

# Evaluating Modeling Options

## Table of Contents

- [Introduction](#)
- [Modeling](#)
- [How to Evaluate](#)
  - [#id](#)
  - [#access](#)
  - [#collection](#)
  - [#super](#)
  - [#schema](#)
  - [#access](#)
  - [#content](#)
- [Comparison](#)
- Reference
  - [#review](#)
  - [#formats](#)

## Introduction

Geographic Information Systems have an interesting problem domain, much like with programming GIS Hacks are involved in modeling the real world.

- Programmers model with Software (and use abstractions like Object and Class).
- Geographers model with Maps (and use abstractions like Feature and FeatureType).

There is more than one way to do it, programmers used to use procedures, geographers used to use a pen and paper. Lets get on with it.

## Modeling

Ack - try and stay awake. Nothing is as silly as trying to explain why creating a model is useful. Why create a model when you could solve a real problem?

Well for starters models are simpler and more flexible.

- A Geographer could model what it would be like to flood a river, this is much more pragmatic than driving around with sticks of dynamite and blowing up a few dikes (although perhaps not as much fun). Hacking models is easier than hacking reality.
- A programmer can model code (using UML diagrams and such), hacking the model is often a lot quicker than hacking the code.

One can also view "proper" Object-Oriented programming as constructing a model of a problem space in the real world - and then using it to solve actual problems. There is even a branch of practice called Model Driven Development & Domain Driven Design to try and steer us back in the direction of OO principles.

As for a Geographer - they can create a Map (a visualization) of their model, this is a bit cheaper than jumping up in a hot air balloon and taking pictures. And we can make maps for things that don't exist yet (such as someone running around and wrecking the dikes that keep a river in check).

## How to evaluate modeling Systems?

Different modeling systems allow for different ranges of expression. It is quite hard to capture the relationships on "Days of Our Lives" using the C programming language. The Java language would do a bit better as it supports a Class system (but may still have a tough time with the inheritance relationships in Deliverance). The closer our means of expression is to the problem domain the less work we have to do.

For Geographers the Story is exactly the same, depending on the modeling system they use they will be able to accomplish different things. It is fairly hard to model projections using a CAD program, a modeling system that supports projection is often more suited to GIS work.

The point? We can only evaluate a modeling system with respect to a problem domain, or a task.

## So what is it we want to Do?

Stop talking about models and get some work done ... ie. Blunt Example: based on a schema construct a query to access data.

What do we want (aka what is our task) - Geographic Markup Language:

- **GML3** - this is the "mandate" mentioned at an IRC meeting a couple of months ago.
  - Programmers would use UML2 these days (right?)

Here are some of the constructs in GML that we need to ensure we handle well:

- **Feature** - something that can be drawn on a map
  - Programmers would call these **objects** (or **instances** if that makes you happy)
- **FeatureType** - kind of thing on a map.
  - Programmers would call these **classes**
- **FeatureCollection**
  - a bunch of features, can also be considered a "Feature" (Cause you can draw it on a map)
  - Programmers would consider this a **data structure**
- **id** - formally FID or "FeatureID"
  - used to uniquely identify the Feature, the real world object

These are informal working definitions, for a formal definition consult a specification.

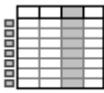
## Formal separation of Concerns

We would like to maintain the following separation of concerns:

Metadata	XMLSchema	Table	FeatureType
Data	XML	Row	Feature
Query	XPath	SQL	Expression

## Handling of ID

Different FeatureModels handle the idea of **id** differently:  
(sorry for the use of database language here - it is what GeoAPI and DeeGree do)

 <p>Id</p>	<p>ID</p> <p>Based on "magic" (like shapefile row)</p> <p>This is the ideal, everything else is an approximation. <b>D JB: shapefile row is a very bad example - this is the worst possible example of an ID since it changes whenever you delete something. The PostGIS DataStore using the system's tuple id (OID) is a much better example.</b></p>
 <p>Key</p>	<p>Key</p> <p>A single KEY column is treated as an ID</p> <p>column is no longer available as an attribute to be queried</p>
 <p>Keys</p>	<p>Keys</p> <p>Several KEY columns are used to derive an ID</p> <p>Columns are not available as an attribute to be queried</p>
 <p>Key Attribute</p>	<p>Key Attribute</p> <p>A single KEY column is marked as the Feature ID</p> <p>Column remains available as an attribute</p>
 <p>Key Attributes</p>	<p>Key Attributes</p> <p>Several KEY columns are used to derive an ID</p> <p>Columns remain available as attributes</p>

❓ Evaluation: Is the idea of a unique unmodifiable ID maintained?

❓ Key Attribute - does this violate encapsulation?

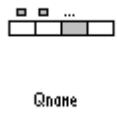
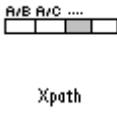
⚠️ if on the off chance the attribute was modifiable people would be able to change their **ID** (and the world would end).

ℹ️ The concept of KEY (and Unique) seems to be available in GML3, it does not have any relationship to FeatureID.

💡 We can consider support of KEY and UNIQUE validation constraints, this does not need to imply any relationship to the generated GML2 FID or GML3 ID

## Attribute Access

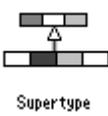
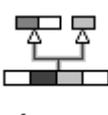
 <p>Index</p>	<p>Access attribute by an integer index</p>
 <p>None</p>	<p>use attribute name to access content</p>

 <p>Qname</p>	use qualified name, or attribute object
 <p>XPath</p>	use xpath to access deeply nested content

⚠️ XPath seems to be too complex for implementations to support, aim for distinct separation between data model and query model

⚠️ index and name both run into limitation when used with super types

## Type System

 <p>Supertype</p>	Type information through extension of a single super type
 <p>Supertypes</p>	Type information from multiple super types

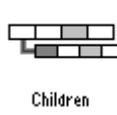
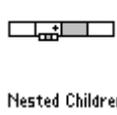
⚠️ A Type system does not have to imply reuse,

ℹ️ Java interfaces are pure type system, the class system is for code reuse.

ℹ️ XML has the same separation by way of substitution groups

## Handling of FeatureCollection

We need to ensure that we can determine the schema of children allowed in a feature collection:

 <p>Children</p>	collection schema refernces child schema
 <p>Mixedchildren</p>	collection schema refernces schema of children
 <p>Nested Children</p>	children represented as a nested feature attribtue with multiplicity

In addition there are the following tradeoffs:

 <p>Childref</p>	<p>Parent contains references (or filter) defining contained children</p>
 <p>Parentref</p>	<p>Children contain a back reference to collection, prevents children from being in more than one collection</p>

## Handling of Schema

	FeatureType
 <p>Flat</p>	<p>Attributes providing name and type information</p>
 <p>Multiplicity</p>	<p>Concept of multiplicity, and nilable</p>
 <p>Restriction</p>	<p>Allow for restrictions on attributes (such as length)</p>
<b>"Complex"</b>	
 <p>Choice</p>	<p>Allow for a choices</p>
 <p>Complex</p>	<p>Allow for complex attributes</p>
 <p>Nested</p>	<p>complex attribute is a feature</p>
FeatureCollection Type	

 <p>Children</p>	access to child feature type
 <p>Mixedchildren</p>	FeatureCollections with mixed content
 <p>Nested Children</p>	FeatureCollection child modeled as an complex attribute

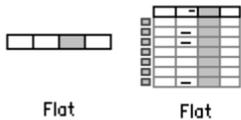
Needs to be able to describe the contents of the previous section:

❓ Evaluation: Is the schema "rich" enough to generate XPath expressions into the content?

## Handling of Content

Here is a description of a "reference" GML document, we can use it as a reference point to see how capable the different modeling systems are.

### Flat



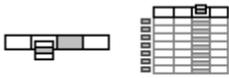
Simple feature content as used by JUMP & Shapefile

```

header
document
  collection1
    road.1= label=hwy1, line=..., group=1,
title=Hiway 1
    road.2= label=hwy1, line=..., group=7,
title=Hiway 2
    ...
    road.N=...

```

### Complex Content



Complex

Complex

Support for complex types, usually represented as an object. Support is required at the schema level to enable the construction of XPath queries into the complex content.

```

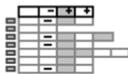
header
document
  collection
    road.1= label=hwy1, line=...,
address=(name=fred,zip=1234}
    road.2= label=hwy1, line=...,
address=(name=fred,zip=1234}
    ...
    road.N=...
  
```

## Multiplicity

Features contain repeating (or optional) elements:



Multiplicity



Multiplicity

**Attribute Multiplicity:** min/max used to constrain the number of times attributes are able to occur



Multiplicity



List

**List multiplicity:** Values are represented as a list, min/max are used to bound the list

```

header
document
  collection
    road.1= label=hwy1, line=..., group=1,
title=Hiway 1
    road.2= label=hwy1, line=..., group=7,
title=Hiway 2, type=7, title=Lovers Lane
    ...
    road.N=...

```

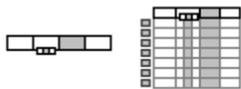


Required for:      Flat      Multiplicity

**⚠** Note the **Lists** representation agree directly with the XPath query model, but does not allow for validation of complex features. It is presented due to its use by several of the toolkits (not because it represents a valid model for handling GML3 content).

### Nested Features

A form of complex content in which Features are allowed to contain other Features.



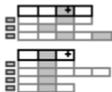
Nested      Nested      Support nested features as normal attributes, depending on implementation no additional may be required over and above that required for complex content.

```

header
document
  collection
    road.1= label=hwy1, line=...,
    road.2= label=hwy1, line=...,
crosses=(label=pond,
polygon=polygon(coordiantes))
  ...
  road.N=...

```

## Multiple Collections



Collections

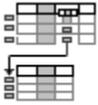
Unsure what metadata support is required to represent this kind of document?

```

header
document
  collection1
    road.1= label=hwy1, line=..., group=1,
title=Hiway 1
    road.2= label=hwy1, line=..., group=7,
title=Hiway 2
  ...
  road.N=...
  collection2
    lake.1= label=pond,
polygon=polygon(coordiantes)
  ...
  lake.N=...

```

## Feature References



Reference

Same problem as before, what captures the schema information for a GML Document?

```
header
```

```
document
```

```
  collection1
```

```
    road.1= label=hwy1, line=..., group=1,  
title=Hiway 1, crosses=#lake.1
```

```
    road.2= label=hwy1, line=..., group=7,  
title=Hiway 2
```

```
    ...
```

```
    road.N=...
```

```
  collection2
```

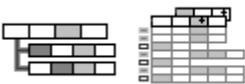
```
    lake.1= label=pond,  
polygon=polygon(coordiantes)
```

```
    ...
```

```
    lake.N=...
```

Note: Refernece should be handled as a lazy Feature (require another pass through the data when resolved?), either that or are we stuck with loading the document into memory.

## Mixed Content



Mixedchildren

Mixed

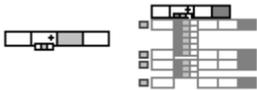
Allow children of several types within a collection. GML may limit this to chilfren within one substiution group.

```

header
document
  collection
    road.1= label=hwy1, line=..., group=1,
title=Hiway 1
    lake.1= label=pond,
polygon=polygon(coordiantes)
    road.2= label=hwy1, line=..., group=7,
title=Hiway 2
    ...

```

## Nested Collections



Nested Children NestedCollection Allow collection contents to be collections themselves.

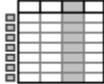
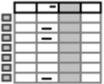
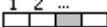
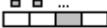
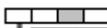
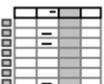
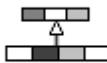
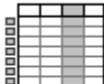
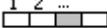
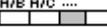
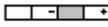
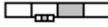
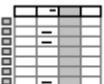
```

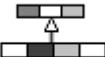
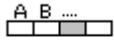
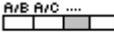
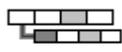
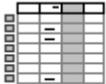
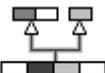
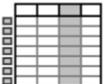
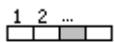
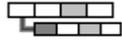
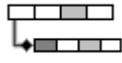
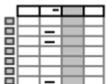
header
document
  collection1
    road.1= label=hwy1, line=..., group=1,
title=Hiway 1
    collection2
      lake.1= label=pond,
polygon=polygon(coordiantes)
      ...
      lake.N=...
    road.2= label=hwy1, line=..., group=7,
title=Hiway 2
    ...

```

# Comparison

Note chart is based on documented intent, not ability.

Name	SuperType	ID	Feature Access	FeatureType	FeatureCollectionType	Feature
JUMP		 <p>Id</p>	<p>A B ...</p>  <p>None</p> <p>1 2 ...</p>  <p>Index</p>	 <p>Flat</p>		 <p>Flat</p>
GeoAPI 2.0		 <p>KeyAttributes</p>	<p>A B ...</p>  <p>None</p> <p>1 2 ...</p>  <p>Index</p> <p>A B ...</p>  <p>QName</p>	 <p>Flat</p>  <p>Multiplicity</p>  <p>Complex</p>  <p>Nested</p>	 <p>Children</p>  <p>Parentref</p>	 <p>Flat</p>  <p>List</p>
GeoTools 2.0	 <p>Supertype</p>	 <p>Id</p>  <p>Keys</p>	<p>A B ...</p>  <p>None</p> <p>1 2 ...</p>  <p>Index</p> <p>A/B A/C ...</p>  <p>Xpath</p>	 <p>Flat</p>  <p>Multiplicity</p>  <p>Complex</p>  <p>Nested</p>		 <p>Flat</p>  <p>List</p>

<b>GeoTools 2.1</b>	 Supertype	 Id   Keys	 None   Index   XPath	 Flat   Multiplicity   Complex   Nested   Restriction	 Children   Parentref	 Flat   List
<b>Deegree</b>	 Supertypes	 Id	 None   Index	 Flat   Multiplicity   Complex   Nested	 Children   Parentref	 Flat   List

## Requirements

We require the feature model to be "big enough" in modeling power to support:

- Level 0 Profile of GML3 for WFS
- XPath and SLD Documents

### Level 0 Profile of GML3

The exact requirements are outlined here:

- [FeatureType Survey#FeatureTypeSurvey-Enhancementrequirements](#)

In brief:

- complex type support
- subset of geometry types

There is some discussion of references and dealing multiple feature collections and feature references.

## XPath and SLD Documents

Our FeatureType system must be complete enough to allow the generation of XPath expressions as used by Expressions in SLD documents.

The XPath view of a document is limited to the following:

- nodelists

To support this model we **do not** need the full set of validation constructs required for GML3 (such as choice, restrictions or even multiplicity).

### Example

Lets practice with the following document:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="application/xml"
href="sco.xsl">
<gml:FeatureCollection
  xmlns:wfs="http://www.opengis.net/wfs"
  xmlns:sco="http://online.socialchange.net.au
  "
  xmlns:gml="http://www.opengis.net/gml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance"

  xsi:schemaLocation="http://online.socialchan
  ge.net.au
  03_multiple_geometric_and_non_geometric_attr
  ibutes.xsd

  http://www.opengis.net/gml
```

```
../schemas.opengis.net/gml/3.1.1/base/gml.xsd">
  <!--this case has multiple geometry
properties and repeated non-geometry
properties (one scalar, once complex)-->
  <gml:boundedBy>
    <gml:Envelope
srsName="http://www.opengis.net/gml/srs/epsg
.xml#4283">
      <gml:coordinates decimal="." cs=","
ts=" " ">8.89999962,143.53399658
200,143.53399658</gml:coordinates>
    </gml:Envelope>
  </gml:boundedBy>
  <gml:featureMember>
    <sco:wq_plus gml:id="_41010901">
      <sco:sitename>BALRANALD
WEIR</sco:sitename>
      <sco:measurement
gml:id="_16JAN94002001002003000000">
        <sco:determinand_description>16/JAN/94</sco:
determinand_description>
          <sco:result>Turbidity</sco:result>
        </sco:measurement>
      <sco:measurement
gml:id="_24JAN94002001002003000000">
        <sco:determinand_description>24/JAN/94</sco:
determinand_description>
          <sco:result>Turbidity</sco:result>
```

```
</sco:measurement>
  <sco:location>
    <gml:Point
srsName="http://www.opengis.net/gml/srs/epsg
.xml#4283">
      <gml:coordinates decimal="."
cs="," ts="
">22,143.53399658</gml:coordinates>
    </gml:Point>
  </sco:location>
  <sco:nearestSlimePit>
    <gml:Point
srsName="http://www.opengis.net/gml/srs/epsg
.xml#4283">
      <gml:coordinates decimal="."
cs="," ts="
">22.1,143.53399658</gml:coordinates>
    </gml:Point>
  </sco:nearestSlimePit>
  <sco:sitename>RWWQ0004</sco:sitename>
</sco:wq_plus>
```

```

</gml:featureMember>
  <gml:name>My Boreholes</gml:name>
</gml:FeatureCollection>

```

Node breakdown of example:

Root:

- xml-stylesheet type="application/xml" href="sco.xsl"
  - gml:FeatureCollection
  - gml:boundedBy: Envelope
  - gml:featureMember
    - sco:wq\_plus
      - sco:sitename: BALRANALD WEIR
      - sco:measurement
        - sco:determineand\_description: 16/JAN/94
        - sco:result: Turbidity
      - sco:measurement
        - sco:determineand\_description: 24/JAN/94
        - sco:result: Turbidity
      - sco:location: Point( 22, 134.5399658, EPSG:4283 )
      - sco:nearestSlimePit: Point( 22.1,143.53399658, EPSG:4283 )
      - sco:sitename: RWWQ004
    - gml:name: My Boreholes

Points of interest:

- Root node is not the same as the root element
- Root node contains the entire document including the xml-stylesheet (xmlns and xmlns:prefix are not covered though)

Location Paths:

Similar to unix shell:

<b>Root node</b>	/	Selects the root node
<b>Child element</b>	gml:Point	Point( 22, 134.5399658, EPSG:4283 ), Point( 22.1,143.53399658, EPSG:4283 )
		Note this is relative

## Reference

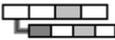
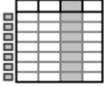
### Data Format Reference

This review will focus on the different modeling systems ability to capture the Shapefile and/or GML document mentioned above.

## Shapefile

The other thing we should look at is the humble shapefile, this was used to drive the expressiveness of the existing GeoTools modeling abilities. We really dont want to backtrack (although shapefile is so simple it will be hard to).

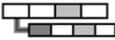
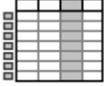
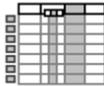
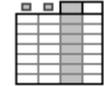
<b>Model</b>	shapefiles, one per feature type
FeatureType	shapefile, details in header
Feature	entry in shp file, using shx to find a row in dxf file
FeatureCollection	n/a

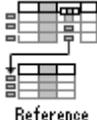
Schema	Model	Collection	ID
 Restriction	 Flat	 Children	 Id
restriction	flat	child	row number

## Databases

Database land is all wrapped up in the Simple Features for SQL specification. The main benefit is a definition of the basic Geometry types we use in GeoTools (implemented by the JTS library).

<b>Model</b>	Relational
FeatureType	table (or view) details in metadata
Feature	Row
FeatureCollection	Result Set

Schema	Model	Collection	ID
 Restriction	 Flat	 Children	 Id
 Complex	 Nested		 Keys

	 Reference		 Key Attributes
			fidmapper function

Note: Not all databases support SFSQL (and Oracle supports a few interesting things likes curves)

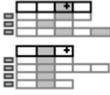
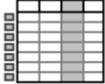
**DJB: you should also mention object-relational DB techniques (like hybernate) or just simple multiplicity-by-table-joining.**

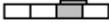
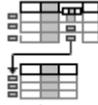
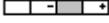
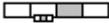
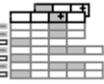
## Unified Modeling Language

Model	Object-Oriented
FeatureType	Class
Feature	Object
FeatureCollection	Aggregation

## GML2

Model	XML
FeatureType	XMLSchema
FeatureCollection	compext type that extends gml:FeatureCollection
Feature	element extending gml:Feature
	child of gml:FeatureCollection's gml:featureMembers element

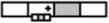
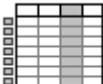
Schema	Model	Collection	ID
 Restriction	 Collections	 Nested Children	 Id
 Complex	 Nested		

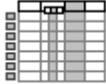
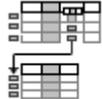
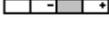
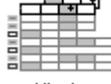
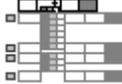
 Choice	 Reference		
 Multiplicity	 Multiplicity		
 Nested	 Mixed		
	 NestedCollection		

The GML2 geometry model is similar to JTS (although based on some ISO specification).

### GML3

<b>Model</b>	XML
FeatureType	Direct/indirect extension of gml:AbstractFeatureType
	Attribute is a local xsd:element
FeatureCollection	complex type that extends gml:FeatureCollection
Feature	element realization of GF_FeatureType
	child of gml:_FeatureCollection's featureMember element
	member of the gml:_Feature substitution group

Schema	Model	Collection	ID
 Restriction	 Collections	 Nested Children	 Id

 Complex	 Nested		
 Choice	 Reference		
 Multiplicity	 Multiplicity		
 Nested	 Mixed		
	 NestedCollection		

Reference: [GML-3.1.pdf](#)

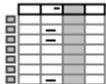
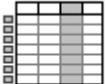
- Page 67 for Feature
- Page 68 & 528 for FeatureCollection
- Page 530 for "Annex E UML-to-GML Application Schema Encoding Rules"

GML3 includes a more complicated Geometry model than supported by JTS. Higher order geometries (like surface) are supported in addition to curves and topologies.

What is interesting is that "Annex E" describes a mechanical way to port a UML model to GML (the bridge between programmer and geographer has been crossed before us).

## Reviews

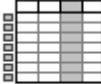
### GeoTools 2.0

Schema	Model	Collection	ID
 Flat	 Flat		 Id

 Multiplicity	 List		
 Complex			
 Nested			

- As you can see the GeoTools 2.0 schema model far outstrips its Modeling abilities.
- Multiplicity is supported as attribute arrays
- Collections are **not** considered Feature in their own right.  
 Schema indicates multiplicity support, resulting values indistinguishable from support of complex content such as a list.
- supertype support and name/index attribute access inconsistent, in practical terms super type is unsupported
- Feature indicates xpath support (but not implemented)
- FeatureType indicates xpath support (but not implemented)

## GeoTools 2.1

Schema	Model	Collection	ID
 Flat	 Flat	 Children	 Id
 Multiplicity	 List		
 Complex			
 Nested			

 Restriction			
--	--	--	--

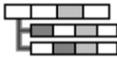
- GeoTools 2.1 added support for restrictions
  - by use of facets (defined as a series of Filter associated with an AttributeType)
- Collections are now considered a Feature
  - Cannot determine allowable child features from collection feature type schema.

## GeoAPI 1.0

Schema	Model	Collection	ID
--------	-------	------------	----

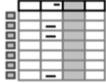
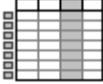
Not yet reviewed

## GeoAPI 2.0

Schema	Model	Collection	ID
 Flat	 Flat	 Mixedchildren	 Keys
 Complex			
 Multiplicity			

- FeatureType is moustable? Until frozen/used?
- Support for Object Attributes
- FeatureCollection is considered a Feature
- FeatureCollectionType incomplete
  - Cannot determine allowable child features

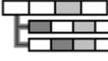
## DeeGree

Schema	Model	Collection	ID
 Flat	 Flat	 Mixedchildren	 Id

 Complex			
 Multiplicity			

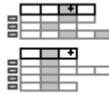
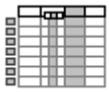
GPL toolkit actually able to take on complex GML, used as an input to the GeoAPI model. The model supports complex types but the implementations do not seem to follow, at least not on the head of their repository.

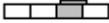
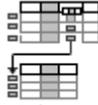
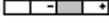
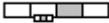
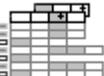
## JUMP

Schema	Model	Collection	ID
 Flat	 Flat	 Mixedchildren	 Key
 Complex			

- Simple FeatureType model, FeatureType is non mutable.
- Support for Object Attributes
- Collections are allowed children of a single type.
- Collections are **not** considered Feature in their own right.
- Collections are held in memory

## GML2 & GML3

Schema	Model	Collection	ID
 Restriction	 Collections	 Nested Children	 Id
 Complex	 Nested		

 <p>Choice</p>	 <p>Reference</p>		
 <p>Multiplicity</p>	 <p>Multiplicity</p>		
 <p>Nested</p>	 <p>Mixed</p>		
	 <p>NestedCollection</p>		

GML3 makes the transition to ISO based Geometries.