Embedding GML Best Practice Paper

# Overview

This note provides a best practice guideline for the embedding of GML inside an XML language that is not specified by a GML application schema, in fact may not be specified by an XML Schema at all. The intent of this guideline is to encourage this practice while at the same time achieving as much standardization as possible, and maintaining consistency with GML itself.

The basic structure of GML is based on the so called “object-property-value” rule. This means that objects (things, entities) are encoded at elements, and their element children are properties of the parent object. The element children of these property elements are the values of the property for complex valued properties, or are just the element content in the case of properties of simple type.

When embedding GML in another language, one should attempt to stick to this basic GML model by using the following best practice guidelines.

1. **Only embed GML objects.** These are things in the GML core schemas, whose content models derive directly or indirectly from AbstractGMLType. Such objects include geometry elements, topological elements, observations, coordinate reference systems and temporal elements.
2. **Include any GML dependent attributes, elements and content models.** The embedded GML object may depend on attributes, elements etc that are defined elsewhere in the GML core schemas. Be sure to include these in your embedding.
3. **Embed GML objects in “properties”.** Where possible, identify container elements in the parent non-GML language that behave like attributes or properties. Examples might include Slots in ebRIM, or the “where” property in GeoRSS.
4. **Create a GML profile schema.** Create a GML profile schema by taking the desired embedded elements and their dependencies and constructing a single schema file, containing them. You need then only import this single profile schema into your non-GML language.

The following sections provide additional details using these simple rules and illustrates them with some simple examples.

# Only Embed GML Objects

Essentially this means do not embed GML properties, GML attributes/attribute groups, nor use GML types to define the content models of elements in the non-GML language. While all of these are possible, GML objects are complete GML components and thus are more readily employed in the non-GML language. Additionally there is less opportunity for confusion with GML objects having the same content model. This practice also recommends against the embedding of GML properties (e.g. pointProperty) as the semantics of such properties is clear in the GML context, but would be much less so in the non-GML language in which GML is being embedded.

Note that this does not mean you cannot define new application objects which use GML content models. For example, in an emergency response language, it is desired to define an ImpactArea using a GML Polygon. It is recommended that this be encoded as follows:

 <abc:ImpactArea>

 <abc:extent>

 <gml:Polygon gml:id = “…”>

 <gml:exterior>

 <gml:LinearRing> … </gml:LinearRing>

 </gml:exterior>

 </gml:Polygon>

 </abc:extent>

 <abc:ImpactArea>

Rather than writing:

 <abc:ImpactArea>

 <gml:exterior>

 <gml:LinearRing> … </gml:LinearRing>

 </gml:exterior>

 <abc:ImpactArea>

The semantics of the former encoding is clearer in that the ImpactArea while having Polygon extent is NOT a Polygon as implied in the second encoding.

If the parent language wishes to define objects already defined in GML (e.g. Point) than this best practice would require that they use the GML defined objects rather than creating new ones with the same content model.

For example, the following encoding (in an XML schema with a non-GML namespace) would not comply with this best practice:

 <element name = “Point” type = “gml:PointType”/>

There are cases where one might like to add properties (e.g. special ID attributes or elements) to a GML defined object. Of course this can be done by inheriting from the GML content models, and adding the desired attributes, or elements. This should be avoided if possible, as it complicates the processing software for the non-GML language (i.e. requires schema parsing techniques).

# Include any dependent attributes, type definitions etc.

Of course this is required to construct a valid schema. It is recommended that the user create a GML profile schema file containing the needed objects, and all dependent attributes, elements, and type definitions. Automated tools for this purpose are available for most versions of GML.

The profile schema file then isolates all GML content used by the non-GML language, and only this file need be imported by the XML Schema for the non-GML language.

# Embed GML Objects in “Properties”

Properties in GML frequently represent associations between the parent and child elements of the property, the property in effect defining the context of the child object in relation to the parent. Consider the following simple example:

 <abc:Road id = “j12k”>

 <abc:centerline>

 <gml:LineString> … </gml:LineString>

 </abc:centerline>

 <abc:rightEdge>

 <gml:LineString> … </gml:LineString>

 </abc:rightEdge>

 </abc:Road>

The properties centerline and rightEdge define the relationship of the geometric objects (both LineStrings) to the Road object.

In GML Application Schemas this structure of objects-properties-values is rigidly enforced. In a non-GML language there may be no such requirement, however, it is considered best practice to look for (or if possible) create properties or elements with a similar role as the container for GML objects.

The following encoding would then not comply with this best practice:

 <abc:Road>

 <gml:LineString> … </gml:LineString>

 </abc:Road>

Most non-GML languages will have some sort of property construct and it will usually be possible to comply with this best practice.

Following this best practice will ensure that GML usage is clearer in the non-GML language.