

GML – User Perspectives

Mr. Don Murray, President
Safe Software
Suite 2017-7445 132nd Street
Surrey, BC Canada V3W 1J8
Tel: (604) 501-9985
Fax: (604) 501-9965
E-mail: dmurray@safe.com

Keywords: GML, OGC, interoperability, XML, WFS

Abstract: One of the most powerful aspects of GML is the freedom it gives users to define their own custom application schemas. While this capability provides extraordinary flexibility to data modelers, it also creates significant challenges, particularly when the data is interpreted.

Since GML is a relatively new and complex technology, there may be misconceptions about what it does and does not solve. These issues will be discussed and clarified. In addition, there have been several significant deployments of GML as a solution to data archive as well as data interchange. Several implementations will be surveyed and an analysis of the different approaches will be presented. The lessons learned from these early adopters would be of great interest to anyone planning to use GML in their own projects for data input, archive, or interchange.

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. The Geography Markup Language (GML) attempts to alleviate these difficulties by increasing organizations' ability to share geographic information. GML, which is based on the eXtensible Markup Language (XML), is an open and non-proprietary specification used for the transport and storage of geographic information. This paper presents a configurable XML translation engine that enables translation between XML-based formats and other GIS formats. It then describes how the XML translation engine can be automatically configured to read datasets that are based on any arbitrary GML user application schema.

The XML specification provides a standard way for defining markup languages for textual documents; it is a meta-language that allows users to design and format the structural relationships of their documents using strict lexical and syntactical constraints. XML documents are stored in plain text, which introduces numerous beneficial consequences. Because XML is human-readable, a plain text editor can be used to view

documents; it is also easily transmitted across platforms and over the Internet. In addition, plain text is vendor-neutral, so information that is stored in XML is not locked into a proprietary binary format. XML thus enables disparate systems to share information easily – and, since GML is defined with XML, it inherits all of XML’s benefits.

GML uses the W3C XML Schema Definition Language to define and constrain the contents of its XML documents. The GML v2.0 Specification defines some basic conformance requirements for users to develop their own application schemas. Software applications attempting to process any arbitrary GML user application schema must understand GML and all of the technologies upon which GML depends, including the W3C XML Schema.

Many free parsers are available for XML. These parsers may be used by GIS applications as a base building block for implementing GML software modules. Most XML parsers provide the option for validating an XML document through a W3C Schema document. GIS applications can utilize these parsers to read and validate documents for arbitrary GML user application schemas. It must be noted that software applications must still interpret the output from the XML parsers into their own local meaningful context. The software application must know what each XML element in the GML dataset means, whether the element refers to a feature, a property of a feature, or a feature collection. It is not enough for the GIS application software to use the XML parser to validate the dataset according to a schema: the application must also understand how GML uses the W3C Schema to define a geographic feature and its properties. GML introduces an extraordinary flexibility by letting users define their own application schemas suitable for their own domains; however, this same flexibility also presents a substantial difficulty for writing GML software applications.

It is more or less trivial to write a software component that works on a particular GML user application schema. The GML software component can even bypass the schema processing, since all the processing logic for that particular domain can be hard-coded into the software component. The W3C XML Schema document only needs to be examined if, in addition to processing, the user wishes to perform validation on the GML datasets with the XML parser.

It is substantially more difficult to write a software component that works on any arbitrary GML user application schema because the component would be expected to understand any GML dataset. Reading GML documents into a system is trivial since users can employ any of the free XML parsers available. The difficulty comes in interpreting the XML elements into a geographic context and then in interpreting that geographic context into a GIS system’s own local context. GML helps software components interpret XML data by constraining the interpretation of the data into a well-defined geographic context.

Since a GML user application schema is expressed through a W3C XML Schema document, a software component can perform type discovery on the schema to identify which XML elements from the GML dataset represent a feature, a feature’s properties, and a feature’s geometric properties. Currently most of the XML parsers available support validation with the W3C XML Schema on a document but do not expose the W3C XML Schema programmatically through an API. This presents an extra obstacle for GML software application programmers who need type discovery in order to work with

their GML datasets, because admittedly, the W3C XML Schema Recommendation is in no way an easy specification to understand.

Another difficulty when working with geographical data in XML – whether the GML software component handles only one or multiple types of GML user application schemas – is that geographical datasets are inherently large in size. There are two standard APIs that are used by software applications to parse XML documents: DOM and SAX. The DOM specification defines a tree-based approach to navigating an XML document. Currently, a DOM Parser creates an in-memory tree-based data structure, which makes it prohibitive to use on GIS data. A SAX Parser is event-based – it does not itself construct an internal representation of the XML document, but instead provides callback functions for software applications to handle events. Using a SAX Parser does not tax the memory usage, but it does require considerable more effort to program against. The application programmer should avoid using the in-memory representation of the parsed XML document when large geographic datasets are being read.

The next section describes an XML translation engine that enables translation between XML-based formats and other GIS formats. The XML translation engine can be dynamically configured to map XML elements into the engine's local notion of geographical objects. These objects are neutral representations of geographic features: they are not tied to any particular GIS format, can hold geographic and non-geographic information, and can be exported through a general-purpose translation hub into diverse GIS systems. The XML translation engine is configured through mapping rules that are themselves expressed in XML. This substantially reduces the amount of effort needed for a user to support new XML-based formats, since users can write new mapping rules in a matter of hours, rather than taking weeks to code software components using traditional programming.

In designing the XML data translation engine and its declarative mapping rules, a stream-based processing model was chosen, since the potential size of geographical data made the tree-based processing model unfeasible.

THE XML DATA TRANSLATION ENGINE

The XML data translation engine is event-driven. It takes two input XML documents: the XML data document and an xfMap document, which contains its declarative mapping-rules. The xfMap mapping-rules are loaded into the translation engine and react to the data document's input stream of XML elements. These mapping-rules may be activated, executed, suspended and deactivated by the translation engine during key events. These key events will be illustrated later in an example. Although there are different types of xfMap mapping-rules, we'll only consider mapping-rules that construct neutral geographical features from the XML elements. These are called "feature mapping-rules".

The XML data translation engine constructs one feature at a time. The first feature mapping-rule that is activated initiates the construction of a new geographical feature. Subsequent feature mapping-rules that are activated do not create a new geographical feature; they help on the construction of the feature that was already created. The geographical feature is completely constructed only after the deactivation of the initial feature mapping-rule.

The following XML document fragment illustrates the usage of the xfMap feature mapping-rules:

Fragment1

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

If we want a geographic feature to be constructed on the <building> element, then a feature mapping-rule must activate when XML translation engine reads the <building> element's start tag. The following feature mapping-rule will activate when the <building> element's start tag is read:

Fragment2

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

The above feature mapping-rule deactivates when the <building> element's end tag is read. The geographic feature that is constructed is vacuous: it has no feature-type, attributes nor geometry. The following is a textual representation of the vacuous feature (it is an actual log of the geographic feature from the data translation hub):

```
+++++
Feature Type: ``
Attribute(string): `xml_type' has value `xml_no_geom'
```

```
Geometry Type: Unknown (0)
```

```
=====
```

The feature-type and attributes of a geographic feature may be constructed by adding a <feature-type> element and an <attributes> element to the feature mapping-rule above:

Fragment3

```
<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 xfMap's <extract> element allows extraction of information from the input data document. Because the input data document is read in a streaming manner, the <extract> element can only locate and extract information from a sub-tree of the data document whose root element start tag caused the activation of the mapping-rule. In the case above, the two <extract> elements can only extract information from *Fragment1*'s <building> and child elements.

The xfMap's <feature-type> element sets the feature-type for the geographic feature. In *Fragment3*, the <extract> element under the <feature-type> element pulls in the content of *Fragment1*'s <theme> element and sets this content as the feature-type of the geographic feature. The feature-type for the geographic feature is set to "City".

The attributes of the geographic feature are set by the xfMap's <attributes> element. An <attributes> element can contain one or more <attribute>. Each <attribute> has a <name> and a <value>. The feature mapping-rule in *Fragment3* specifies one attribute for the geographic feature. The name of this attribute, "featureCode", is set by the <literal> element; it is the string value specified by its "expr" attribute. The value of the "featureCode" attribute is set to be the content of *Fragment1*'s <featureCode> element, "1234". The textual representation of the geographic feature constructed by the feature mapping-rule in *Fragment3* is:

```
+++++
Feature Type: `City'
Attribute(string): `featureCode' has value `1234'
Attribute(string): `xml_type' has value `xml_no_geom'

Geometry Type: Unknown (0)
=====
```

The geographic feature still lacks geometry. The geometry for a feature can be constructed by adding an xfMap's <geometry> element to a feature mapping-rule:

Fragment4

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

The following is a textual representation of the geographic feature constructed by *Fragment4*'s feature mapping-rule on *Fragment1*'s <building> element. Notice that a two-dimensional point geometry feature with coordinates (10,0) is constructed:

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

In *Fragment4*, the <geometry> element in the feature mapping-rule directs the XML translation engine to construct a point-geometry for the geographic feature by using

the “xml-point” geometry builder. The XML translation engine contains several predefined geometry builders that are format neutral. These geometry builders are capable of constructing point, line, area, and aggregate geometries. In addition, the XML translation engine can be easily extended with new format-specific geometry builders as need arises. Every geometry builder receives the information that it needs from the xfMap’s <geometry> element’s <data> elements. The “xml-point” geometry builder requires a <data> element whose “name” attribute must be “data-string”, and its value must be the coordinate string sequence to parse. The “xml-point” geometry builder also accepts other optional <data> elements, which can be used to specify the coordinate string sequence’s 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 (for example, x, y, z, or y, x, z, etc...), and the decimal character for each coordinate (for example, “.”, or “,”). The names and values for the <data> elements are geometry builder-dependent.

The <extract> and <literal> elements are called “expression elements”. Several of the elements in the above feature mapping-rules had expression elements as their children; these included the xfMap’s <feature-type>, <name>, <value> and <data> elements. An xfMap element that accepts an expression element actually accepts a sequence of them. The <data> element in *Fragment4* has a sequence of expression elements that consists of an <extract>, a <literal> and an <extract> element; the “xml-point” geometry builder will receive the string “10.0,0.0” as the value of its “data-string” data parameter. There are several expression elements that were not illustrated in the above mapping-rule fragments: <tclexpr>, <defnval>, <parmval> and <strexpr>.

The XML translation engine uses the xfMap’s feature mapping-rules to map the XML elements into format-neutral geographic features. This allows for flexibility since the XML translation engine is not hard-coded for any particular XML format. The engine allows the processing for different type of XML based documents into geographical features without the need to write a new software module in a traditional programming language. Any GML dataset based on an arbitrary GML user application schema can be processed once an appropriate xfMap is written.

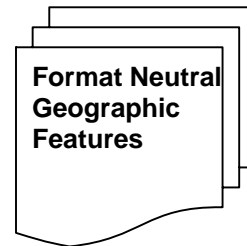
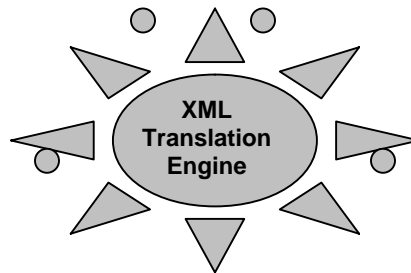
1.1.1 THE XML TRANSLATION ENGINE

1.1.2

1.1.3

1.1.4

1.1.5



1.1.9

1.1.10

1.1.11

1.1.12

1.1.13

1.1.14

1.1.15

1.1.16

1.1.17

1.1.18

1.1.19

1.1.20



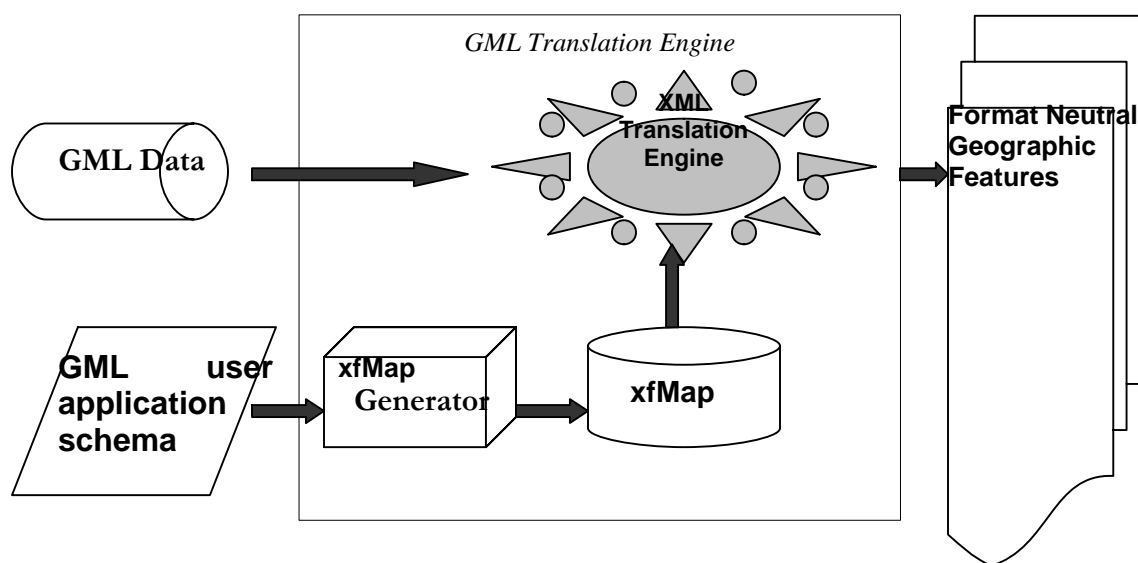
The XML translation engine facilitates the reading of arbitrary GML user application schema datasets as it can interpret arbitrary XML elements with an appropriate xfMap. The xfMap can either be written manually or generated automatically through the examination of the GML user application schema documents. The GML 2.0 Recommendation provides rules and guidelines for users to define their own application schemas with the W3C XML Schema. These rules and guidelines make it possible for a software module to perform type discovery on a GML user application schema. A user may know if an XML element represents a feature, a feature's properties, or a feature's geometric properties by following the type hierarchy for that element in the GML user application schema. When an author creates a GML user application schema, each of its

new feature types, feature collection types, geometry types, and geometry property types must ultimately derive from some of the base types provided by GML. By walking through the type hierarchy, application software can discover what each element in the GML document is supposed to mean.

One can envision a software module that is capable of reading any arbitrary GML user application schema dataset by examining the type hierarchy of a schema to automatically generate an xfMap that maps GML features into the XML translation data engine geographic features.

The GML user application constrains the interpretation of the XML elements from a GML document into a well-defined geographic context. This well-defined GML geographic context can be transformed with an xfMap by the XML translation engine into the engine's neutral geographic features.

THE GML TRANSLATION ENGINE



IMPLEMENTING THE XML AND GML TRANSLATION ENGINE

The XML translation engine leverages the free parsing utilities that are available in the industry. It uses both common XML parsing APIs, namely the DOM and SAX APIs. The DOM API is used to read in the xfMap mapping-rules, while the SAX API is used to read the actual input XML data document in a streaming manner. The consequence of this is that the XML translation engine can handle arbitrarily large XML documents.

The GML translation engine allows the processing of GML datasets that are based on arbitrary user application schemas. We have implemented an API that provides access to a W3C XML Schema document because most of the free cross-platform XML parsers available do not provide such an API. This API allowed us to perform type discovery on the W3C XML Schema document to discover what each of the elements in a GML document represents. Type discovery allows the GML translation engine to automatically generate an xfMap document that is used in conjunction with the XML translation engine behind the scenes.

The XML translation engine is coupled to a general-purpose data translation hub; this hub can take the format-neutral geographic features output by the XML translation engine and translate them into a large array of GIS formats. The hub is capable of performing a large variety of sophisticated transformations on geographic features. These include topology, geometry, attribute, and coordinate system transformations. The hub also provides an API for all of the transformations that the XML translation engine exposes through the xfMap's "group mapping-rules". Briefly, the group mapping-rules allow further processing of the topology, attributes, and geometry for the geographic features constructed by the feature mapping-rules. For example, the feature-mapping-rules may be used to map the XML elements into geographic primitives, while the group-mapping rules may specify geometric operations to construct the topology of the dataset from the geographic primitives.

GML LESSONS LEARNED

We have used the above technology in a number of XML/GML projects. During this we have seen some projects go much more smoothly than others. This has given us insight into GML's strengths and weaknesses when applied to real-world problems. GML is no different than any other technology and has strengths and weaknesses. Some of these may seem obvious but are stated nevertheless.

Firstly, because GML is a means of disseminating data, *it is an interchange format not a database*. The purpose of an interchange format is to move data into a database or other system where the data is to be processed and used. GML is not a format that lends itself to doing any sort of GIS analysis.

For instance, Ordnance Survey uses GML to distribute nationwide data (OS MasterMap). GML enables them to model and distribute their data in a standards-based vendor neutral manner and allows users to get updates frequently. The Ordnance Survey is a prime example of how GML can be used effectively.

Secondly, GML files are an order of magnitude slower to process than binary files. Therefore, *parsing GML is expensive*. Often users will complain that GML files are big, heavy, and slow to read. While this is true, the purpose of GML is to provide a standards-based interchange format. As GML is based on XML, GML will have all the benefits of XML, such as a standard set of tools available for developing an application. On the other hand, GML will also have the disadvantages of XML. Specifically, GML and XML are not designed to be an efficient means of storing data.

Finally, to use GML, users must have access to external documents that describe the structure and the meaning of the data. A big difference exists between being able to read data and use data intelligently. Expecting an application to be written that is going to read and restructure data intelligently without any human intervention is unrealistic.

GML is a very rich and flexible language for expressing data. Unlike the historical highly restrictive formats that the GIS industry is used to, including SHAPE and MIF, GML enables users to represent data in an almost infinite number of ways. However, this richness is no free lunch. For an application to ingest or use data in a schema it has never seen before, it most effectively requires a human user to define the mapping to the schema of the destination system. Fortunately, work is being done now to define industry application schemas. Work is also being done on "profiles" within the OGC. A profile is a set of rules that constrain how GML can be used to ideally increase data sharing by lowering the bar of effort required to read data.

TWO VIEWS

The GML Relay in Netherlands was setup to show how GML data could be read and edited by one tool, saved and then edited by another tool. A number of vendors were invited to participate and each one was given data in the original form. The idea was the data was going to be passed from one tool to another with each tool adding or changing the data. The goal was to determine the state of the art with GML. This exercise was very valuable and may have been one of the catalysts for the push to develop profiles and applications schemas. What was found was that the current definition was too broad, resulting in virtually no data exchange. Although every vendor could read the original data and perform an operation on it, the written data was in a different schema which the next application could not process (without human intervention). The subsequent discussion with different participants in the GML world revealed that there are *two views* of GML – there are those who want restrictive profiles that greatly simplify the task of using GML to move data, whereas others view that GML should not be restricted. Left unrestricted, everyone may then adopt profiles, which would mean that the power of GML will be lost.

TWO USES

Having worked on GML, we do not see two views but rather see two uses of GML. The first use is for large scale data publishing, such as OS MasterMap. In a data publishing scenario like this where the data has great value, the data provider is free to use the full power of GML while knowing that the value of the data alone makes it worthwhile for vendors and developers to develop technology to ingest the data. Our flexible model enables us to process this type of data although it does require time to set up the xmap as described above to get the full value of this data.

The second use of GML is not for data publishing but rather for application level data sharing. For this use, it is much more effective if a profile is defined which constrains how GML can be used. Much can be learned by the formats that are most often used to move data between applications today. The most popular of these is likely the SHAPE format followed closely by MIF/MID. Why? Both of these formats come with issues that will require a more in-depth discussion. The reason that these are both successful as interchange formats is they are simple and completely inflexible. They are simply relational formats with an associated geometry for each relational row. It is simple to develop an application that can read and display any SHAPE or MIF/MID file.

If one accepts the argument that there are two different uses of GML, then it is logical that each of the uses requires a different approach. GML is a fit for both as it has much to offer. Indeed, GML is having great success with data publishing use today.

WHAT IS NEEDED

OGC has done a remarkable job of bringing together industry players towards a common goal of interoperability and has been able to avoid the lowest common denominator standard that would have served no participant. GML is rich and powerful and able to represent any type of information.

GML and associated software is now in its infancy and needs as much success as possible. These successes are dependent on both the OGC and the user community. When designing schemas, one should consider that developers in the industry need assistance for GML to succeed. Schema designers can increase the likelihood of success by developing schemas that are as simple as possible. While the flexibility of GML is

seductive, creating schemas with deep hierarchies of nested objects is not going to increase interoperability. When designing a schema, one should note that it is highly likely that both the system producing the data and the system consuming the data are relational. If this is the case, why not develop a schema that reflects this?

In examples listed earlier, the Ordnance Survey project was a success, yet the Netherlands GML relay was not. Why? The former used GML in a data publishing capacity. Application developers wanting to read it had the time and the documentation to tell them how the data is structured. They also had the motivation to read the data because it contained great value. This is a great fit. The GML Relay demonstrated that to use GML as a means to share data at the application level requires definition of rules so that applications will follow them for data sharing. Working on being able to define profiles will help tremendously. The more a profile constrains the schema definition rules, the better the chance for data sharing in an environment similar to the one created in the GML relay.

The bottom line is that simple schemas are easier to use, which thereby increases interoperability. In some cases, we have seen schemas that are too complicated for the task and data being modeled. While developing a schema that does not consider the state of database technology can be an enriching experience and result in a model that more closely resembles the real world interaction of entities, the resulting complexity may be expensive and schemas requiring custom software will not increase interoperability.

CONCLUSION

The XML translation engine replaces complex programming code with xfMap mapping-rules for the interpretation of XML elements. The XML translation engine can be used to read datasets from any arbitrary GML user application schema. This process can be automated when a type discovery module is used to analyze the schema for the automatic generation of xfMap documents. The XML translation engine enables new XML-based formats to be supported in time that can be measured in hours rather than in weeks.

Critical mass is necessary for GML to be successful over the long term. GML software is in its infancy and if schema designers recognize and take this into consideration when defining schemas then this will help its adoption greatly. GML needs many projects and organizations to adopt it successfully for it to succeed. The GML community must also be open to discuss failures and successes. Consider why some projects are successful, why other projects are unsuccessful and what are the lessons learned from both.

There is no such thing as a silver bullet, however with effective management of expectations, recognizing that there are two different uses of GML and hard work, GML can definitely be a much better bullet.

REFERENCES

GML

Simon Cox, Adrian Cuthbert, Ron Lake, Richard Martell. Eds.
Geography Markup Language (GML) 2.0. Open GIS Consortium Inc.,
<<http://www.opengis.net/gml/01-029/GML2.html>>

xfMap

Map Asia 2004

Beijing, China

Safe Software Inc., *XML Reader: xfMap, Feature Manipulation Engine (FME) Readers and Writers*.

< <ftp://ftp.spatialdirect.com/fme/2002sr1/docs/ReadersWriters2002SR1.zip> >

XML

Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. Eds. *Extensible Markup Language (XML) 1.0 (Second Edition)*. W3C

Recommendation 6 October 2000.

< <http://www.w3.org/TR/REC-xml> >

W3C XML Schema Structures

Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. Eds. *XML Schema Part 1: Structures*. W3C Recommendation 2 May 2001.

< <http://www.w3.org/TR/xmlschema-1/> >