

# Feature Model Discussion

This discussion has resulted in a [Feature Model Proposal](#) resulting in the following work:

- [Feature Model](#) (API including nice diagrams)
- [Feature Model Branch](#) implementation status of the fm branch

This discussion is the subject of development in [2.3.x](#).

## Discussion

This page documents the design discussion used to resolve several of the feature model proposals.

For the following Q&A session please consider JG, GR & CH entirely fictional creations of the authors imagination:

For real chats:

[Feature Model IRC Chat 1](#)

[Feature Model IRC Chat 2](#)

[Feature Model IRC Chat 3](#)

[Feature Model IRC Chat 4](#) - Geometry / Association / Super

Remaining Issues: (see [Feature Model Design Discussion](#) for explanation of options considered)

- Facets, as Filter or Expression or something else?
- Associations, do we need an explicit representation?

## Geometry

Q: We have a question - is Geometry Complex?

A: No.

Geometry started out life as complex, allowing us to xpath into its CRS and Bounds, note it would still be bound to a Geometry class.

We have decided to go with Simple for now, as a consequence we have moved getID to Attribute (as Geometry have IDs)

## Association (and Referencing)

Q: is association directly relizable as Complex? Or does it need to be treated special?

A: Yes, association is not special

We know that when we do support referencing we will need a system to lookup IDs, the benefit being that we can have a Proxy that just stores the ID.

On the surface it appears that an AssociationType would be nice:

```

interface Association extends Complex {
    ID getReferencedID(); // used to report
ID of "contained" content.
    Type getReferenceType(); // report Type
of contain content
    Choice getSchema(); // where schema is a
choice between Foo and Referece<Foo>
}
interface Reference<Type> {
    ID getID(); // returns ID of content to be
aquired
    Type getReferencedType(); // Type of
content referred to
}

```

Or conversly just treat this as a matter of Schema:

```

interface Association<Type> extends
Schema.Choice {
    List<Node> options(); // contents are
Type and ReferenceType
}

```

But this would require us to declare how ID may be resolved, if we did not do this then out data constructs would not be interoperable between systems. It does us no good to represent content and not be able to have it be used:

The obvious way to provide for ID look up witht he XMLSchema concepts of ApplicationSchema and Application:

```

interface Application extends Complex {
    ApplicationSchema getSchema();
    Object ref( String ID );
}
interface ApplicationSchema extends Node {
    Type getType(); // may be null if schema
does not define a "document"
    List<ApplicationSchema> dependencies();
// navigate dependencies to determine all
allowable types
    List<Type> getDefinitions(); // all
Types available in schema, will need to
search dependencies for those
    List<Type> identified(); // view
indicating which Types that may be used with
ref( id )
}

```

You can see this just puts us in a situation where we need to find out the dependencies ... lets assume client code provides us with a SchemaFactory configured in some manner.

```

class SchemaFactory {
    ApplicationSchema create( GenericName
name );
}

```

So long term Association looks like a Complex choice between a Thing and Reference to a Thing.

We are thinking of the short term solution of representing an Association as a Complex choice of a single Thing. This is fully representable in the proposed modeling system and would not require further changes.

```

class Association<Type> extends Complex {
    ...
    Choice getSchema(){
        return SchemaFactory.choice(
Collections.singleton( SchemaFactory.node(
Type ) ) );
    }
    ...
}

```

This represents something that can be accomplished in the short term, and is forward compatible with longer term solutions such as ApplicationSchema above. Note this does speak to the need to explicitly treat Type and IdentifiableType separately, right now we are getting by with Type.isIdentified().

## ID

Q: Does complex need to support ID?

A: Yes

Gabriel has discovered some recommended practices for GML, and knowledge of XML parses indicate, that ID is a concept valuable for more than just Features. Specifically in GML Geometry is known to have a gml:id, and it is used to allow two features to share a single Geometry.

Q: Should it actually move to Attribute?

A: Yes ⚠️ - Moved to Attribute in order to allow GeometryAttribute to be represented as non Complex.

## Super

Q: Multiple?

A: No.

Q: FeatureType ComplexType or Type?

A: Type - used to represent Atomic Types

## Atomic Types

Gabriel would like to capture the XMLSchema atomic types, so that allowable restrictions such as length can be reused. What is the best way to accomplish this?

Q: Explicitly model XMLSchema atomic types as an Enum of Strategy objects and allow SimpleSchema to delegate to them?

A: This can be considered as Gabriel starts to represent GML.

Q: Move FeatureType.getSuper() to Type.getSuper()

A:  We have now done this

## Understanding Schema - Override vs Extention / Restriction

Q: Do we have to support the XMLSchema concept of Extention and Restriction?

A: No, we can provide a way to use plain old "Override" with a set of guidelines

In general we have a problem with our split of Type and Schema and restrictions, are restrictions part of the Schema or part of the Type? The only thing they can do is cause validation errors - so they should be Schema. However our type system (not our Schema system) is setup for inheritance and customization.

Judging from XMLSchema the Type should define allowable/understandable restrictions and the Schema should define them. Example: xs:string understands maxLength, and my:name extends xs:string with a facet of maxLength=10.

Try it: We have an implementation of extension and override in svn.

We planning to understand the relationship between getSchema() and getSuper().getSchema() in terms of override.

So here is a definition of override:

1. Will combine schema of child with schema of parent
2. Super Type schema will be used as a starting point
3. Sub Type schema will be mixed in - at level 0 of the schema (that may be a choice, sequence or set)
  - a. For every Schema in the child sequence it will have a chance to "override" an entry in the parent.

How to model XML extension using "override":

1. Use a single Schema.Sequence it will be tacked on the end.

How to model XML restriction:

1. Override each and every Schema from the parent.
2. if you need to remove, override with multiplicity 0:0

Gabriel needs to ensure that this concept of Override does completely cover his needs for restriction and extension support.

## Restrictions

Q: Facets or Filters

A: Filter - one silly expression language is enough

Facets are an interesting problem, GeoTools has taken to defining facets using the Filter specification. A great move at the object level (Filter is defined as a boolean function: boolean accepts( Feature ) ). The problems are operating on Feature and not on Objects, and being a test of set membership that we can only loosely translate into testing against the set of "valid" and "invalid" content.

It would be desirable to:

- allow the testing of individual values prior to their being assembled into a completed Feature
- generalize Filter to accept other kinds of XPathable content

Other toolkits have taken to including the bare minimum needed to get shapefile working (maxLength and so on). It would be great if we can figure out a way to make use an Expression syntax to solve a general need:

Here is the bare minimum:

```
interface Filter {
    boolean evaluate( Object content );
}
interface Expression {
    Object evaluate( Object content );
}
```

Gabriel has an excellent breakdown of the GeoTools mappings here: [FeatureType Survey#Restrictionsupport](#)

### Schema - or how the heck does Choice work?

Q: How do we represent Schema?

A: Separation of concerns between Data, Type and Ordering

So far it appears clear that **three** peices of information are needed (Content, Reference, and Type) to capture validation (or construction) at the level of GML. It is also fairly clear that this need goes beyond that requested by most data sources.

	Java	GeoTools 2.1	Proposal
Content:	Object	Object, Feature, FeatureCollection	Object, Complex, Feature, FeatureCollection
Schema:	Field	AttributeType	Schema
Type:	Class	FeatureType	Type, ComplexType, FeatureType, FeatureCollectionType

You can see that the intended separation is maintained by both GeoTools and this Proposal.

It should be noted that XPath works off the Type system, validation works of the Reference system.

At first draft we had:

	<a href="#">Jody</a>	<a href="#">Gabriel</a>	<a href="#">Revision</a>
Content:	Object, Complex, Feature, FeatureCollection	Attribtue, Feature	Object, Feature, FeatureCollection
Schema:	Schema, ...	Schema, ...	AttributeType, FeatureAttributeType

Type:	Type, ComplexType, FeatureType, FeatureCollectionType	AttributeType, FeatureType	FeatureType
-------	---	----------------------------	-------------

The two drafts mostly differ in:

- the direction of associations (strangely enough aligned with the authors primary concerns of Validation/Schema and XPath/Containment)
- treatment of Atomic and Collection Types

These how now been merged.

## First Draft

Diagram	Differences
	<p>Attribtue referes to Schema          Schema refers to FeatureType          NodeSchema refers to AttributeType          Schema stands between AttributeType and FeatureType</p>

This is a different take on the same problem.

## Q&A

Q: Association from Attribute to Node Schema?

JG: Should point to Schema (so we can have complex Attribtues)

Q: Association from Attribute to Schema and Feature to ComplexSchema

JG:: This is reversed from the proposal

Q: NodeSchema extends Schema?

JG: Does getRestriction(): List<Filter> present a problem?

JG: Does getFeatureType(): FeatureType present a problem?

Q: Where are the names name?

GR: AttributeType.getName(): GenericName

JG: That works for attributes, where is name for Complex content?

GR: FeatureType.getNamespace(): URI and FeatureType.getTypeName()

JG: Oh right then ... that makes schema strongly typed with objects, and all the soft squish stuff is separated out, nice.

Q: But what about AtomicTypes?

GR: They are their see right under SimpleAttributeType?

JG: But how are you going to reuse xs:string? Do you need SimpleAttributeType.getSuper(): AtomicAttributeType?

JG: You can even move it up a level and have AttribtueType.getSuper(): AttributeType, so everyone has the same modeling power that you have available when you defined the AtomicAttribtueTypes from XMLSchema.

XMLSchema is all very well and standard, but there are other standards (like GML) that would like to have the same power of reuse (for things like gml:srs ).

Q: Where is the Super? FeatureType

JG: I don't see it ... were you thinking of having it on FeatureType? Or could it be done on Schema?

Q: Where is ComplexType (or Complex for that matter)?

GR: It is not needed ...

JG: I don't get it.

## Revision of GeoTools 2.1 API

⚠ This approach was rejected, as was the revision of GeoAPI in favour of a shared model. It is only included for comparison.

Revision of GeoTools would require the following minimum changes:

-  added
-  removed
-  changed
-  explained

		Feature	Description
	String	getID()	Feature ID unique across instances
	Envelope	getBounds()	may be null
	FeatureType	getFeatureType()	
	FeatureCollection	getParent()	OSG Reference Model has this relationship reversed *
	int	getNumberOfAttributes()	
	Object[]	getAttributes( Object array[] )	Copy as many attribute values as fit into array (and return it)
	Object[]	getAttributes()	Array of values of length getNumberOfAttributes()
	AttributeType[]	getAttributeTypes()	Array of attributeTypes of length getNumberOfAttributes()
	Object	getAttribute( int index )	
	AttributeType	getAttributeType( int index )	
	Object	getAttribute( AttributeType )	

	Object	getAttribute( String name )	Geotools 2.1 said xpath, but implemented name **
		<b>BoundedFeature</b>	<b>Bounds are mandatory</b>
	Envelope	getBounds()	required to be non null
	FeatureType	getFeatureType()	required to have FeatureType.BOUNDED as a super type
		<b>"FlatFeature"</b>	
		<b>(DefaultFeatureType)</b>	attribute multiplicity not supported
	int	getNumberOfAttributes()	return getFeatureType().getAttributeCount()
	AttributeType[]	getAttributeTypes();	return getFeatureType().getAttributeTypes()
	Object[]	getAttributes()	Values in order described by getAttributeTypes()
	Object	getAttribute( AttributeType )	We can be sure of one value or null
	Object	getAttribute( String name )	We can be sure of one value or null
		<b>"ComplexFeature"</b>	
		<b>(FeatureImpl)</b>	Supports multiplicity and complex attributes
	int	getNumberOfAttributes()	number of attributes for <b>this</b> feature
	AttributeType[]	getAttributeTypes()	matches actual content, in order provided
	Object[]	getAttributes()	values in order described by getAttributeTypes()

	Object	getAttribute( AttributeType )	single instance or List based on multiplicity
	Object	getAttribute( String name )	single instance or List based on multiplicity
		<b>FeatureCollection</b>	<b>has child features</b>
		<b>(FeatureCollectionImpl)</b>	extends Collection, BoundedFeature
	FeatureType	getFeatureType()	required to have FeatureType.COLLECTION (and BOUNDED) as a super type
	Envelope	getBounds()	non null (because we are a BoundedFeature)
	FeatureIterator	features()	Sames as Iterator<Feature> if we had Java 5
	void	addListener( CollectionListener )	not hooked up for most implementations
	void	removeListener( CollectionListener )	not hooked up for most implementations
	int	size()	Note this is at worst O(N) as per Collection.size() spec
		Collection.*	Implement as per Collection specification
		<b>FeatureList</b>	<b>has ordered features</b>
		<i>proposed</i>	Support random access for Shapefile and Postgis, Think Filter 1.1 "SortBy"
	int	size()	To be useful this will be O(1)
	Object	get( int index )	acquire Feature based index in internal order, RandomAccess!

	RandomAccess	features()	A FeatureIterator that is also a ListIterator
		List.*	Implement as per List specification

\* [OGC Reference Model](#) - Page 11 "Parent Features contain References to their children."

\*\* Note on xpath - I am still hopeful we can make xpath external to the data model (it is a query model after all).

Note on multiplicity & getAttribute w/ name or attributeType:

- a single instance when attribute type minOccurs=1 and maxOccurs=1
- null or single instance when attributeType minOccurs=0 and maxOccurs=1
- List when minOccurs=0 and maxOccurs=unbounded