AuScope

A4.23 Data Grid milestone 494: Interoperability of GPML, GML & GPlates.

This report describes the need for interoperability between GPML and other GML application schemas (including, in particular, GeoSciML) as well as the need for interoperability between GPlates and other GML-compatible AuScope software applications (including, in particular, those which use GeoSciML). It includes a rationale for interoperability; a description of perceived interoperability issues; and an enumeration of potential use cases. Potential solutions to the issues and use cases raised in this document will be examined in detail in the document to be created for Data Grid milestone 495.

James Clark (jclark@geosci.usyd.edu.au), 2008-01-10

Table of Contents

Introduction

What is GML?

GML stands for Geography Markup Language. It is defined by the Open Geospatial Consortium, and has been accepted as ISO standard number 19136. GML has been designed to act as the backbone of many different geographic data formats, to unify the assorted geographic information systems (GIS) by providing a common set of geographic geometries and structures that can be re-used by anyone developing a data model derived from GML.

GML promotes interoperability between GIS software by allowing developers to standardise on GML constructs for the basic elements common to all geographic software – points, lines, polygons, and the various interactions between them and supporting data structures.

GML is a *feature-oriented* language. Under GML, individual geographic elements in the data file are referred to as *features*, which can have a number of *properties* associated with them. This is in contrast to the PLATES4 Line Format, or the ESRI Shapefile format, which are *geometry-oriented* – each item in these files are defined by its geometric presence, which in turn can have attributes associated with it. The GML approach allows for the definition of features with no geometric presence, or features with multiple geometries for different purposes.

What is the GPGIM?

The GPGIM is the GPlates Geological Information Model. The careful documentation of data structures that will be used by GPlates is very important for the development process. It is the primary source of documentation for both the XML/GML-based GPlates Markup Language (GPML) and the C++ object model used by GPlates internally.

The purpose of modelling (in the software-development sense) is to develop a model of the "real world" which is able to be expressed abstractly in the form of an information model. The GPGIM is our abstract representation of our model of plate tectonics (as far as our plans for GPlates functionality are concerned); as our information model, it serves as our primary requirement for what needs to be implemented in GPlates in order for GPlates to be able to represent our model of plate tectonics. To put it another way: The GPGIM is what we have determined we need to implement in GPlates to enable GPlates to do what we collectively want it to do.

What is GPML?

GPML, the GPlates Markup Language, is an *application schema* of GML. It takes the basic geographic primitives that GML provides, and uses them to define the structure of features that GPlates can be expected to understand. The definition is written using the XML Schema language.

GPML can also be considered the GML/XML *serialisation* of the GPGIM. While the GPGIM is an information model and defines features in general terms, the GPML schema sets out the structure of a GPML document precisely. GML was chosen early on as a base for GPML for its superior flexibility, extensibility and feature-oriented model.

By developing GPML as a GML application schema, GPML offers us the ability to augment the set of GML primitives by defining our own data types, such as inter-feature references, time-dependent geometry, or spherical volume geometries. GPML is extensible enough that new features can be added without breaking compatibility with older versions.

It is also able to store nested structures, such as an aggregation of an arbitrary number of inter-feature references to define a plate boundary topologically, or a sequence of values sampled at a variety of geological times. GPML also supports the inclusion of structured meta data on any of its features via the GML meta data property. Fixed-field formats lack this quality, which forces users to resort to packing meta data into comment fields in unstructured ways, or dropping the extra information altogether.

The case for interoperability

Rather than detail the advantages of interoperability, first consider the alternative – the disadvantages of not having any form of interoperability between different software packages and the data they use.

Without applying the effort required to achieve interoperability, data created by software will quickly degrade into inscrutable, proprietary formats which are only usable by that specific software package – quite possibly by a single version of that software package. Partial format compatibility is sometimes worse than no compatibility at all, as the file may load, but silently introduce data loss and formatting errors when it is re-saved. In a grid environment where multiple applications must interact with the same data, this is totally unacceptable.

XML by itself is not sufficient

Just because a format uses plain text labels inside its structures, does not mean that the format will be comprehensible to other software applications.

As an example, two competing office document formats have emerged in the last year. From Microsoft, there is 'Office Open XML' (MS-OOXML), used in the latest versions of their office suite. There is also the Oasis Open Document Format for Office Applications (ODF), which recently became an international standard (ISO 26300).

At first glance, the two formats might seem similarly supportive of interoperability. Neither is a binary format; both use XML encoding of data. However, *merely using an XML-based format is not sufficient for interoperability.* While the schema for ODF describes the generic structures required to encode a word-processing document, the MS-OOXML format uses terms such as 'autospaceLikeWord95', 'useWord97LineBreakRules' and 'useWord2002TableStyleRules' – without defining precisely what these terms mean. Without a proper definition, anyone hoping to render a MS-OOXML document correctly must have the official Microsoft Word code base available to them, as that is the only formal description of how to implement those formatting elements correctly.

The need for explicit documentation

Thus, it is clear to see that any truly interoperable format must not only be 'open' with a plain-text or well known encoding, but also defined precisely and unambiguously. Because of the effort involved in meticulously documenting every single aspect of the data a software package uses, it is common to find proprietary formats in use. However, the workload can be reduced by applying existing open standards where possible. Many of these standards even have implementations available as libraries for a variety of programming languages, which can reduce the complexity of developing software further.

Any format which does not encode information explicitly will create many problems with interoperability, both with existing programs and future programs – without documentation, developers will have to reverse-engineer the data format and rely heavily on guesswork to determine the original intent of the data.

PLATES4 Rotation files are a perfect example of this: The PLATES4 method of encoding reconstructions uses implicit Total Reconstruction Pole interpolation. Each line represents a pole, and there is an implicit ability to interpolate between some poles, but not others. As this behaviour is not documented and is subject to a variety of special cases, it is a challenge to reverse-engineer the reconstruction logic behind the rotation format. Many other subtleties like this exist in the PLATES4 file formats.

As another example of the need for documentation which is precise and unambiguous, consider the PLATES4 Line Format. PLATES4 used simple two-letter codes to record the type of each line in the data file – 'CS' for *Coastline*, 'XR' for *Extinct Ridge*, and so on. Unfortunately, many different local standards for codes have emerged, not all of which are documented, so it is quite common to encounter an unknown code. The codes also suffer from conflicts – 'IS' has been used for both *Isochron* and *Ice Shelf*, while the currently-accepted practice is to use 'IC' for *Isochron (Cenozoic),* 'IM' for *Isochron (Mesozoic),* leaving 'IS' for *Ice Shelf.* Naturally, this causes problems for software wishing to convert from the old PLATES4 format to a newer one.

Interoperability without careful planning does not scale

When looking at the problem of making two software applications share data and communicate, there are two changes that need to be made. Application A needs to understand the data used by Application B, and vice versa. However, in any environment larger than this, such as the AuScope Data Grid, a carefully-planned process is vital. As the number of applications requiring interoperability increases, the size of the problem scales quadratically. For example, in an infrastructure that has four applications, Application A now needs to be changed to interoperate with applications B, C, and D, and likewise for the other three. In this naive approach, twelve different changes need to be made to get all the infrastructure programs to cooperate.

With the proper planning, development and implementation of standards, the workload can be reduced significantly. If each application in the infrastructure has a single interface which it can use for interaction with other applications, the problem of interoperability can scale more efficiently in an environment with many applications. Each application added to the infrastructure only needs to become interoperable with the standards set by the group, and existing applications should not need any major modifications to support the new software.

Benefits of well-designed interoperability

When interoperability is done right, it should be transparent to the user. The user will be aware that they are using different applications on the Data Grid, but will not need to concern themselves with file format conversion, data loss, or other incompatibilities between grid applications. The process should be as seamless as possible.

In an environment of interoperable software programs, users will be able to explore the possibilities offered to them by each tool, rather than locking their data to the one tool that can understand it.

Perceived interoperability issues

While the application of GML provides a leg-up for development of interoperable standards-based formats, it is not a panacea.

References between Features and Value Objects

GML prefers to use its '*xlink:href*' and '*gml:id*' method for linking between a property and a remote Feature or Value Object. There are a few disadvantages to this approach which create implementation issues and may create problems when attempting to parse GML-based files. Two main problems are dual-purpose links and a high potential for broken links:

Dual-purpose links

GML uses the '*xlink:href*' linking attribute in two situations which should be distinct; referencing another feature (in cases such as observations, bridges noting which rivers they cross, etc.) and referencing a *property* of another feature (geometry sharing common points, etc.). These are two very different applications which should be separated rather than using the same method of referencing the appropriate snippet of XML.

The GPGIM uses the Feature Reference when talking about a feature, and the Property Delegate to point to the value of a property owned by another feature.

High potential for broken links

The '*xlink:href*' attribute selects it's target XML element by the '*gml:id*' attribute attached to it. These IDs are

arbitrary, and are guaranteed to be unique only in the scope of the current document. These references can include either a local document or document to be accessed via a web service as part of the target name; if this local document is missing, or the network location is unresolvable, or if the document is available but is of an outdated revision, there is a very good chance that the link will fail.

If the link fails to resolve, the property using the link will contain invalid data. Since almost all of the GML properties make use of this dual in-line/reference nature in their definition, there are many possible ways for a feature to fail that must be accounted for at load time.

GML Dynamic Features are also supposed to use duplicate '*gml:ids*' when using the 'snapshot' temporal model; this naturally introduces a problem when it comes to reference that feature. A proposed workaround is to place different snapshots in different instance documents[3;p182], but that seems unsuitable as it would cause unnecessary fragmentation of files.

Still requires integration of foreign application schemas

Merely using GML in your application does not mean that all software using GML can make sense of it; there is no such thing as a '.gml document', only specific dialects of GML. While two application schemas may share common geometric primitives, obtaining documentation and the schema file for the application schema for that application's dialect of GML is the only way to be sure of interpreting a 'foreign' GML document properly. Having the schema and associated documentation, software projects can be designed to handle the types found in instance documents of that type - so that the foreign file format is now no longer foreign at all. GML makes this development process a little easier due to the common primitive types used, but it is important to realise use of GML does not make the process unnecessary.

As there is no official reference implementation detailing the subtleties of parsing a GML-based file format, it is infeasible to expect to be able to parse a document on guesswork alone; even if the schema file is available, few reliable inferences can be made based on inheritance and extension of core GML primitives – good documentation is necessary.

Interoperability with a foreign application schema can be grouped into three main levels:

1. Data structures will not be damaged and no information will be lost.
2. Useful information can be extracted from documents with foreign schemas.
    1. Partial information can be comprehended, such as a few basic properties of the schema.
    2. Complete information can be comprehended.
3. Documents from external programs can be edited and the original application will still be able to read the document.
    1. The edited document is written in a hybrid format, which preserves original information in the original format, but may need to store new information in a non-standard way which the original application will ignore.
    2. The edited document is written in a format that can be completely understood by the original application, including the additional data and modifications.

For the most basic level of interoperability with other application schemas, GPlates needs to be aware of the general structure of the application's documents – how the feature-property model is applied, which features might be related to others, and some form of unique identifier per feature. This would allow GPlates to load and save the data without damaging it, and allow the user to inspect some of the features' properties.

For a more useful level of interoperability, GPlates also needs to be able to identify properties of interest within the data file. GML can provide some assistance with this due to the common primitives used for geometry, however without understanding the application schema fully, GPlates will only be able to guess at the purpose of these geometries. Since GML allows a single feature to have multiple geometries for different purposes, it is necessary to know which geometries should be displayed to the user and which should not. Geometry such as bounding polygons or geometry used for annotation are clearly not meant to be used in the same way as the real geometry of the feature, so this sort of knowledge about the external application schema is necessary for interoperability.

For more advanced interoperability between applications, allowing for the editing of external features and insertion of properties, a more advanced understanding of the data structures is necessary. Internal consistency of the data must be maintained – if a property of one feature is modified, does another feature related to it also need

to be updated? In what manner are two geometries of a feature related to each other, if a user wishes to move a feature to a different location? Questions such as these must have detailed, documented answers for interaction between applications sharing data.

Interoperability considerations with other GPlates-supported formats

GPML and GML do not exist in a vacuum. It would be naive to assume that the only formats that needed to be considered when designing GPlates are GML-based ones such as GeoSciML. GPlates has specific plate tectonic reconstruction requirements, and needs to support other formats which are not GML-based. For example, the PLATES4 Line format and PLATES4 Rotation format – the latter being essential to define the plate rotation model.

Having met its own initial requirements for loading data in older formats, GPlates is free to push forwards to support newer GML application schemas. As stated earlier, GML is a feature-oriented language – features come first, and geometry is an optional property of those features. Using the GML feature-oriented approach is more flexible, but the existence of the older, more established formats will still influence the development of interoperable GML-based formats.

The primary reason for this influence is that many existing formats such as ESRI Shapefile and PLATES4 Line format are geometry-oriented. This introduces some challenges when converting between the two models. GPlates uses a feature-centric model internally, so it is able to comprehend both models easily. The problem is that many existing data files have been created to 'work around' the limitations of a geometry-oriented format, such as splitting a single feature into several different segments. Doing this allows for more than one geometry for the feature, but discards the information that they should be considered part of one logical feature.

Thus, processes outside of the definition of file formats can still have effects on interoperability. A problem with data loss in one format can remain in data sets even after the conversion to newer formats.

Interoperability use cases

Borehole Reconstruction

The reconstruction of borehole data through time using GPlates is an interesting example, given the importance of boreholes to the exploration and mining industries. Take borehole data stored in XMML or GeoSciML - To be reconstructed by GPlates, a feature needs a '*gpml:reconstructionPlateId*' property, and some existing geometry that can be recognised. Since XMML and GeoSciML deal with present-day data only, they do not have any plate IDs associated with boreholes in their schema. Similarly, GPlates may not have a Borehole feature with all the detailed properties available in GeoSciML.

As this case may not be limited to boreholes, but potentially any interesting feature from any schema, it is impractical to assume that GPlates will natively handle every possible feature from every possible markup language. A level of interoperability between GPlates and external schemas is required.

For the basic ability to read data from a GML application schema, then visualise and reconstruct the data in GPlates, it will need to be able to parse the file to determine which elements are features, which elements are properties of those features, and so on. The extra plate ID property, along with all other necessary data for plate tectonic reconstruction, will need to be added to the feature or otherwise associated with it.

GPlates gives us the flexibility to add the extra properties in memory as required. GML gives us the possibility of recognising the geometry of the borehole and being able to display it to the user (although some geometries may not be relevant to the user, such as bounding boxes). For the basic purposes of viewing and reconstructing borehole data, there should be no need to parse additional GeoSciML-specific meta data about the borehole.

The more complex case of writing this reconstructed borehole, along with the necessary information for the reconstruction, has a more significant impact on interoperability considerations.

A strategy must be devised for creating such a hybrid output file. As mentioned earlier, GeoSciML does not have plate tectonic properties, and it is impractical to believe that GPML will handle every possible feature from every language natively. Some means of associating the two sets of data is required that will not break compatibility with their native applications.

The simplest possible solution, the amalgamation of properties from both schemas into a single feature, is unfortunately denied to us by XML Schema rules. It would also no doubt confuse any parser expecting only what was defined by the schema, such as a Web Feature Service (WFS).

This leaves us with the implementation of some kind of proxy feature, which can then be included in the '*gml:metaDataProperty'* of the main feature (or vice versa), or included with the main feature collection and associated using xlinks. Doing things this way may prevent loss of the original data when re-loading the file in the original application, although there is no guarantee that the plate-tectonic data association will not be broken.

An additional step is necessary for this complex case when reading foreign application schemas. GPlates must be careful to preserve the contents of the file in memory, even if it cannot understand it. Any unknown data structures will need to be preserved as a tree of XML elements, so they can be written out with the rest of the file – otherwise, important information may be lost.
Further potential solutions and the ramifications of those solutions will be discussed in Data Grid milestone 495.

Instantaneous Export for GMT to produce paleo-graphic maps

The Generic Mapping Tools (GMT) is a collection of around 60 tools for manipulating geographic data sets and producing high quality Encapsulated PostScript illustrations. GMT is capable of over 30 map projections and transformations.

GMT is not able to do plate tectonic reconstructions by itself, however the ability to create maps from data reconstructed by GPlates is a highly desirable feature. GPlates stores all reconstructable geometry in terms of their present day positions, along with a plate ID and the geological time period for which the feature is valid.

For GPlates to support this interoperability with GMT, GPlates will need the ability to export 'Instantaneous' features, which are not directly associated with a plate ID and store their geometries in terms of their reconstructed paleo-position. The user will be able to instruct GPlates to create an instantaneous snapshot of the data for a particular time and rotation tree and export that data for GMT to use.

Once GMT and GPlates can be combined in this fashion, more complex combinations of software can be used to integrate with WFS and map-making services. This will be discussed in a later use-case.

GPlates as a web reconstruction service

As GPlates' ability to perform plate-tectonic reconstructions will be one of its most highly valued abilities on the data grid, it makes sense to provide this ability in a variety of forms. Interactive reconstruction using the GPlates interface is perfect for fine-tuning data and “what if” scenarios. For the benefit of other applications wishing to do reconstructions, a GPlates reconstruction service could be created.

This service would not require the GPlates GUI, but would instead create a web services-based front end to the core GPlates reconstruction engine. With this engine, requests for reconstructions and accompanying data can be encoded in GPML, then sent to the GPlates web service. GPlates will respond to the application with the reconstructed geometry, encoded using Instantaneous GPGIM features.

Having GPlates available as a service will enable automated reconstruction tasks across the grid, and effectively endow all grid applications using the service with the plate-tectonic reconstruction power of GPlates.

Integrating a GPlates web reconstruction service with GMT to create an online map-making service

Once GPlates can be invoked as a service, many other integration possibilities present themselves. The combination of the map-making ability of GMT with an AuScope WFS and a GPlates-based reconstruction service, would create an online paleo-map making service.

For a sample request to this web map service, a source of data along with reconstruction and transformation parameters must be supplied. A WFS would be used to obtain the data requested. Features would be returned from a call to the WFS using GPML. Note that GeoSciML might also be a possible encoding, but without Plate ID and associated reconstruction data it makes little sense to use it for an automated reconstruction procedure. The data would be passed to GPlates' reconstruction service, along with suitable parameters such as the target reconstruction time.

The output of the reconstruction service would use GPML instantaneous features as appropriate input to GMT, as detailed in an earlier use-case. GMT would then be used to render the data, again drawing on parameters supplied to the paleo-map service in the initial request.

Connecting geological models and observations via plate kinematic models

CitcomS and UnderWorld are finite element mantle convection simulation programs. One potential application for interoperability between these programs and GPlates is to use global plate boundary data along with the plate rotation model to create surface constraints for UnderWorld and CitcomS to use in their models.

GPlates would need to be able to calculate plate velocities and produce a point-velocity grid feature which could then be imported and used in a plate-driven convection model. Provided the grid format is one both programs are able to understand, the import should be hassle-free. Any additional meta data can be handled by an accompanying GPML document describing the grid.

Conclusions

GML enables GPlates and GPML to get a head-start on interoperability design. However, mere application of GML by itself is not sufficient. Documentation and good design are essential to the development of interoperable software. There are different levels of interoperability that can be achieved, and each level introduces more requirements and requires additional design. Specific use-cases may only need limited interoperability between applications to work successfully, or require extensive work in multiple applications.

References

[1] http://www.opengeospatial.org/standards/gml, last accessed 2007-12-21.
[2] http://www.earthbyte.org/Resources/GPGIM, last accessed 2007-12-21.
[3] *Geography Mark-Up Language: Foundation for the Geo-Web* by Ron Lake et al. 2004 John Wiley & Sons Ltd.
[4] UnderWorld home page, http://www.mcc.monash.edu.au/Software/Underworld, last accessed 2008-01-04.
[5] *CitcomS User Manual*, http://geodynamics.org/cig/software/packages/mc/citcoms/citcoms.pdf, last accessed 2007-12-21