” expression elements. The “parmval” element allows access to the mapping rule’s parameters (a mapping rule may have an optional parameters section, a mapping rule calling that calls another mapping rule may use these parameters to passed in information). The “defnval” element allows access to expression sequences that are defined earlier in the mapping rule (a mapping rule can have an optional define section which defines expression sequences by name). The “tclexpr” element allows various TCL commands that return a value to be execute, the arguments to these commands are themselves passed in as expression sequences.

The above was a brief description on how the XML translation engine uses activated feature mapping rules to interpret the information of XML elements into the data translation hub’s neutral geographic features. Many GIS XML-based formats structure their data topologically. The primitive geographic features may be constructed out of these formats with the feature mapping rules, while their topology may be constructed with other mapping rules that take advantage of the hub’s inherent geographic feature processing capabilities, these mapping rules are called “group mapping rules”.

Leveraging the Hub’s Feature Manipulation Capabilities

The XML translation engine is coupled to a data translation hub that is capable of performing a large variety of sophisticated transformations on geographic features. These include topological, geometrical, attributes, and coordinate system transformations. The data translation hub provides an API for all these transformations and for which the XML translation engine exposes through “group mapping rules”.

When group mapping rules activate they create group objects. A group object is destroyed when the group mapping rule, that created the group, de-activates (actually, the lifetime of the group object can become a little more complicated than this, but what was stated is usually true). A group object is like a tunnel in which geographic features enter, are transformed, and may or may not exit, all depending on the processing that is performed inside the tunnel. If there are group objects that are “alive” in the XML translation engine, then, before being output to the hub the geographic features that are created by feature mapping rules must first pass through the group objects. The processing that is performed inside a group object is dependent on the structure of the group. The XML translation engine constructs a group object based on the contents of the group mapping rules.

A group mapping rule may contain an “apply-properties” and/or an “apply-pipelines” element.   The ‘apply-properties’ element instructs the XML translation engine to construct one or more property structures in the group object. The property structures can append additional attributes to the geographic features that pass through the group. Consider the following XML fragment:

Fragment 5.

```
<topologicalDataset theme=”parcels”>
  <primitiveEdge id=”1”>...</ primitiveEdge >
  <primitiveEdge id=”2”>...</ primitiveEdge >
  <primitiveEdge id=”3”>...</primitiveEdge>
  <polygon id=”4”>
    <edges>
      <edge>2</edge>
      <edge>3</edge>
      <edge>1</edge>
    </edges>
  </polygon>
</topologicalDataset>
```

Lets assume that we have feature mapping rules that create a geographic features for each the “primitiveEdge” and “polygon” elements above. Recall that after the geographic features are constructed they, before being output to the hub, pass through group objects. The following group mapping rule when activated will instruct the XML translation engine to construct a group object that contains one property structure; these group object will be destroyed after the group mapping rule de-actives – the group mapping rule activates when the XML translation engine reads the ‘topologicalDataset’ element’s start-tag and de-activates when it reads its end-tag:

Fragment 6.

```
<mapping match=”topologicalDataset”>
  <apply-properties>
    <property>
      <attributes>
        <attribute>
          <name> <literal expr=”the-theme”/> </name>
          <value> <extract expr=”@theme”/> </value>
        </attribute>
      </attributes>
    <property>
  </apply-properties>
</mapping>
```

The group object that is defined by the above group mapping rule will append the attribute with “the-theme” as name and “parcels” as value to each of the geographic features that passes through the group.

The ‘apply-pipelines’ element instructs the XML translation engine to construct one or more pipelines that contain the definition of one or more of data translation hub’s feature factories. The feature factories in the data translation hub can perform, on a single and or on group of geographic features, topological, geometrical, attribute, and coordinate system transformations. These factories are specified in a separate text file in the syntax of the data translation hub. The group mapping rule below is activated by the XML translation engine when the “topologicalDataset” element start-tag is read; it creates a group object containing one pipeline that is defined in the “myPipeline.fmi” file:

Fragment 7.

```
<mapping match=" topologicalDataset">
   <apply-pipelines>
      <pipeline>
         <file name="myPipeline.fmi"/>
      </pipeline>
   </apply-pipelines>
</mapping>
```

Lets assume that we have feature mapping rules that construct features out of the XML fragment 5. Each of the geographic features constructed will have their id attribute, the geographic feature constructed for the "polygon" element will in addition contain a list of ids that references the edges. When all of these geographic features pass through the pipeline of the constructed group object, a polygon feature will be topologically constructed by the factories that were defined in the "myPipeline.fmi" file.

Geographic features may also be filtered out of group objects. The group mapping rules also contain a mechanism to filter out features from property and pipeline structures, users may selectively write their group mapping rules to allow and/or disallow the entry of geographic features based on their attribution.

Up to now the gist of the XML translation engine has been given through the descriptions of its mapping rules. The following section briefly describes the engine's implementation.

Implementing the XML Translation Engine

The XML translation engine leverages the free parsing utilities that are available in industry. There are two standard APIs that are used by application software to parse XML documents these are DOM and SAX. The DOM specification defines a tree-based approach to navigating an XML document. Currently, DOM Parsers create an internal in-memory tree based data structure; usage of a DOM Parser on GIS data is thus prohibitive. A SAX Parser is event based, it does not itself construct an internal representation of the XML document, but instead provides callback functions so that application software may handle events. Using a SAX Parser then, does not tax the memory usage, but it does require considerable more effort for application software to build meaningful content out of the XML documents.

A hybrid solution is used to implement the XML translation engine. The DOM Parser is use to read in the mapping rules, while a SAX Parser is use to read the actual input XML document. The consequence of this is that the XML translation engine can read arbitrarily large XML documents. Using SAX means that unless provisions has been made information that came before the current element in the input stream will not be available at hand to the application program. There are various mechanisms that allow mapping rules to access information of elements other than elements in the sub-tree of the current element being read in the input stream by the SAX parser. The mechanisms include mapping rule parameters and reference mapping rules.

The data translation hub to which the XML translation engine is coupled to, is Safe Software's Feature Manipulation Engine (FME). The XML translation engine utilizes two of Safe Software's APIs, the Plug-in Builder API and the FME Objects API. The Plug-in Builder API is used to enable XML to be translated into any of the FME supported formats. The FME Objects API are used extensively in the implementation of the group objects that group mapping rules create. The FME Objects API allows the group objects to access all of the FME's feature factories and features functions. The XML translation engine is Safe Software's XML Reader and its mapping rules are defined in xMap documents.

Conclusion

The XML translation engine replaces complex hard-coded programming code with declarative mapping rules for the interpretation of XML elements – the meaning of the XML elements. This enables new XML-based support to be added by anyone thereby enabling new formats to be configured in time measured in hours rather than weeks.

To date, the XML translation engine is being used to successfully read SOTF, GML2, and DNF data.

References

Extensible Markup Language (XML). Eds. Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler.
<http://www.w3.org/TR/REC-xml>.

Laser-Scan Inc., Spatial Object Transfer Format (SOTF): Initial High-Level Design, v1.2, 18 November 1999.

Open GIS Consortium Inc., Geography Markup Language (GML) 2.0
<http://www.opengis.net/gml/01-029/GML2.html>

Ordnance Survey, The Digital National Framework (DNF).
<http://www.ordsvy.gov.uk/dnf/prod-spec.htm>