Validating Web Feature Server
Design Document

Submitted To: Program Manager
GeoConnections
Victoria, BC, Canada

Submitted By: Jody Garnett
Brent Owens
Refractions Research Inc.
Suite 400, 1207 Douglas Street
Victoria, BC, V8W-2E7
jgarnett@refractions.net
Phone: (250) 885-0632
Fax: (250) 383-2140

TABLE OF CONTENTS

TABLE OF FIGURES

1 INTRODUCTION

This document describes the creation of a Validating Web Feature Server. A Web Feature Server distributes, allows modification of, feature information over the Web using XML.

GeoServer is the reference Java implementation of the Open GIS Consortium's Web Feature Server specification. This project will extend the GeoServer framework by integrating Integrity tests into the Transaction operation.

2 VALIDATION TESTS

A validating web feature server is designed to provide additional checks to safe guard geodatabase integrity. This requirement is implemented by providing a series of Validation Tests that may be configured for use.

The specification of Validation Tests Plug-Ins, and their design guidelines, is described in a separate document.

This section provides an overview of Validation Tests, their classification and their intended purpose in the context in a VWFS.

2.1 Reference Validation Tests

To illustrate our design we will refer to the following validation tests:

- Duplicate Vertex:
  A simple topology test that checks for duplicate vertices within a single geometry. The test requires a feature, and geometry to test.
- Null Zeros:
  An attribute test used to detect zeros or NULL values. The test requires a feature and attribute to check.
- Intersection:
  A geodatabase integrity test to ensure two FeatureTypes do not intersect. The test requires the name of two layers.

These Validation Tests will be referenced in later sections.

2.2 Validation Test Classification

By classifying tests according to their scope we can offer a simplified design for the vast majority of validation tests.

2.2.1 Feature Validation Tests

The Duplicate Vertex and Null Zeros are example of Feature Validation Tests that are limited in scope to a single Feature.

The VWFS will perform these tests on insert and update feature information prior to any spatial database involvement

2.2.2 Integrity Validation Tests

The Overlaps reference test is an example of an integrity test. These tests are run to ensure the spatial database is in a consistent state.

The VWFS will run these tests prior to committing a transaction.

3 REQUEST INTEGRATION

The OpenGIS Web Feature Server specification describes facilities for serving GIS Feature information over the Internet.

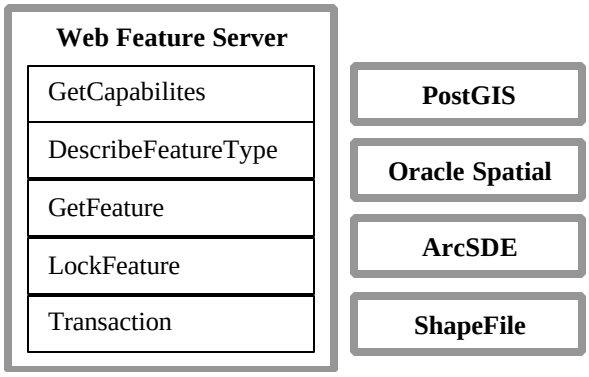


Figure 1 - Web Feature Server

The standard defines the following operations:

- GetCapabilities
- DescribeFeatureType
- GetFeature
- GetFeatureWithLock
- Transaction
- LockFeature

The Transaction operation is of interest for our present development.

A transaction request is defined as one composed of operations that modify features; that is ‘create’, ‘update’ and ‘delete’ operations on geographic features.

The capability of the Transaction Operation to handle multiple operations as a single entity is extremely fortunate. It allows us to package together large enough editing operations to leave the WFS in a consistent state at the end of a transaction without requiring Long Transactions.

3.1 Transaction Operation Example

This section provides an example transaction operation and defines some of the specifics we will be interested in considering in the implementation of a Validating Web Feature Server.

3.1.1 Reference Operation

The following reference operation is used to describe the Web Feature Server Workflow, Communications, and Transaction Requests.

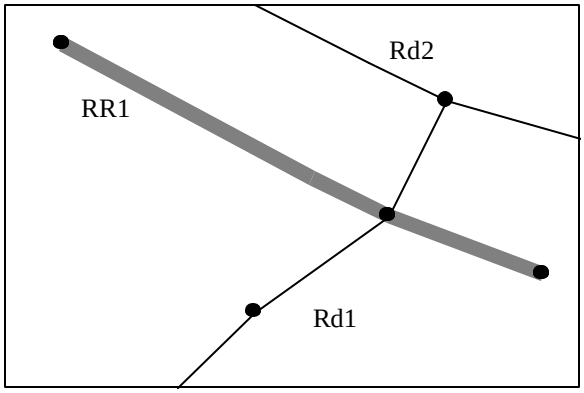


Figure 2 - Railway Overpass Construction Before

A construction project involving placing an overpass over a railway has finished. Previously a railway crossing was used.

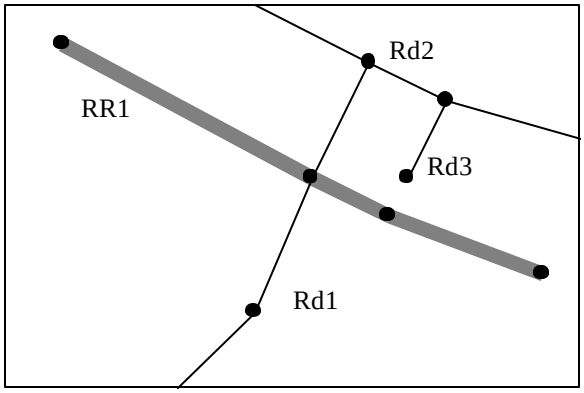


Figure 3 - Railway Overpass Construction After

After the construction was completed the railway crossing was retired and the road was diverted onto the overpass to a new intersection. On one side of the rail way crossing the road was closed off and the other side continues to be maintained as an access road.

3.1.2 Web Feature Server Use Case

The following Use Case describes a sequence of operations required to update a road network after a construction project.

Overpass Construction Use Case

1. Client Setup
   - 1.1. Client Checks for transaction support.
   - 1.2. Client submits a <GetCapabilities> request to the WFS
   - 1.3. The WFS responds with a <GetCapabilities> document including transaction support.
2. Client obtains schema and encoding information about the required layers.
   - 2.1. Client submits a <GetFeatureType> request to the WFS
   - 2.2. WFS responds with a <GetFeatureType> document describing the following layers: Road, Railway, RoadIntersection and RailwayCrossing and associated attributes.
3. Client locks features for modification
   - 3.1. Client submits a <LockFeature> request using a bounding box to limit the lock the area of the construction site.
   - 3.2. WFS responds with a <LockFeature> document, containing a lock id and used for subsequent transactions operations.
4. Client obtains locked features for modification
   - 4.1. Client submits a <GetFeatures> request
   - 4.2. WFS responds with a <GetFeatures> document list the required features in GML notation.
5. The client edits the feature set and constructs a transaction to represent the required update as described in the reference modification (see Section 2.1.1)
6. The client updates the WFS with the resulting edits
   - 6.1. The client submits a <Transaction> containing the following elements:
     - A <Delete> element removing the Rd1/RR1 RailwayCrossing
     - A <Delete> element removing the Rd1/Rd2 RoadIntersection
     - An <Update> element modifying the Rd1 road to reflect the new route
     - An <Insert> element defining the new Rd3 road.
     - An <Update> element modifying the RR1 railway to have a node at the location of the crossing with the new Rd1 road.
     - An <Update> element modifying the Rd2 road to have a nodes at the location of the crossing with the new Rd1 and Rd3 roads.
     - An <Insert> element defining the Rd1/RR1 RailwayCrossing as an overpass.
     - An <Insert> element defining the Rd1/Rd2 and Rd3/Rd2 RoadIntersection entries
   - 6.2. The WFS responds with a <Transaction> Document.
7. The client releases the locks held
   - 7.1. The client constructs the a <Transaction> Request to release held locks
   - 7.2. The WFS responds with a <Transaction> Document.

3.1.3 Network Communication

The following diagram illustrates Web Feature Server Communication during our reference workflow from section

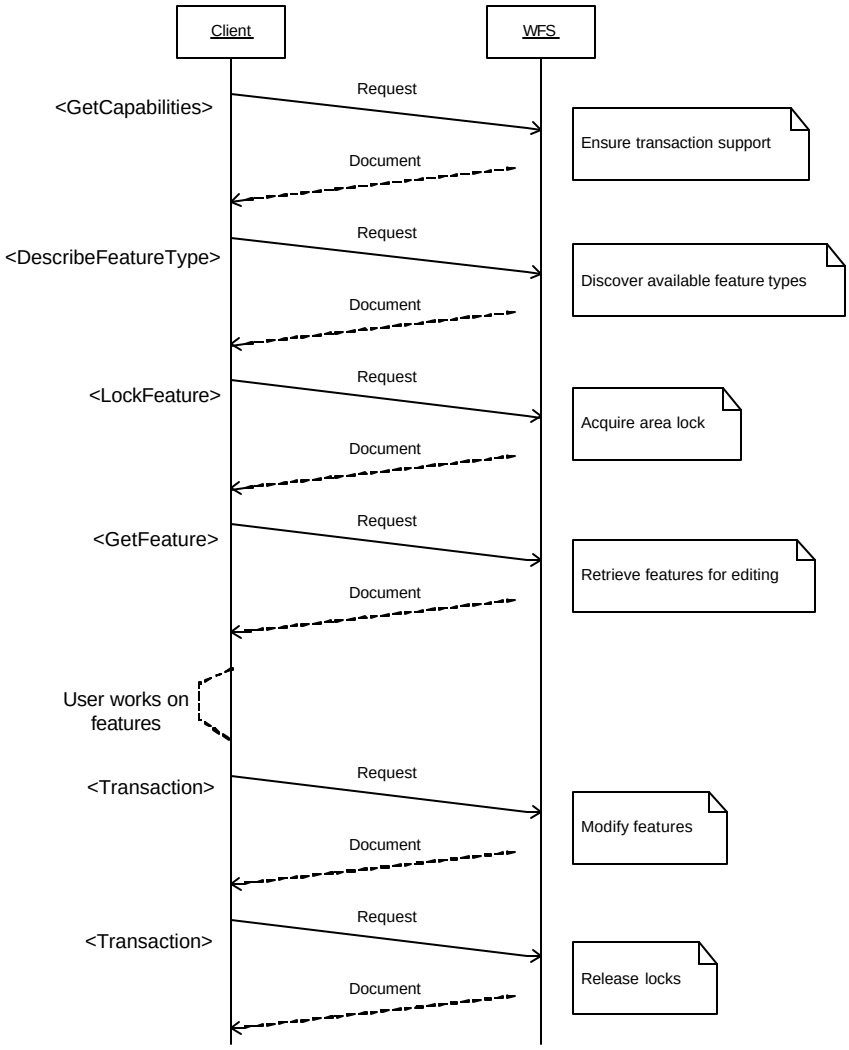


Figure 4 - Web Feature Server Communication Diagram

The Web Feature Server uses HTTP is used to handle requests. For more information on the protocol used please consult the Web Feature Server Implementation Specification

3.1.4 WFS Transaction Request

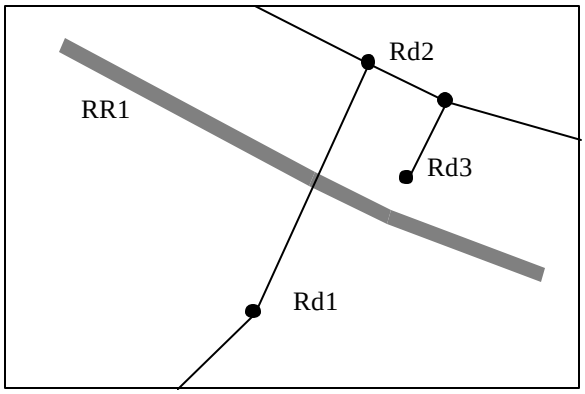
Sample outline of a WFS <Transaction> Request.

```
<?xml version="1.0" ?>
<wfs:Transaction version="1.0.0" service="WFS"
  xmlns=”http://www.roadnetwork.com/myns”
  xmlns:gml=”http://www.opengis.net/gml”
  xmlns:wfs=”http://www.opengis.net/wfs”
  xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
  xsi:schemaLocation=
    “http://www.openplans.org/myns
     http://bee:8080/geoserver/DescribeFeatureType?typeName=myns:localshapes
     http://www.opengis.net/wfs
     http://schemas.opengis.net/wfs/1.0.0/WFS-basic.xsd”
  <!-- delete Rd1/RR1 -->
  <wfs:Delete typeName="RailwayCrossing>
    <ogc:Filter><ogc:FeatureId fid="RailwayCrossing.1032"/></ogc:Filter>
  </wfs:Delete>
  <!-- delete Rd1/Rd2 -->
  <wfs:Delete typeName="RailwayCrossing">
    <ogc:Filter><ogc:FeatureId fid="RoadIntersection.457"/></ogc:Filter>
  </wfs:Delete>
  <wfs:Update typeName="RailRoad">  <!-- update RR1 -->
    <wfs:Property>
      <wfs:Name>the_geom</wfs:Name>
      <wfs:Value>
        <gml:Linestring gid=”RailRoad.47.the_geom”
          srsName=”http://www.opengis.net/gml/srs/epsg/xml#6269”>
         <gml:coordinates decimal=”.” cs=”,” ts=” “>
           1292907.3840915519, 450910.9250699766 ...
         </gml:coordinates>
        </gml:Linestring>
      </wfs:Value>
    </wfs:Property>
    <ogc:Filter><ogc:FeatureId fid="RailRoad.47"/></ogc:Filter>
  </wfs:Update>
  <!-- Insert Rd3 -->
  <wfs:Insert>
    <Road>
      <the_geom>
        <gml:Linestring gid="Road.1397.the_geom"
          srsName="http://www.opengis.net/gml/srs/epsg.xml#6269">
          <gml:coordinates>
            1292907.3840917452,450911.2453253453...
          </gml:coordinates>
        </gml:Linestring>
      </the_geom>
      <NAME>Dead End</NAME>
    </Road>
  </wfs:Insert>
  <!-- Update RR1 -->
  <!-- Update Rd2 -->
  <!-- Insert Rd2/RR1 RailwayCrossing -->
  <!-- Insert Rd2/RR1 RailwayCrossing -->
  <!-- Insert Rd2/Rd1 and Rd2/Rd3 RoadIntersection -->
</wfs:Transaction>
```

3.2 Transaction Operation with Validation

3.2.1 Reference Operation

To aid in describing the operation of the Validating web feature server will refer to the inconsistencies present the following diagram. While visually correct the following diagram represents a logically inconstant attempt to render the previous workflow.



Road.Rd1
- the_geom: linestring: 3 coordinates
- name: Sydney Ave
- speedlimit: 0 km/ph

Railroad.RR1
- the_geom: linestrin: 3 coordinates
- name: CN 4329

Figure 5 - Invalid Intersection

The above diagram is inconsistent in that:

- The road and railway feature types require an intersection
- The feature Road.Rd1 requires a valid speedlimit attribute.

3.2.2 Validating Web Feature Server Use Case

The use case for the validating web feature server is consistent with the WFS in the course of normal operation. In the presence of a transaction operation additional checks are performed.

The following use case is limited to the processing of the above invalid intersection update. The transaction is configured to allow "SOME" rather than "All" allowing partial success.

Validating Web Feature Server Transaction with Error Reporting

1. The client submits a <Transaction> request containing the following elements:
   An <Update> element modifying the Rd1 road to reflect the new route
   An <Insert> element defining the new Rd3 road.
   An <Update> element modifying the Rd2 road.
   An <Insert> element defining the Rd1/Rd2 and Rd3/Rd2 RoadIntersection entries
2. The VWFS processes the transaction
   2.1. A checkpoint is set on the Datasource
   2.2. The transaction elements are processed
      2.2.1. Road.Rd1 <Update> element
         2.2.1.1. Null Zeros feature test fails for the speedlimit attribute
      2.2.2. Road.Rd3 <Insert> element
         2.2.2.1. Null Zeros check passes
         2.2.2.2. Insert operation is performed by the Datasource
   2.3. Road.Rd2 <Update> element
      2.3.1. Null Zeros check passes
      2.3.2. The update operation is performed by the Datasource
   2.4. RoadIntersection <Insert> operation
      2.4.1. No feature test exist for RoadIntersection
      2.4.2. The insert operation is performed by the Datasource
   2.5. The VWFS checks performs Integrity checks
      2.5.1. The Intersection tests are performed
         2.5.1.1. The Railway and Road information is checked for intersection
         2.5.1.2. The pass succeeds (remember Road.Rd1 was not updated)
3. The WFS responds with a <Transaction> Document containing the following errors
   3.1. Success: Rd3 and Rd2 updates
   3.2. Failures: describes the Road.Rd1 speedlimit Null Zero failure
4. The client submits a <Transaction> with a the following:
   An <Update> element modifying the Rd1 road to reflect the new route and correct speedlimit
5. The VWFS processes the transaction
   5.1. A checkpoint is set on the Datasource
   5.2. Road.Rd1 <Update> element passes checks and the Datasource is Updated
   5.3. The VWFS checks performs Integrity checks
      5.3.1. The Intersection tests are performed
         5.3.1.1. The Railway and Road information is checked for intersection and fails as Road.Rd1 and Railway.RR1 intersect.
   5.4. The Datasource is rolled back to the previous checkpoint
6. The VWFS responds with a <Transaction> Document containing the following the failure
7. The client submits a <Transaction> with a the following:
   An <Update> element modifying the Rd1 road to reflect the corrected route and correct speedlimit
   An <Update> element with the changed Railway.RR1 geometry
8. The Validation checks pass and the operations are performed
9. A <Transaction> document is returned to the client

3.2.3 Transaction Document

The WFS specification allows the Transaction Document to report partial success, a facility we plan to use in the VWFS document to report multiple errors from a single transaction.

The Web Feature Server Specification allows us to:

- Describe features using Locator tags.
  Locator tags require features be marked using Handle tags by the request.
- Describe error message using the Message tag.

The reporting of warnings will be returned in XML comments in the resulting Transaction Document in addition to being available as log information.

```
<?xml version="1.0" ?>
<WFS_TransactionResponse version="1.0.0"
  xmlns:wfs="http://www.opengis.net/wfs"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://www.opengis.net/wfs ../wfs/1.0.0/WFS-transaction.xsd">
  <wfs:InsertResult handle="Reroute Road One">
    <ogc:FeatureId fid="Road.rd3"/>
  </wfs:InsertResult>
  <TransactionResult handle="Reroute Road One">
    <Status><PARTIAL/></Status>
    <Locator>Road One</Locator>
    <Message>Speed limit required</Message>
  </TransactionResult>
</WFS_TransactionResponse>
```

The above Transaction Document describes:

- Road.rd1 (known as “Road One”) failing a NullZeros check on speedlimit
- Road.rd2 being inserted

4 CONFIGURATION

Configuration of validation tests is an important usability issue for this project.

The following strategies are proposed for validation configuration:

- Provide specific guidelines for where configuration information is stored
- Provide User Interface support for generation of configuration information

4.1 GeoServer Configuration Files

GeoServer configuration is currently specified by a series of XML files:

- **GeoServer Configuration:**
  GeoServer stores its configuration and capabilities in a configuration.xml file.

```
configuration.xml
```

- **Feature Type Configuration:**
  Each FeatureType directory contains a `schema.xml` and an `info.xml` file.

  The `schema.xml` file describes geometry and attribute format information.

```
roads/schema.xml
railroads/schema.xml
```

  The `info.xml` file describes the connection information for data access.

```
roads/info.xml
railroads/info.xml
```

For more information on these files consult the GeoServer documentation.

4.2 Validating Web Feature Server Configuration Files

The VWFS proposes adding the following configuration files:

- **Plug-in Configuration:**
  XML files stored in a plug-in directory provide Plug-In configuration. Each file contains configuration information for a Plug-Ins that will perform a validation test. The file format is defined by the XML schema file `plugin.xsd` outlined in a separate document on Validation Language.

```
plugins/nullzero.xml
plugins/duplicatevertex.xml
plugins/intersection.xml
```

- **Test Suites Definition:**
  XML files stored in validation directory define test Suites. Each file contains configuration information for running validation tests. Test suites are defined by the XML Schema file `suite.xsd`.

```
validation/toponomy.xml
validation/roads.xml
```

More specific information about the format of these files is provided in the Validation Language Design Document.

4.3 Validating Web Feature Server Initialization Process

The Validating Web Feature server stores validation tests in two data structures according to test type. This section describes the initialization process by which Validation tests are constructed and organized for later retrieval.

4.3.1 Processing Plug-In Configuration Files

Each Plug-In Configuration File defines the following:

- A name identifying the Plug-In
- A classname of the Java Bean implementing the Plug-In
- Attributes used for configuration

This information will be stored in a lookup table referenced by plug in name.

```
public void addPlugIn( String name, String bean, String text, Map config ){
  PlugIn plugin = new PlugIn( name, bean, text, config );
  plugins.put( name, plugin );
}
```

The PlugIn API used above records this configuration information and provides instance creation and helper functions.

```
public class PlugIn {
  public PlugIn( String name, String bean, String description, Map config );
  public ValidationTest getInstance();
  public static ValidationTest configure( ValidationTest test, Map defn );
}
```

4.3.2 Processing Test Suite Definitions

For each test defined in the Test Suite file a test is constructed, initialized, and placed in the appropriate Lookup table.

1. The plugin element will be used to lookup the appropriate PlugIn information.

```
PlugIn plugin = (PlugIn) plugInLookup.get( plugin_element );
```

2. The information will be used to obtain a Java Bean validation test as described by the Plug-In configuration file.

```
ValidationTest test = plugin.getInstance();
```

3. The validation test will be configured with the test definition:

```
Map configure = arguments.copy();
configure.put("name", name );
configure.put("description", text);

PlugInUtilities.configure( test, configure );
```

4. The validation test will be placed in a lookup table based on its type:

```
String types[] = test.getFeatureTypes();

for( int i=0; i<types.length;i++){
  String featureType = types[i];
  if( test instanceof FeatureValidation ){
    featureLookup.put( featureType, test );
  }
  else {
    integrityLookup.put( featureType, test );
  }
}
```

4.4 Feature Validation Test Lookup

The featureLookup provides a mapping from FeatureType from to FeatureValidation tests.

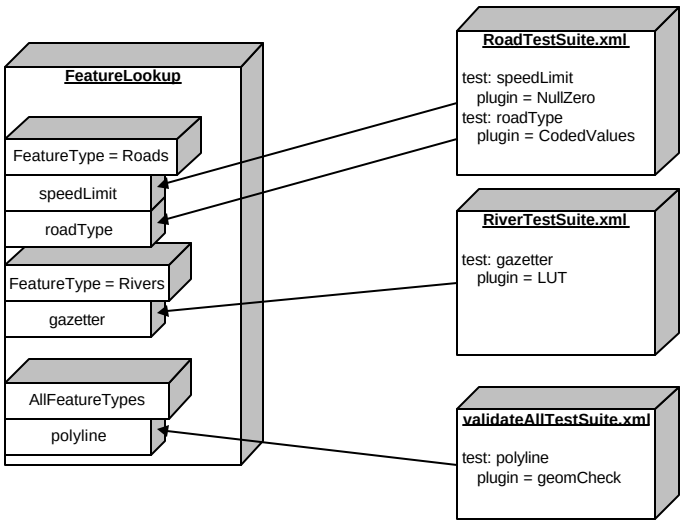


Figure 6 - featureLookup

A test suite can specify a placeholder "AllFeatureTypes" to describe test that should be run on every FeatureType.

4.5 Integrity Validation Test Lookup

The integrityLookup provides a mapping from FeatureType to IntegrityValidation tests.

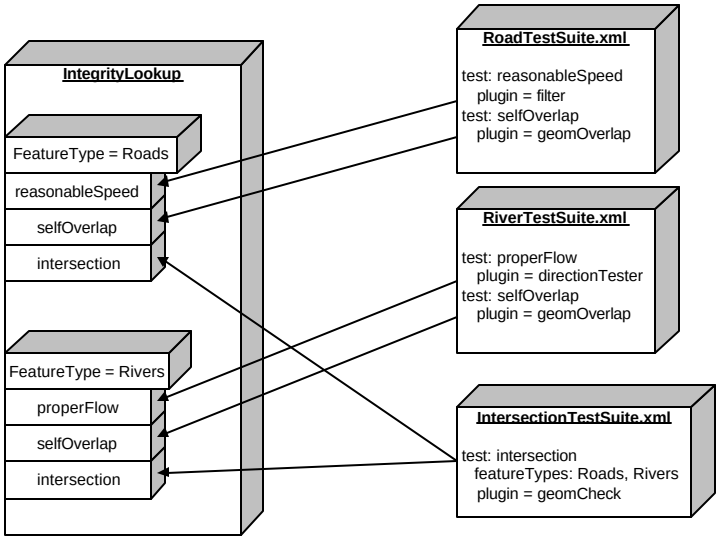


Figure 7 - IntegrityLookup

5 VALIDATION PROCESSING

The Validation Processor is used during the course of a Web Feature Server Transaction operation to execute validation tests.

5.1 The Validation Process

The validation process consists of:

1. The user requests a transaction
2. The Transaction is processed in order:
   - INSERT and UPDATE elements are validated
   - The modified area is recorded
3. Database integrity tests are performed on the modified area
4. A final validation result returned to the user

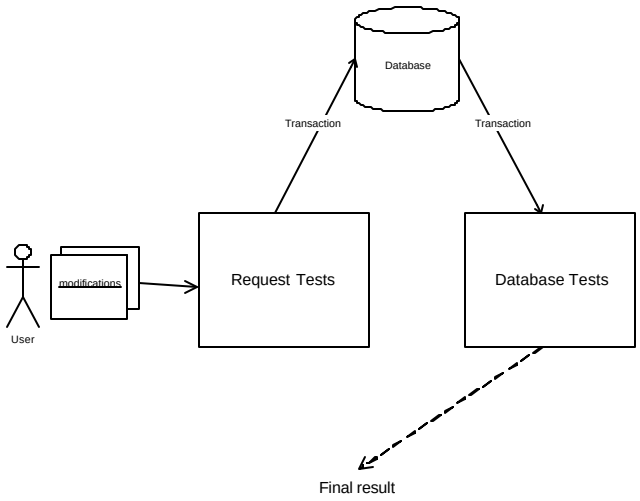


Figure 8 - Validation Process

The Transaction Document has three options for operation success.

- SUCCESS:
  The Transaction has been performed and the database is in a consistent state after modification
- FAILED:
  The Transaction has been rolled back, citing invalid modifications
- PARTIAL:
  The valid subset of the Transaction has been applied, and a partial result is returned citing invalid modifications

5.2 Processing Transaction Requests

Transaction operations are processed as a series of Insert, Update and Delete elements. Feature Validation Tests can be performed while processing Insert and Update elements.

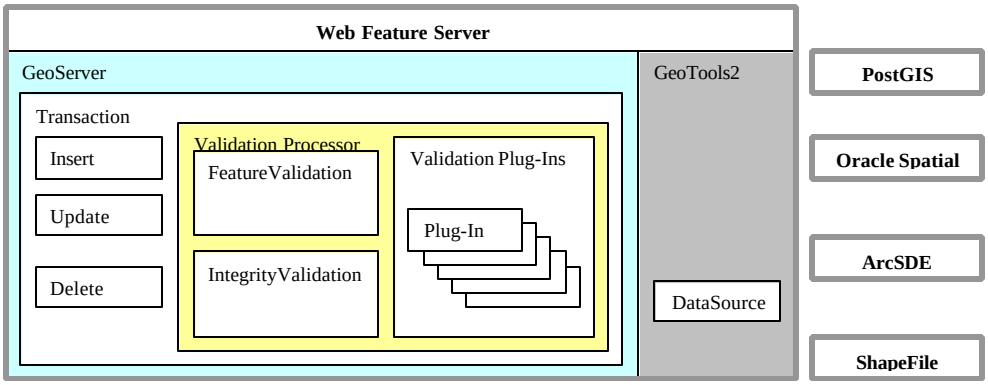


Figure 9 - Validation Processor

Integrity tests will need to be performed prior to committing the transaction. A Transaction Document can be created from the lists of failed tests.

5.2.1 Processing Insert Elements With Feature Validation

When processing an INSERT element, the data being passed in can be tested on the fly because the whole feature is available at that moment.

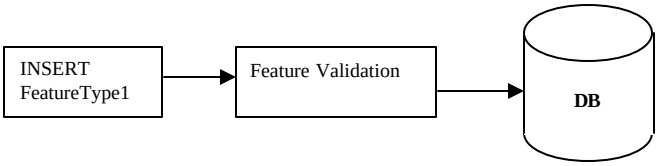


Figure 10 - Insert Feature Validation

The required FeatureValidation tests for are run as the data is passed in and the results are recorded either as a success or fail.

The bounding box of the modified Features should be recorded for future Integrity Validation tests.

5.2.2 Processing Update Element With Feature Validation

When processing an UPDATE, after the modification the relevant features are pulled from the database.

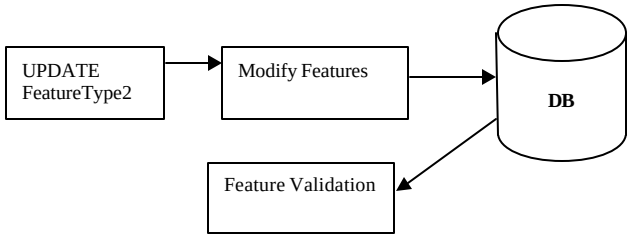


Figure 11 - Update Feature Validation

The required Feature validation tests are on the modified Features, and the results recorded as a success or fail.

The bounding box of the modified Features should be recorded for future Integrity Validation tests.

5.2.3 Processing Delete Elements

When processing a DELETE a bounding box must be queried from the database for use by future Integrity Validation tests.

The Delete operations also require a transaction to the database, but no FeatureType Tests are performed on Deletes.

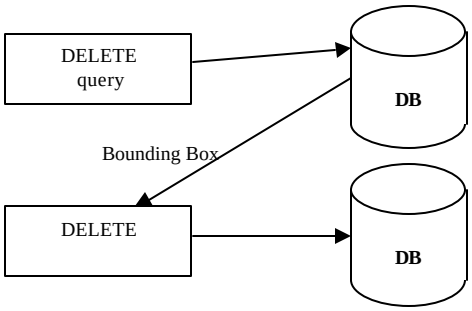


Figure 12 - Delete Feature Validation

5.2.4 Database Integrity Testing

Over the course of processing Transaction Elements the Validation Processor has recorded:

- The Modified Feature Types
- A Bounding box of the modified area for each Feature Type

A two step process is needed to figure out what information will be required form the database for Integrity Validation.

1. Build a list of Integrity Tests using integrityLookup
2. Cycle through the Integrity Tests recording required Feature Types

Using the complete list of required Feature Types and the modified area we can query the database for features needed to Integrity Validation Tests.

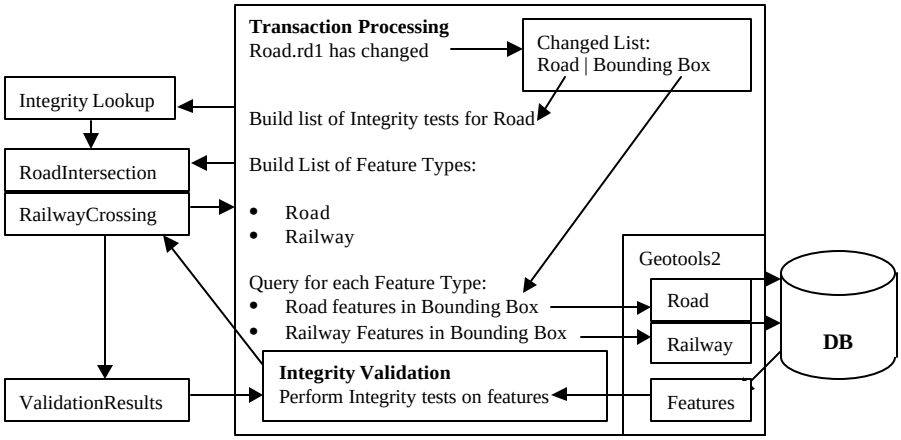


Figure 13 - Integrity Testing

Integrity Validation Tests are executed according to priority, as described in the Validation Plug-In API.

5.3 Validation Results

The results of the Validation process can be used to create a Transaction Document.

The Web Feature Server Implementation Specification defines:

Transaction Result	Definition
SUCCESS	The transaction was successfully completed.
FAILED	An exception was encountered while processing one or more elements contained in a Transaction request

The Validating Web Feature Server may use SUCCESS and FAILURE to return the results of the validation process.

5.3.1 Partial Success

The Web Feature Server Implementation Specification also supports the notion of PARTIAL success of a Transaction Operation.

Transaction Result	Definition
PARTIAL	The transaction partially succeeded and the data may be in an inconsistent state. For systems that do not support atomic transactions, this outcome is a distinct possibility

To support the partial success model, a more complicated validation processing mechanism is needed.

Briefly:

- One pass of the previous Validation Processing is attempted
- A list of Failures and Passes is for constructed
- We recursively try processing the request as follows:
  - If the list of Passes is empty we exit with complete failure
  - The Transaction Operations are redone excluding any features mentioned in the Failure list.
    (Feature Tests will no longer need to be reapplied)
  - The Integrity Tests are redone
  - If there are no more failures to be added we exit with a known list of Failures and Passes.

We may wish to limit the depth of recursion as a performance consideration, there is an opportunity to cache integrity tests by bounding box and limit duplicate work.

6 REFERENCES

Web Feature Server Implementation Specification, OCG 02-058.pdf