Process API Design Alternatives

One long standing request of the GeoTools code base is to offer an operations api for working on Features (similar to what is available for grid coverage).

The idea here is to have a low level interface to handle a very simple kind of operations on data.

Refractions has been working on a "Web Process Service" extension for GeoServer and has thus defined the following modules:

- gt-process - and API outlined on this page for defining a process with parameter/result information handled as a Map (described in a manner similar to how DataStore does it)
- gt-wps - a client to talk to a WPS; also contains a bridge advertising the services of a WPS using the gt-process API

These are currently two unsupported modules - and we don't need a proposal to cover them. This page has thus been moved out of the way.

Scope

Here are examples of what I call "usual operations on data" :

- http://docs.codehaus.org/display/GEOTOOLS/Widget-tools+list
- http://udig.refractions.net/gallery/linecleaner/
  Previous attempts have been limited in scope to lack of collaboration and focus. Often they were created in isolation by people wanting to write feature hacking stuff; rather than by desktop applications.
- http://udig.refractions.net/docs/uDig-DataAccessGuide.pdf
- IOp
- ProgressListener

Practical Examples

- ogr2ogr binded in a process
  FWtools gives a set of .exe files very interesting, ogr2ogr is one of them. This exemple is to demonstrate a process can be linked to native executables.
- shoal technology to send process on different processors
  this exemple is to raise the problem of dispatching processes anywhere, so process should not have relation with threads.

Hot-Swap and Dynamically Defined Process

We have the request from Simone to support hot swapping of process implementations; as such assume he would use OSGi to swap out an old plug-in and swap in a new one. A method to **kick** the ProcessFinder into searching the classpath again would thus be needed.

The problem of dynamically defined processes (say as a groovy script) is a bit more tricky since we are a Java program and like to work with classes and instances. I hope that Dynamic Proxy can be used here by whatever developer wants to try the idea out.

Motivation

Eclesia has three main reasons:

1. Offer a structure for usual operations
2. Once a low level process is written, Swing or SWT Widgets can be made separately.
3. Thoses processes can be quickly adapted in ETL applications

Jody has two reasons:

1. Kick valuable code out of the uDig codebase for wider use (such as the reshape operation)
2. Make use of Eclesia's enthusiasm
3. Want to make sure this stays simple enough to work; the last four attempts failed to catch on

Relationship to Web Processing Service:

- none!
- no really we did not look at it first; consider this API as raw ability that you could wrap up as a WPS
- you could also hide an external WPS behind this API

Requirements and Acceptance Tests

Acceptance tests are the **best** definition of scope, the final API we present here will meet the following requirements.

Controlling scope:

- The best solution is the ones that is easiest for implementors to make a new Process for us; that is simply measured as number of lines of code (the burden can be eased with a nice abstract super class).
  Controlling the scope of "process description":
- The ability to create a Swing user interface based **just** on the Process description
- The ability to create a WPS GetCapabilities and DescribeProcess

Controlling the scope of "process":

- The ability to execute a process
- The ability to track a process that is executing
- The ability to interrupt a process that is executing

Not quite in scope:

- The ability to split a process in two, and merge the result (see FeatureVisitor for examples)
- The ability to chain several processes together

While our design will address these concerns, we will only implement what we need at this time. It is way better to wait until someone has a real live problem in hand in order to test the solution.

API

The API is currently being defined using code - the final API will meet several acceptance tests.

First Attempt - ProcessFactory and Parameter

Goals:

- Write down an initial starting point

Feedback:

- When implementing a process you have a strange dance where the inputParameters have to be provided along with you factory as part of the constructor You also need to run over to your ProcessFactory and make references to the static final Parameter implementations it has in order to lookup keys in your inputParameters, and store your resultParameters
- Not having the inputParameters as part of the Process interface made it impossible to use just a Process object on its own in normal Java code
- Not able to handle multiplicity
- unclear that you can only call process once? or can you call it multiple times ...

```
/** Used to describe the parameters needed, and to create a Process for use. */
public interface ProcessFactory {
    public InternationalString getTitle();
    public InternationalString getDescription();
    public Map<String,Parameter> getParametersInfo();
    public boolean isValid(Map<String, Object> parameters);
    public Process create(Map<String, Object> parameters) throws
IllegalArgumentException;
    public Map<String,Parameter> getResultInfo(Map<String, Object> parameters) throws
IllegalArgumentException;
}
public class Parameter {
    public final String key;
    public final InternationalString description;
    public final Class type;  // expected value, may be in a list based on maxOccurs
    public final boolean required; // Can the value be missing? Or is null allowed...

    public final Object sample; // default value used to prompt user
    public final Map<String,Object> metadata;
    public static final String FEATURE_TYPE = "featureType";
    public static final String CRS = "crs";
    public static final String LENGTH = "length";

    /** Mandatory information */
    public Parameter(String key, Class type, InternationalString description ){
        this( key, type, description, false, null, null );
    }
    /** Addition of optional parameters */
    public Parameter(String key, Class type, InternationalString description,
                     boolean required, Object sample, Object metadata){
        this.key = key;
        this.type = type;
        this.description = description;
        this.required = required;
        this.sample = sample;
        this.metadata = metadata;
    }
}
/**
 * Used to Process inputs, process is reported using a ProgressListener. Results are
 * available after being run.
 */
public interface Process {
    public Map<String,Object> process(ProgressListener monitor);
    public ProcessFactory getFactory();
}
```

Single thread example (say from a main method):

```
ClipProcessFactory descriptor = new ClipProcessFactory();

// note can only use this once!
Process clip = factory.create(myParams);
Object result = clip.process( new PrintListener() );
```

Multiple threads example (say from a Swing Button):

```
ClipProcessFactory descriptor = new ClipProcessFactory();
final Process clip = factory.create(myParams);
Thread t = new Thread( new Runnable(){
   public void run(){
      Object result = process.process( new SwingProcessDialog() );
   }
});

// dispatch process!
t.start();
```

Second Attempt - Process and Parameter

Goals:

- makes the process method accept input parameters; as a result is is much easier to code up this method, the inputParameters are visible in our interface and can be documented
- this has the side effect of making processfactory / process split non useful (since the process method is not stateful)
- strictly admit to isNillable, minOccurs, maxOccurs in the Parameter api (leaving metadata to address user interface concerns)
- name has been provided for Process; to allow for dynamicly generated processes that cannot be strictly identified using a classname

Feedback:

- panic that there is not an explicit **Process** object around that can be manged in a queue (this was a surprise to me since whatever job system is used you will need to write a wrapper around the bundle of inputParameters and Process; even if it is just a SwingWorker or Runnable).
  To repeat the design goal: we need to make Process easy to implement - wrapping the result up for a job system or dispatch to a grid system is not our concern.
- need to handle up multiplicity

```
/**
 * Used to describe the parameters needed, and provide a single process method to call
 * with input parameters.
 */
public interface Process {
    public String getName(); // machine readable name - may just be
getClass().getName()
    public InternationalString getTitle(); // human readable title changes with locale
    public InternationalString getDescription();
    public Map<String,Parameter> getParametersInfo();
    public boolean isValid(Map<String, Object> parameters);
    public Map<String,Parameter> getResultInfo(Map<String, Object> parameters) throws
IllegalArgumentException;

    public Map<String,Object> process(Map<String,Object> input, ProgressListener
monitor);
}
public class Parameter {
    public final String key;
    public final InternationalString description;
    public final Class type;  // expected value, may be in a list based on maxOccurs
    public final boolean required; // Can the value be missing? Or is null allowed...

    // Hints for the User Interface / GetCapabilities
    public final Object sample; // default value used to prompt user
    public final Map<String,Object> metadata;
    public static final String FEATURE_TYPE = "featureType";
    public static final String CRS = "crs";
    public static final String LENGTH = "length";
    public static final String MIN = "min";
    public static final String MAX = "max";

    /** Mandatory information */
    public Parameter(String key, Class type, InternationalString description ){
        this( key, type, description, false, null, null );
    }
    /** Addition of optional parameters */
    public Parameter(String key, Class type, InternationalString description,
                     boolean required, Object sample, Object metadata){
        this.key = key;
        this.type = type;
        this.description = description;
        this.required = required;
        this.sample = sample;
        this.metadata = metadata;
    }
}
```

Normalize using ISO 19111
This proposal looks like the most reasonable and the easiest to wrap for any use.

We could even have something nicer by replacing our parameter class by GeoAPI "Parameter" - ISO 19111
cf : http://geoapi.sourceforge.net/snapshot/javadoc/org/opengis/parameter/package-summary.html

Even if a Parameter as defined in GeoAPI is a bit hard, we can make an abstract class to simplify and fit our basic needs.

Oh please no! Parameter and ParameterGroup come from a metadata background and serve as description, but do not provide

enough info to build a good user interface from. Please see the WMS GridCoverageExchange implementation for an example of trying to dynamically build up a ParameterGroup describing the available layers; styles and so on.

If you want a decent replacement look at PropertyDescritpror and the bridge is supplied to a PropertyEditor.

If you must consider ISO stuff please review ISO 19119; there are no good public things I can point you to. Oh wait ... have a look at Diagram 10 of WRS.pdf The diagram was genreated from ISO 19119 stuff, and shows you the realtionship between Parameter

We should move this stuff to the comments section...

Third Attempt - ProcessFactory and Parameter with a standalone Process

Goals:

- make Process a stand alone object so that it can be more easily managed / wrapped by job systems such as Runnable, SwingWorker, Eclipse Jobs, etc...
- Create an AbstractProcess that takes care of a lot of the grunt work so that the poor implementer has a small number of lines of code to write
- Strictly seperate out value constraints (type,isNillable,minOccurs,maxOccurs) used by the framework from the general metadata used to assist the end user
- Define a subclass of Process that has the ideas of Splitting and Merging in order to handle distribution of a process across multiple threads (the alternative is some more control metadata at the Process level)

Feedback:

- pending

```
/**
 * Used to describe the parameters needed, and provide a single process method to call
 * with input parameters.
 */
public interface ProcessFactory {
    public String getName();
    public InternationalString getTitle();
    public InternationalString getDescription();
    public Map<String,Parameter> getParametersInfo();
    public boolean isValid(Map<String, Object> parameters);
    public Process create();
    public Map<String,Parameter> getResultInfo(Map<String, Object> parameters) throws
IllegalArgumentException;
}
public interface Process {
    setInput( Map<String,Object> input );
    setResult( Map<String,Object> output );
    public void process(ProgressListener monitor) throws Exception; // call only once
}
public interface DistributedProcess extends Process {
    List<Process> split( int number ); // split process for a number of threads
    join( List<Process> children ); // results will be merged
}
public class Parameter {
    public final String key;
    public final InternationalString description;
    public final Class type;  // expected value, may be in a list based on maxOccurs
    public final boolean required; // Can the value be missing? Or is null allowed...
    public final int minOccurs; // either 1, or indicates value should be a list
    public final int maxOccurs; // either 1, or indicates value should be a list

    // Hints for the User Interface / GetCapabilities
    public final Object sample; // default value used to prompt user
    public final Map<String,Object> metadata;
```

```
    public static final String FEATURE_TYPE = "featureType";
    public static final String CRS = "crs";
    public static final String LENGTH = "length";
    public static final String MIN = "min";
    public static final String MAX = "max";

    /** Mandatory information */
    public Parameter(String key, Class type, InternationalString description ){
        this( key, type, description, false, 1,1, null, null );
    }
    /** Addition of optional parameters */
    public Parameter(String key, Class type, InternationalString description,
                     boolean required, int minOccurs, int maxOccurs, Object sample,
Object metadata){
        this.key = key;
        this.type = type;
        this.description = description;
        this.required = required;
        this.minOccurs= minOccurs;
        this.maxOccurs= maxOccurs;
        this.sample = sample;
        this.metadata = metadata;
```

```
    }
}
```

4th Attempt - Process and Beans

Goals:

- Benefit all the bean property description to build user interface "on the fly"
- Less code for implementers to write

Feedback:

- NetBeans RCP : really easy to embed
- Down side is that minOccurs, maxOccurs kind of information cannot be provided, poor integration with List<Geometry> and internationalization support is poor (based on GeoServer experience)
- desruisseaux asked : why not extending the runnable interface (java standard) and replace method process by run ?

```
public interface Process <I,O>{
    public String getName(); //Name is a unique ID
    public InternationalString getTitle(); //Process Title, human readable
    public InternationalString getAbstract(); //Process Description, human readable

    // the InputBean is a simple Bean which has all the input properties
    // needed for the process
    public I getInputBean();

    // The outputBean is a simple bean which has all the output properties
    // in read access only.
    // this is a dynamic bean, so once you have initialize the inputbean you can grab
    // this bean and "parse" it
    public O getOutputBean();

    //Start the process operations
    public void process(ProgressListener monitor);
}
```