FeatureTypes for GML

Proposal for GeoTools Features to more fully handle GML Schemas

Prepared by:	Chris Holmes, The Open Planning Project and David Zwiers, Refractions Research

Introduction

The GeoTools Feature model was written with the thought of dealing with the full complexity of GML in mind - to be able to fully translate a GML document into a structure of Features and FeatureTypes (to hold an XML document and its XML Schema, respectively). It was designed quite well, but the problem is that up until now all of our feature datastores have only produced Simple Features - that is ones with no nested or complex structures. The point of this proposal is to review and slightly redesign our interfaces and implementations and interfaces to deal with the lacking aspects of our Feature model that are highlighted by David's GML parser.

Plan of Attack

We had a few basic goals that we wanted to hit:

- Make the AttributeType hierarchy more obvious (it was hidden in DefaultAttributeType)
- Introduce Choice, Multiplicty, and Set constructions from XML to the Feature model
- Have FeatureCollection *be* a Feature, and allow similar complex nestings
- Support Type restrictions (facets in xml).

And the other design consideration that drove our choices was the desire to change things as little as possible, to use existing structures as much as we could, and to make the change over relatively painless. We have no desire to introduce radical change, we simply want to extend and hash out the implications of Features that are not simple. The changes should not break any client code, though we did allow ourselves to deprecate a few methods, to be removed at a later date, as there were a few subtle assumptions about features being simple.

The work

The results of our work can be seen on a branch of subversion, called features-exp2. See http://svn.geotools.org/geotools/branches/features-exp2/ And most of the work is in the feature module: http://svn.geotools.org/geotools/branches/features-exp2/module/main/src/org/geotools/feature/

We were able to accomplish quite a lot without changing too much around. This is mostly due to the excellent forward thinking to use xpath to specify features. This made everything a lot easier, we ended up putting xpath a lot more front and center, to specify how to get out all the new complex constructors.

AttributeType hierarchy

The first step we did was to have FeatureType extend AttributeType. This was already more or less done, as there was a DefaultAttributeType.FeatureType static inner class. We just wanted to make the fact that you can do this a little more front and center. In addition we split out the rest of the DefaultAttributeTypes - Numeric, Textual, and Temporal, into their own classes, in the types/ directory. This puts their documentation more at the fore for javadocs, and emphasizes that implementors can make their own AttributeTypes.

- Future Work: The DefaultAttributeType.Geometric code should probably be refactored into GeometryAttributeType.

The next step was to add AttributeTypes to deal with the new complexity. We ended up making three complex attributeTypes: ChoiceAttributeType , ListAttributeType and SetAttributeType . Each represent different core concepts available in XML Schema. A Choice is an xml construct - it is the same as a Union in C - a single structure than can hold a few different types. In XML it is a single field, but you are allowed a choice of the types that are allowed. The SetAttributeType is the ANY construct in XML, it is the same as an unordered set. The XML Sequence construct is our ListAttributeType. This is the most familiar to GeoTools land, as our FeatureType construct is one, it specifies an order for the elements (attributes in feature speak). Indeed in our first stab FeatureType was an abstract class that extended from NestedAttributeType, which extended from ListAttributeType. We ended up refactoring, so that FeatureType could be an interface, but DefaultFeatureType still extends NestedAttributeType. NestedAttributeType does remain - it is a useful construct to think about things - an embedded ListFeatureType. A nested attribute is basically a Feature, but without a featureid. So this allows various nested complex objects.

Multiplicity

We decided to not go with explicit Multiplicity classes, but to give all the option to support it. We added getMin() and getMax() functions to AttributeType. DefaultAttributeTypes would be 0 and 1, or 1 and 1 for those values. We left in isNilleable, since in XML nilleable and min are

different constructs. Having multiplicity would imply that you should return a list for all calls to getAttribute(), and that anybody that wanted the first attribute would have to specify a 0 in the xpath. We decided to get around this annoyance by creating a SimpleFeature class, where a getAttribute() call would imply a 0, since there would not be multiplicity in that class. You could call instanceof SimpleFeature, and determine whether you could use the shortcuts. Or you could just do things safely, with xpaths. All our current DataStores, except GML, should then just return SimpleFeatures.

- Future Work: I would like to revisit the getAttribute calls a bit, I think they may need a bit more hashing out, getting user feedback.

FeatureCollection as Feature

We changed FeatureCollection to extend Feature. This was a good test of the work, seeing if all the concepts of a FeatureCollection could be accessed through the Feature interface. What ended up happening is that most of the collection calls became just shortcuts to xpath calls, optimized for how a FeatureCollection is backed.

- Future work: We didn't fully hash out TypedFeatureCollections and the like. FeatureCollections take a FeatureType in the constructor, but they are not smart about validation and using it, enforcing what FeatureTypes are allowed in and not. I think this is where the FeatureCollection implements Feature can really shine, to do some very smart stuff with how FeatureTypes are handled in conjunction with FeatureCollections, enforcing various conditions about Features in a collection, and allowing users to get a lot more information about Features in the collection. You could do things like making a ChoiceAttributeType between several FeatureTypes, so that it is a multi-typed collection - each feature must validate against one of three feature types. I do believe we set up the structure to make this possible, someone just needs to spend some good time on it.

AttributeType restrictions (aka XML Facets)

The last piece we wanted to at least get a shot at is XML Facets. Being able to restrict the basic types in fairly arbitrary ways. A simple case is setting a string length. This was done in a very half assed manner in our current api, with the getFieldLength() method. This was not thought through, since field length could mean many things in different contexts, like a double vs. an int vs. a string. It is a very necessary thing, as our datastores also support it, and that information is currently lost in the GeoTools model. In a database a varchar(9) (specifying that the string can not be longer than 9 characters) is simply translated to a String in GeoTools. This is also the field length in a shapefile. We wanted to deal with that construct, but also with much more complex situations - in XML you can say things like strings must meet a certain regular expression, or that numbers must be below a certain value. Rather than re-invent the wheel we decided to make base restrictions on Filter. We are not too sure if this experiment will work out, as it needs test cases. But it has a high potential, and it needs little investement, since we already have all the code to do Filters. And it does handle the first few use cases we could think of. This ended up as a getRestriction method in AttributeType. This easily bootstraps on our exisiting stuff, since most of them will just return Filter.NONE - there is no restriction.

- Future work: This all needs a lot of use cases, to see if it fits the needs of DataStores and GML and users. It is now only mentioned, it needs to be plugged in to the current code. The first place to do this would be the validate() method, as the restriction could invalidate certain objects. It remains to be seen if these restrictions will be in the main attribute type implementations - if it is a lot of code it may make sense to add a set of RestrictionAttributeTypes to the attributeType hierarhcy.

Desired but not completed Work.

There were a few requests that were wanted, but that we did not get to. There simply wasn't enough time, and indeed both are slightly different issues, which were just going to piggy backed on this work, since it was the 'core geotools changes' work. I am hoping that we can start to shift the perception that core geotools changes have to be massive efforts that only happen once in awhile. This work is a start in that direction, with a fairly compact set of changes, that are performed on a branch (instead of hacking main, even though it is technically the 'unstable' branch), and easily rolled in when approved. The two changes that didn't get in are FeatureSource returning a backed FeatureCollection instead of a FeatureResults object that is super similar. This was actually approved, but needs to be coded. The other is some sort of Meta type object, perhaps the FeatureTypeInfo idea that has been kicking around, as there are structures in udig and geoserver that want more info. I was always hesitant on these before, basically because I was scared people would start to put in tons of additional datastore specific information that our feature model did not handle. Now that we have set up the structure to handle a much larger percentage of GML and DataStores, I am fine with some meta objects, to handle the really specific stuff, hints and the like, that should not be in the core geotools feature model.

Proposal

We would very much like to get this work on head this week, as I am taking off for Africa next week. There are definitely a few places that need more work, but right now everything compiles, and should work. We did not end up changing the exisiting interfaces that much at all, it was mostly extending them and refining the meanings of the methods to be a bit more specific, to deal with more. I can spend some good time this week doing a lot more documentation for all of the new changes, and then we can let the implementations shake themselves out in an open source manner, finding and fixing the bugs. These should just be with the newer work, not with any exisiting things (or at least hopefully not). David is also going to switch over his XML Parser to use these classes as the default, which will mean that everything in GML will be parsed into GeoTools objects, which is a huge win. This work will also shake out a number of additional bugs, and put solid implementations behind the interfaces. People will start dealing with more and more complex GML documents, and leveraging and bug spotting on the new geotools classes. We feel this work is of vital importance to move GeoTools ahead, as the feature model has felt a bit constrained for awhile, from several directions. Indeed not having these structures was part of what made Social Change Online's work so hard, and why they ended up just hacking in GeoServer instead of doing things right in GeoTools. This allows people to do things right, and to represent much more information natively in GeoTools, instead of resorting to nasty hacks.