Open Geospatial Consortium

Publication Date: 2016-01-28

Approval Date: 2015-10-26

Posted Date: 2015-09-08

Reference number of this document: OGC 15-010r4

Reference URL for this document: http://www.opengis.net/doc/PER/tb-11-WFST_IEA

Category: Public Engineering Report

Editor: Panagiotis (Peter) A. Vretanos

OGC® Testbed-11 WFS-T Information Exchange Architecture

Warning

This document is not an OGC Standard. This document presents a discussion of technology issues considered in an initiative of the OGC Interoperability Program. This document does not represent an official position of the OGC. It is subject to change without notice and may not be referred to as an OGC Standard. However, the discussions in this document could very well lead to the definition of an OGC Standard. Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type: Public Engineering Report
Document subtype: NA
Document stage: Approved for public release
Document language: English

License Agreement

Contents

Page

Figures

Page

Tables

Page

Abstract

This document presents an assessment of the conformance level, with respect to the WFS standard (OGC 09-025r2), of the web feature servers used in the OGC Testbed-11. Each server is accessed to determine if it conforms to the minimum requirements of the WFS standard. Each server is further accessed to determine whether the server offers additional, upcoming and complimentary capabilities just as support for the WFS REST API and GeoJSON.

This document offers recommendations to aid implementers of the WFS standard (OGC 09-025r2).

This document presents options available to WFS implementers for achieving interoperability between WFS clients and server at the schemas level.

This document includes a survey of available WFS clients and an assessment of their capabilities.

This document reviews tools and standards, such as the GeoSynchronization Service (OGC 10-069r3), that are complimentary components that may be used with a WFS to address requirements such as verification and notification, data and access security, exception handling and system hardening.

Finally, this document includes a FAQ composed of questions raised during the OGC Testbed-11.

Business Value

For parties interested in implementing and/or deploying web feature servers, this document offers a survey of available web feature servers and client, implementation recommendations for achieving interoperability at the schemas level and a description of complimentary components that may be used with a web feature service to address additional requirements that are beyond the scope of the WFS standard (OGC 09-025r2).

Keywords

ogcdocs, testbed-11, WFS, WFS-T, transactions, REST, GSS, synchronization, geosynchronization, access control, schema, translation,

Testbed-11 WFS-T Information Exchange Architecture

1 Introduction

1.1 Scope

This purpose of this document is to analyze the current market situation with regard to available Web Feature Service (WFS) implementations from vendors and open source implementations that are participating in the OGC Testbed-11. The report covers the following aspects.

- The report identifies the differences and limitations of support and implementation of service standards (particularly WFS-T, REST, GML, and GeoJSON) between vendors;
- Provides a review of available WFS-T clients (or the lack thereof).
- This report offers recommendations to aid GIS vendors implementing support for these standards. The recommendations include detailed advice to avoid discrepancies between implementations of these standards.
- Review the capabilities of tools and standards such as GeoSynchronization Service (GSS) and others to provide additional engineering and workflow aspects needed to be addressed such as, verification and notification, data and access security, exception handling and system hardening before being robustly implementable.
- Provides a Q&A section of questions raised during the Testbed about the other topics discussed in this document.

1.2 Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

Name	Organization
Panagiotis (Peter) A. Vretanos	CubeWerx Inc.

1.3 Forward

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

1.4 Future work

- Clause 7.3.2 discusses schema translation between GML application schemas using XSLT. XSLT can also be used to support a transaction from GML to GeoJSON but not the other way around. A comparison should be made between this XSLT approach and the one in OWS10 using OWL which seems to offer a more general solution providing both schema translation and format translation.
- There is a need to investigate exception handling and system hardening for systems of WFSs.

2 References

The following documents are referenced in this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

OGC 02-069, Geography Markup Language

OGC 02-058, Web Feature Service

OGC 02-059, Filter Encoding

OGC 03-003r3, Basic XML Feature Schema

OGC 03-038, OGC Distributed Access Control System

OGC 03-105r1, OpenGIS Geography Markup Language (GML) Encoding Standard

OGC 04-094, OpenGIS Web Feature Service (WFS) Implementation Specification

OGC 04-095, OGC Filter Encoding Implementation Specification

OGC 06-121r3, OGC Web Services Common Standard

OGC 06-042, OpenGIS Web Map Service (WMS) Implementation Specification

OGC 07-036, OpenGIS Geography Markup Language (GML) Encoding Standard

OGC 09-025r2, OGC Web Feature Service 2.0 Interface Standard – With Corrigendum

OGC 09-026r2, OGC Filter Encoding 2.0 Encoding Standard – With Corrgendum

OGC 10-069r3, OWS-7 Engineering Report – Geosynchronization Service

OGC 10-100r3, Geography Markup Language (GML) simple features profile 2.0

OGC 11-080r1, A REST binding for WFS 2.0 (Change request)

OGC 11-117, Add service id field to service identification section

OGC 13-100, Geospatial Extensible Access Control Markup Language (GeoXACML)

OGC 14-102, OGC Web Feature Service 2.5 Interface Standard (pending)

OGC 14-103, OGC Filter Encoding 2.5 Encoding Standard (pending)

OGC 15-011, OGC Testbed-11 Multiple WFS-T Interoperability ER

OGC 15-022, Testbed 11 - Implementing Common Security Across the OGC Suite of Service Standards ER

OGC 15-052, OGC Testbed-11 REST Engineering Report

OGC 15-053, OGC Testbed-11 JSON/GeoJSON in OGC Stds ER

OGC 15-066, Testbed-11 Use of Semantic Linked Data with RDF for National Map NHD and Gazetteer Data Engineering Report

OGC 15-068r2, Testbed-11 GeoPackaging Engineering Report

3 Terms and definitions

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Standard [OGC 06-121r3] shall apply. In addition, the following terms and definitions apply.

3.1
attribute <XML>
name-value pair contained in an **element** (4.6)

[ISO 19136:2007, definition 4.1.3]

NOTE In this document an attribute is an XML attribute unless otherwise specified.

3.2
client
software component that can invoke an **operation** (4.17) from a **server** (4.28)

[ISO 19128:2005, definition 4.1]

3.3
coordinate
one of a sequence of n numbers designating the position of a point in n-dimensional space

[ISO 19111:2007, definition 4.5]

3.4
coordinate reference system
coordinate system (4.5) that is related to an object by a datum

[ISO 19111:2007, definition 4.8]

3.5
coordinate system
set of mathematical rules for specifying how **coordinates** (4.3) are to be assigned to points

[ISO 19111:2007, definition 4.10]

3.6
element <XML>
basic information item of an XML document containing child elements, **attributes** (4.1) and character data

[ISO 19136:2007, definition 4.1.23]

3.7
feature
abstraction of real world phenomena

[ISO 19101:2002, definition 4.11]

NOTE A feature can occur as a type or an instance. The term "feature type" or "feature instance" should be used when only one is meant.

3.8
feature identifier
identifier that uniquely designates a **feature** (4.7) instance

3.9
filter expression
predicate expression encoded using XML

[OGC 09-026r2, definition 4.11]

3.10
interface
named set of **operations** (4.17) that characterize the behavior of an entity

[ISO 19119:2005, definition 4.2]

3.11
join predicate
filter expression (4.9) that includes one or more clauses that constrain properties from two different entity types

[OGC 09-026r2, definition 4.16]

NOTE In this International Standard, the entity types will be **feature** (4.7) types.

3.12
Multipurpose Internet Mail Extensions (MIME) type
media type and subtype of data in the body of a message that designates the native representation (canonical form) of such data

[IETF RFC 2045:1996]

3.13
namespace <XML>
collection of names, identified by a URI reference which are used in XML documents as **element** (4.6) names and **attribute** (4.1) names

[W3C XML Namespaces:1999]

3.14
operation
specification of a transformation or query that an object may be called to execute

[ISO 19119:2005, definition 4.3]

3.15
property
facet or attribute of an object, referenced by a name

[OGC 09-026r2, definition 4.21]

3.16
resource
asset or means that fulfils a requirement

[OGC 09-026r2, definition 4.23]

NOTE In this International Standard, the resource is a **feature** (4.7), or any identifiable component of a feature (e.g. a property of a feature)

3.17
request
invocation of an **operation** (4.17) by a **client** (4.2)

[ISO 19128:2005, definition 4.10]

3.18
response
result of an **operation** (4.17) returned from a **server** (4.28) to a **client** (4.2)

[ISO 19128:2005, definition 4.11]

3.19
schema
formal description of a model

[ISO 19101:2002, definition 4.25]

NOTE In general, a schema is an abstract representation of an object's characteristics and relations to other objects. An XML schema represents the relationship between the **attributes** (4.1) and **elements** (4.6) of an XML object (for example, a document or a portion of a document).

3.20
schema <XML Schema>
collection of **schema** (4.26) components within the same target **namespace** (4.16)

[ISO 19136:2007, definition 4.1.54]

EXAMPLE Schema components of W3C XML Schema are types, **elements** (4.16), **attributes** (4.1), groups, etc.

3.21
server
particular instance of a **service** (4.29)

[ISO 19128:2005, definition 4.12]

3.22
service
distinct part of the functionality that is provided by an entity through **interfaces** (4.10)

[ISO 19119:2005, definition 4.1]

3.23
service metadata
metadata describing the **operations** (4.17) and geographic information available at a **server** (4.28)

[ISO 19128:2005, definition 4.14]

3.24
Uniform Resource Identifier
unique identifier for a resource, structured in conformance with IETF RFC 2396

[ISO 19136:2007, definition 4.1.65]

NOTE The general syntax is <scheme>::<scheme-specified-part>. The hierarchical syntax with a **namespace** (4.16) is <scheme>://<authority><path>?<query>

3.25
filter capabilities XML
metadata, encoded in XML, that describes which **predicates** defined in this International Standard a system implements

3.26
function
rule that associates each **element** from a domain (source, or domain of the function) to a unique element in another domain (target, co-domain, or range)

[ISO 19107:2003, definition 4.41]

3.27
predicate
set of computational **operations** applied to a data instance which evaluate to true or false

3.28
predicate expression
formal syntax for describing a **predicate**

3.29
base URL
HTTP GET URL for a server's OGC capabilities document without the GetCapabilities request parameters attached

NOTE: this base URL must match the HTTP GET base URL reported in the Capabilities document of the service

NOTE: the base URL is used to identify the server in lieu of a service id which is not currently define in OWS common but has been posted a change request to OGC (see OGC 11-117)

3.30
category document
documents that describe the categories allowed in Collection

3.31
change feed
collection of ATOM entries that describe changes to a data store expressed using the WFS Transaction syntax (see OGC 04-094)

3.32
collection
resource that contains a set of member resources

NOTE In this candidate standard, collection are implemented as ATOM feeds (see IETF 4287)

3.33
collector
a person or entity that proposes changes to data

3.34
entry resource
members of a collection that are represented as ATOM entry documents (see IETF RFC 4287)

3.35
event
any detectable or discernable occurrence that has significance for the management of an SDI

3.36
follower
person or process that accesses or subscribes to the replication feed of a GSS for the purpose of data synchronization

3.37
integrator
person or process that reviews proposed data changes and then makes a determination (based on established criteria) if the proposed change is acceptable or not

3.38
member resource
resource whose IRI is listed in a Collection with a atom:link element with a relation of "edit" or "edit-media"

3.39
publisher
synonym for collector (see X.X)

3.40
replication feed
collection of ATOM entries containing a log of changes that have been applied to a data store that can be used for the purpose of replicating or synchronizing with that data store

3.41
representation
entity included with a request or response (see IETF RFC 2616)

3.42
resolution feed
collection of ATOM entries describing the disposition of proposed changes listed in a change feed

3.43
reviewer
synonym for integrator (see 4.15)

3.44
service document
XML document that describes the location and capabilities of one or more Collections grouped into Workspaces

3.45
topic
collection of ATOM entries that satisfy some query predicates

NOTE: this is also referred to as a filtered feed because a topic is generated by querying a base feed and applying some predicate; for example a topic could consist of all the entries that lie within some defined boundary

3.46
workspace
named group of collections

4 Conventions

4.1 Abbreviated terms

Some more frequently used abbreviated terms:

API Application Program Interface

AtomPub ATOM Publishing Protocol

CGDI Canadian Geospatial Data Infrastructure

CGI Common Gateway Interface

COTS	Commercial Off The Shelf
CRS	Coordinate reference system
DCE	Distributed Computing Environment
DCOM	Distributed Component Object Model
DCP	Distributed Computing Platform
EPSG	European Petroleum Survey Group
FES	Filter Encoding Specification
GML	Geography Markup Language
GSS	GeoSynchronization Service
HTTP	Hypertext Transfer Protocol
HTTPS	Secure Hypertext Transfer Protocol
IDL	Interface Definition Language
IETF	Internet Engineering Task Force
KVP	Keyword-value pairs
MIME	Multipurpose Internet Mail Extensions
OGC	Open Geospatial Consortium
OWS	OGC Web Service
REST	Representational State Transfer
SDI	Spatial Data Infrastructure
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
UCR	Urban Climate Resilience thread
UML	Unified Modelling Language
URI	Uniform Resource Identifier

URL Uniform Resource Locator

URN Uniform Resource Name

VSP Vendor Specific Parameter

WFS Web Feature Service

WNS Web Notification Service

WSDL Web Services Description Language

XML Extensible Markup Language

5 WFS servers

5.1 Introduction

This clause presents a survey of WFS implementations used in the OGC Testbed-11. Both commercially off the shelf servers as well as open source servers are surveyed.

In addition to surveying support to the minimum requirements of the WFS standard this clause also looks at support for additional, upcoming and complimentary capabilities. For example, this clause describes if the server support transactions; if the server supports the Simple Feature Profile of GML; if the server offers GeoJSON (or any JSON format) as one of its output formats; if the server offers a RESTful API (even though that API has not as yet been standardized in the OGC).

5.2 Assessment criteria

The following set of capabilities was reviewed for each server:

- Versions of the WFS standard supported
- Operations supported
    - For servers supporting version 2.0 and below, the specific list of operations is presented.
    - For servers supporting version 2.5 and the REST binding, the supported HTTP methods are listed.
- Available output formats
    - Support for JSON and GeoJSON output formats are specifically noted
- Filtering capabilities

- Spatial operators supported
- Temporal operators supported
- Scalar operators supported
- Logical operators supported
- Additional functions supported

- Number of spatial reference system supported (i.e. can handle a variety of CRS')
- Schema support
  - Support for GML Simple Feature Profile (see OGC 10-100r3) is specifically noted
- Additional capabilities
  - WFS REST binding support
  - ATOM support
  - XSLT support / Schema translations capability
  - Binary data handling / multimedia support

5.3 Survey of WFS servers

5.3.1 List of servers

This clause presents a list of commercially and open source WFS implementations that participated in the UCR thread of the OGC Testbed-11 and analyses the capabilities of these servers using the criteria outlined in clause 5.2.

The following tables list the WFS implementations that participated in the UCR thread of the OGC Testbed-11 and their endpoints.

Table 1 - List of WFS server participating in the UCR thread

Vendor	Product	Supported WFS Versions
CubeWerx	CubeWerx Suite 8.1.1	2.5, 2.0, 1.1.0, 1.0.0
Geomatys	Constelation	2.0,
IBM Cloudant	RESTful JSON WFS	2.5

Luciad	LuciadFusion	2.0,1.1.0,1.0.0
OSGeo	Geoserver 2.7.1.1	2.0, 1.1.0, 1.0.0

Table 2 – List of server endpoints for Testbed-11

Vendor	Endpoint
CubeWerx	http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11
Geomatys	http://ows11.geomatys.com/constellation/WS/wfs/ows11/2.0.0
IBM/Cloudant	http://ogcwfs.mybluemix.net/wfs/2.5
Luciad	http://demo.luciad.com:8080/LuciadFusion/wfs?REQUEST=GetCapabilities&SERVICE=WFS
GIS-FCU/ OSGeo	internal server

5.3.2 Server capabilities

The following table lists the capabilities of each WFS server that participated in the OGC Testbed-11 UCR thread. This list was derived from the components page of the UCR Thread on the project wiki at:

https://portal.opengeospatial.org/wiki/Testbed11/UcrSoftwareComponents

Table 3 – Capabilities review of UCR servers

	CubeWerx	Geomatys	IBM Cloudant	Luciad	GIS-FCU/ Geoserver
Versions	2.5 2.0 1.1.0 1.0.0[2]	2.0 1.1.0	2.5	2.0 1.1.0[2] 1.0.0[2]	2.0 1.1.0 1.0.0
Operations	GetCapabilities DescribeFeatureType GetFeature ListStoredQueries DescribeStored	GetCapabilities DescribeFeatureType GetFeature ListStoredQueries DescribeStored	GET POST PUT DELETE	GetCapabilities DescribeFeatureType GetFeature	GetCapabilities DescribeFeatureType GetFeature ListStoredQueries DescribeStored

	Queries GetPropertyValue Transaction Sync (new of Testbned-11) GET PUT POST DELETE	Queries GetPropertyValue Transaction CreateStoredQuery DropStoredQuery GET POST PUT DELETE			Queries GetPropertyValue Transaction CreateStoredQuery DropStoredQuery LockFeature GetFeatureWithLock GET[5] PUT[5] POST[5] DELETE[5]
Output Formats	GML v3.2 GML v3.1.1 GML v2.1.2 GeoJSON KML SHAPE ATOM RSS HTML	GML v3.2 GML v3.1.1 GeoJSON	GeoJSON	GML v3.2 GML v3.1.1 GML v2.1.2 JSON	GML v3.2 GML v3.1.1 GML v2.1.2 GeoJSON KML SHAPE CSV
Spatial operators	Disjoint Equals Intersects Touches Crosses Contains Overlaps BBOX Within	Disjoint Equals DWithin Beyond Intersects Touches Crosses Contains Overlaps BBOX	BBOX	BBOX	Disjoint Equals DWithin Beyond Intersects Touches Crosses Contains Overlaps BBOX Within
Spatial operands	gml:Envelope gml:Point gml:LineString gml:Polygon gml:CircleByCenterPoint	gml:Envelope gml:Point gml:LineString gml:Polygon"		gml:Envelope gml:Point gml:LineString gml:Polygon"	gml:Envelope gml:Point gml:LineString gml:Polygon gml:MultiPoint gml:MultiLineString gml:MultiPolygon gml:MultiGeometry

Temporal Operators					After Before Begins BegunBy TContains During TEquals TOverlaps Meets OverlappedBy MetBy EndedBy
Temporal Operands					gml:TimeInstant gml:TimePeriod
Scalar operators	PropertyIsBetween PropertyIsEqualTo PropertyIsGreaterThan PropertyIsGreaterThanOrEqualTo PropertyIsLessThan PropertyIsLessThanOrEqualTo PropertyIsLike PropertyIsNotEqualTo PropertyIsNull	PropertyIsBetween PropertyIsEqualTo PropertyIsGreaterThan PropertyIsGreaterThanOrEqualTo PropertyIsLessThan PropertyIsLessThanOrEqualTo PropertyIsLike PropertyIsNotEqualTo PropertyIsNull		LessThan[3] GreaterThan LessThanEqualTo GreaterThanEqualTo EqualTo NotEqualTo Like Between NullCheck	PropertyIsBetween PropertyIsEqualTo PropertyIsGreaterThan PropertyIsGreaterThanOrEqualTo PropertyIsLessThan PropertyIsLessThanOrEqualTo PropertyIsLike PropertyIsNotEqualTo PropertyIsNull PropertyIsNil
Logical	And, Or, Not	And, Or, Not		And, Or, Not	And, Or, Not
Available Stored Queries	GetFeatureById NearestNeighbours	GetFeatureById GetFeatureByType			GetFeatureById
Number of CRSs	>10	<10	<10	<10	<10
Support for GML SF	Yes	No	No	Yes	No
REST API	Yes	Yes	Yes	No	Yes[5]
GeoJSON	Yes	Yes	Yes	Perhaps[4]	Yes
ATOM	Yes	No	No	No	No

XSLT vendor extension[1]	Yes	No	No	No	No

Note 1: This extension was included since it was anticipated that in the schema translation component of the Transactions Scenario in the UCR thread, XSLT would be used to transform the standard GML output in one schema from one server to conform to the schema of another server.

Note 2: These versions of the service have passed the OGC compliance test suite and a certificate of compliance has been issued by OGC.

Note 3: Although the server is claiming to be a WFS 2.0, these operator names are from WFS 1.1.0. The capabilities document actually validates with Xerces but that is because the schemaLocation attribute is referencing the WFS 1.1.0 schemas rather than the 2.0 schemas.

Note 4: Despite best efforts, the editor was unable to coax JSON out of the server and received only a blank response document. The other XML-based output formats seemed to work fine.

Note 5: Geoserver support a REST API but is it not base on the REST binding of WFS 2.5

5.3.3 Sanity checks

In order to access the status of each server participating in the UCR Thread, the following series of sanity checks were performed on each service:

1. Retrieve the server's capabilities document;
2. Retrieve the server's application schema;
3. Retrieve 1 feature from a feature type offered by the server without any filter
4. Retrieve 1 feature from the feature type used in (3) but include a bbox filter that contains the feature;
5. Create a feature of one of the feature types offered by the server;
6. Do a GetfeatureById to retrieve the feature created in (5) and verify that the feature was created correctly;
7. Update the feature created in 5 and change the value of one of its properties;
8. Do a GetfeatureById to retrieve the feature updated in (6) and verify that the properties' value was updated correctly;
9. Delete the feature created in step 5; and
10. Do a GetFeatureById on the feature created in step 5.

Tests 1 thru 4 were executed on all the servers and the results are presented in Table 3a.

Table 3a – Results of Sanity Checks 1 through 4

Vendor	TEST	URL
CubeWerx	1	http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11
	2	http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/schema
	3	http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/wwAccess?count=1
	4	http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/wwAccess?count=1&bbox=-44,-43,172,173
Geomatys	1	http://ows11.geomatys.com/constellation/WS/wfs/ows11/2.0.0
	2	http://ows11.geomatys.com/constellation/WS/wfs/ows11/2.0.0/schema
	3	http://ows11.geomatys.com/constellation/WS/wfs/ows11/2.0.0/GMLJP2ReferenceableGridCoverage?count=1
	4	http://ows11.geomatys.com/constellation/WS/wfs/ows11/2.0.0/GMLJP2ReferenceableGridCoverage?count=1&bbox=40,41,-4,-3
IBM Cloudant	1	http://ogcwfs.mybluemix.net/wfs/2.5
	2	http://ogcwfs.mybluemix.net/wfs/2.5/schema
	3	http://ogcwfs.mybluemix.net/wfs/2.5/highway?count=1
	4	http://ogcwfs.mybluemix.net/wfs/2.5/highway?count=1&BBOX=-123,-122,37,38
Luciad	1	http://demo.luciad.com:8080/LuciadFusion/wfs?REQUEST=GetCapabilities&SERVICE=WFS
	2	http://demo.luciad.com:8080/LuciadFusion/wfs?REQUEST=DescribeFeatureType&SERVICE=WFS&version=2.0.0
	3	http://demo.luciad.com:8080/LuciadFusion/wfs?service=WFS&version=2.0.0&request=GetFeature&&typeNames=buoya1316240__nav__buoybcnpType&count=1

	4	http://demo.luciad.com:8080/LuciadFusion/wfs?service=WFS&version=2.0.0&request=GetFeature&&typeNames=buoya1316240__nav__buoybcnpType&count=100&bbox=-121.0,-120.0,36.0,37.0
NOTES on fails (red) or partial success (yellow)		
Geomatys-3	Appending the feature name from the capabilities document fails but appending the feature title from the capabilities document works.	
IBM-2	Does not recognize the /schema path	
IBM-4	Using the keyword “bbox” fails. Using the keyword “BBOX” works.	
Luciad-3	The request actually produced good results. However, the server advertises that it supports version 2.0, the request is requesting version 2.0 but the response is a 1.1.0 response.	
Luciad-4	The request failed with a Tomcat stack trace.	

Editor’s Note:

Only the CubeWerx server and the IBM server were tested for Transactions. The CubeWerx server was tested with transactions to create features that included a multimedia property such as a photo using a mobile client. The IBM server was tested during the enterprise sync experiment since it was ingesting changes from the CubeWerx server and syncing its own WFS with those changes. The details of both experiments can be found in document OGC 15-011, Multiple WFS-T interoperability.

5.4 Implementation guidance

5.4.1 Introduction

This clause offers recommendations to aid implementers of WFS servers. The recommendations include detailed advice to avoid discrepancies between implementations of these standards.

5.4.2 Recommendations

5.4.2.1 Pass the OGC Compliance test suite

All servers should attempt to pass the OGC Compliance test suite for WFS – even if they do not intend to pay the fee to obtain a compliance certificate. Passing the conformance

test suite will ensure that a certain level of interoperability has been achieved and will also catch a number of the other discrepancies described in this clause. Further details about OGC compliance can be found here: http://www.opengeospatial.org/compliance.

5.4.2.2 Read the capabilities document

A very common impediment to WFS interoperability is assuming something about the server without actually reading its capabilities document. Common assumptions include: which CRS's are supported; what filters are available; what output formats are supported; etc. Clients should always commence their interaction with a WFS by reading and parsing the capabilities document to understand the capabilities of the server. The job of matching client and server capabilities is aided in the WFS 2.0 standard (see OGC 09-025r1) with the inclusion, in the capabilities document, of explicit statements that declare which conformance classes from the WFS standard a server implements.

5.4.2.3 Truth in advertising

The first step that a client **must** take in commencing an interaction with a WFS is to read its capabilities document in order to determine the server's capabilities. For this reason, it is critical that a server's capabilities document accurately reflects the actual capabilities of the server. Implementers should not simply copy the examples in the specification or from some other server but should carefully tailor the capabilities document to their server implementation.

Specifically, implementers should be mindful of the following when generating the server's capabilities document

- Accurately list the supported versions of the WFS standard in the service identification section.
- Ensure that the end point for each operation is accurately specified in the operations metadata section; otherwise clients will be using the wrong address to access your server.
- Pay particular attention that the conformance declarations in the operations metadata section are correctly listed. This is critically important because clients making requests to your server will generate their requests in large part based on the information provided here. For example, if the capabilities document of a server advertises that it supports spatial joins then the client will assume that queries to that server may include spatial join predicates.
- If any capacity constraints, such as CountDefault, are included in the operations metadata section ensure that accurate values are specified. It is not nice for your server to advertise a count default of 10 but then return all the features is result set – especially if the result set contains thousands of features.
- Ensure that the DefaultCRS and OtherCRS values listed for each feature type in the feature type list accurately reflect the actual storage CRS of your data and also accurately list the CRS' into which your server can project geographic data. A common error is for servers to list an inaccurate default CRS and a large number of other CRS' and yet only return geometric values in EPSG 4326 – regardless of

what the client requests specifies based on the information in your server's capabilities document.

- If at all possible, include WGS84BoundingBox values that closely bound the data of each feature type that your server offers. The multiplicity of the ows:WGS84BoundingBox element is unbounded so multiple bounding boxes may be used to accurately locate your feature data. Specifying a single bounding box that covers the entire surface of the Earth it typically useless information. If all that data of a particular feature type is concentrated in New Zealand, for example, then a tighter bounding box around that country would be much more useful not only for WFS clients but also for discovery via catalogues harvesting your server.
- It is critical that the filter capabilities section accurately lists the filter conformance classes, operators and operands that your server actually implements. This seems obvious and yet many servers specify that they support conformance classes and operators that they do not in fact support. Inaccurately advertising supported filter functions is a particularly common occurrence. For example, many WFS server are built on top of an RDBM such as Oracle. Such server typically list support for functions such as min(), max(), etc. in the filter capabilities section because the underlying database supports those function. However, we a WFS request that uses those function is sent to the server it often fails because the service is incorrectly translating the WFS request to a SQL request to the underlying RDBMS.

5.4.2.4 Rich metadata

Server implementations should endeavor to provide as much metadata about the service as possible in the capabilities document. The ServiceIdentification section should list all supported version; not just the one being requested. The ServiceProvider section should be as complete as possible. Liberal use should be made of the ows:Parameter and ows:Constraint elements in the ows:OperationsMetadata section to convey domain information such as the list of supported output formats. The more metadata the server can provide the better the interaction with the client will be.

Rich metadata in the capabilities document also makes the service more discoverable when harvested by a catalogue. Rich metadata in the capabilities document equates to more ways to find the service.

For good examples of what not to do, review the capabilities documents of the servers that participated in the Testbed (see Table 2).

5.4.2.5 Canonical GML versions

Ensure that the correct version of GML is implemented for the advertised WFS version. If the server supports multiple versions of the standard, then it will have to support multiple versions of GML since each version of the standard defines a canonical version

of GML. The following table lists the canonical GML version based on the version of the WFS standard:

Table 4 – Canonical GML versions

WFS Standard Version	Canonical GML Standard Version
1.0.0	2.1.2
1.1.0	3.1.1
2.0	3.2
2.5	3.2

5.4.2.6 Valid XML

GML is an XML vocabulary and as such is subject to all the rules of XML. It is critical that a server implementation generate correct and valid XML response documents in every instance where it generates XML. More specifically, the response to a GetFeature request must validate against the application schema the server offers; this is obtained from the server using the DescribeFeatureType operation.

An example of a common interoperability problem results from the fact that in XML element and attribute names are case sensitive. If the WFS's underlying data source supports case insensitive names but the schema it advertises uses case sensitive names then the server must do the necessary work to map the case insensitive names from the source data to case sensitive names in the response document.

5.4.2.7 Correct CRS

Servers must ensure that geometries in response documents are labelled with the correct CRS. Interoperability is severely hampered when response data is incorrectly labeled; it does not help a client when a server generates geometries in UTM-5 and labels the output as WFS84.

If no CRS labels are used, then geometries should be generated in the default CRS advertised in the server's capabilities document for the feature type being queried.

5.4.2.8 GML simple features profie

Although not a requirement of the WFS standard, server implementers should consider supporting the GML simple features profile (see OGC 10-100r3) as one of the output formats for a DescribeFeatureType request. See Clause 5.4.3 for further discussion on this topic.

The GML simple features profile defines three conformance levels, each of increasing complexity: level 0, level 1, and level 2. A best interoperability practice is to provide, if possible, multiple schema representations of the feature types that a server offers starting with GML simple feature profile, level 0. This can be accomplished by either offering multiple representations of the same feature types with different names from a single WFS end point (e.g. INWATERA_1M_LEVEL0, INWATERA_1M_LEVEL1 and INWATERA_1M using full, unrestricted, GML) or by providing different end points offering the same information but encoded using schemas of different complexities. To illustrate the point, consider the following sample end points:

- http://www.acme.com/gmlsf/level0/wfs
- http://www.acme.com/gmlsf/level1/wfs
- http://www.acme.com/wfs

In each case these servers offer the same list of feature types. The "…gmlsf/level0/…" server offers these feature types encoded using a schema that is compliant with GMLSF level 0. The "…gmlsf/level1/…" server offers these feature types encoded using a schema that is compliant with GMLSF level 1. Finally, the "…/wfs" server offer the features type encoded using an unrestricted GML application schema. Thus clients of various capabilities can usefully access your feature types. **NOTE:** It should be noted that the mapping feature types using restricted schema profiles such as GMLSF L0, L1 or L2 may result in a lossy mapping when compared to feature type encoded using an unrestricted GML application schema.

5.4.2.9 Implement only what you need

Be mindful of the fact that servers are not obliged to implement the entire WFS standard so a careful analysis of the requirements or goals of the server implementation will save a lot of un-necessary work.

5.4.3 WFS and GML

This section discussed the relationship between WFS and GML and provides some implementation guidance concerning supported output formats.

Although the WFS standard mandates the use GML, it does not restrict the availability of other output formats. As can be seen in Table 1, server implementations typically offer more than just GML as an output format.

The reason for mandating GML is to foster interoperability by providing some minimal level of capability. However, GML is a large and complex specification and may not be the best choice as a baseline for supporting interoperability – although at the time WFS was being developed it was the only feature encoding format available.

Over the years several attempts have been made to simplify the canonical output format that WFSs must support. The most notable of these efforts include:

a. Basic XML Feature Schema (BXFS) (see OGC 03-003r3) and
b. GML simple features profile (see OGC 10-100r3).

In the end, the GML simple feature profile (GMLSF) (see OGC 10-100r3) was adopted as an OGC Standard. It defines a very restricted subset of GML and goes so far as to proscribe exactly how application schemas should be encoded in GML (i.e., in a template-like fashion). The result is application schemas that can be readily parsed by clients thus making the interpretation of the output generated by a WFS that much easier. An additional benefit of GMLSF is the two independent vendors, implementing the same database as a GMLSF application schema should end up with similar if not semantically and syntactically identical schemas documents.

Although GMLSF does not offer the full range of GML capabilities it does define several levels of conformance that cover a wide range of requirements.

Thus, it is strongly recommended that server implementations support the GML simple features profile. This satisfies the WFS Standard's requirement for supporting GML but restricts the vocabulary to a manageable subset.

5.4.4 WFS and not GML

Although GML is the mandated canonical feature representation, the OGC Testbed-11 has shown that the WFS API can function quite successfully without any GML at all – this being especially true for the REST binding which is not strongly coupled to the specific feature representation. In the UCR thread, JSON and specifically GeoJSON, were used as the feature encoding format for the enterprise-to-enterprise synchronization scenario (see OGC 15-011) which involved two WFSs acting as clients for each other.

The implication for implementers is that they can implement their server to support an outputFormat suitable to their requirement without have to also incur the burden of implementing GML support. Of course, this will result in a non-compliant, but still usable WFS. The IBM-Cloudant WFS is just such an example; that server implements the WFS REST API and uses GeoJSON as its only feature encoding.

The use of GeoJSON in the enterprise-to-enterprise use case also signals that future versions of the WFS standard may decouple GML from the specification. So, while GML is currently required, future versions of the WFS specification may allow clients and servers to negotiate a mutually agreeable output format - in much the same manner as described in the WMS Standard (see OGC 06-042) – and that output format will not need to be GML. NOTE: A change request was posted to the OGC portal against the WFS standard requesting such a change but the request was tabled because it was posted late in the process and also involved a large number of complex changes in addition to the large number of complex changes already implemented in the standard to reach version 2.5.

5.4.5 WFS interoperability at the schema level

5.4.5.1 Introduction

This clause discusses the various options available for achieving interoperability at the schema level between clients and WFS servers. Schema interoperability in this context means that a client is able to read a server's schema and is able to syntactically interpret that schema sufficiently well to be able to formulate queries and perhaps transactions against that server.

No particular approach is advocated here, rather the various options are presented for achieving schema interoperability.

5.4.5.2 Community schemas

The technically simplest way to achieve WFS interoperability at the schema level is to have participating servers implement the identical application schema. While this might be feasible in specialized communities of interest such as hydrography or AIXM, in the real world this is general not the case. In such an environment clients are typically custom built to understand

5.4.5.3 Simple feature profile

The next level of WFS interoperability at the schemas level is to ensure that participating servers implement the GML simple features profile. The GMLSF specification is a restricted profile of GML that proscribes how spatial and non-spatial properties of features are encoded in application schemas offered by a WFS.

Using a GMLSF compliant schema allows for the possibility of implementing a dynamic client that can parse and syntactically interpret a previously unknown schema. Injecting semantic information into such an environment may also allow the client to perform semantic mapping between dissimilar schemas.

5.4.5.4 Rich client

Achieving schema interoperability with a rich client is similar to the approach taken in 5.4.5.3 except that the schema is an unrestricted GML application schema and the client has the ability to parse and interpret the full breadth of GML. Such a client would be exceedingly difficult to implement.

Editor's Note: I am not aware of any client that can generically and robustly handle a GML schema that includes elements from the full scope of GML. I only include this clause here for completeness.

5.4.5.5 Schema translation agent

The final level of WFS interoperability at the schema level is achieved by having some intermediate client or agent act as a schema translation server (see 7.3) that can take the

output from one WFS and map it to the schema of another WFS. Such an agent could be a general processing module that incorporates both syntactic and semantic elements in its translation. Another benefit of this approach is that the WFS clients and servers participating in an interaction would not need to be modified; any impedance mismatch would be resolved by the schema translation agent.

6 WFS clients

6.1 Introduction

This clause looks at available WFS clients with particular emphasis on WFS-T clients. The list of clients surveyed includes standalone commercial clients, open source clients and web-based frameworks that can be used to build browser-based WFS-T clients.

The intent was to survey each client for their specific capabilities however this was not possible due to limited time and resources. As such, the information provided in this clause can be considered a starting point for further investigation.

Coding examples are provided for some of the framework clients in order to try and convey the effort required to build web-based WFS clients (see Annex B).

6.2 Available WFS clients

Table 5 – COTS WFS Clients

Vendor	Produce	WFS-T	Web site
ESRI	ArcGIS	N	http://resources.arcgis.com/en/help/main/10.2/index.html#//00370000000p000000
Bentley	Bentley Map	N	http://www.bentley.com/en-US/Products/Bentley+Map/
Carbon	Gaia	Y	http://www.thecarbonproject.com/Products
Mapinfo	Mapinfo Pro	Y	http://www.mapinfo.com/product/mapinfo-professional/
Safe Software	FME	Y	http://www.safe.com

Table 6 – Open Source WFS Clients

Product	WFS_T	Web site
QGIS	Y	http://qgis.org/en/site/

uDig	Y	http://udig.refractions.net/
gvSig	Y	http://gvSig.org

Table 7 – Open Source Web Frameworks that include WFS support

Product	WFS-T	Web site	Tutorial
GeoExt	Y	http://geoext.org/index.html	http://workshops.boundlessgeo.com/geoext/wfs/wfst.html
Geotools	Y	http://www.geotools.org/	http://blogs.law.harvard.edu/jreyes/2007/08/03/geotools-wfs-t-update-request/
Leaflet	Y	http://leafletjs.com	http://blog.georepublic.info/2012/leaflet-example-with-wfs-t/
Openlayers	Y	http://openlayers.org	http://dev.openlayers.org/releases/OpenLayers-2.8/examples/wfs-t.html

6.3 Light testing

Some light tests we performed on the some of the components listed in Tables 5, 6, and 7. All the tests were performed using the CubeWerx server and the manhole cover feature type wwAccess.

The testing consisted of:

a. Connecting to the target WFS and seeing if the list of feature types offered by the client appeared in the component's catalogue;
b. Select a layer and render a small number of features;
c. Attempt to add a new feature;
d. Modify that feature; and
e. Delete the added feature to return the server to its original state.

The following table summarizes the results of those tests:

Table 8 – Light client testing results

	Test (a)	Test (b)	Test (c)	Test (d)	Test (e)
Gaia	Pass	Pass	Pass	Pass	Pass
QGIS	Pass	Pass	Pass	Pass	Pass
uDIG	Pass	Pass	Fail	Fail	Fail

7 Complimentary services or capabilities

7.1 Introduction

The clause reviews the capabilities of tools and standards such as GeoSynchronization Service (GSS) and others that provide additional engineering and workflow aspects needed to be addressed such as: verification and notification, data and access security, exception handling, and system hardening before being robustly implementable

7.2 Crowdsourcing (GeoSynchronization service)

7.2.1 Introduction

During the OGC Testbed-11 the following synchronization uses cases were defined and tested.

1. Geopackage-to-geopackage synchronization between two mobile clients that encounter each other in the field. (see OGC 15-068r2) .

2. Geopackage-to-WFS synchronization whereby a mobile client arrives at a control node and synchronizes its contents with an enterprise WFS (see OGC 15-068r2).

3. WFS-to-WFS at the enterprise level where a source WFS synchronizes with a target WFS (see OGC 15-011).

This clause describes how OGC's Geosynchronization service (GSS) can be used to mediate the synchronization workflows in use cases 2 and 3.

The GeoSynchronization Service standard (see OGC 10-069r3) (GSS) was developed within the OGC over several test beds previous to Testbed-11.

The purpose of the GSS is to support crowdsourced collection of data for OGC data services including WFS. The GSS is somewhat analogous to Open Street Maps but built using OGC technologies. A GSS sits between the crowd and a WFS mediating changes to that data in that server; in other words, members of the crowd do not have direct transactional access to the WFS and their changes must flow through the GSS for validation before being applied to the data.

7.2.2 Characteristics of a GSS

The main features of a GSS are:

- The standard assumes that some identity management and a roll based access control system is implemented by the GSS.
    - This means that members of the crowd must be registered as users of the GSS.
    - The GSS assumes that the following roles are defined.
        - Data Publisher: a member of the crowd that can log into the GSS and propose changes to be made to a source WFS.
        - Reviewer: a system user that has the authority to review proposed changes and decide on their disposition.
- Supports crowdsourcing with verification.
    - This means that the crowd can propose changes to a source WFS but those changes are not applied to the server until they have been validated.
    - The validation process can be a manual or automated process.
        - In either case the entity performing the validation must be assigned the role of “Reviewer.”
    - The validation process can also be NULL meaning that all changes are applied directly to the server.
    - The validation process is subject to identity and access control rules meaning, for example, that some uses may be privileged and their changes are accepted unverified while others members of the crowd go through the formal validation process.
- Supports push and pull notification of events via a subscription sub-system.
    - Events include the creations of a proposed change, the disposition of a proposed change (accepted or rejected), and application of an accepted change to the source WFS.
    - Subscriptions can be created on any of the base feeds (CHANGE, RESOLUTION, REPLICATION) and any topic created on those feeds.
    - When a subscription is created a handler is specified to indicate how the notification is to be delivered (e.g. email, SMS, etc.).

- Supports synchronization of a source WFS with one or more target WFSs.
    - The subscription sub-system includes a special event notification handler named “sync” which triggers synchronization between a source and target WFSs.
- Provides a mechanism whereby schema translation can be applied between the source WFS and the target WFSs during synchronization.

7.2.3 GSS components

7.2.3.1 Introduction

Figure 1 illustrates the components of a Geosynchronization service. A GSS is composed of a set of ATOM feeds, which are used to maintain the information the service uses to manage change workflows, and a service API that defines the operations of the service.

7.2.3.2 The feeds

The ATOM feeds are labeled the CHANGE feed, the RESOLUTION feed and the REPLICATION feed.

The CHANGE feed is where proposed changes are tracked. When a proposed change is created, it is stored in the change feed.

The RESOLUTION feed is where the disposition of proposed changes (i.e. accepted or rejected) is stored.

The REPLICATION feed is used to track accepted changes. Each change that is accepted and applied to the source WFS that the GSS is managing is stored in the REPLICATION feed.

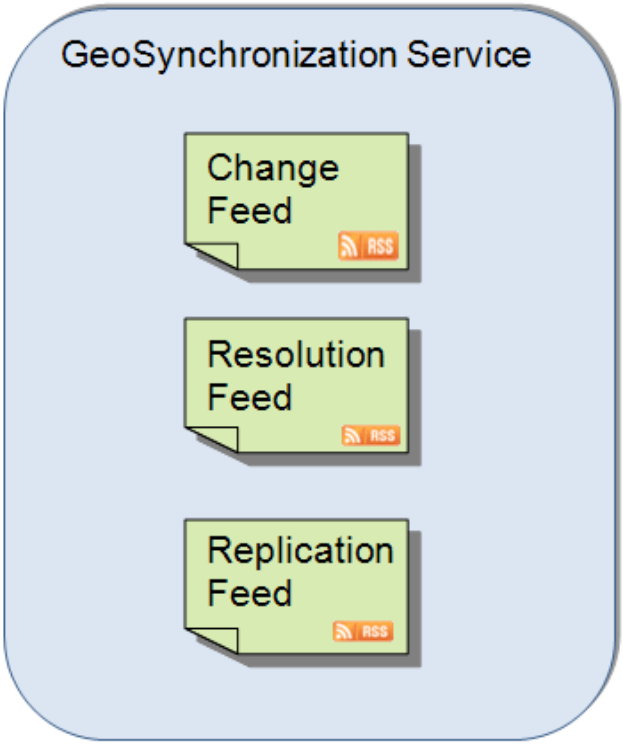

Figure 1 – Components of a GSS

The service interface is defined in Table 9. It lists the conformance classes and operations that the GSS standard defines.

Table 9 – GSS Operations

Conformance Class	API	Operation
Core	Discovery	GetCapabilities[1]
	Transaction	Insert, Update, Delete
	Query	GetEntries
	Topic Management	ListTopics
Extensions: Topics	Topic Management	CreateTopic
		RemoveTopic
Extension: Review	Change Management	AcceptChange

		RejectChange
		ReviewChanges
Extension: Active Notification	ActiveNotification	Subscribe
		ListSubscription
		PauseSubscription
		ResumeSubscription
		CancelSubscription
Extension: Active Synchronization	Synchronization[2]	Subscribe
NOTE 1: This class includes the GetCapabilities operation that generates an OGC capabilities document, the AtomPub service document accessible via a published URL and the OpenSearch description document also accessible via a published URL. NOTE 2: The active synchronization class is an implementation of the Subscribe operation, from the Active Notification class, with support for the "sync" delivery method.		

7.2.3.3 Core class

The Core class defines the basic operations that every GSS must implement. These operations provide service metadata about the GSS as well as define the basic operations required to query, input, update and delete entries from the various feeds.

The GSS API supports a full predicate language allowing the feeds to be queried using complex predicates including spatial and temporal operators. Topics are predefined stored queries on a feed and within the system, topics behave just like read-only feeds. The purpose of topics is the have persistent predefined views of a feed for the purpose of notification. Since topics are considered feeds, an interested party can subscribe to a topic and be notified whenever a new event satisfies the predicates used to define the topic. For example, consider an interested party that lives in the province of Quebec in Canada. Such a party could, using the boundary of the province of Quebec, define a topic with the title “Quebec Change Requests.”

7.2.3.4 Topics class

The Core conformance class supports the ability to read predefined system topics via the GetEntires operation. The operations in the Topics extension class add the ability to create and remove topics from the system subject the access control rules.

7.2.3.5 Review class

The Review class defines the operations that a reviewer needs in order to determine which new changes have been added to the system – so that they can be validated -- and then accept or reject those changes.

7.2.3.6 Active Notification class

Because the GSS uses ATOM feeds as its basic data structure, passive notification is supported right out of the box. Any user with a feed reader can subscribe to any of the feeds or topics the GSS offers and receive notifications whenever changes in the feeds or topics occur. The Active Notification class, however, defines an active subscription and notification subsystem within the GSS that pushes notifications out to subscribers using some delivery protocol (e.g. mailto) whenever events of interest occur. Using the Quebec example from clause 7.2.3.3, the user could subscribe to that topic and receive email notification whenever a change occurs within the boundary of the province of Quebec.

7.2.3.7 Active Synchronization class

Finally, the Active Synchronization class supports the synchronization of a source WFS with one or more subscribed target WFSs. As described in the previous paragraph, interested parties can subscribe to feeds or topics and request active notification delivery via some deliver method such as email. Other delivery methods might include SMS, Twitter, etc. Active Synchronization is simply another delivery method but it is special because the delivery mechanism used is the wfs:Transaction operation.

Active Synchronization is triggered when an interested party subscribes to the REPLICATION feed using the "sync" delivery method (as opposed, for example, to the mailto protocol). When the GSS is ready to notify a user who has subscribed with the "sync" delivery protocol, the GSS behaves like a WFS client and posts the wfs:Transaction contained in the event to the target WFS specified when the subscription was initially created. In this way, the GSS can synchronize one or more target WFSs with a source WFS.

User case 3 from the OGC Testbed-11, described in clause 7.2.1 can be implemented using the GSS's active synchronization capabilities.

Editor's Note: Although the GSS could have been used to support enterprise-to-enterprise synchronization a different approach was implemented in tested in Testbed-11 based on the replication protocol defined at http://replication.io. The details of this test can be found in the document, OGC 15-011 Reference Case Study of Multiple WFS-T Interoperability ER.

7.2.4 Crowdsourcing workflow

Figure 2 illustrates the entire change management and synchronization workflow of a GSS.

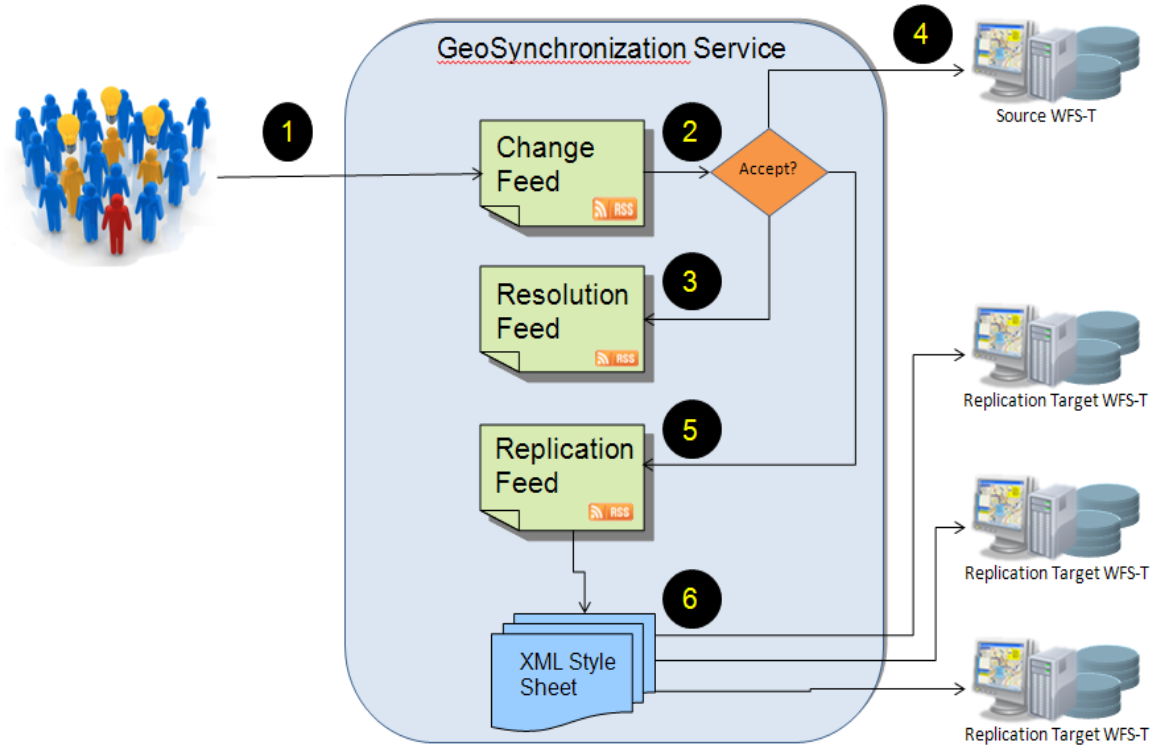

Figure 2 – GSS workflow

The flow of information through the GSS proceeds as follows.

1. The "crowd", on the left side of the figure, proposes changes to the data in the Source WFS-T which are posted to the CHANGEFEED.
2. A user with the role of the "reviewer" would be notified of the new change proposal in the CHANGEFEED and using a GSS client would review and validate the proposed change.
   In this discussion it is assumed that a manual validation process occurs but this could just also well be an automated process.
   Once the proposed change has been reviewed and validate a decision is made concerning its disposition.

3. Whatever that decision is: accept or reject, an entry is posted into the RESOLUTIONFEED. This allows the member of the crowd who posted the change to determine what happened with it. If it was rejected, the entry in the RESOLUTIONFEED can include a description of why it was rejected.
4. Assuming that in Step 2 the proposed change was accepted, the GSS would, acting as a WFS-T client, apply the change to the source WFS-T via the wfs:Transaction operation.
5. After the change has been applied to the source WFS-T, the transaction used to apply the change is posted to the REPLICATIONFEED. This allows interested parties to see that the change was applied and either actively or passively fetch the change and apply it to some target WFS-T thus keeping the two systems synchronized.
6. Assuming that one or more target WFS-T system have been subscribed to the GSS for active synchronization, the GSS would – once again acting as a WFS-T client – post the proposed change (already applied to the source WFS-T) to one or more subscribed target WFS-T systems. Figure 2 shows an XML Style Sheet at this point to illustrate that, if necessary, the transaction being posted to the target WFS-T servers can be modified to accommodate schema differences between the source and the target WFSs. In this case, the GSS is acting as a schema translation agent as described in clause 7.3.

7.3 Schema translation

7.3.1 Introduction

As discussed in 5.4.5, one approach to achieving WFS interoperability at the schema level is by means of an intermediate agent that can translate instance documents from the schema of a source WFS into a WFS transactions that can be executed on a target WFS with a different target schema. Figure 3 below illustrates this situation.

It is anticipated that the provisioning of the agent is flexible. The agent can be a stand-along component, part of a smart WFS client or even built into a WFS client.

7.3.2 Schema translation work flow

Figure 3 illustrates a schema translation work flow. Schema translation commences when a request to be executed on a source WFS, WFS 1, is passed to the schema translation agent with the intention of synchronizing the results with a second target WFS, WFS 2, that has a different schema. The workflow proceeds as follows.

1. The schema translation agent fetches the source schema from WFS 1 and notes its namespace.
2. The schema translation agent fetches the target schema from WFS 2 and notes its namespace.

3. The schema translation agent then accesses the schema registry to see if a translation script (e.g. XSLT) has been registered for the source and target schemas.
4. The schema translation agent then executes the WFS request on the source WFS, WFS1, and fetches the results. The features in the response are translated into the target schema using the script obtained from the registry.
5. A wfs:Transaction is created using the translated features and executed on the target server, WFS2.

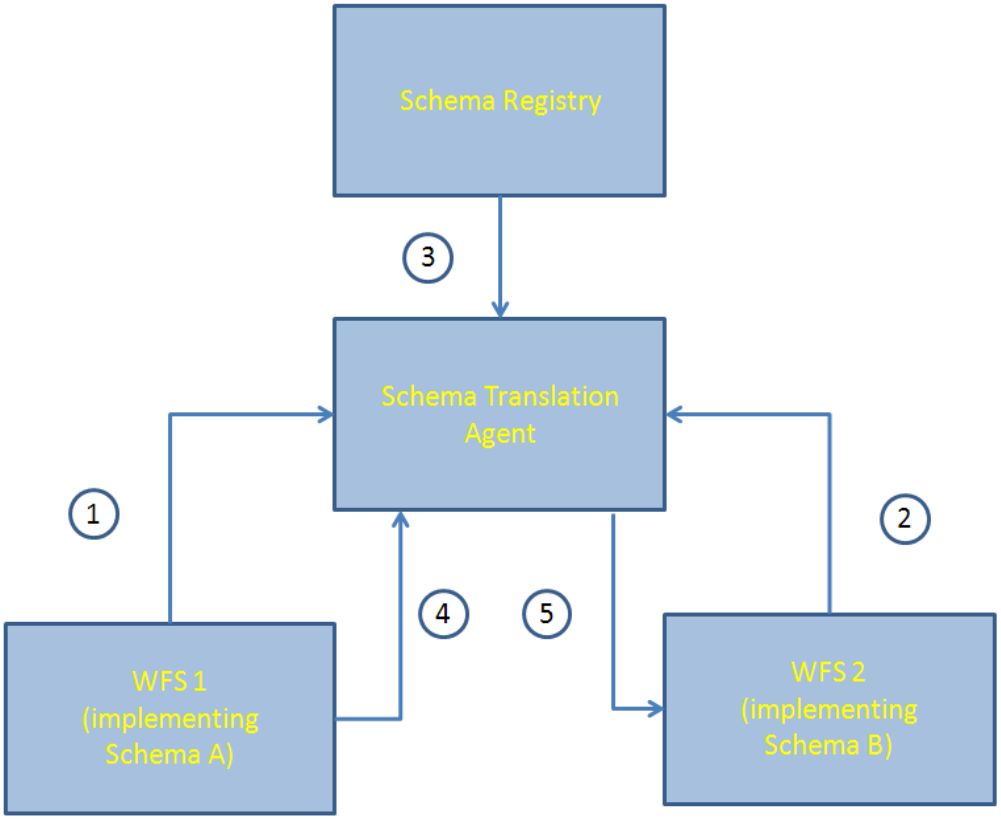

Figure 3 – Schema translation work flow

This description of the workflow assumes a non-REST WFS using GML as the feature encoding. However, the same flow would be possible with GeoJSON and REST – although without an XSLT equivalent more programming would be required in the schema translation agent to actually transform the source features into target features.

7.3.3 Schema registry

In the **UCR thread** a schema registry service was deployed based on the CSW-ebRIM 1.0.1 profile, at this endpoint: http://demo1.wrs.galdosinc.com/ows11/. The endpoint implemented the following capabilities:

- Schema registry

- Schema publishing
- Notifications of schema updates

Several extensions to the core information model were required to provide this functionality. A `Schema` object was defined as a new type of `ExtrinsicObject` (Fig. 4). The actual schema resource—that conforms to some specified schema language—is associated with the Schema object as a repository item. A Schema object belongs to a `SchemaPackage` that contains all schemas that reside in the same target namespace.

As an additional, support for registering schema mapping resources that may be used to transform instances of the **source** schema to instances of the **target** schema using a script (Fig. 5). A *SchemaMapping* is a type of `Association` (link) that relates the source and target schemas.

A mapping may be implemented by any number of `Script` resources (a type of ExtrinsicObject) that are written in some scripting language. The "scriptLanguage" slot identifies the scripting language. For example, XSLT is often used to transform XML documents; in this case the scripting language is denoted by the URI "http://www.w3.org/1999/XSL/Transform."

Predefined (stored) queries may be used to provide a simple means of querying and accessing registry content. We defined and implemented several of these as described in Table 5.

A pre-release version of a browser-based registry client application currently under development was also deployed for OGC Testbed-11. The client presents a graphical interface for the convenience of human users who wish to browse the registry; it is implemented using HTML5 and JavaScript, and should work in most current browsers.

Figure 4 -- Model extensions for schema management

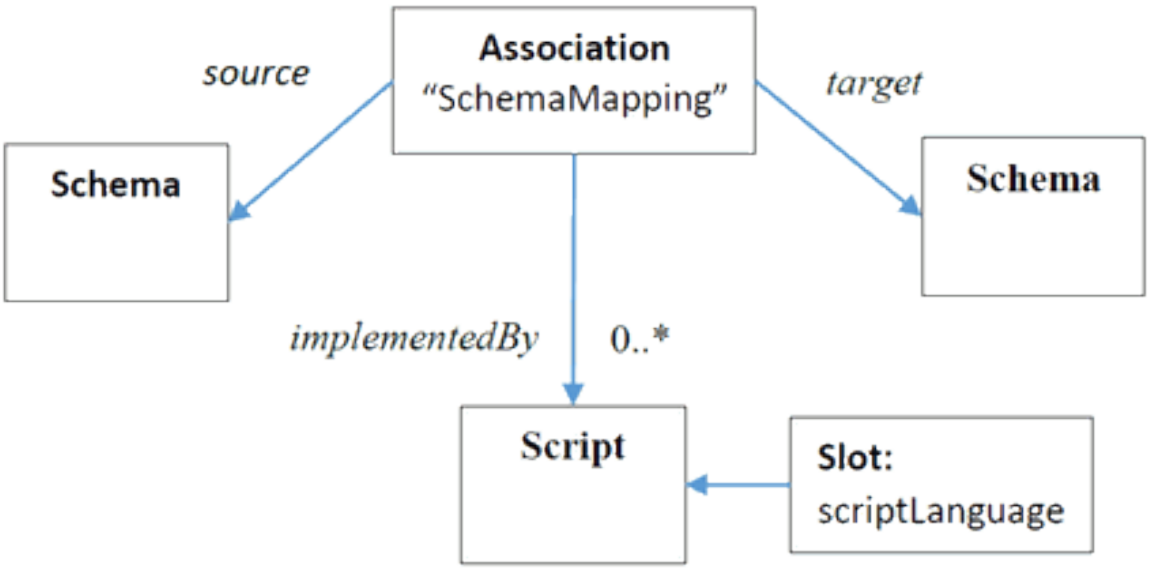

Figure 5 -- Schema mapping resources in the registry

Table 5 – Predefined registry queries

Stored query name	Description	Parameters
Find Source Schemas	Returns a list of Schema objects which are sourceObjects in a SchemaMapping Association	None
Find Target Schemas	Returns a list of Schema objects which are targetObjects in a SchemaMapping Association	None
Find Schema Mappings	Returns list of SchemaMapping Association objects	None
Find Schema Mappings by Script Type	Returns list of SchemaMapping Association objects for the specified Script type	scriptLanguage (required)

7.4 Data and access security

7.4.1 Introduction

The work done on security and access control in the OGC Testbed-11 are describe in document OGC 15-022, "Testbed 11 - Implementing Common Security Across the OGC Suite of Service Standards ER"

This clause shall describe some of the editor's thoughts concerning security and access control in relation to the WFS. It is anticipated that aspects of such as access control framework would be layered on top of a WFS allowing fine-grained operation and data security and incorporate rules based on spatial, temporal and non-spatial predicates.

7.4.2 Requirments

The framework should support the following requirements:

- Services can be configured to allow fine-grained access to operations and/or content to users based on the following types of credentials:
  - IP address
  - HTTP Basic Authentication
  - Vendor-specific authentication mechanism
- The security model should be based on an authentication server that has the following properties.

- It is secure. Credentials cannot be spoofed or manipulated. To accomplish this, a PGP public-key encryption mechanism may be employed. Credentials can only be authenticated and served by the Authentication Server, and all servers involved can verify the authenticity of a set of credentials by successfully decrypting them with the public key of a known (and trusted) Authentication Server. Furthermore, as long as all connection endpoints are HTTPS, malicious third parties cannot gain access to credentials in transit.

- It is flexible. It is compatible with the various standard distributed-application architectures, including "thin" browser-based clients, hybrid server-side web applications, and "thick" desktop application. It does this by communicating the credentials via an HTTP cookie and documenting how servers and desktop applications should intercept, interpret and propagate this cookie. It is also adaptable to various backend authentication mechanisms.

- It is role-based. In addition to indicating a username (i.e., a specific individual), a set of credentials can also indicate one or more project-defined roles. The access-control rules for a set of services can then be formulated based on these roles, providing a much more natural and flexible mechanism for access control.

- It is cascadable. In a service-chaining scenario (where, for example, a Web Map Server gets its data from a Web Feature Server), credentials can be passed down from service to service (as long as they all have the same second-level domain name) so that all entities along the chain are aware of the user's credentials and can control access at each level accordingly.

- It is single sign-on. A user that has logged on to (i.e., received authenticated credentials from) an Authentication Server within a particular domain can then access any server within that domain without having to log in again.

- It is simple. By employing the standard HTTP cookie mechanism and standard public-key encryption technology, the credentials mechanism is easy to understand, implement and configure.

- It is efficient. In most situations, once the user has logged in, the various entities never need to contact the authentication server. Validation of credentials is achieved solely by successfully decrypting the credentials cookie with the authentication server's public key (which has been configured beforehand).

- The authentication server should implement a flexible vocabulary for defining rules that support:

- Access control at the operation level;
- Access control at the feature type level;
- Access control based on geographic extent (exclusion and inclusion zones); and
- Access control based on temporal extent (exclusion and inclusion).

8 Q & A from the UCR thread of Testbed-11

8.1 Introduction

This clause is a FAQ for answering some of the questions posed in the UCR thread in relation to WFS, GSS, REST, etc.

8.2 How to handle replication/synchronization between enterprise DBs?

This question is answered in clause 7.2 of this document. The short answer is that whenever changes occur in a source WFS, the GSS acts as a WFS client and propagates those changes – possibly applying schema translation – to a target WFS.

8.3 WFS-T REST: what is the difference to traditional request/response?

Traditional OGC web services employ a request-response model based on an XML encoding of messages that are passed to a web service using the HTTP POST method (this includes SOAP) or a KVP encoding of messages that are passed to a web service using the HTTP GET method. The response is specific to the service but in the case of WFS, the canonical response is GML (see OGC 07-036) which is an XML vocabulary. Other responses are also possible but are not defined in the WFS standards.

From the response perspective, there is not much difference between a traditional OGC web service and a REST based web service. A RESTful web feature service still generates an XML container (wfs:FeatureCollection) full of features and a RESTful coverage service still responds with a coverage. This is currently done intentionally in the OGC in order to maintain backward compatibility with existing OGC services. However, with the advent of JSON, linked data and other modern web technologies, the response side will inevitably change. Several engineering reports in the OGC Testbed-11 are concerned with just these topics (see OGC 15-053, OGC 15-066).

Considering the request side, some of the disadvantages of the current approach include the following.

- Clients need to be intimately aware of the interface in order to interact with the service.
  - Unlike REST services where very simple clients such as a web browser can be used to navigate the service and obtain useful results.

- The interface from service to service is different requiring specialized clients be written from each service and significantly reducing the possibility of code reuse.
- The encoding of the resource is, in many cases, bound to the encoding of the request itself. For example, in order to insert, update or delete a feature using a non-REST WFS, an XML document containing a wfs:Transaction needs to be created and embedded within that are encoded the features to be manipulated. In other words, the interface and the resource are tightly coupled.
    - In contract, the REST architectural pattern has a common and consistent interface for all services, namely the HTTP methods GET, PUT, POST and DELETE. The interface and the resource are decoupled, allowing the same interface to service many different types of resources.
    - To illustrate the point, consider two objects: a feature and a coverage. Using the REST architectural style, creating a new feature or adding a new coverage to a repository is performed in a consistent, uniform way; the resource in question, feature or coverage, is sent to the client as the body of a POST message. A header in the message (i.e. Content-Type) is used identify the specific resource by its MIME type.

A more detailed discussion concerning REST service in OGC can found in the document OGC 15-052, Testbed-11 REST Engineering Report. Clause 7.4 of that document describes in more detail a REST binding for WFS.

8.4 Who needs a service using the REST architecture and what are the implications of using it?

The short answer is that all OGC services should be migrated to, or at least make available, a REST binding. A simple web search will uncover many justifications for using REST and the implications of doing so but Fielding summarizes the salient points here:

REST demands the use of hypertext, which scales very well since the client and server are very loosely coupled. With REST, the server is free to change the exposed resources at will. There is no fixed API above and beyond what REST itself defines. The client needs only know the initial URI, and subsequently chooses from server-supplied choices to navigate or perform actions. A server may download code to the client which aids in navigation and state representation.

All of this is in stark contrast with the various remote procedure call (RPC) schemes in which the client and server must agree upon a detailed protocol that typically needs to be compiled into both ends (e.g. URIs of a particular form accessed in a particular order at one extreme, SOAP/WSDL/WS at the other). This approach is brittle, because any changes need to implemented on both the server and client sides at the same time. It rapidly becomes untenable as the number of servers and/or clients grows. Servers in particular suffer because evolution of the published API becomes progressively more difficult as popularity increases.*

In light of these factors, REST is always the better choice when possible. It allows for rapid evolution of servers and allows an astronomical number of applications to interact freely on an ad hoc basis (e.g. the whole Internet).

But what about the "when possible" part? REST works best when there is a human in the loop. After all, a human has a good chance of being able to make a rational choice when presented with a previously unknown set of options. Machines aren't there yet. Web RPC protocols were born precisely to handcuff both sides to a fixed protocol. This makes it easier for automated processes to communicate when the human is removed from the picture. An RPC is a valid design choice when purely automated operation is more important than evolution and scalability (in Internet time and on an Internet scale).

Scale and Coupling?

"Scale" here is meant in a broad sense. It includes numbers of users and sessions, yes, but also application size and development process. Tight coupling presents a severe impediment to application size. It is hard to imagine the existence of the largest known application, the World-Wide Web, without the extremely loose coupling afforded by the REST architecture. Millions of developers around the globe have collaborated to build this application that supports billions of users. Yet the developers do this while remaining blissfully unaware of each other (or at least they would be unaware of each other if it weren't for StackOverflow ;).

The primary enabling principle of REST is hypertext. The other elements of the architecture exist to support that principle in very large scale (in every sense). Is REST the only conceivable way that the Web could have been built? No. But it happens to be the wildly successful de facto standard. It should be the default choice for any new entry into the ecosystem, discarded only after careful and explicit design consideration.

8.5 What about URL patterns

One of the “really neat” features of a REST architecture is that, as Fielding points out above, a client needs only know the initial URI, and subsequently chooses from server-supplied choices to navigate or perform actions. The use of URL patterns diminishes this benefit.

The following sequences of URIs -- with explanations -- illustrate the advantages of hypermedia controls and HATEOAS:

1. The initial URI is the WFS’ server root: http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11
2. Figure 6 shows a fragment of the capabilities document from (1). Each feature listed in the FeatureTypeList includes an ATOM link with rel=”collection.”

3. Resolving this link takes the client to the collection of features of this type and executes a default query: http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/wwAccess
4. Figure 7 shows that each feature in the response contains a set of ATOM links. The link with rel=”service” links back to the service that offers this features. The link with rel=”collection” links back to the collection (i.e. feature type) of which this feature is a member. There are also a number of links with rel=”alternate” that link to alternative representations of this feature. In this case, GeoJSON: http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/wwAccess/CWFID.WWACCESS.0.0.BA89DF77E5626F761F20020000?outputFormat=application%2Fvnd.geo%2Bjson

```
+ <ows:OperationsMetadata>
- <FeatureTypeList>
  + <FeatureType>
  + <FeatureType>
  - <FeatureType>
      <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="collection"
        href="http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/wwAccess" />
      <Name>cw:wwAccess</Name>
      <Title>wwAccess</Title>
      <DefaultCRS>urn:ogc:def:crs:EPSG::4326</DefaultCRS>
      <OtherCRS>urn:ogc:def:crs:EPSG::42110</OtherCRS>
      <OtherCRS>urn:ogc:def:crs:EPSG::3857</OtherCRS>
      <OtherCRS>urn:ogc:def:crs:EPSG::4267</OtherCRS>
      <OtherCRS>urn:ogc:def:crs:EPSG::4269</OtherCRS>
      <OtherCRS>urn:ogc:def:crs:EPSG::32758</OtherCRS>
```

Figure 6 – Capabilities document fragment

```
- <wfs:member>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="service,"
      href="http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11" />
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="collection,"
      href="http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/wwAccess" />
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" type="application/gml+xml; version=2.1"
      href="http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/wwAccess/CWFID.WWACCESS.0.0.BA89DF77E5626F761F20020000?
      outputFormat=application%2Fgml%2Bxml%3B%20version%3D2.1" />
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" type="application/gml+xml; version=3.1"
      href="http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/wwAccess/CWFID.WWACCESS.0.0.BA89DF77E5626F761F20020000?
      outputFormat=application%2Fgml%2Bxml%3B%20version%3D3.1" />
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" type="application/gml+xml; version=3.2"
      href="http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/wwAccess/CWFID.WWACCESS.0.0.BA89DF77E5626F761F20020000?
      outputFormat=application%2Fgml%2Bxml%3B%20version%3D3.2" />
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" type="application/x-bxfs+xml; version="0.0.3""
      href="http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/wwAccess/CWFID.WWACCESS.0.0.BA89DF77E5626F761F20020000?
      outputFormat=application%2Fx-bxfs%2Bxml%3B%20version%3D%220.0.3%22" />
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" type="application/rss+xml"
      href="http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/wwAccess/CWFID.WWACCESS.0.0.BA89DF77E5626F761F20020000?
      outputFormat=application%2Frss%2Bxml" />
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" type="application/vnd.google-earth.kml+xml"
      href="http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/wwAccess/CWFID.WWACCESS.0.0.BA89DF77E5626F761F20020000?
      outputFormat=application%2Fvnd.google-earth.kml%2Bxml" />
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" type="application/atom+xml"
      href="http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/wwAccess/CWFID.WWACCESS.0.0.BA89DF77E5626F761F20020000?
      outputFormat=application%2Fatom%2Bxml" />
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" type="text/html"
      href="http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/wwAccess/CWFID.WWACCESS.0.0.BA89DF77E5626F761F20020000?
      outputFormat=text%2Fhtml" />
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" type="application/vnd.geo+json"
      href="http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/wwAccess/CWFID.WWACCESS.0.0.BA89DF77E5626F761F20020000?
      outputFormat=application%2Fvnd.geo%2Bjson" />
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" type="application/vnd.shp+octet-stream"
      href="http://www.pvretano.com/cubewerx/cubeserv/default/wfs/2.5.0/ows11/wwAccess/CWFID.WWACCESS.0.0.BA89DF77E5626F761F20020000?
      outputFormat=application%2Fvnd.shp%2Boctet-stream" />
  - <wwAccess gml:id="CWFID.WWACCESS.0.0.BA89DF77E5626F761F20020000">
    - <Geometry>
      - <gml:Point gml:id="GID1" srsName="urn:ogc:def:crs:EPSG::4326">
          <gml:pos>-43.59451450970821 172.5609996851931</gml:pos>
        </gml:Point>
      </Geometry>
      <uid>WWMH-3044</uid>
      <AccessType>Standard Manhole</AccessType>
      <SourceId>2504S00145</SourceId>
      <LocationCe>Non Verified</LocationCe>
```

Figure 7 – GetFeature response showing hypermedia controls

8.6 REST principles such as "all you need is a mime-type" are not sufficient in geo domain. How to handle this?

This may be true for other OGC services but does not appear to be the case for web feature services.

8.7 How to use HTTP headers?

In general HTTP headers should be used as described in RFC 2616. The use of the common headers, Content-Type, Language, Accept, etc. is only now becoming well understood in the context of OGC service. Further experimentation and investigation, which was not part of the UCR thread, would be required to understand what the other headers might mean.

8.8 What are the implications of moving from XML to JSON/GoeJSON?

This topic is covered in document OGC 15-053, JSON/GeoJSON in OGC Standards ER.

8.9 How to use JSON with WFS 2.5?

In the WFS REST binding, a JSON/GeoJSON encoding is simply another representation of a feature that can be used to interact with the service. As long as the server advertises in its capabilities document that JSON/GeoJSON is an acceptable format/representation for features creating or modifying features using JSON/GeoJSON is simply a matter of POSTing or PUTing a JSON/GeoJSON-encoded feature to the appropriate feature URI. Retrieving a JSON/GeoJSON-encoded feature is simply a matter of appropriately setting the Accept header when GETing the feature via it URI.

Further details can be found in the document OGC 15-052, OGC Testbed-11 REST Engineering Report.

8.10 How can JSON be used with the GSS?

There is fundamentally no theoretical impediment to using JSON with GSS. However, there several issues that would need to be considered. These issues include the following.

- The use of JSON/GeoJSON to encode proposed changes within the context of the existing standard.
- The use of JSON/GeoJSON, rather than XML, to encode the feeds.
- The lack of validation tools for JSON.

Although not currently covered in the draft GSS specification, encoding changes as JSON/GeoJSON is easily supported in the GSS since the JSON/GeoJSON text can either be included in the ATOM entry as an escaped text string or as a CDATA section. Using JSON-encoded features would, among other things, require some careful capability coordination to ensure that any synchronized target WFSs support the REST and the JSON feature encoding.

GSS is based on ATOM and XML so all the GSS components that use ATOM and XML (i.e. change feed, resolution feed, replication feed, request encodings for the operations, etc.) would need to be mapped or translated to JSON. This is not a trivial task and would require and entirely new profile of GSS to be written that describes how just feed would be managed by the service. It should be noted that work being done in the CCI thread concerning XML to JSON translation may help should in such an endeavor.

Annex A
REST binding for WFS

The content of this annex can be found at this URL:
https://portal.opengeospatial.org/wiki/pub/Testbed11/CciSysArchRest/11-080r1.pdf

Openlayers WFS-T Client example

The following code fragment illustrates how to code a simple WFS-T client using the Openlayers framework.

This code fragment was copied from http://demo.boundlessgeo.com.

```
var map, wfs;
OpenLayers.ProxyHost = "proxy.cgi?url=";

var DeleteFeature = OpenLayers.Class(OpenLayers.Control, {
    initialize: function(layer, options) {
        OpenLayers.Control.prototype.initialize.apply(this, [options]);
        this.layer = layer;
        this.handler = new OpenLayers.Handler.Feature(
            this, layer, {click: this.clickFeature}
        );
    },
    clickFeature: function(feature) {
        // if feature doesn't have a fid, destroy it
        if(feature.fid == undefined) {
            this.layer.destroyFeatures([feature]);
        } else {
            feature.state = OpenLayers.State.DELETE;
            this.layer.events.triggerEvent("afterfeaturemodified",
                                           {feature: feature});
            feature.renderIntent = "select";
            this.layer.drawFeature(feature);
        }
    },
    setMap: function(map) {
        this.handler.setMap(map);
        OpenLayers.Control.prototype.setMap.apply(this, arguments);
    },
    CLASS_NAME: "OpenLayers.Control.DeleteFeature"
});

function init() {
    var extent = new OpenLayers.Bounds(
        -11593508, 5509847, -11505759, 5557774
    );


    map = new OpenLayers.Map('map', {
        projection: new OpenLayers.Projection("EPSG:900913"),
        displayProjection: new OpenLayers.Projection("EPSG:4326"),
        restrictedExtent: extent,
        controls: [
            new OpenLayers.Control.PanZoom(),
            new OpenLayers.Control.Navigation()
        ]
    });
```

```
    var gphy = new OpenLayers.Layer.Google(
        "Google Physical",
        {type: google.maps.MapTypeId.PHYSICAL, sphericalMercator: true}
    );

    var saveStrategy = new OpenLayers.Strategy.Save();

    wfs = new OpenLayers.Layer.Vector("Editable Features", {
        strategies: [new OpenLayers.Strategy.BBOX(), saveStrategy],
        projection: new OpenLayers.Projection("EPSG:4326"),
        protocol: new OpenLayers.Protocol.WFS({
            version: "1.1.0",
            srsName: "EPSG:4326",
            url: "http://demo.boundlessgeo.com/geoserver/wfs",
            featureNS :  "http://opengeo.org",
            featureType: "restricted",
            geometryName: "the_geom",
            schema:
"http://demo.boundlessgeo.com/geoserver/wfs/DescribeFeatureType?version
=1.1.0&typename=og:restricted"
        })
    });

    map.addLayers([gphy, wfs]);

    var panel = new OpenLayers.Control.Panel({
        displayClass: 'customEditingToolbar',
        allowDepress: true
    });

    var draw = new OpenLayers.Control.DrawFeature(
        wfs, OpenLayers.Handler.Polygon,
        {
            title: "Draw Feature",
            displayClass: "olControlDrawFeaturePolygon",
            multi: true
        }
    );

    var edit = new OpenLayers.Control.ModifyFeature(wfs, {
        title: "Modify Feature",
        displayClass: "olControlModifyFeature"
    });

    var del = new DeleteFeature(wfs, {title: "Delete Feature"});

    var save = new OpenLayers.Control.Button({
        title: "Save Changes",
        trigger: function() {
            if(edit.feature) {
                edit.selectControl.unselectAll();
            }
            saveStrategy.save();
        },
        displayClass: "olControlSaveFeatures"
```



```
    });

    panel.addControls([save, del, edit, draw]);
    map.addControl(panel);
    map.zoomToExtent(extent, true);
}
```