Validating Web Feature Server Implementation Report

Submitted To: Program Manager
GeoConnections
Victoria, BC, Canada

Submitted By: Jody Garnett
Refractions Research Inc.
Suite 400 – 1207 Douglas Street
Victoria, BC, V8W-2E7
jgarnett@refractions.net
Phone: (250) 383-3022
Fax: (250) 383-2140

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

1 INTRODUCTION

This document outlines our experience in extending the Validating Web Feature Server to support a robust configuration model.

A limited number of changes have been made with respect to our GeoServer Web Based Configuration Design Document and Validation Implementation Report.

The GeoServer configuration process has been modified to:

- Clarify separation of application layers
- Clarify use of FeatureType References
- Allow internationalization support

This design has been very well received by the development community and is being used as an example for further development.

The validation API has remained stable through this phase of development. Minor modifications have been made to allow for internationalization and to clarify the handling of FeatureType references.

2 GeoServer Configuration Design

The Web Based Configuration Design document provided a series of design goals for updating the GeoServer Configuration System:

- Separate out the Configuration Model from the GeoServer Application
- Build a Struts Web Interface against the Configuration Model
- Allow XML persistence of the GeoServer Application Configuration state
- Maintain the existing configuration file format

To meet these requirements, Layered Architecture was proposed (Figure 1).

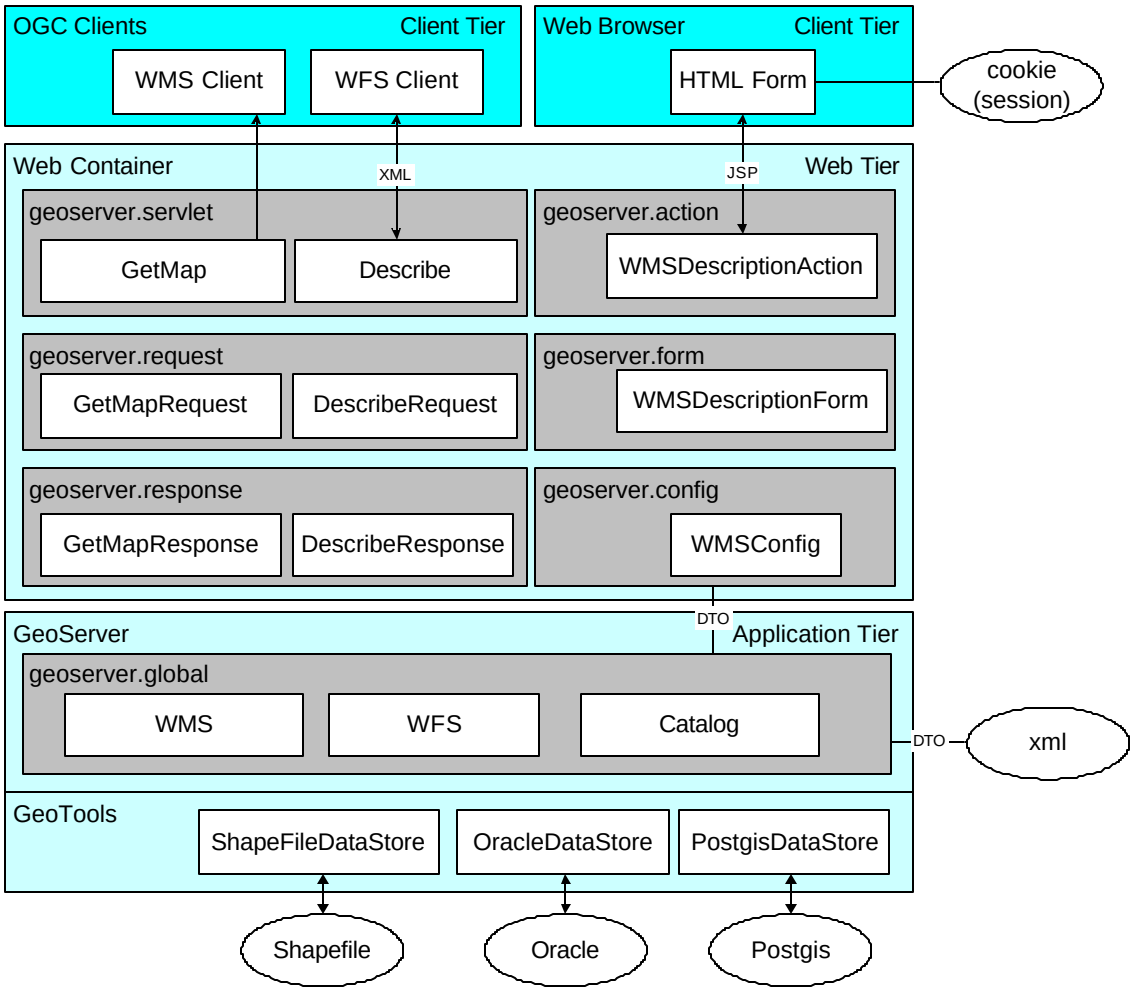

Figure 1 - GeoServer Layer Diagram

2.1 Design Successes

The design for the GeoServer Configuration System has:

- Succeeded in meeting our Requirements.
- Succeeded in separating the GeoServer application, configuration, and persistence subsystems
- Succeeded in allowing unit testing of Core GeoServer components
- Succeeded in maintaining a clear separation between GeoServer and the Validation Framework

2.2 Design Extensions

The previous Web Based Configuration Design document outlined the layer structure of the GeoServer application.

Table 1 lists the GeoServer classes by both package and subsystem (interfaces are marked in italics):

GeoTools	Global	DTO	Config	Form
	GeoServer Contact	GeoServerDTO ContactDTO	GeoServerConfig	GeoServerConfigurationForm
	WMS	WMSDTO	WMSConfig	WMSDescriptionForm WMSContentForm
	WFS	WFSDTO	WFSConfig	WFSDescriptionForm WFSContentForm
Catalog	Data	DataDTO	DataConfig	
NamespaceMetaData	NamespaceInfo	NamespaceDTO	NamespaceConfig	DataNamespacesForm
DataStoreMetaData	DataStoreInfo	DataStoreInfoDTO	DataStoreInfoConfig	DataDataStoresEditorForm DataDataStoresNewForm DataDataStoresSelectForm
FeatureTypeMetaData	FeatureTypeInfo	FeatureTypeInfoDTO	FeatureTypeInfoConfig	DataFeatureTypesEditorForm DataFeatureTypesNewForm DataFeatureTypesSelectForm DataAttributeTypesNewForm DataAttributeTypesSelectForm
	Styles	StyleDTO	StyleConfig	DataStylesFormForm

Table 1 - GeoServer Data Matrix

The GeoTools data module has been extended with Catalog and Metadata interfaces. The use of GeoTools2 interfaces by classes in the global layer allows the Validation Processor to be written in an application independent manner.

2.2.1 Validation Framework Extension

The extension of our design for the validation processor to accommodate a web based user interface was accomplished without incident.

Table 2 lists the Validation classes and interfaces by package (interfaces are marked in italics):

Validation	Validation DTO	Config	Form
ValidationProcessor		ValidationConfig	
PlugIn	PlugInDTO	PlugInConfig	
TestSuite	TestSuiteDTO	TestSuiteConfig	ValidationTestSuiteNewForm ValidationTestSuiteSelectForm
Validation	TestDTO	TestConfig	ValidationTestEditorForm ValidationTestNewForm ValidationTestSelectForm

Table 2 - Validation Processor Data Matrix

We have maintained a clear separation between the Validation Processor and the GeoServer application.

2.3 Design Recommendations

Over the course of development we have arrived at several recommendations for the design of the existing GeoServer application. These recommendations will be placed in the GeoServer bug tracking system (http://jira.codehaus.org/).

2.3.1 Centralize Application Logic

The advent of the WMS and WFS classes in the global package presents an opportunity to improve the GeoServer Application. Currently GeoServer has dispersed application logic across several Servlets.

These Servlets are used to implement the WFS and WMS protocols according to the Transaction Script pattern (Fowler 110).

Transaction Scripts represent an Object Oriented Idiom similar in effect to procedural programming. The use of this design has resulted in significant duplication of functionality.

By moving common functionality to the WMS and WFS classes in the global package we can reduce duplication.

Request	Servlet	Duplication
WFS GetCapabilities	wfs/GetCapabilities	WFS
WFS DescribeFeatureType	wfs/DescribeFetureType	WFS
WFS GetFeature WFS GetFeatureWIthLock	wfs/GetFeature	WFS, Data, FeatureTypeInfo
WFS LockFeature	wfs/LockFeature	WFS, Data, FeatureTypeInfo
WFS Transaction	wfs/Transaction	WFS, Data, FeatureTypeInfo
WMS GetCapabilities	wms/GetCapabilities	WMS
WMS GetMap	wms/GetMap	WMS
WMS GetFeatureInfo	wms/GetFeatureInfo	WMS, Data, FeatureTypeInfo
WMS Describe Layer	wms/DescribeLayer	WMS, Data, FeatureTypeInfo

Table 3 - Duplicate Application Logic

Specific recommendations:

- Move FeatureType Schema generation from DescribeFeatureType to FeatureTypeInfo. This functionality is currently duplicated.
- Allow WFS to publish a selection of FeatureTypes defined in Data. This functionality is currently duplicated in the FeatureTypeInfo.
- Allow WMS to publish a selection of FeatureTypes defined in Data. This functionality is currently duplicated in the FeatureTypeInfo.
- Allows WMS to control the style associated with a Layer.

It may be noted that the global Data classes module has already accomplished this centralization for the GeoServer Data classes and the Validation Processor.

2.3.2 Proposed Data Layer

WMS and WFS Servlets should not access Catalog information directly. A more robust implementation would access the required data through the WFS or WMS global implementations.

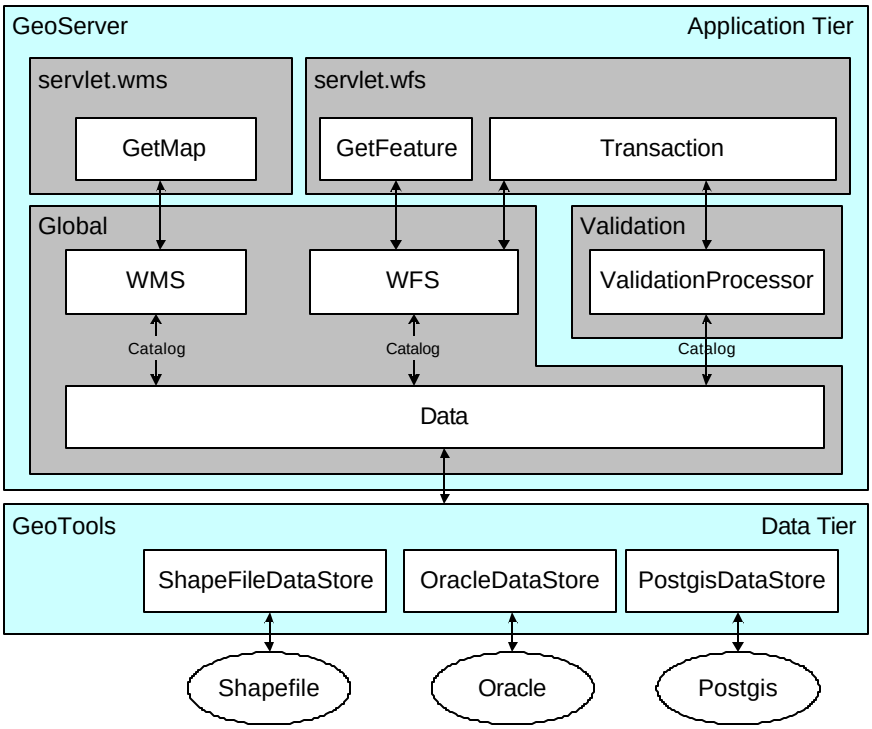

Figure 2 - Proposed Data Layer

Advantages of this modification:

- Allows separate enabling of FeatureTypes for WFS and WMS
- Allows the relocation of Style to WMS

We may choose to have the WMS and WFS classes access the Data class through the GeoTools2 Catalog interface in a manner similar to the Validation Processor.

2.3.3 FeatureType Definition

GeoServer currently uses XML Fragments for the `schema.xml` and `info.xml` used to describe FeatureTypes. These fragments are stored in a directory based on the FeatureType name.

Recommendations for FeatureType Definition:

- Use complete XML Documents for `schema.xml` and `info.xml`
- Remove orphaned <attributes> tag from `info.xml`.
- Store FeatureType configuration by DataStore Id

These changes would allow the use of a validating XML parser and prevent FeatureType name conflicts across data sources.

2.3.4 Configure Level of WFS Support

GeoServer is compliant with the full "Transactional Web Feature Server" level of service as defined by the OGC. The ability to limit the capabilities of GeoServer has been frequently requested.

The OGC defines two levels of service that a Web Feature Server may provide and still remain compliant:

- Basic WFS:
  GetCapabilities, DescribeFeatureType and GetFeature
- Transactional WFS:
  Transaction operation, with optional LockFeature

To meet user requests we propose adding a service level tag to the configuration process.

Intended modification to `services.xml`:

```
<service type="WFS">
 <title> My WFS </title>
 ...
 <serviceLevel>Basic</serviceLevel>
 ...
</service>
```

The following table defines our recommended level of Service.

Level of Service	Operations (additive)	OGC Compliant
Basic	GetCapabilities DescribeFeatureType GetFeature	Basic Web Feature Server
Transactional	Transaction	Transactional Web Feature Server
Complete	LockFeature	

Table 4 - WFS Level of Service

This recommendation does not interact with the Validation Processor. The Validation Processor, if configured, will be used for all Transaction operations regardless of level of service.

3 GeoServer Configuration Implementation

3.1 Implementation Successes

While we are always pleased to work with a dynamic project such as GeoServer we are particularly happy with several aspects of our recent work.

3.1.1 Removed Dependence on Singletons

We have been able to remove the previous design's dependence on the Singleton Pattern (GOF 127).

Singletons represent an Object Oriented Idiom similar in effect to global variables. The previous Servlet implementations used this pattern to locate the current application's configuration and data source connections.

Example Singleton use:

```
class MyResponse extends Response {
  public void execute(Request request) {
    GlobalConfig config = GlobalConfig.getInstance();
    …
  }
}
```

The use of a global singleton for application configuration prevents the use of unit testing during development. In order to test any part of the application, all of the application had to be working.

The storage of this information has been migrated to the Web Container. We have made use of the Web Container's life cycle to manage the startup and shutdown process of the GeoServer application.

Prior to these changes the first call to GeoServer was used to configure the application. This practice often resulted in a time-out error. Configuring the application during startup has eliminated this problem.

Our solution to the Singleton issue in the previous design included the following observations:

- AbstractService is used as the super class for all GeoServer Servlets.
- AbstractService makes use of the following workflow:
  1. Get a Request Reader
  2. Ask the Request Reader for the Request object
  3. Ask the Response Handler to execute the Request
  4. Set the http response content type
  5. Write to the http response's output stream

Our solution to the problem included modifying the workflow to provide the original HttpServletRequest which generated the GeoServer Request. The HttpServletRequest received from the user allows the system to locate the web container, which contains a copy of the application's configuration and data source connections.

This refactoring of the Servlets using the Web Container was handled by extending the base Request class:

```
class Request {
  GeoServer getGeoServer()
  ValidationProcessor getValidationProcessor()
  UserContainer getUserContainer()
  HttpServletRequest getHttpServletRequest()
  void setHttpServletRequest(HttpServletRequest httpServletRequest )
}
```

Example Use of Request:

```
class MyResponse extends Response {
  public void execute(Request request) {
    GeoServer config = request.getGeoServer();
  }
}
```

Our solution was also instrumental in sharing the current GeoServer application configuration with the configuration management user interface. Both sub-systems could now access copies of the configuration contained in the web container.

The user interface is provided with access the GeoServer application configuration through the extension of GeoServerAction:

```
class GeoServerAction extends Action {
  UserContainer getUserContainer( HttpServletRequest httpServletRequest )
  GeoServer getGeoServer( HttpServletRequest httpServletRequest )
}
```

The preceding classes make use of the utility methods of the Requests class:

```
getGeoServer(HttpServletRequest httpServletRequest )
getValidationProcessor(HttpServletRequest httpServletRequest )
getUserContainer(HttpServletRequest httpServletRequest )
```

These changes have resulted in the following benefits:

- Improved startup process
- Ability to unit test core GeoServer components

3.1.2 Data Transfer Objects (DTO)

Our implementation of Data Transfer Objects for GeoServer configuration proceeded smoothly.

This layer of indirection has afforded parallel development of the Web Based Configuration User Interface and the GeoServer application.

One unexpected consequence of the creation of well-documented Data Objects has been the temptation to extend them through subclassing.

In practice this has several drawbacks:

- Application Objects have different documentation requirements then a DTO Object. An Application Object needs to document properties in terms of use, rather than configuration.
- Application Objects need to limit modification to well-known configuration methods. An Application Object should not have set methods for many properties supported by a DTO Object.

In response to these issues we have been forced to mark the Data Transfer Objects as final.

3.1.3 Ported JUMP Network Builder

Several desired validation tests are defined in terms of network based constraints.

We have ported a well-documented Network Builder from the JUMP Feature Model to the Geotools2 Feature Model.

This has succeeded in allowing the Validation Processor access to powerful technology from another sphere of open source development.

3.2 Implementation Extensions

3.2.1 Attribute Based XMLSchema Generation

The use of XMLSchema to represent FeatureType schema information has resulted in several interesting developments.

GeoServer has included support for a user defined XMLSchema fragment, or the generation of an XMLSchema document based on a live Geotools2 data store.

The definition of schema information in GeoServer is provided by:

- The `schema.xml` file defines an XMLSchema fragment:

```
<xs:complexType name="Deletes_Type">
  <xs:complexContent>
    <xs:extension base="gml:AbstractFeatureType">
      <xs:sequence>
         <xs:element name="id" type="xs:string" minOccurs="0"/>
         <xs:element ref="gml:pointProperty" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

- The `info.xml` file defines a list of attributes:

```
<?xml version="1.0" encoding="UTF-8"?>
<featureType datastore="cite_postgis">
  <name>Deletes</name>
  <SRS>32118</SRS>
  <title>test delets</title>
  <abstract>This is some cite features we can delete</abstract>
  <keywords>delete, New York City, TOPP</keywords>
  <latLonBoundingBox dynamic="false"
          minx="-74.27000" miny="40.50000" maxx="-73.80000"
maxy="40.94000"/>
  <attributes/>
  <styles default="normal"/>
</featureType>
```

- The `schema.properties` file was added during the release of GeoServer 1.1. This file captures the `minOccurs` and `maxOccurs` information from the XMLSchema fragment
- The live data store provides schema information at runtime.

To resolve the duplicated schema information we have:

- Implemented limited parsing of the XMLSchema fragment
- Implemented AttributeTypeDTO for user defined XMLSchema information
- Implemented a mapping of attribute name and type to GML and XMLSchema types allowing the use of GML property types such as `gml:pointProperty`.

These extensions allow improved handling of schema information without duplication.

3.2.2 Handling of Type References

We have made a slight modification to the Validation Plug-In API presented in the Validation Processor Implementation Report.

The Validation interface (from Validation Implementation Report) was:

```
interface Validation {
  String getDescription();
  String getName();
  int getPriority();
  String[] getTypeNames();
  void setDescription( description );
  void setName(String name );
  setTypeNames( String typeNames[] )
}
```

The limitations of this designed were revealed in two ways.

1. Writing a BeanInfo class to describe a Validation Plug-In
2. Connecting a validation test to spatial information.

To address these limitations we have revised the Validation interface:

```
interface Validation {
  String getDescription();
  String getName();
  int getPriority();
  String[] getTypeRefs();
  void setDescription( description );
  void setName(String name );
}
```

The modifications are:

- Removal of `setTypeNames` method.
  This change allows the specification of a property for each FeatureType used.
- Addition of `getTypeRefs()` method (replacing `getTypeNames()` method)
  This change allows the property name to match the use of typeRefs to identify FeatureTypes.

We have chosen to use the reference pattern `dataStoreId:typeName` to identify FeatureType. This allows the XML prefix to vary freely without impact to the Validation Processor.

3.3 Implementation Recommendations

We have arrived at several recommendations for the final phase of development.

3.3.1 Sample Configuration

We would like to package the GeoServer Web Application Archive (war file) with a sample configuration based on publicly available data stores and internal shape files. By providing a working example for users to modify we hope to avoid any initial frustration users may experience setting up the application for their first time.

This also presents an opportunity to showcase the validation processor to the existing GeoServer installation base.

3.3.2 Phase out the use of Document Object Model XML Parsing

The current XML reader used to process configuration information is based on the Document Object Model. The rest of the GeoServer application makes use of SAX based parsers.

GeoTools2 has recently deprecated their DOM based OGC Filter and GML parsers. The configuration subsystem and the validation subsystem use these components.

To account for these changes we will need to port our configuration readers to a SAX based parser.

3.3.3 GML Harmonization

Progress has been made on several fronts in the handling of GML information. We would like to share our recent experience in generating accurate XMLSchema fragments with the GeoTools2 library.

3.3.4 Remove Data Transfer Object Delegates

Several classes in the global package have taken to making use of an internal Data Transfer Object as a delegate. This approach provides a level of code reuse, and lacks many of the drawbacks of sub-classing discussed earlier.

Some of the more apparent drawbacks to this approach include:

- Wrong Granularity/Abstraction:
  By definition a DTO object provides a fine level of detail; it is used to group a collection of configuration information for communication. Direct delegation to the DTO object results in an unwieldy Application Object that requires client code to access several properties to discern state.
- Integrity Gap:
  By directly using the DTO provided as part of the configuration process, an application is tempted to present information that it may be unable to deliver. This is most often seen when accessing dynamic content.

When the above limitations are recognized, making use of a delegate can be made to work. In the interests of maintaining a public code base we would like to phase out this approach.

4 BIBLIOGRAPHY

Fowler, Martin. *Patterns of Enterprise Application Architecture*. Addison Wesley, 2003.
http://www.martinfowler.com/eaaCatalog/transactionScript.html
http://www.martinfowler.com/eaaCatalog/domainModel.html

GOF Eric Gamma ... [et al.]. *Design Patterns Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.