Open Geospatial Consortium

Publication Date: 2014-02-24

Submission Date: 2013-09-05

Reference number of this document: OGC 13-053r1

Reference URL for this document: http://www.opengis.net/doc/PER/chisp1-er

Category: Engineering Report

Editor(s): Panagiotis (Peter) A. Vretanos

OGC® CHISP-1 Engineering Report

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

License Agreement

Contents

Page

OGC® CHISP-1 Engineering Report

1. Introduction

1.1 Scope

This document provides a technical description of the work completed for the Climatology-Hydrology Information Sharing Pilot, Phase 1 project.

This document describes a profile of SOS, the NRCan GIN SOS 2.0 profile, developed in order to define a baseline of interoperability among the sensor observation services used in the project.

This document describes the use cases used to drive the component development during the project. The first use case was a flood scenario that involved exchanging cross-border hydrologic data with a unified alert service. The second use case involved calculating nutrient loads to the Great Lakes, which also involved the cross-border exchange of analytic data.

This document describes each component developed during the project and the challenges encountered and overcome during the development. The newly developed components include a nutrient load calculation client, a SOS integrating water quality data form the U.S. and Canada, a nutrient load calculation service, an upstream gauge service, a subscription client, and an event notification service composed of a number of sub-components including a subscription broker, an observation harvester and a CAP alert client.

1.2 Document contributor contact points

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

Name	Organization	Email Address
Dave Blodgett	USGS	dblodgett [at] usgs.gov
Eric Boisvert	NRCan	Eric.Boisvert [at] RNCan-NRCan.gc.ca
Nate Booth	USGS	Nlbooth [at] usgs.gov
Denis Boutin	NRCan	Denis.Boutin [at] RNCan-NRCan.gc.ca
Jean-Francois Bourgon	NRCan	Jean-Francois.Bourgon [at] RNCan-NRCan.gc.ca
Boyan Brodaric	NRCan	Boyan.Brodaric [at] NRCan-RNCan.gc.ca
欣永 Buck	GIS-FCU	Buck [at] gis.tw
Spencer Cox	Explorus	spencer.cox [at] explorus.org
Alex Crosby	ASA	acrosby [at] asascience.com
Laura DeCicco	USGS	ldecicco [at] usgs.gov
振宇 How	GIS-FCU	How [at] gis.tw
Alex Joseph	Explorus	alex.joseph [at] explorus.org
Tom Kralidis	EC	Tom.Kralidis [at] ec.gc.ca
Lew Leinenweber	OGC	lleinenweber [at] opengeospatial.org
友華 Orange	GIS-FCU	Orange [at] gis.tw
Yves Richard	Explorus	yves.richard [at] explorus.org
育縉 Tericky	GIS-FCU	Tricky [at] gis.tw
袁琿 Thomas	GIS-FCU	Thomas [at] gis.tw
Panagiotis (Peter) A. Vretanos	CubeWerx Inc.	pvretano [at] cubewerx.com

1.3 Future work

1.3.1 Introduction

This clause identifies work items that might be considered for future initiatives.

1.3.2 Catalogue

This project used the ISO profile of the OGC Catalogue specification. Such a catalogue is specifically designed to maintain metadata about services and data and their relationships. During the CHISP-1 project however, it was clear that other objects and relationships needed to be catalogued. For example, metadata about gauge stations and their relationship to the hydrographic network needed to be maintains and accessed and this was not easily handled in an ISO based catalogue. A better choice would have been an ebRIM based catalogue which includes a rich set of structures for cataloguing objects of all kinds including classifying objects and maintaining arbitrary relationships between those objects.

1.3.3 Big data handling

During this project an SOS profile was developed which, among other things, was designed to compensate for shortcomings in the SOS standard related to handling large networks of sensors. For example, in situations where large networks of sensors are made accessible via SOS, managing the capabilities document of these services becomes cumbersome because the content section can become quite large. A future work item would be to enhance the SOS standard to handle large networks of sensors. This would include work to enhance the GetCapabilities operation to allow large content sections to be accessed more efficiently – perhaps employing paging or some simple query capability to limit the number of items appearing in the response at one time.

1.3.4 Semantic mediation

The nutrient load calculation use case illustrated a need that commonly arises in cross-border projects which is the need for semantic mediation of information. An example of this was the need to mediate analyte names. During the CHISP-1 projects a SPARQL server was deployed to investigate its use in this mediation role. However, the server was never populated and so a future work item would be to complete the integration of the SPARQL server into the system.

1.3.5 Service performance

The CHISP-1 project deployed a large number of services that interacted with each other. Some of these services were not stable resulting in frequent service outages and connection problems which required robust exception handling. This was particularly true for components that operated on a periodic basis such as the Harvester module (see 6.3.4.1). Another example in clause 5.3.4.4 describes a performance issues with the upstream WPS that required modifications to how notification actions were executed. A future work item would be to (a) determine why the underlying services were having performance and stability issues and (b) consider more fault tolerant system designs.

1.3.6 Subscription client enhancements

The following enhancements could be considered for the web-based subscription client:

- Include a search box to allow geo-search by name (e.g. user enters “Milk River” and map zooms to Milk river area)

- The current set of data sources presented on the map is fixed and displayed in a legend in the upper right. Allow sources to be dynamically discovered and added to the map and make the legend dynamic to reflect which sources are currently being displayed.

- Suggestion capability – when creating a subscription it would be useful if the web-based subscription client could access historical information to suggest values for the various input parameters.

1.3.7 Event Notification System enhancements

The CHISP-1 project has only considered subscription and notification via a web-based browser client and email. However, there are many other standards that might be useful in real world situations. Some examples include GeoSMS (see OGC 11-030r1), that can send notification via SMS and can include all the spatial information that a subscriber needs to know; GeoPackage (see OGC 12-128r1), a draft standard that allows mobile applications to describe and store spatial information locally in a user's mobile devices; Geosync. (see OGC 10-069r1), that allows users to sync in-situ information back to emergency management centres for further integration.

1.3.8 Nutrient Load Calculation Model enhancements

Because of issues with data availability and sparseness on the Canadian side, a simple and ultimately non-scientifically correct model was used in the CHISP-1 project to compute nutrient load. This approach – while yielding invalid results -- allowed us to show that OGC services could be used to (a) provide the data to drive the model and (b) make the model available as a service via WPS. A future work item would be to enhance the model to account for the sparseness of observation and thus generate scientifically valid results.

1.4 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.

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 04-094, OpenGIS Web Feature Service (WFS) Implementation Specification, Version 1.1.0

OGC 05-005r1, OpenGIS Web Processing Service, Version 1.0

OGC 06-042, OpenGIS Web Map Service (WMS) Implementation Specification, Version 1.3.0

OGC 06-095r1, OpenGIS Web Notification Service, Version 0.0.9

OGC 07-006r1, OpenGIS Catalogue Services Specification, Version 2.0.2

OGC 07-045, OpenGIS Catalogue Services Specification 2.0.2 – ISO Metadata Application Profile, Version 1.0.0

OGC 10-126r2, OGC WaterML 2.0: Part 1 – Timeseries

OGC 12-006, OGC Sensor Observation Service Interface Standard, Version 2.0

Common Alerting Protocol, Version 1.2, (http://docs.oasis-open.org/emergency/cap/v1.2/CAP-v1.2.html)

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**

[ISO 19136:2007]

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

3.2
client
software component that can invoke an **operation** from a **server**

[ISO 19128:2005]

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

[ISO 19111:2007]

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

[ISO 19111:2007]

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

[ISO 19111:2007]

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

[ISO 19136:2007]

3.7
feature
abstraction of real world phenomena

[ISO 19101:2002]

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** instance

3.9
filter expression
predicate expression encoded using XML

[ISO 19143]

3.10
harvester
a module of the event notification system responsible for monitoring the last observed value of water monitoring stations

3.11
Harvest operation
an operation defined by the CSW standard that may be used to automatically register resources (e.g. services) with the catalogue

3.12
interface
named set of **operations** that characterize the behaviour of an entity

[ISO 19119:2005]

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

[W3C XML Namespaces]

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

[ISO 19119:2005]

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

[ISO 19143]

3.16
request
invocation of an **operation** by a **client**

[ISO 19128:2005]

3.17
response
result of an **operation** returned from a **server** to a **client**

[ISO 19128:2005]

3.18
server
particular instance of a **service**

[ISO 19128:2005]

3.19
service
distinct part of the functionality that is provided by an entity through **interfaces**

[ISO 19119:2005]

3.20
service metadata
metadata describing the **operations** and geographic information available at a **server**

[ISO 19128:2005]

3.21
Uniform Resource Identifier
unique identifier for a resource, structured in conformance with IETF RFC 2396

[ISO 19136:2007]

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

3. Conventions

3.1 Abbreviated terms

AOI	Area of interest
API	Application Program Interface
BBOX	Bounding box
CHISP	Climatology-Hydrology Information Sharing Pilot
CRS	Coordinate Reference System
EC	Environment Canada
EMA	Emergency Management Analyse
ER	Engineering Report
FES	Filter Encoding Standard
GDA	GetDataAvailability operation
GML	Geography Markup Language
HTTP	Hypertext Transfer Protocol
KVP	Keyword-Value Pair
NLCS	Nutrient Load Calculation Service
NRCan	Natural Resources Canada
SOS	Sensor Observation Service
UML	Unified Modelling Language
USGS	United States Geological Survey
WFS	Web Feature Service
WPS	Web Processing Service
WQA	Water Quality Analyst
WML	Water Markup Language
WSGI	Web Server Gateway Interface
XML	Extensible Markup Language

3.2 UML notation

Most diagrams that appear in this standard are presented using the Unified Modelling Language (UML) static structure diagram, as described in Sub-clause 5.2 of [OGC 06-121r3].

4. NRCan profile of SOS

4.1 Introduction

This clause describes a profile of SOS 2.0 (see OGC 12-006) developed by NRCan and used in the CHISP-1 project as an interoperable baseline of capabilities.

The profile is named *"The NRCan Profile of SOS 2.0 for CHISP-1"* but is typically referred to as the "*NRCan profile*" or the *"NRCan profile of SOS"* in this document.

All SOS's deployed in the CHISP-1 project were implemented or were modified to conform to this profile.

4.2 UML model

Figure 1 illustrates the UML model that describes the NRCan profile. The main features of the profile as implemented for the CHISP-1 project are:

1. The service shall implement the GetCapabilities, DescribeSensor, GetObservations, GetFeatureOfInterest and GetDataAvailability operations.

2. The service shall support the spatial filtering profile.

3. The service shall support the BBOX operator for the KVP binding (see http://www.opengis.net/spec/SOS/2.0/req/kvp-core/go-bbox-encoding, requirement 116).

4. The service shall implement WaterML 2.0.

5. For the GetDataAvailability operation, the "offering" parameter shall be mandatory.

6. The feature of interest shall be a SamplingFeature.

7. Extracting the last observation from a time series shall be performed using a GetObservation request (see 4.3.3)

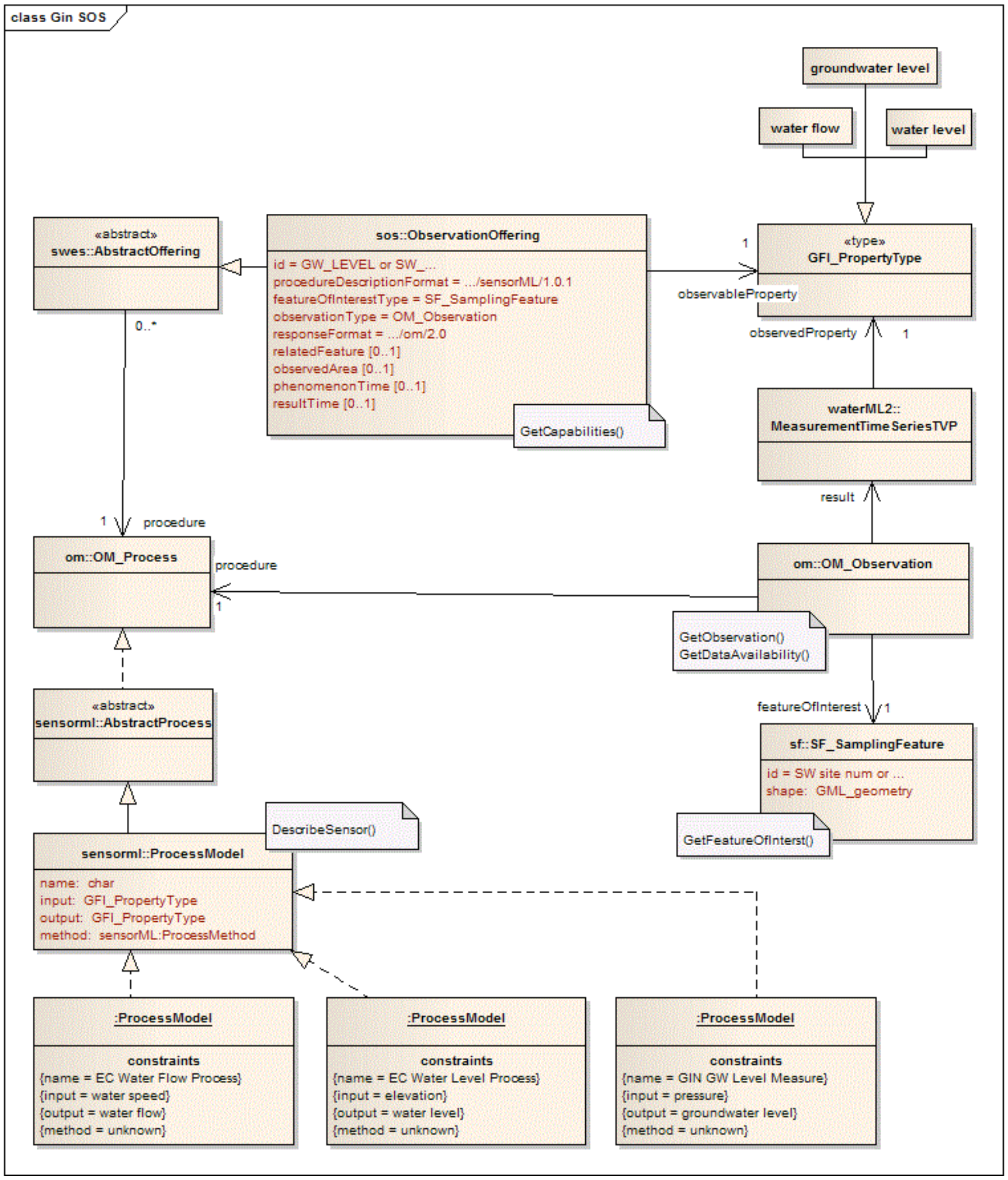


Figure 1 – UML Model for the NRCan SOS 2.0 Profile for CHISP-1

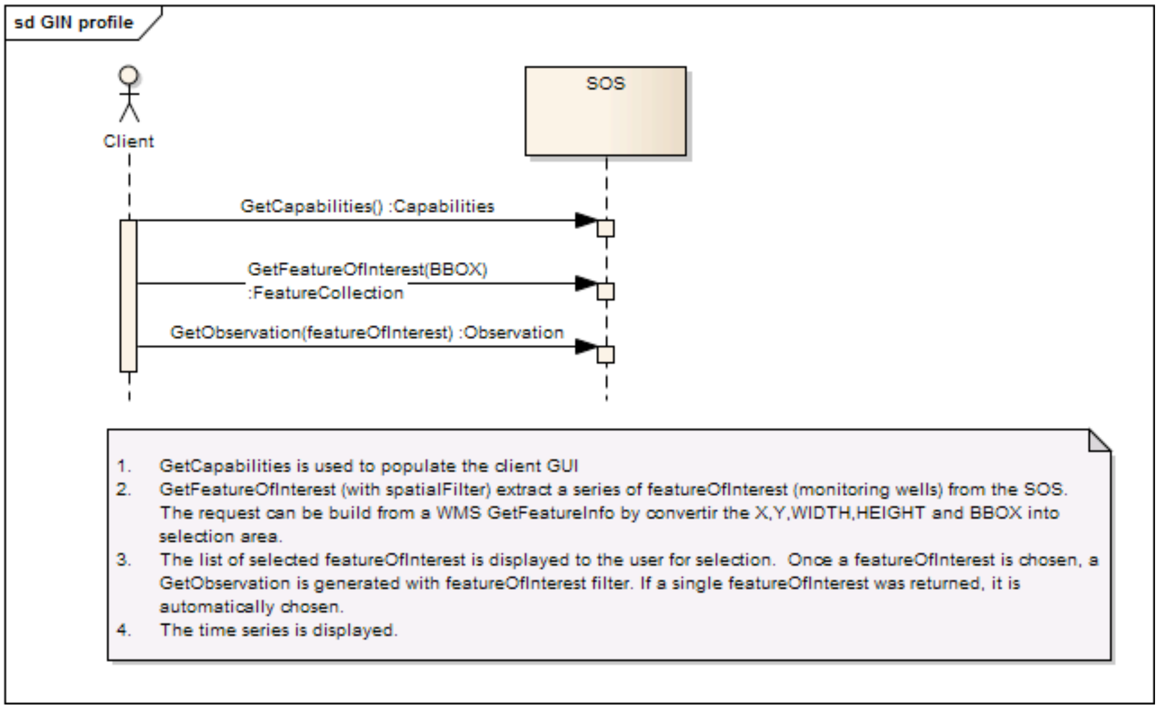
4.3 Sequence diagrams

4.3.1 Introduction

The following sequence diagrams illustrates how to obtain a time series for a specific feature of interest and also illustrates the Harvester algorithm (see 4.3.3) used to obtain the last observed value for a procedure, which is loaded into the system catalogue.

4.3.2 Retrieving a time series

The following sequence diagram (see Figure 2) illustrates how to retrieve a time series from an SOS that is compliant with the NRCan profile.


sd GIN profile
Client
SOS
GetCapabilities() :Capabilities
GetFeatureOfInterest(BBOX) :FeatureCollection
GetObservation(featureOfInterest) :Observation

1. GetCapabilities is used to populate the client GUI
2. GetFeatureOfInterest (with spatialFilter) extract a series of featureOfInterest (monitoring wells) from the SOS. The request can be build from a WMS GetFeatureInfo by convertir the X,Y,WIDTH,HEIGHT and BBOX into selection area.
3. The list of selected featureOfInterest is displayed to the user for selection. Once a featureOfInterest is chosen, a GetObservation is generated with featureOfInterest filter. If a single featureOfInterest was returned, it is automatically chosen.
4. The time series is displayed.

Figure 2 – Sequence diagram for fetching a time series

4.3.3 Harvester algorithm

The following sequence diagram illustrates the harvesting algorithm that is used to populate the catalogue with the last observed value of an observable.

The metadata that needs to be gathered by the harvester and stored in the catalogue includes the URL of the sensor observation service, the feature of interest, the observable, its last value and the timestamp.

The harvesting algorithm (see Figure 3) proceeds as follows:

- The harvester module uses the GetCapabilities request to determine the available observables and their extents (observedArea).
- Using a GetFeatureOfInterest request, the harvester gets all the features of interest (using a BBOX) over the entire AOI.
- For each returned feature of interest, the GetDataAvailability request is used to determine the parameter values necessary to formulate a GetObservation request to read the last value.
- The GetObservation request is executed to extract the last value.
- The last value is stored in the catalogue along with the URL of the service, the FOI, the observable, the last value and a timestamp.

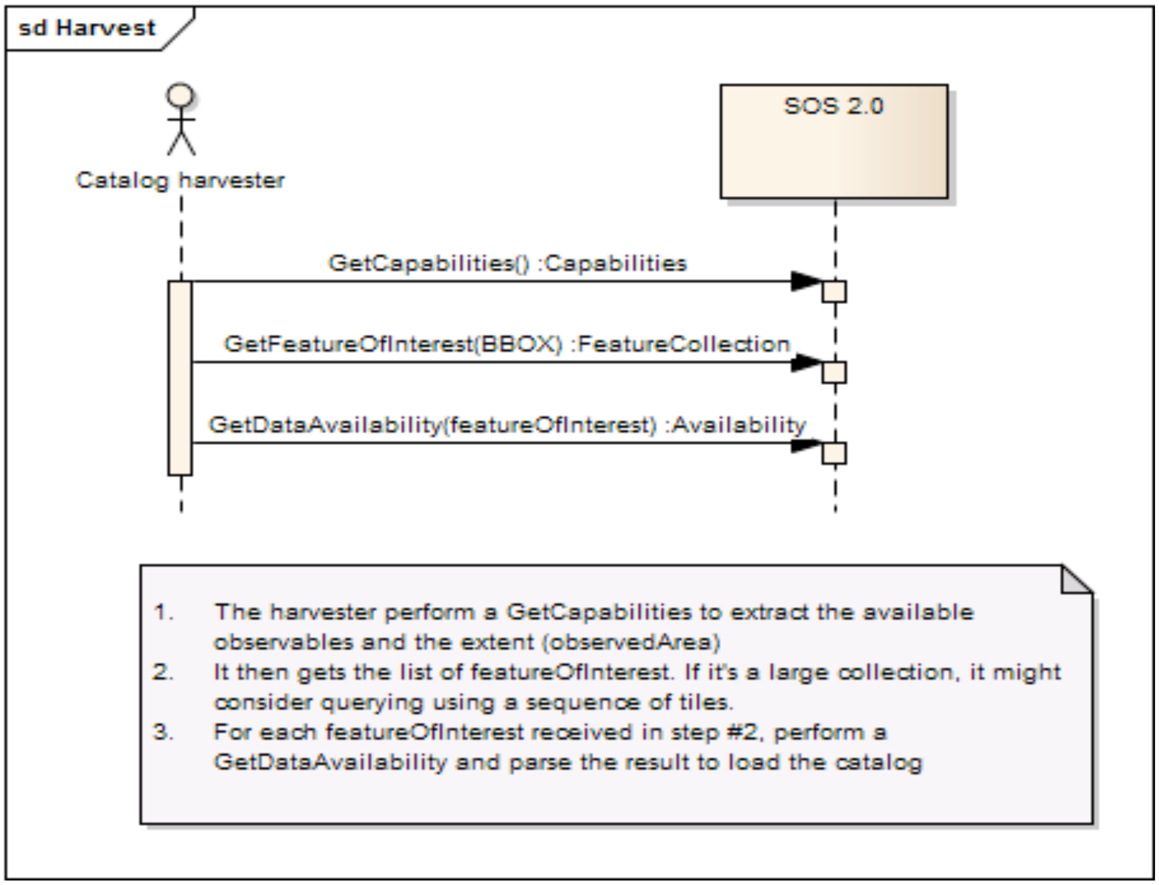

sd Harvest
Catalog harvester
SOS 2.0
GetCapabilities() :Capabilities
GetFeatureOfInterest(BBOX) :FeatureCollection
GetDataAvailability(featureOfInterest) :Availability
1. The harvester perform a GetCapabilities to extract the available observables and the extent (observedArea)
2. It then gets the list of featureOfInterest. If it's a large collection, it might consider querying using a sequence of tiles.
3. For each featureOfInterest received in step #2, perform a GetDataAvailability and parse the result to load the catalog

Figure 3 – Sequence diagram for the harvester algorithm

It should be noted that the AOI can be segregated into sub-regions and the entire last value harvesting method can be parallelized. That is, multiple harvesters can be running processing sub-regions of the AOI thus allowing the harvesting process to be scaled as required.

4.3.4 SOS operations

4.3.4.1 Introduction

This clause discusses the features of the NRCan profile by using example SOS requests, encoded using both KVP and XML.

4.3.4.2 GetCapabilities

The following request causes a compliant SOS to generate service metadata in the form of an OGC capabilities document:

http://ngwd-bdnes.cits.nrcan.gc.ca:8080/proxy/GinService/sos/gw?REQUEST=GetCapabilities&SERVICE=SOS&VERSION=2.0.0

The salient portion of the response is presented here and illustrates the profile directives that a SOS compliant with the NRCan profile would include in its capabilities document:

```
<ows:Profile>http://www.opengis.net/spec/SOS/2.0/conf/gfoi</ows:Profile>

<!--  TODO: add KVP BBOX profile when implemented -->
<!-- Observation can be queries using spatial geometry expressed in param --></pre>
<ows:Profile>http://www.opengis.net/spec/SOS/2.0/conf/spatialFilteringProfile</ows:Profil
e>

<!-- sampling feature must have a point geometry -->
<ows:Profile>http://www.opengis.net/spec/OMXML/2.0/conf/samplingPoint</ows:Profile>

<!-- Observation encoded with GML 3.2 XML-->
<ows:Profile>http://www.opengis.net/spec/OMXML/2.0/conf/observation</ows:Profile>

<!-- this service implement WaterML 2.0 -->
<ows:Profile>http://www.opengis.net/spec/waterml/2.0/conf/xsd-measurement-timeseries-tvp-
observation</ows:Profile>
```

4.3.4.3 DescribeSensor

The following two examples encode the same DescribeSensor request -- one using the KVP encoding and the other using the XML encoding.

KVP-encoded example:
http://198.103.103.7/GinService/sos?REQUEST=DescribeSensor&VERSION=2.0.0&SERVICE=SOS&procedure=urn:ogc:object:Sensor::EC_WaterFlowProcess&procedurDescriptionFormat=http://www.opengis.net/sensorML/1.0.1

XML-encoded example:

```
<swes:DescribeSensor
   service="SOS"
   version="2.0.0"
   xmlns="http://www.opengis.net/swes/2.0"
   xmlns:sos="http://www.opengis.net/sos/2.0"
   xmlns:fes="http://www.opengis.net/fes/2.0"
   xmlns:gml="http://www.opengis.net/gml/3.2"
   xmlns:swe="http://www.opengis.net/swe/2.0"
   xmlns:swes="http://www.opengis.net/swes/2.0"
   xmlns:xlink="http://www.w3.org/1999/xlink">
   <procedure>urn:ogc:object:Sensor::EC_WaterLevelProcess</procedure>
<procedureDescriptionFormat>http://www.opengis.net/sensorML/1.0.1</proc
edureDescriptionFormat>
</swes:DescribeSensor>
```

4.3.4.4 GetFeatureOfInterest

The following KVP-encoded request illustrates a GetFeatureOfInterest request employing a BBOX spatial filter:

http://198.103.103.7/GinService/sos?REQUEST=GetFeatureOfInterest&VERSION=2.0.0&SERVICE=SOS&spatialFilter=om:featureOfInterest/*/sams:shape,-101.2,49,-99.5,50.1&namespaces=xmlns(sams,http://www.opengis.net/samplingSpatial/2.0),xmlns(om,http://www.opengis.net/om/2.0

This following XML-encoded request similarly includes a spatial predicate on the GetFeatureOfInterest request:

```
<sos:GetFeatureOfInterest
   service="SOS"
   version="2.0.0"
   xmlns:sos="http://www.opengis.net/sos/2.0"
   xmlns:fes="http://www.opengis.net/fes/2.0"
   xmlns:gml="http://www.opengis.net/gml/3.2"
   xmlns:sams="http://www.opengis.net/spatialSampling/2.0">
   <sos:spatialFilter>
      <fes:Intersects>
         <fes:ValueReference>sams:shape</fes:ValueReference>
         <gml:Envelope srsName="urn:ogc:def:crs:EPSG::4326">
            <gml:lowerCorner>49 -140</gml:lowerCorner>
            <gml:upperCorner>60 -110</gml:upperCorner>
         </gml:Envelope>
      </fes:Intersects>
   </sos:spatialFilter>
</sos:GetFeatureOfInterest>
```

Finally, the following KVP-encoded request illustrates a GetFeatureOfInterest request employing an identifier to retrieve a specific feature of interest:

http://198.103.103.7/GinService/sos?REQUEST=GetFeatureOfInterest&VERSION=2.0.0&SERVICE=SOS&featureOfInterest=ca.gc.ec.station.11AA001

4.3.4.5 GetObservation

The following KVP and XML encoding examples encode the same request:

KVP-encoded example:
http://198.103.103.7/GinService/sos?REQUEST=GetObservation&VERSION=2.0.0&SE

RVICE=SOS&offering=WATER_LEVEL&featureOfInterest=ca.gc.ec.station.05NG021 &observedProperty=urn:ogc:def:phenomenon:OGC:1.0.30:waterlevel

XML-encoded example:

```
<sos:GetObservation
   service="SOS"
   version="2.0.0"
   xmlns:sos="http://www.opengis.net/sos/2.0"
   xmlns:fes="http://www.opengis.net/fes/2.0"
   xmlns:gml="http://www.opengis.net/gml/3.2"
   xmlns:swe="http://www.opengis.net/swe/2.0"
   xmlns:swes="http://www.opengis.net/swes/2.0"
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/sos/2.0
                       http://schemas.opengis.net/sos/2.0/sos.xsd
                       http://www.opengis.net/gml/3.2
                       http://schemas.opengis.net/gml/3.2.1/gml.xsd">
   <sos:offering>WATER_LEVEL</sos:offering>
<sos:observedProperty>urn:ogc:def:phenomenon:OGC:1.0.30:waterlevel</sos
:observedProperty>
<sos:featureOfInterest>ca.gc.ec.station.05NG021</sos:featureOfInterest>
</sos:GetObservation>
```

4.3.4.6 GetDataAvailability profile

4.3.4.6.1 Introduction

For the NRCan profile is it mandatory to implement the GetDataAvailability request that plays an important role in this project because it provides the means by which the last value of a procedure can be monitored by the notification system.

This clause discusses various aspects of the GetDataAvailabilty operation including why the profile makes the offering parameter mandatory and how to use the operation to extract the last value.

4.3.4.6.2 Offering parameter

For CHISP, it is proposed that the offering be a **mandatory** item in the request. The reason for this is illustrated in the following example:

The NRCan SOS service (see 5.3.3.2) provides the same observed property and procedure in two different offerings, WATER_FLOW and WATER_FLOW_LIVE. The only difference between them is the time span and the features of interest.

Offering: WATER_FLOW_LIVE

- WATER_FLOW_LIVE has only 10 features of interest and values from around 2011 to now
- Observed property : water flow ("urn:ogc:def:phenomenon:OGC:1.0.30:waterflow")
- procedure : Water flow process ("urn:ogc:object:Sensor::EC_WaterFlowProcess")

Offering: WATER_FLOW

- WATER FLOW has historical data for a large number of features of interest, including the 10 present in LIVE, but the observation are all BEFORE 2011.
- Observed property : water flow ("urn:ogc:def:phenomenon:OGC:1.0.30:waterflow")
- procedure : Water flow process ("urn:ogc:object:Sensor::EC_WaterFlowProcess")

Because some features of interest actually appear in both offerings (05NB036 is an example), a GetDataAvailability response will be ambiguous if the request does not specify an offering (the service considers all offerings to be in scope - WATER_FLOW and WATER_FLOW_LIVE). This might further encourage the omission of an offering from an SOS GetObservation request, which could result in only one value returned instead of two.

Although not clearly stated in the SOS specification (see OGC 12-006), the time span should probably be an aggregation of both time spans from both offerings.

4.3.4.6.3 Extracting the last value

This clause discusses how the GetDataAvailability operation may be used to determine the request parameters for a GetObservation request that may be used to extract the last observed value for an offering of a feature of interest.

Consider the following KVP-encoded request is an example of the GetDataAvailability operation.

http://198.103.103.7/GinService/sos?REQUEST=GetDataAvailability&VERSION=2.0.0&SERVICE=SOS&featureOfInterest=ca.gc.ec.station.01AJ013&offering=WATER_FLOW

The GetDataAvailability operation can includes parameters to specify a procedure, an observation, a feature of interest, the offering and a time range. By default, if a value is not specified for a parameter, then the entire range of values for that parameter is included in the response.

Consider the following example which is requesting all procedures and observations related to feature of interest 05NB036 for offering WATER_FLOW_LIVE during 1900-01-01T12:00:00 to 2013-12-12T23:59:59.

- procedure = ? (response will include all matching procedures)
- observation = ? (response will include all matching observations)
- featureOfInterest = 05NB036
- offering = WATER_FLOW_LIVE
- time range = 1900-01-01T12:00:00 to 2013-12-12T23:59:59

Because we did not specify an observed property or a procedure in the request, the response will return all observed properties and procedures in the context of that offering, feature of interest and time span. Here is the request encoded as XML:

```
<gda:GetDataAvailability
   version="2.0.0"
   service="SOS"
   xmlns:gml="http://www.opengis.net/gml/3.2"
   xmlns:gda="http://www.opengis.net/sosgda/1.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/sosgda/1.0
file:///W:/LCNP/Normes/SOS/SOS2/GetDataAvailability_Extension/xsd/sosGe
tDataAvailability.xsd">
   <gda:availabilityTimeframe>
      <gml:TimePeriod gml:id="x">
         <gml:beginPosition>1900-01-01T12:00:00</gml:beginPosition>
         <gml:endPosition>2013-12-12T23:59:59</gml:endPosition>
      </gml:TimePeriod>
   </gda:availabilityTimeframe>
   <gda:featureOfInterest>05NB036</gda:featureOfInterest>
   <gda:offering>WATER_FLOW_LIVE</gda:offering>
</gda:GetDataAvailability>
```

Notice, that as per the NRCan profile the request includes a mandatory offering. A minimal response to this request might be:

```
<?xml version="1.0" encoding="UTF-8"?>
<gda:GetDataAvailabilityResponse
   xmlns:swes="http://www.opengis.net/swes/2.0"
   xmlns:gml="http://www.opengis.net/gml/3.2"
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xmlns:gda="http://www.opengis.net/sosgda/1.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/sosgda/1.0
file:///W:/LCNP/Normes/SOS/SOS2/GetDataAvailability_Extension/xsd/sosGe
tDataAvailability.xsd">
   <gda:featurePropertyRelationship>
      <gda:FeaturePropertyTemporalRelationship
         xmlns:gml="http://www.opengis.net/gml/3.2">
         <gda:phenomenonTime>
            <gml:TimePeriod gml:id="x1">
               <gml:beginPosition>2011-05-
15T23:16:00</gml:beginPosition>
               <gml:endPosition>2013-01-14T14:00:00</gml:endPosition>
            </gml:TimePeriod>
         </gda:phenomenonTime>
         <gda:targetFeature xlink:href="#foi_05NB036"/>
         <gda:targetProperty xlink:href="#water_flow"/>
      </gda:FeaturePropertyTemporalRelationship>
   </gda:featurePropertyRelationship>
   <gda:propertyEntryPoint>
      <gda:ObservedPropertyInfo swes:id="water_flow">
<gda:property>"urn:ogc:def:phenomenon:OGC:1.0.30:waterflow"</gda:proper
ty>
      </gda:ObservedPropertyInfo>
   </gda:propertyEntryPoint>
   <gda:featureOfInterestEntryPoint>
      <gda:FeatureOfInterestInfo swes:id="foi_05NB036">
         <gda:feature>05NB036</gda:feature>
         <gda:relatedProperty xlink:href="#water_flow"/>
      </gda:FeatureOfInterestInfo>
   </gda:featureOfInterestEntryPoint>
</gda:GetDataAvailabilityResponse>
```

The salient fragment from this response is:

```
<gda:FeaturePropertyTemporalRelationship
   xmlns:gml="http://www.opengis.net/gml/3.2">
   <gda:phenomenonTime>
      <gml:TimePeriod gml:id="x1">
         <gml:beginPosition>2011-05-15T23:16:00</gml:beginPosition>
         <gml:endPosition>2013-01-14T14:00:00</gml:endPosition>
      </gml:TimePeriod>
   </gda:phenomenonTime>
   <gda:targetFeature xlink:href="#foi_05NB036"/>
   <gda:targetProperty xlink:href="#water_flow"/>
</gda:FeaturePropertyTemporalRelationship>
```

which essentially says that the feature of interest is related to water flow observation over the time period 2011-05-15 to 2013-01-14. Given this fragment, the two methods that may be employed to extract the last observed water flow value for the feature of interest 05NB036:

1. Because we know the time period from the fragment above, we now have all the information necessary to formulate a GetObservation request to obtain the last value.
2. Alternatively, we could take advantage of the fact that the specification for the GetDataAvailability operation supports an extension point that could be exploited to allow us to include the last value in question in the response above without forcing a second request from the client application as proposed by option (1).

In order to avoid the use of an extension point in an ad-hoc manner the NRCan profile proposed to use the first method (i.e. formulate a GetObservation request).

5. Use cases

5.1 Introduction

Two use cases were developed in the initial stages of the CHISP-1 project that were used to drive the development of the system architecture and ultimately the components developed by the project participants. The first use case was a flood event use case. The second use case was a nutrient load calculation use case. This clause describes these use cases as they ultimately manifested themselves during the course of the CHISP-1 project and were present at the final webinar.

5.2 Project participants

5.2.1 Introduction

This clause lists the CHISP-1 project sponsors and participants.

5.2.2 Sponsors

The CHISP-1 project was sponsored by the following organizations:

- Environment Canada
- Natural Resources Canada (NRCan)
- GeoConnections
- NRCan Groundwater Geoscience Program
- US Geological Service (USGS)

- Environmental Protection Agency (EPA)
- National Oceanic and Atmospheric Administration (NOAA)

In addition to sponsoring the project, Environment Canada, NRCan and USGS participated in the project by enhancing their offerings to conform to the NRCan profile (see Clause 4).

5.2.3 Participants

The following organizations were projects participants responsible for the bulk of the new components developed during the CHISP-1 project (see Table 2, Table 4):

- Explorus (http://www.explorus.org)
- GIS.FCU (http://www.gis.tw/En/)
- RPS ASA (http://www.asascience.com/)

5.3 Flood event and notification use case

5.3.1 Introduction

The flood event and notification use case focuses on monitoring cross-border stream flow and groundwater levels in order to detect when a potential flood event may be occurring. When a potential flood event is detected a unified alert service is used to notify EM analysts who have subscribed with the system to be notified of such events.

The use case can be broken down into three activities:

1. An on-going monitoring activity where the system periodically monitors stream and ground water gauges and tracks water flow and water level values.
2. A subscription activity where an emergency management analyst indicates his/her desire to be notified of an event of interest – such as an imminent flood – by choosing a monitoring point and specifying threshold parameters that signal the beginning of a potential flood. The event notification system automatically subscribes the EM analyst to monitor all the stations upstream of the specified point on interest.
3. A notification activity where the system checks the monitored values from activity (1) against the thresholds specified in activity (2) to determine if a flood is imminent and notification of the EM analyst is required.

This clause described the components used and the basic course of action for each of these activities as developed during the CHISP-1 project.

5.3.2 Area of interest

Two basins with potions in both the U.S. and Canada were considered as areas of interest for the CHISP-1 project.

Milk River Basin:
The Milk River basin includes parts of Alberta, Saskatchewan, and Montana (see Figure 4).

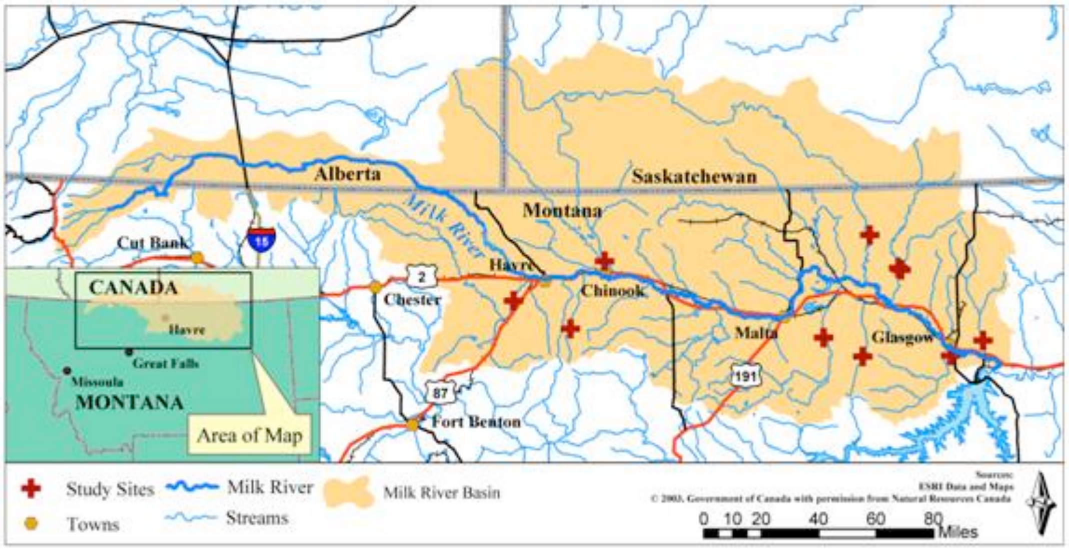


Figure 4 – Milk River Basin

Source: http://www.umt.edu/watershedclinic/images/clip_image002.jpg

Souris River Basin:
The Souris River basin includes parts of Manitoba, North Dakota, and Saskatchewan (see Figure 5). The Souris River flows into the Assiniboine River, and then into the Red River and Lake Winnipeg, which is part of the Hudson Bay. The Souris River basin shows the locations of stream gauges as green circles.

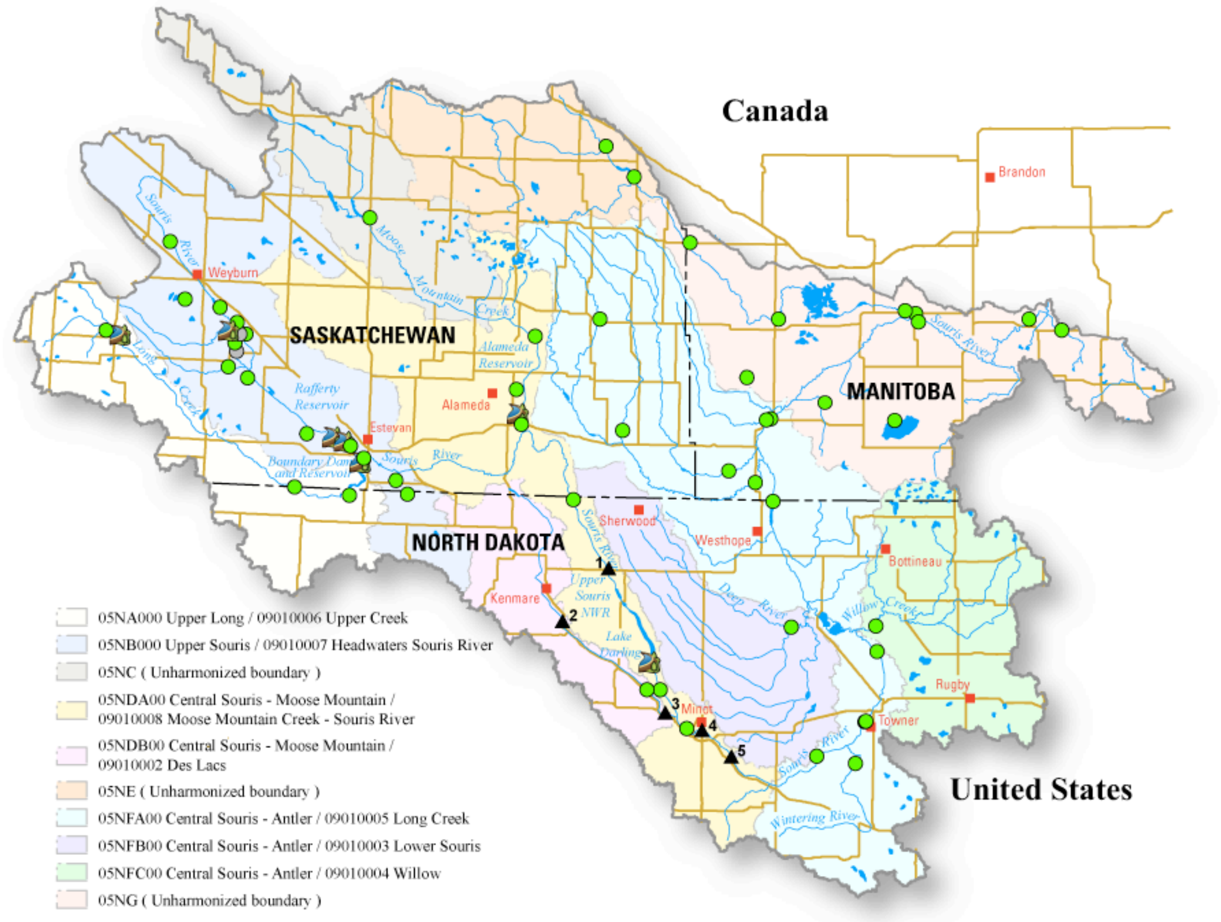

Canada
Brandon
SASKATCHEWAN
MANITOBA
Weyburn
Souris River
Moose Mountain Creek
Alameda Reservoir
Alameda
Rafferty Reservoir
Estevan
Boundary Dam and Reservoir
Long Creek
NORTH DAKOTA
Sherwood
Westhope
Bottineau
Kenmare
Upper Souris NWR
Lake Darling
Minot
Deep River
Willow Creek
Rugby
Towner
Wintering River
United States
05NA000 Upper Long / 09010006 Upper Creek
05NB000 Upper Souris / 09010007 Headwaters Souris River
05NC (Unharmonized boundary)
05NDA00 Central Souris - Moose Mountain / 09010008 Moose Mountain Creek - Souris River
05NDB00 Central Souris - Moose Mountain / 09010002 Des Lacs
05NE (Unharmonized boundary)
05NFA00 Central Souris - Antler / 09010005 Long Creek
05NFB00 Central Souris - Antler / 09010003 Lower Souris
05NFC00 Central Souris - Antler / 09010004 Willow
05NG (Unharmonized boundary)

Figure 5 – Souris River Basin

Source: http://nd.water.usgs.gov/floodinfo/souris.html

The available data for both river basins was inspected to find a suitable historical flood event that could be used to drive component developed in CHISP-1 and that could also be used for the demo at the end of the project. The chosen event was the 2011 flood on the Milk River that occurred in the months of April and May.

5.3.3 Components

5.3.3.1 Introduction

This clause lists the components used for the flood event and notification use case. The components are segregated into pre-existing components that were immediately available for the CHISP-1 profile and components developed during the CHISP-1 project.

It should be noted that some of the pre-existing SOS components were modified during the course of the CHISP-1 project to conform to the NRCan profile (see Clause 4) .

5.3.3.2 Pre-existing components

Table 1 lists the components for the flood event and notification use case that already existed at the start of project.

The table lists the name of the component and what kinds of information it provides, the organization providing the component, the standard to which the API of the component conforms and the standard to which the output format provided by the component conforms.

Table 1 – Pre-existing components for the flood event and notification use case

Component	Provider	Standard	Output Format
1. Sensor Observation Service (Water Level, Water Flow, historic & live)	Environment Canada (via NRCan)	SOS 2.0	WaterML 2.0
2. Sensor Observation Service (Groundwater Level)	NRCan	SOS 2.0	WaterML 2.0
3. Sensor Observation Service (Water Level, Water Flow)	USGS	SOS 2.0	WaterML 2.0
4. Web Feature Service (Station info)	USGS	WFS 1.1.0	WaterML 2.0
5. Web Processing Service (Upstream geometry NHD/NHN)	NRCan	WPS 1.0	WPS 1.0
6. Web Map Service (Stream segments)	NRCan	WMS 1.3.0	Map image (png,gif,jpg,wbmp,svg)
7. Multi-agency Situational Awareness System (MASAS Common Alert Protocol system)	Government of Canada	ad-hoc	Common Alert Protocol (CAP) Messages

5.3.3.3 Components developed during CHISP-1

Table 2 lists the components for the flood event and notification use case that were developed during the CHISP-1 project.

The table lists the name of each component and brief statement about what the component does and/or what kind of information it provides, the organization providing the component, the standard to which the API of the component conforms and the standard to which the output format provided by the component conforms.

Table 2 -
Components developed during CHISP-1 for the flood event and notification use case

Component	Provider	Standard	Output Format
1. Web-based subscription client (Subscribe to upstream stations)	Explorus	N/A	N/A
2. Event Notification Service (Monitor stations, flood even notification)	GIS.FCU	N/A	N/A
3. Web Notification Service (Does notifications to subscribers)	GIS.FCU	WNS 0.0.9	email
4. Web Processing Service (Upstream stations/gauges)	ASA	WPS 1.0	WPS1.0
5. Catalogue (Service metadata, Station metadata)	Explorus (Hosted pycsw instance)	CSW 2.0.2 APISO 1.0.0	ISO19115, ISO19119, OGC Core (csw:Record)

The event notification service, Component 2 in Table 2, is composed of the sub-components:

- A harvester module that monitors upstream stations and gauges. This component is also referred to as the "*harvester*" in this document.

- A subscription broker that handles subscriptions and checks whether notification is required. This component is also referred to as the "*broker*" in this document.

- An OGC compliant web notification service (see Table 2, Component 3) that performs the notification to subscribers.

5.3.4 Basic course of action

5.3.4.1 Introduction

The flood event and notification use case can be segregated into three actions:

1. An ongoing monitoring action where the system periodically monitors stream and ground water gauges and tracks their last value.

2. A subscription action where an emergency management analyst indicates his/her desire to be notified of an event of interest – such as a flood – by choosing a monitoring point, specifying event threshold parameters and having the system automatically subscribe to all the stations upstream of the specified point of interest.

3. A notification action where the system checks the monitored values from activity (1) against thresholds specified in activity (2) to determine if a flood is imminent and notification is required.

5.3.4.2 Monitoring action

The following course of actions, as illustrated in Figure 6, is performed by the event notification system to monitor the last observed value for gauges on the network. Each numbered item in this list corresponds to a circled number in Figure 6.

1. The harvester periodically reads the last observed value from gauges on the network.

2. The harvester stores these values as part of the metadata maintained in the CSW (see Table 2, Component 5) for each station.

3. Whenever the harvester reads a value that has changed it notifies the broker of the change.

4. This causes the broker to read the changed value from the CSW (see Table 2, Component 5).

At this point, the broker executes the course of action required to determine if a notification must be send out (see 5.3.4.4).

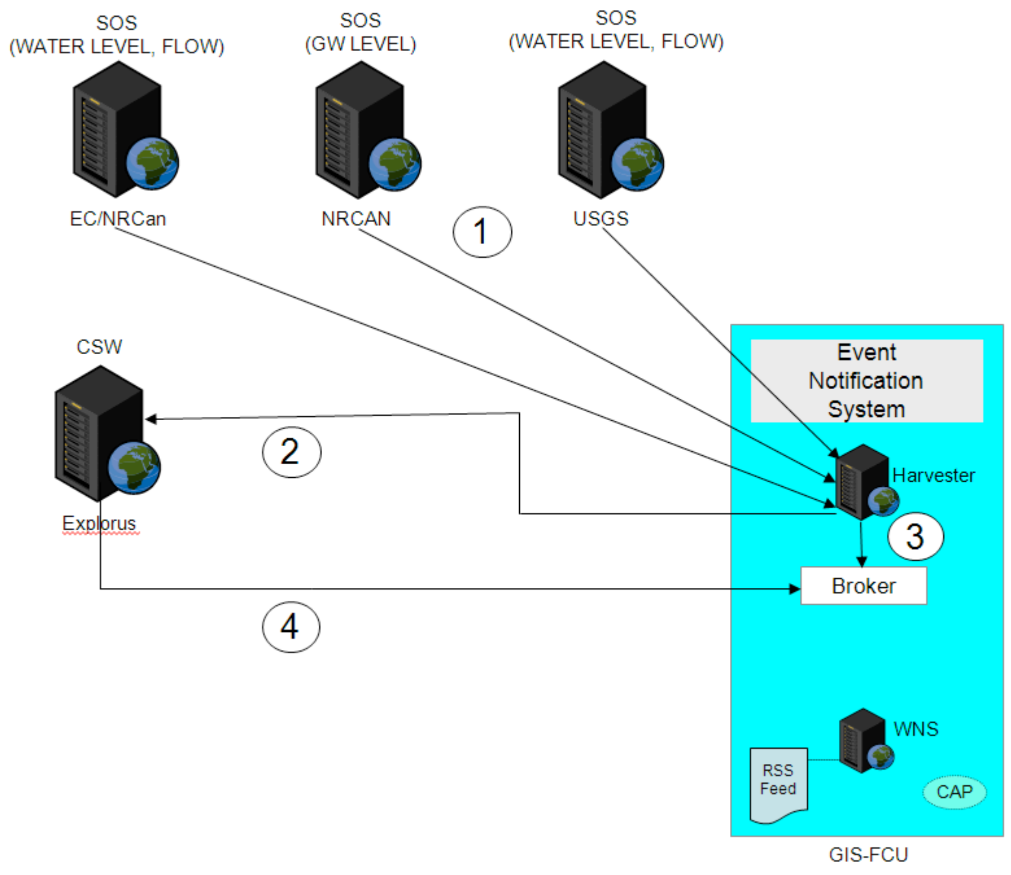


Figure 6 – Basic course of action for the monitoring activity

5.3.4.3 Subscription activity

The following course of actions, as illustrated in Figure 7, is performed by the event notification system to register a subscription for an emergency management analyst. The subscription indicates to the system that the analyst is interested in being notified of an imminent flood at a point of interest. Each numbered item in this list corresponds to a circled number in Figure 7.

1. An emergency manager analyst uses the web client to show water monitoring stations on a map. Figure 17 is a detailed example of such a map.

2. The analyst chooses a station or monitoring point of interest.

3. The causes the broker to get the list of upstream stations to monitor using the upstream WPS and is then used to create a subscription. Figure 18 provides a detailed illustration of the dialogue box presented to the client in order to create a subscription.

4. The broker then registers the subscription with the WNS.

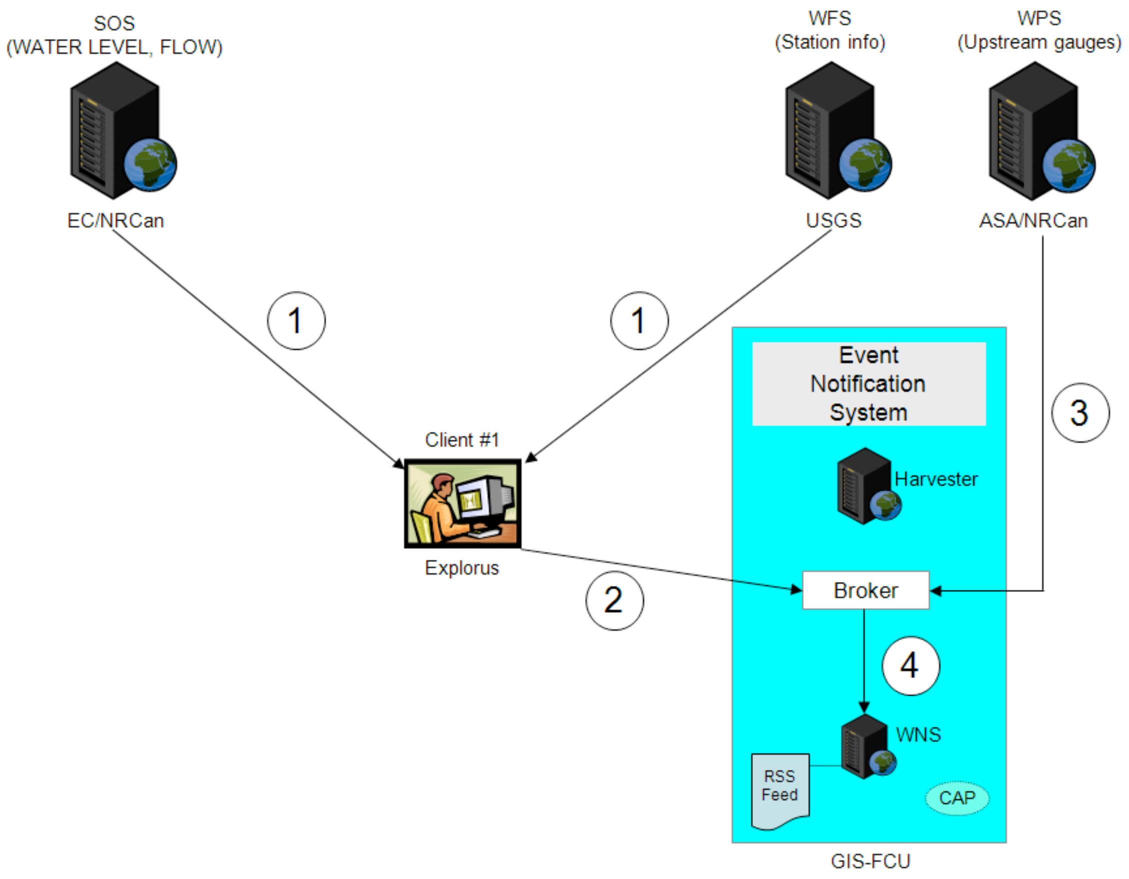


Figure 7 – Subscription action

5.3.4.4 Notification action

The following course of actions, as illustrated in Figure 8, is performed by the event notification system to determine if an event of interest has occurred and if it has to notify subscribers. Each numbered item in the list bellow corresponds to a circled number in Figure 8.

Editor's Note: The original intent, with the notification action, was to have the ENS call the upstream WPS during each polling interval to get an up-to-date sensor list. Thus, if any new sensors were added to the network they would automatically be picked up for monitoring by the ENS. However, because of performance reasons -- the upstream WPS was slow to respond – this functionality was disabled and the ENS only checked the upstream WPS for the list of sensors once when the subscription was created. For the CHISP-1 pilot this was a reasonable fix for this performance issue because the sensor network was static. Thus, this clause describes the notification action as it was actually executed during the pilot.

The notification action proceeds as follows:

1. When notified by the harvester (see 5.3.4.2), the broker reads the last value from the CSW.
2. If the value violates a subscription threshold (e.g. the water level has risen above a set threshold) the broker creates a notification email.
3. The event notification system uses the WNS to do the notification.
4. The WNS sends the notification email created in step 2 to the EM analyst.
5. Among other things, the notification email includes a link to invoke a CAP Alert
6. The EM analyst can click the CAP Alert link which brings up a dialogue box to create the alert (see Figure 10).
7. The alert is then sent to the MASAS hub which is responsible for distributing the alert.

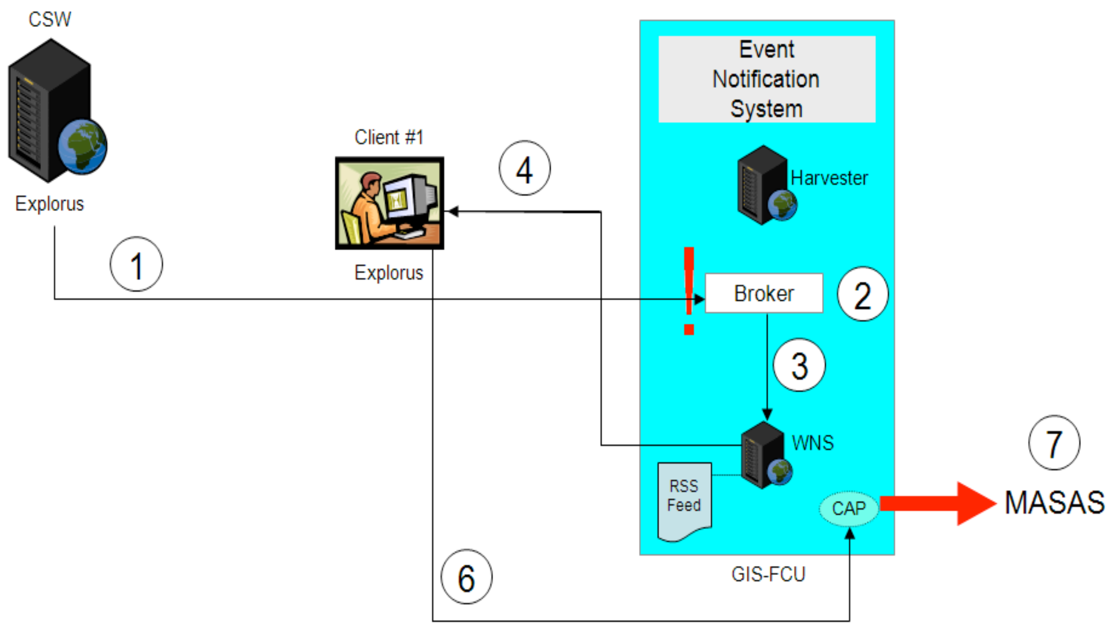
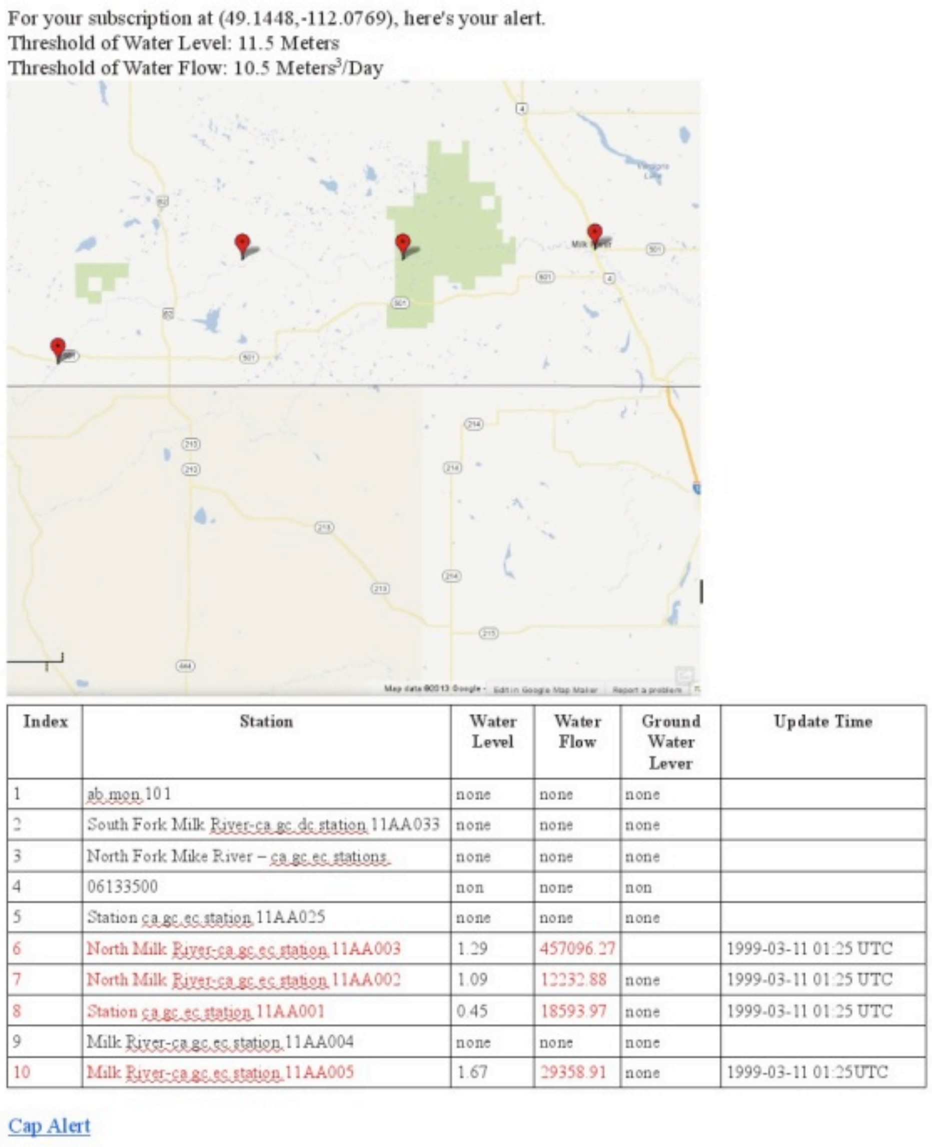


Figure 8 – Notification action

Figure 9 is a sample of a notification email generated by the event notification system and sent to subscribers by the WNS when an event of interest occurs. The email includes

- Metadata about the subscription
- A map showing the station locations
    - the status of each station in indicated by color
    - blue indicates that the last observed values from the corresponding station has not exceeded subscription thresholds
    - red indicates that the last observed value from the corresponding station has exceeded subscription threshold
- A table listing all the upstream stations. The table includes the station id, the last measured values for water level, water flow and ground water level and the time the value was read. Any stations whose measured values exceed the specified subscription thresholds are highlighted in red in the table.
- Depending of the role of user, the email may also contain a link that may be invoked to send a CAP alert using the MASAS system.

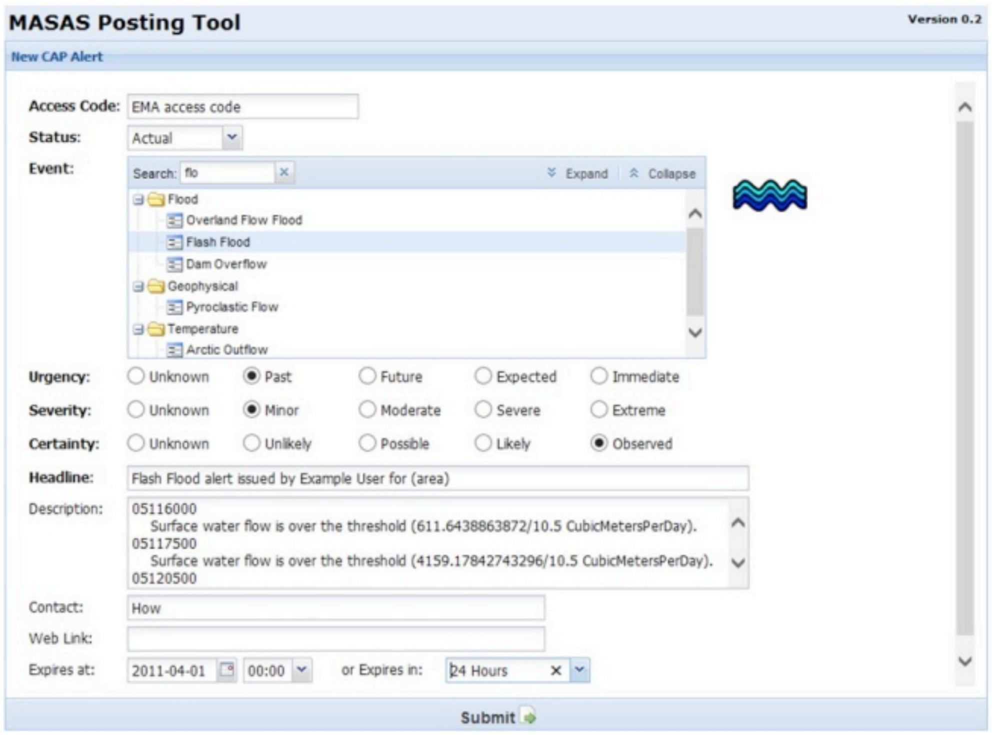

For your subscription at (49.1448,-112.0769), here's your alert.
Threshold of Water Level: 11.5 Meters
Threshold of Water Flow: 10.5 Meters^3/Day

Index	Station	Water Level	Water Flow	Ground Water Lever	Update Time
1	ab.mon.101	none	none	none	
2	South Fork Milk River-ca.gc.dc.station.11AA033	none	none	none	
3	North Fork Mike River – ca.gc.ec.stations.	none	none	none	
4	06133500	non	none	non	
5	Station ca.gc.ec.station.11AA025	none	none	none	
6	North Milk River-ca.gc.ec.station.11AA003	1.29	457096.27		1999-03-11 01:25 UTC
7	North Milk River-ca.gc.ec.station.11AA002	1.09	12232.88	none	1999-03-11 01:25 UTC
8	Station ca.gc.ec.station.11AA001	0.45	18593.97	none	1999-03-11 01:25 UTC
9	Milk River-ca.gc.ec.station.11AA004	none	none	none	
10	Milk River-ca.gc.ec.station.11AA005	1.67	29358.91	none	1999-03-11 01:25UTC

Cap Alert

Figure 9 – Notification email

Figure 10 show the dialogue box that is presented to the EM analyst if he/she chooses to invoke the “Cap Alert” link. The dialogue allows the EM analyst to fill in the detail of the alert which is submitted to the MASAS hub for distribution when the “Submit” button is clicked.



MASAS Posting Tool

Version 0.2

New CAP Alert

Access Code:	EMA access code
Status:	Actual
Event:	Search: flo Flood Overland Flow Flood Flash Flood Dam Overflow Geophysical Pyroclastic Flow Temperature Arctic Outflow
Urgency:	Unknown [✓ Past] Future Expected Immediate
Severity:	Unknown [✓ Minor] Moderate Severe Extreme
Certainty:	Unknown Unlikely Possible Likely [✓ Observed]
Headline:	Flash Flood alert issued by Example User for (area)
Description:	05116000 Surface water flow is over the threshold (611.6438863872/10.5 CubicMetersPerDay). 05117500 Surface water flow is over the threshold (4159.17842743296/10.5 CubicMetersPerDay). 05120500
Contact:	How
Web Link:	
Expires at:	2011-04-01 00:00 or Expires in: 24 Hours

Figure 10 – MASAS posting tool

5.4 Nutrient load calculation use case

5.4.1 Introduction

The Nutrient Load Calculation use case computes known nutrient loads to the Great Lakes from tributaries, using web-accessible inputs of water quality observations and flow rates, to produce web-accessible outputs of nutrient loads. The use case focuses on the analytes Phosphorus and Nitrogen.

5.4.2 Components

5.4.2.1 Pre-existing components

Table 3 lists the components for the nutrient load calculation use case that already existed at the start of the CHISP-1 project.

The table lists the name of the component and what kinds of information it provides, the organization providing the component, the standard to which the API of the component conforms and the standard to which the output format provided by the component conforms.

Table 3 – Pre-existing components for the nutrient load calculation use case

Component	Provider	API Standard	Output Format Standard
1. Sensor Observation Service (Water Level, Water Flow, historic & live)	Environment Canada (via NRCan)	SOS 2.0	WaterML V2.0
2. Sensor Observation Service (Water Level, Water Flow)	USGS	SOS 2.0	WaterML V2.0

5.4.2.2 Components developed during CHISP-1

Table 4 lists the components for the flood event and notification use case that were developed and deployed during the CHISP-1 project.

The table lists the name of each component and brief statement about what the component does and/or what kind of information it provides, the organization providing the component, the standard to which the API of the component conforms and the standard to which the output format provided by the component conforms.

Table 4 – Components developed during CHISP-1 for the nutrient load calculation use case

Component	Provider	API Standard	Output Format Standard
1. Web Processing Service (Nutrient load calculation)	ASA	WPS 1.0	WPS 1.0
2. Sensor Observation Service (Integrates US and CAN water quality servers)	ASA	SOS 2.0	IOOS SWE XML
3. Nutrient Load Calculation Web Client	ASA	N/A	N/A
4. Local catalogue (Catalogue of streams, tributaries and water quality gauges)	ASA	Django Database Abstraction API	N/A
5. SPARQL Server (Analyte equivalents US, CAN)	NRCan	SPARQL	RDF

5.4.3 Basic course of action

Figure 11 illustrates the basic course of actions for the nutrient load calculation use case. Each numbered item in this list corresponds to a circled number in Figure 11:

1. A WQA initiates the nutrient load calculation web client and provides these inputs: a Great Lake of interest, the name of an analyte, a time period of interest

2. The web client invokes the nutrient load calculation WPS to run the model.

3. The nutrient load calculation WPS queries the local catalog for tributaries on the lake of interest that have both a stream gauge and water quality samples available for the nutrient of interest.

4. For the stations returned by the catalogue query, the NLCS makes requests to the water quality and stream flow SOS services

5. The nutrient load calculation WPS interpolates the water quality and stream flow measurements for the period specified by the request. It calculates nutrient flux from the interpolated measurements and numerically integrates the results to determine the total load over the period of interest for each tributary

6. It sums all of the tributaries' contributions for the total load on the lake (converting units between standard and SI where appropriate) ... and presents the results to the WQA

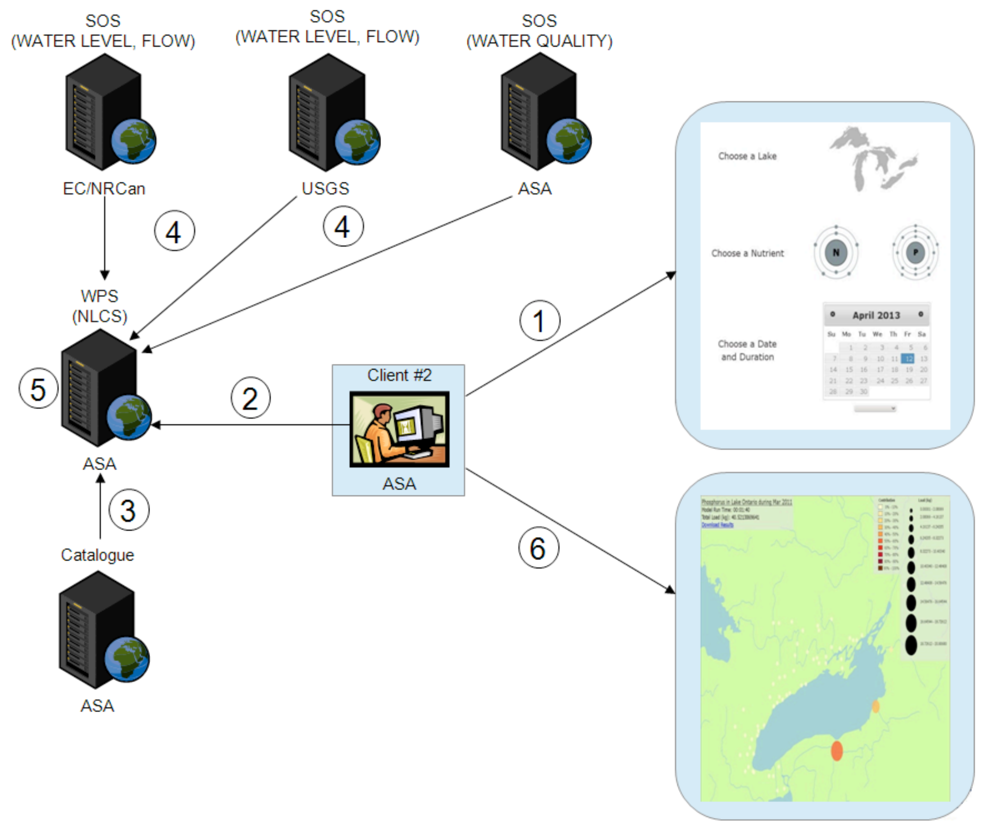


Figure 11 – Nutrient Load Calculation Use Case

6. Component details

6.1 Upstream WPS

6.1.1 Introduction

The function of the upstream WPS is, given a point of interest, to determine which water monitoring stations or gauges exist upstream of that point.

6.1.2 Service endpoint

The service endpoint for the upstream WPS can be found at:

http://64.72.74.103:8080/wps/?request=GetCapabilities&version=1.0.0

6.1.3 Implementation details

The upstream WPS was implemented in Python using a custom developed WPS framework for flexibility and depends on the following libraries:

- For scientific computing: numpy (http://www.numpy.org)
- Web framework: django (http://www.djangoproject.com)
- For making HTTP requests: requests (http://docs.python-requests.org)
- A python-based HTTP server: gunicorn (http://www.gunicorn.org)
    - was used as a WSGI server for the service

6.1.4 Operational details

6.1.4.1 Introduction

The upstream WPS service implements the following operations: GetCapabilities, DescribeProcess and Execute.

The upstream WPS offers the following methods or processes: add_gauge_to_stream, remove_gauge_to_stream and find_upstream_gauges.

6.1.4.2 Process add_gauge_to_stream

The add_gauge_to_stream method allows a list of gauges to be related with a river segment.

The process accepts as input a list of gauge id's and a river reach identifier (in the Canadian NHN ID format) and then updates its local database with the relationship.

The response is either an acknowledgement that the relationship was successfully registered or an exception message.

6.1.4.3 Process remove_gauge_from_stream

The remove_gauge_from_stream process removes the relationship between a list of gauges and a river segment.

The process accepts as input a list of gauge id's and a river reach identifier (in the Canadian NHN ID format) and then removes the relations from the local database.

The response is either an acknowledgement that the relationship was successfully removed or an exception message.

6.1.4.4 Process find_upstream_gauges

Give a point of interest, the find_upstream_gauges process finds all the gauges upstream of that point.

The find_upstream_gauges process queries the NRCan upstream river segment WPS (see Table 1, Component 5) and based on the river segments, returns the stream gauge id's that are located on those rivers segments (if any exist).

The find_upstream_gauges process then uses a local database to manage the river/gauge relationships – originally provided by the sponsors as an Excel spread sheet – via django's database abstraction API.

The response from the find_upstream_gauges process is an XML document that encodes the stream id/gauge id relationships. The following XML fragment is an example of a response to the find_upstream_gauges process:

```
<Stream>
   <id>7bb9c0305c7f4802956c8f7277819f7e</id>
   <name>Souris River</name>
   <Stations>
      <Station latitude="49.9888916"
       longitude="-104.19000244">ca.gc.ec.station.05NB031</Station>
   </Stations>
</Stream>
<Stream>
   <id>64744610f8e74928890282eafe7f73ad</id>
   <name>Souris River</name>
   <Stations>
      <Station latitude="49.33666992"
```

```
          longitude="-103.54110718">ca.gc.ec.station.05NB037</Station>
      </Stations>
   </Stream>
   <Stream>
      <id>50633f5b9f7e425786545276de3bec58</id>
      <name>Souris River</name>
      <Stations>
         <Station latitude="49.49361038"
          longitude="-103.66221619">ca.gc.ec.station.05NB017</Station>
      </Stations>
   </Stream>
   <Stream>
      <id>2876fe821ae34b49a0d91428a5ebd308</id>
      <name>Souris River</name>
      <Stations>
         <Station latitude="49.57556152"
          longitude="-103.76889038">ca.gc.ec.station.05NB020</Station>
      </Stations>
   </Stream>
   <Stream>
      <id>42ddf48f83fe47e49ac8aaf1da906f5b</id>
      <name>None</name>
      <Stations>
         <Station latitude="49.5616684"
          longitude="-103.67443848">ca.gc.ec.station.05NB024</Station>
      </Stations>
   </Stream>
   <Stream>
      <id>a536171eecbf4d66afc93629df2488dc</id>
      <name>Souris River</name>
      <Stations>
         <Station latitude="48.9963913"
          longitude="-100.95806122">ca.gc.ec.station.05NF012</Station>
         <Station latitude="48.9964079"
          longitude="-100.9584889">05124000</Station>
      </Stations>
   </Stream>
```

6.1.5 Examples

- Capabilities document:
  - http://64.72.74.103:8080/wps/?service=WPS&version=1.0.0&request=GetCapabilities
- Process descriptions:
  - http://64.72.74.103:8080/wps/?service=WPS&version=1.0.0&request=DescribeProcess&identifier=all
- Find upstream stations:

- http://64.72.74.103:8080/wps/?service=WPS&version=1.0.0&request=execute&identifier=find_upstream_gauges&datainputs=latitude=49.37833023%3Blongitude=-100.78943634
- http://64.72.74.103:8080/wps/?service=WPS&version=1.0.0&request=execute&identifier=find_upstream_gauges&datainputs=latitude=49.1448%3Blongitude=-112.0769

6.2 Catalogue

6.2.1 Introduction

The catalogue deployed for the CHISP-1 project was used to store metadata about servers participating in the project and metadata about water monitoring stations. The primary function of the catalogue within the CHISP-1 project was to support dynamic discovery of the registered components by other components participating in the project. For example, the harvester module (see 6.3.4.1) of the event notification system uses the CSW to discover the available sensor observation services to monitor and also updates the station metadata stored in the CSW to store the last observed value for each offering.

6.2.2 Service endpoint

The service endpoint for the catalogue can be found at:

http://107.22.84.193/pycsw/csw.py?service=CSW&version=2.0.2&request=GetCapabilities

6.2.3 Implementation details

The initial deployment of the catalogue used the GeoNetwork catalogue (see http://sourceforge.net/projects/geonetwork/). However, several issues were encountered that forced the project to seek an alternative catalogue.

The specific issues encountered using the GeoNetwork catalogue included:

1. A number of documented and undocumented bugs were encountered that consumed a significant amount of project resource to try and resolve. Eventually a properly configured CSW service was deployed as described in the GeoNetwork documentation.

2. Despite being properly configured the server did not always respond to CSW requests as described in the OGC implementation specification (see OGC 07-006r1).

3. The catalogue did not implement the CSW Harvest operation[1] which allows services to be easily registered. The lack of this operation would require a much more involved and likely manual process to register each service in the project.

NOTE 1: The CSW Harvest operation should not be confused with the Harvester module of the event notification system (see 6.3.4.1). The CSW Harvest operation allows the catalogue to read resource metadata (such as the capabilities document of an OGC service) and register the resource to make it discoverable. The Harvest module, on the other hand, is the sub-component of the event notification system that reads the last observed value from a monitoring station and stored that value in the catalogue (see 6.2).

As a result of these issues, the pycsw (see http://pycsw.org/) catalogue was chosen to replace the GeoNetwork catalogue. The pycsw catalogue is an OGC CSW (see OGC 07-006r1) server implementation written in Python and is certified OGC Compliant. The pycsw server offered the following capabilities:

1. The server provided a stable CSW with transactional support that the project required.
2. The pycsw catalogue supports a number of profiles of the OGC CSW but for the CHISP-1 project, the ISO application profile (see OGC 07-045) was selected.
3. The server supports the Harvest operation; although the operations did not initially support harvesting OGC sensor observation service (see item 4 below).
4. One of the primary implementers of pycsw was part of the sponsor team from Environment Canada which allowed any issues encountered to be addressed quickly and which also allowed the pycsw to be extended to suite the requirements of the project (see 6.3.4).

Enhancements:

The pycsw catalogue supports the ability to register many OGC compliant services by reading their capabilities document and then automatically registering the service (e.g. WFS, WMS, WCS, etc.). This is accomplished using the CSW Harvest operation. The pycsw catalogue, however, did not support registering OGC sensor observation services (see OGC 12-006) in this way.

To resolve this issue, the pycsw source code was extended to add the ability to use the Harvest operation to register OGC sensor observation services. These changes were checked back into the pycsw code base maintained on GitHub (see https://github.com/geopython/pycsw).

6.2.4 Operational details

Operationally, the catalogue was used to support the following activities:

- Register CHISP-1 services into the catalogue to make them discoverable.
- Register the available water monitoring stations to make then discoverable.
- Use the CSW to get the list of services and stations within the AOI.
- Periodically update the water monitoring station metadata to include the last observed value.

The examples in Clause 6.2.6 illustrate CSW requests that perform each of these activities.

6.2.5 Examples

Example 1: Register a sensor observation service with the CSW

```
<?xml version="1.0" encoding="UTF-8"?>
<Harvest
   xmlns="http://www.opengis.net/cat/csw/2.0.2"
   xmlns:ogc="http://www.opengis.net/ogc"
   xmlns:gmd="http://www.isotc211.org/2005/gmd"
   xmlns:ows="http://www.opengis.net/ows"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:dct="http://purl.org/dc/terms/"
   xmlns:gml="http://www.opengis.net/gml"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/cat/csw/2.0.2
      http://schemas.opengis.net/csw/2.0.2/CSW-publication.xsd"
   service="CSW"
   version="2.0.2">
   <Source>http://ngwd-
bdnes.cits.nrcan.gc.ca/GinService/sos?SERVICE=SOS&REQUEST=GetCapabiliti
es</Source>
   <ResourceType>http://www.opengis.net/sos/2.0</ResourceType>
   <ResourceFormat>application/xml</ResourceFormat>
</Harvest>
```

Example 2: Register a water monitoring station

```
<?xml version="1.0" encoding="UTF-8"?>
<csw:Transaction
   xmlns:csw="http://www.opengis.net/cat/csw/2.0.2"
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:dct="http://purl.org/dc/terms/"
   xmlns:ows="http://www.opengis.net/ows"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/cat/csw/2.0.2
      http://schemas.opengis.net/csw/2.0.2/CSW-publication.xsd"
   service="CSW"
   version="2.0.2">
   <csw:Insert>
      <csw:Record>
         <dc:identifier>ca.gc.ec.station.05NB004</dc:identifier>
         <dc:title>BEAVERDAM CREEK NEAR WEYBURN</dc:title>
         <dc:type>station</dc:type>
         <dc:subject>offering:WATER_FLOW</dc:subject>
         <dc:subject>offering:WATER_LEVEL</dc:subject>
<dc:subject>observedProperty=urn:ogc:def:phenomenon:OGC:1.0.30:waterflo
w</dc:subject>
<dc:subject>observedProperty=urn:ogc:def:phenomenon:OGC:1.0.30:waterlev
el</dc:subject>
         <dc:relation/>
         <dct:modified>2013-02-20T16:10:00-08:00</dct:modified>
         <dct:abstract></dct:abstract>
         <ows:WGS84BoundingBox>
            <ows:LowerCorner>49.5972 -103.9652</ows:LowerCorner>
            <ows:UpperCorner>49.5972 -103.9652</ows:UpperCorner>
         </ows:WGS84BoundingBox>
      </csw:Record>
   </csw:Insert>
</csw:Transaction>
```

Example 3: Query to obtain the set of sensor observation services within the AOI

```
<?xml version="1.0"?>
<GetRecords
   service="CSW"
   version="2.0.2"
   maxRecords="100"
   resultType="results"
   xmlns="http://www.opengis.net/cat/csw/2.0.2"
   xmlns:csw="http://www.opengis.net/cat/csw/2.0.2"
   xmlns:fes="http://www.opengis.net/ogc"
   xmlns:gmd="http://www.isotc211.org/2005/gmd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/cat/csw/2.0.2
                    http://schemas.opengis.net/csw/2.0.2/csw.xsd">
   <Query typeNames="csw:Record">
      <ElementSetName>full</ElementSetName>
      <Constraint version="1.0.0">
         <fes:Filter>
           <fes:And>
               <fes:PropertyIsEqualTo>
                  <fes:PropertyName>dc:type</fes:PropertyName>
                  <fes:Literal>service</fes:Literal>
               </fes:PropertyIsEqualTo>
               <fes:PropertyIsEqualTo>
                  <fes:PropertyName>dc:format</fes:PropertyName>
                  <fes:Literal>OGC:SOS</fes:Literal>
               </fes:PropertyIsEqualTo>
               <fes:BBOX>
                  <gml:Envelope srsName="urn:ogc:def:crs:EPSG::4326">
```

```
                    <gml:lowerCorner>48.25 -112.56</gml:lowerCorner>
                    <gml:upperCorner>59.32 -109.21</gml:upperCorner>
                 </gml:Envelope>
              </fes:BBOX>
           </fes:And>
        </fes:Filter>
     </Constraint>
  </Query>
</GetRecords>
```

Example 4: Update the metadata for a station to store the last value

```
<?xml version="1.0" encoding="UTF-8"?>
<csw:Transaction
   xmlns:csw="http://www.opengis.net/cat/csw/2.0.2"
   xmlns:fes="http://www.opengis.net/ogc"
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:dct="http://purl.org/dc/terms/"
   xmlns:ows="http://www.opengis.net/ows"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/cat/csw/2.0.2
      http://schemas.opengis.net/csw/2.0.2/CSW-publication.xsd"
   service="CSW" version="2.0.2">
   <csw:Update>
      <csw:RecordProperty>
         <csw:Name>dct:abstract</csw:Name>
         <csw:Value>type:water_flow;value:6.792</csw:Value>
      </csw:RecordProperty>
      <csw:Constraint version="1.0.0">
         <fes:Filter>
            <fes:PropertyIsEqualTo>
               <fes:PropertyName>dc:identifier</fes:PropertyName>
               <fes:Literal>ca.gc.ec.station.05NB004</fes:Literal>
            </fes:PropertyIsEqualTo>
         </fes:Filter>
      </csw:Constraint>
   </csw:Update>
</csw:Transaction>
```

6.3 Event Notification Service

6.3.1 Introduction

The function of the event notification service is to notify subscribers when an event of interest occurs. The event of interest is defined at the time the subscription is created. Thereafter the event notification service monitors the system and if the defined event occurs, it sends a notification message to all affected subscribers. For the CHISP-1 project the event of interest was a flood event (see 5.3).

The overall architecture of the event notification service is illustrated in Figure 12. The event notification system is composed of the following sub-components:

- Harvester
- Subscription Broker
- Notification service

The implementation and operational details of each component are described in this clause.

6.3.2 Service end points

6.3.2.1 Servers and control components

- Harvester control program: https://github.com/gisfcuchisp1/DemoTool
  Note: The Harvester control program allows the harvester to be invoked for a specific period of time thus allowing historical flood events to be demonstrated using the monitoring system developed for the CHISP-1 project.
- Subscription broker: http://59.125.87.213/WNS/Broker/RegisterInfo.ashx
- Web Notification service: http://59.125.87.213/WNS/notification/wns.ashx

6.3.2.2 Test clients

The following components are not strictly part of the system developed for CHISP-1 but rather are test harnesses built to test various sub-components of the event notification server and the GetDataAvailability extensions built by GIS.FCU.

- Broker API test client: http://59.125.87.213/wns/broker/testbroker.aspx
- Harvester test client: http://59.125.87.213/Harvester/Default.aspx
- CAP test client: http://59.125.87.213/wns/broker/CAPAlertPosting/new_cap.aspx
- GetDataAvailability test client: http://59.125.87.213:443/sos-4.0.0/client

6.3.3 Implementation details

Harvester, Subscription Broker, Web Notification Service

The Harvester, broker, notification service and CAP alert client were implemented using C#. The development environment consisted of:

- Windows Server 2008
- MS SQL Server 2008
- NET Framework4.0.

CAP Alert Client

The MASAS Tool (https://sandbox2.masas-sics.ca/portal/main) was used for publishing to the MASAS HUB from the CAP Alert client (see 6.3.4.2.5).

GetDataAvailability operation

Version 4.0 of the 52North code was used to implement the SOS GDA operation that complied with the NRCan profile (see Clause 4). Documentation for 52North's open source sensor observation service can be found at: https://wiki.52north.org/bin/view/SensorWeb/SensorObservationServiceIVDocumentation). The development environment consisted of the following components:

- Java runtime environment (JRE) 7.0
- Tomcat7.0
- PostgreSQL9.2
- PostGIS 2.0.3

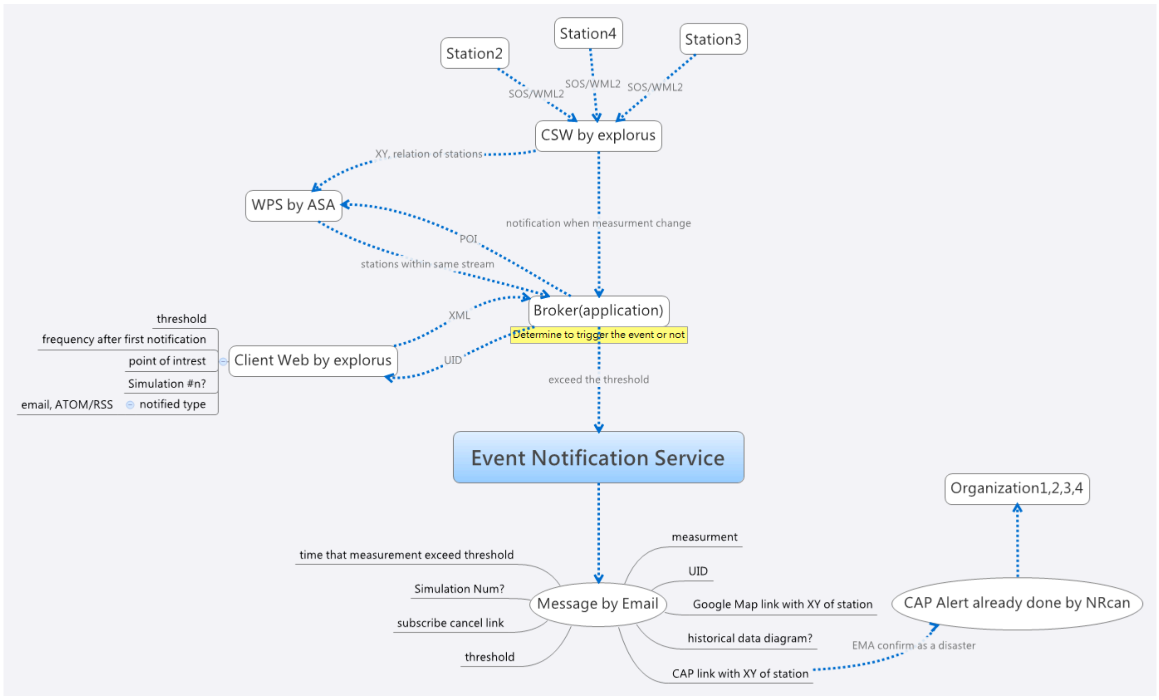


Figure 12 – ENS architecture

6.3.4 Operational details

6.3.4.1 Harvester

In order for the event notification system to operate it must, for each subscription, read the last observed value and compare it to threshold values defined at the time the subscription was created. When the monitored value exceeds a defined threshold the system sends a notification of the event to each affected subscriber.

One approach for accomplishing this monitoring task would be to fire off an agent responsible for each subscription. This agent would read the last observed value for each station that needs to be monitored to satisfy each subscription. The problem with this approach is that it does not scale. As the number of subscriptions grows, the number of agents that are reading the same last value from any particular water monitoring station would grow and eventually overwhelm the SOS's providing the observed values.

A better, more scalable approach would be to have a single process reading the last observed value from each water monitoring station and storing that value is a cache. That process is called the Harvester in the CHISP-1 project. The Harvester reads the last observed value from each SOS – independent of any particular subscription – and caches that value. More than one Harvester can be run in parallel, each reading last observed values from different sets of water monitoring stations, and thus scale as the number of stations that need to be monitored increases. The event notification system can then read the last observed value from the cache and perform its threshold checks to determine if an event of interest has occurred.

For the CHISP-1 project, the initial implementation proposal for the last value cache was a lightweight REST-based service that used a JSON payload to encode the last value. This proposal however, was abandoned in favor of a more standards based approach using the OGC catalogue (see OGC 07-006r1). Since the catalogue was already being used to stored metadata about each water monitoring station (see 6.2.4), it was decided to simply add the last observed value as part of the metadata and the event notification service could simple access that value using the standard catalogue GetRecord request.

The catalogue approach is a more heavyweight approach than the initially proposed REST-based caching service but it was decided to push the OGC technology and see if it could satisfy the requirements of the event notification system. For the CHISP-1 projects using the catalogue for the last value cache worked perfectly. However, due to resource limitation in the project, the system was not sufficiently stress tested, with a large number of subscribers, to determine if indeed an OGC catalogue could act as a suitable last-value cache in most real-world situations with hundred or perhaps thousands of subscribers.

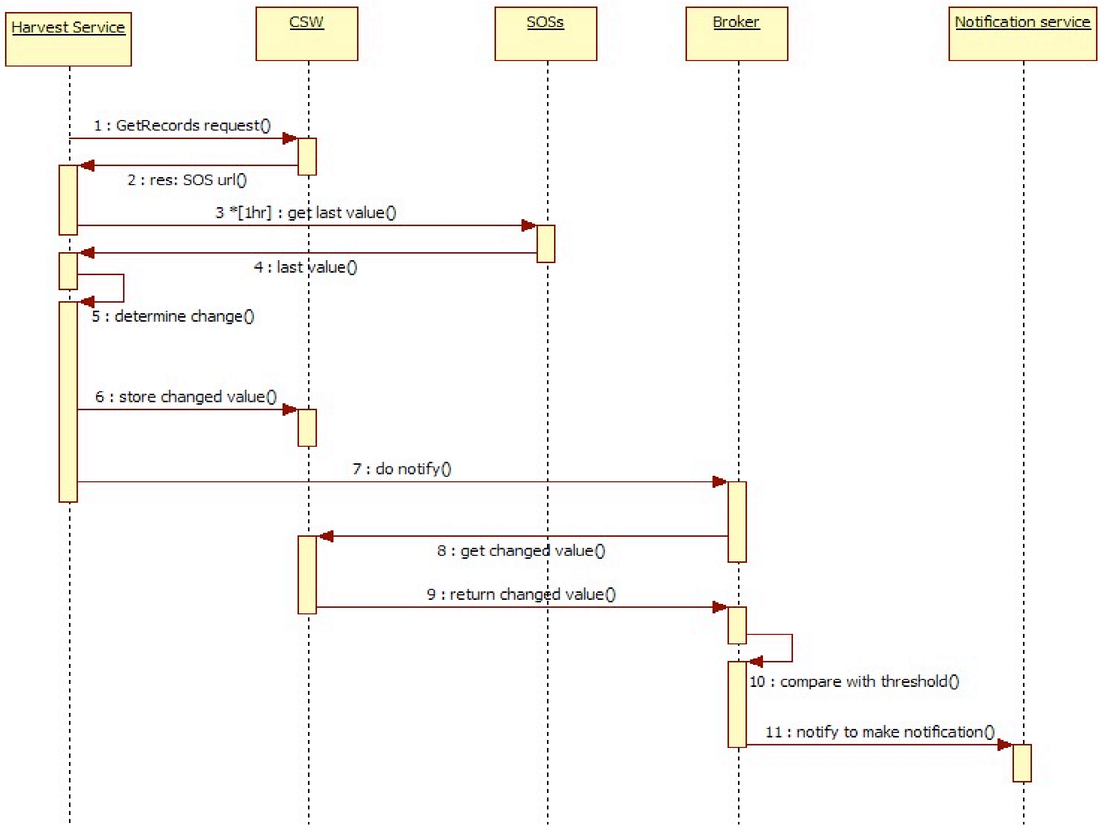


Figure 13 - Sequence diagram of Harvester and its operations

Figure 13 illustrates, in detail, the operation of the harvester. The sequence of actions is as follows:

- Initially the Harvester uses a GetRecords request to obtain the URLs for the list of available sensor observations services from the CSW.
- For each SOS obtained from the CSW:
    - The Harvester executes a GetObservation request to read the last observed value. The parameters for the GetObservation request are determined using the harvester algorithm described in the NRCan profile (see 4.3.3).
    - The Harvester then compares the previous "last value", stored in the CSW, with the value that it just read.
    - If the values are different, the Harvester updates the metadata in the CSW with this new value and notifies the subscription broker that the value has changed.
    - When notified by the Harvester, the subscription broker will read the changed value from the CSW and for each subscription will:
        - Compare the changed value with the threshold value defined for each subscription
        - If the changed value exceeds the threshold, the broker will generate the content of a notification message for that subscription and deliver that notification message to the WNS to actually do the notification.

6.3.4.2 Subscription broker

6.3.4.2.1 Introduction

The primary functions of the subscription broker are to manage subscriptions to the flood monitoring system and to notify subscribers when an event of interest (e.g. flood) occurs.

The primary user interface to the broker's API is the web subscription client (see 6.5) which calls the broker to execute the actions performed by the user. Specifically, the broker supports the subscription process whereby subscriptions are created; the update process whereby existing subscriptions are modified; the unsubscribe process whereby subscriptions are canceled; and the notification process whereby subscribers are notified that the event of interest has occurred. The broker also supports user (i.e. subscriber) management operations that are currently only accessible through the broker test harness (see 6.3.2.2).

This clause describes the operations implemented by the broker that support the capabilities offered by the web subscription client. It also describes the operations implements in the broker to support the management of subscribers.

6.3.4.2.2 Subscribe process

A subscription declares an emergency management analyst's desire to be notified by the ENS when an event of interest occurs. In the CHISP-1 project, subscriptions are related to an emergency management analyst though his/her email address.

Figure 14 illustrates the sequence diagram for the subscription process. An EMA, logs into the web subscription client (see 6.5) using his/her email address. The EMA can then subscribe to a specific water monitoring station or to a point of interest on the map. In either case, the EMA also provides additional subscription parameters (see 5.3.4.3) that include threshold and frequency values for the subscription. The web subscription client then uses these user-supplied values to execute the Subscribe operation of the ENS and thus create the subscription.

If the subscription is for a point on interest, the broker automatically determines the entire set of upstream stations that are in the same catchment by calling the upstream WPS (see 6.1). Once the broker receives a response from the upstream WPS, it generates an email to the subscriber that reports the status of the subscription. Note, that it is a valid response for the WPS to report no upstream stations for a point of interest.

The Subscribe operation of the ENS is the means by which a subscription – to a point of interest or specific water monitoring stations – is created. Table 5 defines the parameters of the Subscribe operation:

Table 5 – Subscribe operation parameters

Parameter Name	Description	Value type
op	Operation name	String (fixed "Subscribe")
email	email to which notifications will be sent	String
swLevelThreshold	Surface water level threshold.	Number
swLevelThresholdUnit	Unit of measure for the value of the swLevelThreshold parameter.	String (one of "Meters" or "Feet")

swFlowThreshold	Surface water flow threshold	Number
swFlowThresholdUnit	Unit of measure for the value of the swFlowThreshold parameters.	String (one of CubicMetersPerDay or CubicFeedPerSecond)
frequency	The event notification frequency AFTER an event of interest, such as a flood, has been detected.	Number (in minutes)
lat (mutually exclusive with stationId parameter)	Latitudes of the subscription point of interest.	Number
lng (mutually exclusive with stationId parameter)	Longitude of the subscription point of interest	Number
stationId (mutually exclusive with lat and lng parameters)	Identifier for the subscription station of interest.	String

The response to a Subscribe operation is a JSON document containing the details of the response or an exception message, again encoded using JSON.

The following javascript fragments show how to create a subscription using a point or interest or specifying a specific station id.

```
/*
 *subscribe by position
 */
$.ajax({
url : "http://59.125.87.213/WNS/Broker/RegisterInfo.ashx?op=subscribe",
type : 'post',
dataType : 'json',
cache : false,
data : JSON.stringify([
{"email":"user1@test.com","swLevelThreshold":11.5,"swLevelThresholdUnit
":"Meters","swFlowThreshold":10.5,"swFlowThresholdUnit":"CubicMetersPer
Day","frequency":15,"lat":48.257599,"lng":-
100.511856},{"email":"user1@test.com","swLevelThreshold":13,"swLevelThr
esholdUnit":"Feet","swFlowThreshold":13,"swFlowThresholdUnit":"CubicFee
tPerSecond","frequency":20,"lat":49.37833023,"lng":-100.78943634}]),
success : function (obj, status, jqXHR) {
             if (obj.poiIDs == null) {
                alert(obj.code+"\n"+obj.message);
             } else {
                alert(obj.code+"\nIDs:"+obj.poiIDs.join(","));}
          },
error : function (req, message) {
            alert(req.statusText);
         }

});

/*
 * subscribe by station id
 */
$.ajax({
url:"http://59.125.87.213/WNS/Broker/RegisterInfo.ashx?op=subscribe" ,
type : 'post',
dataType : 'json',
cache : false,
data : JSON.stringify([
{"email":"user1@test.com","swLevelThreshold":11.5,"swLevelThresholdUnit
":"Meters","swFlowThreshold":10.5,"swFlowThresholdUnit":"CubicMetersPer
Day","frequency":15,"stationID":"ca.gc.ec.station.05NB007"},{"email":"u
ser1@test.com","swLevelThreshold":13,"swLevelThresholdUnit":"Feet","swF
lowThreshold":13,"swFlowThresholdUnit":"CubicFeetPerSecond","frequency"
:20,"stationID":"05116000"}]),
success : function (obj) {
              if (obj.poiIDs == null) {
                 alert(obj.code+"\n"+obj.message);
              } else {
                 alert(obj.code+"\nIDs:"+obj.poiIDs.join(","));}
           },
error : function (req, message) {
            alert(req.statusText);
         }

});
```

The following JSON documents show a successful response by the ENS to a Subscribe operation.

```
// return sample of subscribing with a position
[{"email":"user1@test.com",
  "frequency":15,
  "swLevelThreshold":11.5,
  "swLevelThresholdUnit":"Meters",
  "swFlowThreshold":10.5,
  "swFlowThresholdUnit":"CubicMetersPerDay",
  "lat":48.257599, "lng":-100.511856,
  "poiType": "P",
  "status": "Pending",
  "poiID": 58},
 {"email":"user1@test.com",
  "frequency":20,
  "swLevelThreshold":13,
  "swLevelThresholdUnit":"Feet",
  "swFlowThreshold":13,
  "swFlowThresholdUnit":"CubicFeetPerSecond",
  "lat":49.37833023,
  "lng":-100.78943634,
  "poiType": "P",
  "status": "Pending",
  "poiID": 59}
]

// return sample of subscribing with a station ID
[{"email":"user1@test.com",
  "frequency":15,
  "swLevelThreshold":11.5,
  "swLevelThresholdUnit":"Meters",
  "swFlowThreshold":10.5,
  "swFlowThresholdUnit":"CubicMetersPerDay",
  "stationID":"ca.gc.ec.station.05NB007",
  "poiType": "S",
  "status": "Pending",
  "poiID": 58},
 {"email":"user1@test.com",
  "frequency":20,
  "swLevelThreshold":13,
  "swLevelThresholdUnit":"Feet",
  "swFlowThreshold":13,
  "swFlowThresholdUnit":"CubicFeetPerSecond",
  "stationID":"05116000",
  "poiType": "S",
  "status": "Pending",
  "poiID": 59}
]
```

The following JSON document shows an exception response by the ENS to a Subscribe operation that has failed.

```
// return sample of failed subscribing
{"code":"EmailMustBeSet",
 "message":"Emails of subscription requests should be set."}
```

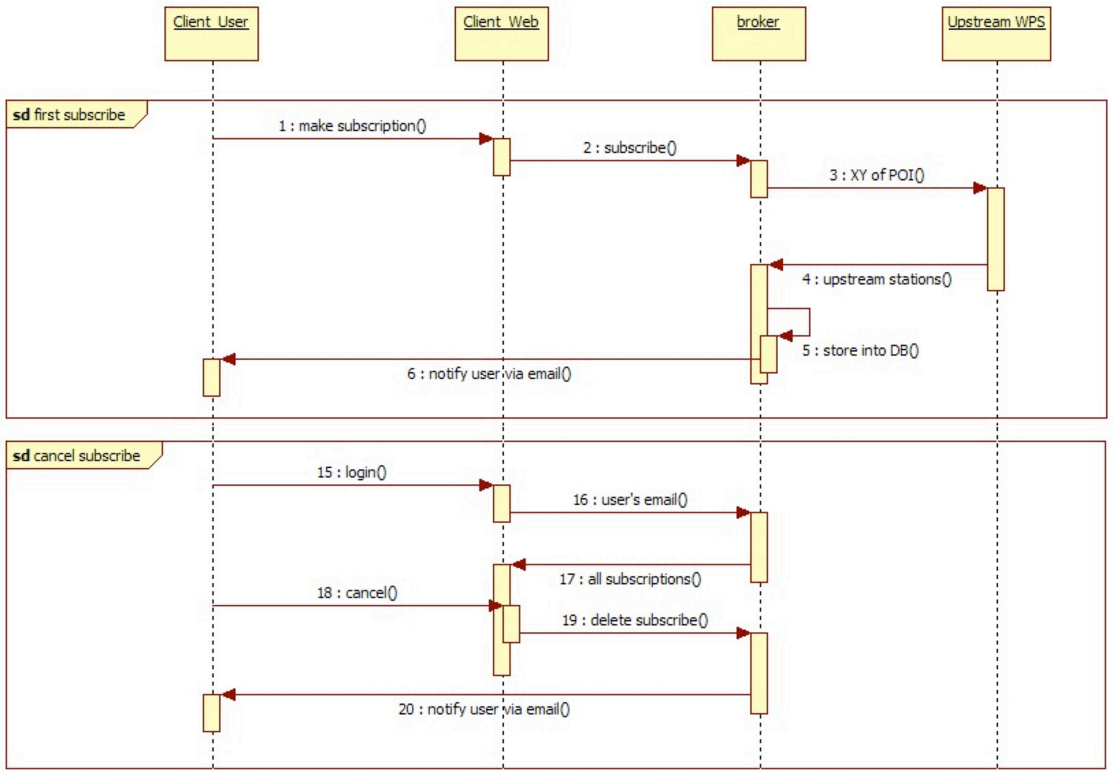


Figure 14 - Sequence diagrams for Subscribe and Unsubscribe processes

6.3.4.2.3 Unsubscribe process

The last sequence diagram in Figure 14 illustrates the steps the web subscription client must perform in order to cancel a subscription. After the EMS logs onto the system, the of process of unsubscribing involves two steps. First, the web based subscription client must determine which subscriptions the EMA. Second the web base subscription client must let the EMA select which subscription to cancel.

The GetUserSubscription operation is used to determine a EMA's subscriptions. Table 6 defines the parameters of the GetUserSubscription operation.

Table 6 – GetUserSubscription operation parameters

Parameter Name	Description	Value type
op	Operation name	String (fixed "GetUserSubscription")
email	email to which notifications will be sent	String

The following URL is an example of the GetUserSubscription operation:

```
http://59.125.87.213/WNS/Broker/RegisterInfo.ashx?op=GetUserSubscriptio
n&email=user1@test.com
```

and the following JSON document is an example of the response that such a request might generate.

```
[{"email":"user1@test.com",
  "poiID":0,
  "frequency":15,
  "swLevelThreshold":11.5,
  "swLevelThresholdUnit":"Meters",
  "swFlowThreshold":10.5,
  "swFlowThresholdUnit":"CubicMetersPerDay",
  "poiType": "S",
  "stationID": "ca.gc.ec.station.05NB007",
  "status": "Valid" },
 {"email":"user1@test.com",
  "poiID":1,
  "frequency":20,
  "swLevelThreshold":13,
  "swLevelThresholdUnit":"Feet",
  "swFlowThreshold":13,
  "swFlowThresholdUnit":"CubicFeetPerSecond",
  "poiType":"P",
  "lat":48.257599,
  "lng":-100.511856,
  "status": "Pending" },
 {"email":"user1@test.com",
  "poiID":2,
  "frequency":15,
  "swLevelThreshold":11.5,
  "swLevelThresholdUnit":"Meters",
  "swFlowThreshold":10.5,
  "swFlowThresholdUnit":"CubicMetersPerDay",
  "poiType":"P",
  "lat":49.37833023,
  "lng":-100.78943634,
  "status":"Invalid"}
]
```

The following table describes the fields of the return document.

Table 7 – GetUserSubscription response parameters

Parameter Name	Description	Type/Domain

email	User’s email address.	String
poiID	The subscription identifier	String
frequency	The even notification frequency after the event has been detected	Number (in minutes)
swLevelThreshold	The surface water level threshold set when the subscription was created	Number
swLevelThresholdUnit	The units used to express the value of the swLevelThreshold parameter	String (one of meter or feet)
swFlowThreshold	The surface water flow threshold set when the subscription was created.	Number
swFlowThresholdUnit	The units used to express the value of the swFlowThreshold value (one of CubeMeterPerDate or CubicFeetPerSecond)	String (one of CubicMetersPerDay or CubicFeetPerSecond)
poiType	Whether the point of interest is a water monitoring station or some arbitrary point on a map.	String (one of “S” or “P”)
lat	If poiType=”P” then this is the latitude of the subscription point	Number (in WGS84)
lng	If poiType=”P” then this is the longitude of the subscription point	Number (in WGS84)
stationId	If poiType=”S” then this is the id of the subscription station	String
Status	The status of the subscription. The subscription can be Valid, meaning that this is an active subscription; Pending meaning that the Subscribe request has been made and a response from the ENS is pending; Invalid meaning that the subscription point is not valid.	String (one of “Pending”, “Valid” or “Invalid”)

The EMA can then pick one or more subscriptions to cancel. The subscriptions are identified using the poiID value obtained from the response of the GetUserSubscription operation.

Table 8 defines the parameters of the UnSubscribe operation.

Table 8 – Unsubscribe operation parameters

Parameter Name	Description	Value type

email	Email to which notifications will be sent	String
poiID	Point of interest identifier as determined using the GetUserSubscription operation.	List of String.

The response to an UnSubscribe operation is a JSON document indicating acknowledging the success of the operation or an exception message.

The following javascript fragment illustrates how the web subscription client might invoke the Unsubscribe operation:

```
$.ajax({
    url :
"http://59.125.87.213/WNS/Broker/RegisterInfo.ashx?op=Unsubscribe&email
=user1@test.com" ,
    type : 'post',
    dataType : 'json',
    cache : false,
    data : JSON.stringify([21, 54, 34]), // the poi id
    success : function (obj) { alert(obj.code); },
    error : function (req, message) { alert(req.statusText); }
});
```

and the following JSON document represent a successful response.

```
{success : {"code":"Success"}}
```

6.3.4.2.4 Subscriber management

6.3.4.2.4.1 Introduction

The broker implements two operations, RegisterUser and GetUser, for the management is subscribers. These functions allow a user to be registered with the system and then subsequently get the user profile including their role within the system. The role of the user is important because only certain user roles have the ability to invoke a CAP alter (see Clause 2, Common Alerting Protocol).

6.3.4.2.4.2 RegisterUser operation

The RegisterUser operation is used to create and assign a role to a user within the event notification system. The CHISP-1 project defines two roles. The “Normal” role is the role assigned to an EMA without the privileges necessary to invoke a CAP alert. The “EM” role is the role assigned to an EMS with the necessary privileges to send a CAP alert using the MASAS hub. As a result, a notification message sent to an EMA with the “EM” role would include a link to the MASAS posting tool (see Figure 10).

Table 9 defines the parameters of the RegisterUser operation.

Table 9 – RegisterUser operation parameters

Parameter Name	Description	Value type
op	Operation name	String (fixed value of RegisterUser)
email	The subscriber's email address.	String
role	The role the user has within the ENS.	String (one of "Normal" or "EM")

The response to a RegisterUser operation is a JSON document echoing the detailed of the registered user or an exception message, again encoded as a JSON document.

The following javascript fragment illustrates how to use the RegisterUser function may be invoked:

```
var req = {"email" : "user1@test.com", "role" : "EM"};
$.ajax({
    url: "RegisterInfo.ashx?op=registerUser",
    type: 'post',
    dataType: 'json',
    cache: false,
    data: JSON.stringify(req),
    success: function (obj, status, jqXHR) {
        alert(jqXHR.responseText);
    },
    error: function (req, message) {
        alert(req.statusText);
    }
});
```

And the following JSON fragment is the response:

```
{"email":"user1@test.com", "role":"EM"}
```

6.3.4.2.4.3 GetUser operation

The GetUser operation may be used to determine a user's role within the system. In the CHISP-1 project this function was used primarily to determine if the notification email sent to a subscriber (see 6.3.4.2.5) should include a link to invoke a CAP alter or not.

Table 10 defines the parameters of the GetUser operation:

Table 10 – GetUser operation parameters

Parameter Name	Description	Value type
op	Operation name	String (fixed value of GetUser)
email	The subscriber’s email address.	String

The response to a GetUser operation is a JSON document containing the profile of the user within the ENS or an exception message, again encoded as a JSON document.

The following URL is an example of the GetUser operation:

```
http://59.125.87.213/WNS/Broker/RegisterInfo.ashx?op=GetUser&email=user
1@test.com
```

and the following JSON fragment is a sample response:

```
{"email":"user1@test.com", "role":"EM"}
```

6.3.4.2.5 Notification process

The notification process is the logic that the broker executes to determine if a notification is required. Figure 15 illustrates the sequence document for the notification process.

Once broker gets the last observed value for a station from the CSW, the broker will compare that value with the threshold defined when a subscription is created. In this way the broker filters the stations that exceed the threshold and then prepares a notification message. The content of the notification message includes:

- The station location or POI to which the user subscribed when the subscription was originally created.
- The water level threshold and water flow thresholds specified by the user when the subscription was created
- The locations of the stations that have exceeded the thresholds are presented on a map showing the entire upstream extent using the Google Map API.
- A table containing a list of **<u>all</u>** upstream stations and their last observed value. The stations exceeding the thresholds highlighted in RED within the table (see Figure 9).

- If the user's role within the ENS is "EM", the notification message will also include a link to invoke the CAP Alert GUI.

Figure 9 illustrates an example notification message. The notification message is then sent to the web notification service that does the subscriber notification via email. The WNS also offers an ATOM and RSS feed that contains the notifications that can be monitored by a subscriber using their favourite feed reader.

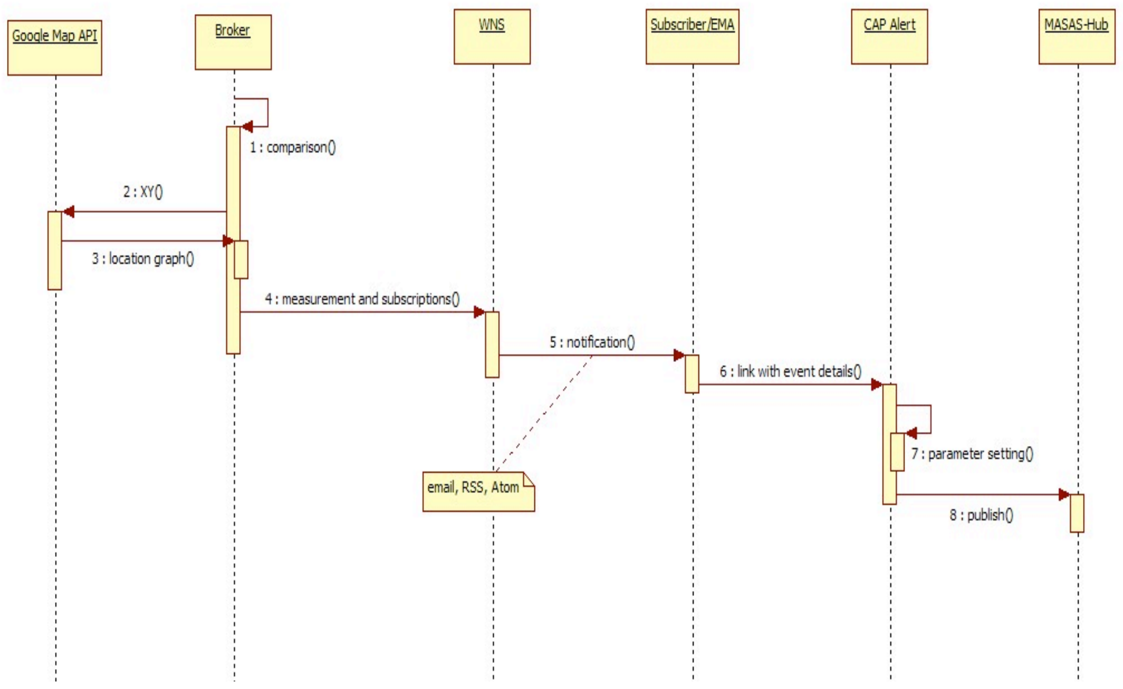


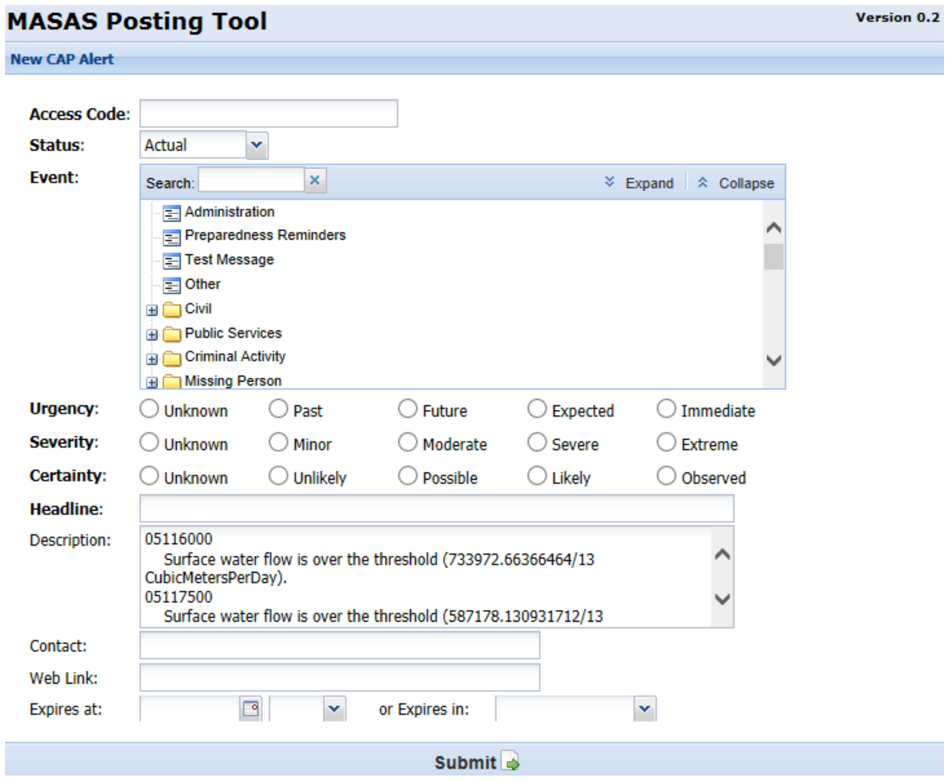
Figure 15 - Sequence diagram for notification process

If the user's role within the system is "EM" the notification message will include a link to invoke the CAP Alter GUI. Figure 15 also include the sequence of event should this link be invoked.

When the CAP Alter link is clicked the user is presented with a dialogue box shown in Figure 16. Using this form, the user entire the following information:

- The user's MASAS access code
- The status of the Event
- The type of Event
- The urgency of the event

- The severity of the event
- A certainty classification for the event
- A human-readable headline for the even
- A more detailed human-readable description of the event
- Contact information for the EMA.
- An optional web link providing more information about the event
- An expiry time or period.



05116000
Surface water flow is over the threshold (733972.66366464/13 CubicMetersPerDay).
05117500
Surface water flow is over the threshold (587178.130931712/13

Figure 16 – MASAS posting tool

Clicking the "Sumbit" button causes the alter to be published to the MASAS hub.

6.3.4.3 Web notification service

The web notification component within the ENS is an implementation of OGC's Web Notification Service (see OGC 06-095r1) and its primary function within the system is to send notification emails to subscribers once the broker (see 6.3.4.2) has determines that a notification is required.

6.3.5 Issues

The primary issue encountered implementing the ENS was related to the stability and performance of the other components of the CHISP-1 project and in particular the stability and performance of the sensor observation services offering data from the water monitoring stations. The Harvester module, for example, was particularly sensitive to performance issues because of its periodic access to the sensor observation services.

In this kind of framework, the broker – the element that needs to integrate and communicate with other components – must consider a lot of exception in order to avoid connection problems.

6.4 GetDataAvailability

In this pilot, GIS.FCU applied 52North code base to implement NRCan SOS GDA extension which follow the requirement of NRCan's GIN profile (https://portal.opengeospatial.org/wiki/CHISP1/CHISP1GINSOSProfile).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<sos:GetDataAvailability
    xmlns:sos="http://www.opengis.net/sos/2.0" service="SOS" version="2.0.0">
    <sos:observedProperty>test_observable_property_1</sos:observedProperty>
    <sos:observedProperty>test_observable_property_2</sos:observedProperty>
    <sos:observedProperty>test_observable_property_3</sos:observedProperty>
    <sos:observedProperty>test_observable_property_4</sos:observedProperty>
    <sos:observedProperty>test_observable_property_5</sos:observedProperty>
    <sos:featureOfInterest>test_feature_1</sos:featureOfInterest>
    <sos:featureOfInterest>test_feature_4</sos:featureOfInterest>
    <sos:procedure>http://www.example.org/sensors/101</sos:procedure>
    <sos:procedure>http://www.example.org/sensors/104</sos:procedure>
</sos:GetDataAvailability>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<sos:GetDataAvailabilityResponse
  xmlns:sos="http://www.opengis.net/sos/2.0"
  xmlns:om="http://www.opengis.net/om/2.0"
  xmlns:gml="http://www.opengis.net/gml"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:swes="http://www.opengis.net/swes/2.0">
  <sos:featurePropertyRelationShip>
    <sos:featurePropertyTemporalRelationShip>
      <om:phenomenonTime>
        <gml:TimePeriod gml:id="tp_1">
          <gml:beginPosition>2012-11-19T13:00:00.000+08:00</gml:beginPosition>
          <gml:endPosition>2012-11-19T13:20:00.000+08:00</gml:endPosition>
        </gml:TimePeriod>
      </om:phenomenonTime>
      <sos:targetFeature xlink:href="#test_feature_1"/>
      <sos:targetProperty xlink:href="#test_observable_property_1"/>
    </sos:featurePropertyTemporalRelationShip>
  </sos:featurePropertyRelationShip>
  <sos:propertyEntryPoint>
    <sos:observedPropertyInfo swes:id="test_observable_property_1">
      <sos:property>"http://www.example.org/sensors/101"</sos:property>
    </sos:observedPropertyInfo>
  </sos:propertyEntryPoint>
  <sos:featureOfInterestEntryPoint>
    <sos:featureOfInterestInfo swes:id="foi_test_feature_1">
      <sos:feature>test_feature_1</sos:feature>
      <sos:relatedProperty xlink:href="#test_observable_property_1"/>
    </sos:featureOfInterestInfo>
  </sos:featureOfInterestEntryPoint>
  <sos:featurePropertyRelationShip>
    <sos:featurePropertyTemporalRelationShip>
      <om:phenomenonTime>
        <gml:TimePeriod gml:id="tp_2">
          <gml:beginPosition>2012-11-19T13:00:00.000+08:00</gml:beginPosition>
          <gml:endPosition>2012-11-19T13:09:00.000+08:00</gml:endPosition>
        </gml:TimePeriod>
      </om:phenomenonTime>
      <sos:targetFeature xlink:href="#test_feature_4"/>
      <sos:targetProperty xlink:href="#test_observable_property_4"/>
    </sos:featurePropertyTemporalRelationShip>
  </sos:featurePropertyRelationShip>
  <sos:propertyEntryPoint>
    <sos:observedPropertyInfo swes:id="test_observable_property_4">
      <sos:property>"http://www.example.org/sensors/104"</sos:property>
    </sos:observedPropertyInfo>
  </sos:propertyEntryPoint>
  <sos:featureOfInterestEntryPoint>
    <sos:featureOfInterestInfo swes:id="foi_test_feature_4">
      <sos:feature>test_feature_4</sos:feature>
      <sos:relatedProperty xlink:href="#test_observable_property_4"/>
    </sos:featureOfInterestInfo>
  </sos:featureOfInterestEntryPoint>
</sos:GetDataAvailabilityResponse>
```

6.5 Subscription client

6.5.1 Introduction

The subscription client is the web-based user interface for the event notification system (see 6.3). It allows an emergency management analyst to view, on a map, the river network and the locations of water monitoring stations. The EM analyst can then chose one or more stations, or a point of interest on the map and subscribe to be notified if a flood event occurs at the chosen stations or upstream of the point of interest.

6.5.2 Client endpoint

The subscription client can be found at: http://184.73.217.132/map

6.5.3 Implementation details

The subscription client was built as a web application in javascript using the following components:

- The primary UX is based on Twitter's Bootstrap (see http://twitter.github.io/bootstrap/)

- The map component leverages Leaflet and associated plugins (see http://leafletjs.com/)

6.5.4 Operational details

Figure 17 shows the web-based subscription client. The subscription client is composed of a live map showing the river network and monitoring stations. The client implements the following features:

1. At the top right there is a login field where the EM analyst enters his/here email address and logs into the system. Subscription notifications will be sent to this email address.

2. In the upper right, on the map, there is a color-coded legend show the sources of data presented on the maps. Individual sources can be turned on an off by checking the box on the right within the legend. For the CHISP-1 project the following sources of data were presented: stream segments (see Table 1, Component 6), USGS surface water monitoring stations, Environment Canada (via NRCan) water monitoring stations, NRCan groundwater stations.

3. Water monitoring stations are shown on the map as color coded circles. The color of the circle corresponds to the data source presented in the legend.

4. A color coded circle may also contain a number indicating that at the current zoom level the circle represents more than one water monitoring station. The number represents the specific number of water monitoring stations represented. As the client zooms in, numbered circles will eventually separate out into individual water monitoring stations.

5. Mousing over a water monitoring station causes the web client to draw the catchment monitored by the station.

6. At the bottom of the map is a table that shows chosen subscription points. The columns of the table are:

    a. Actions – allows the EM analyst to Subscribe, Cancel, Update or Unsubscribe from the point of interest.

b. Show – allows the EM analyst to turn points of interest on and off on the map.

c. Description – shows the exact coordinate of the point of interest

d. Status – shows the subscription status of the point of interest. Possible values are:

    i. Selected – indicates that the point of interest has been chosen on the map but has not been subscribed to

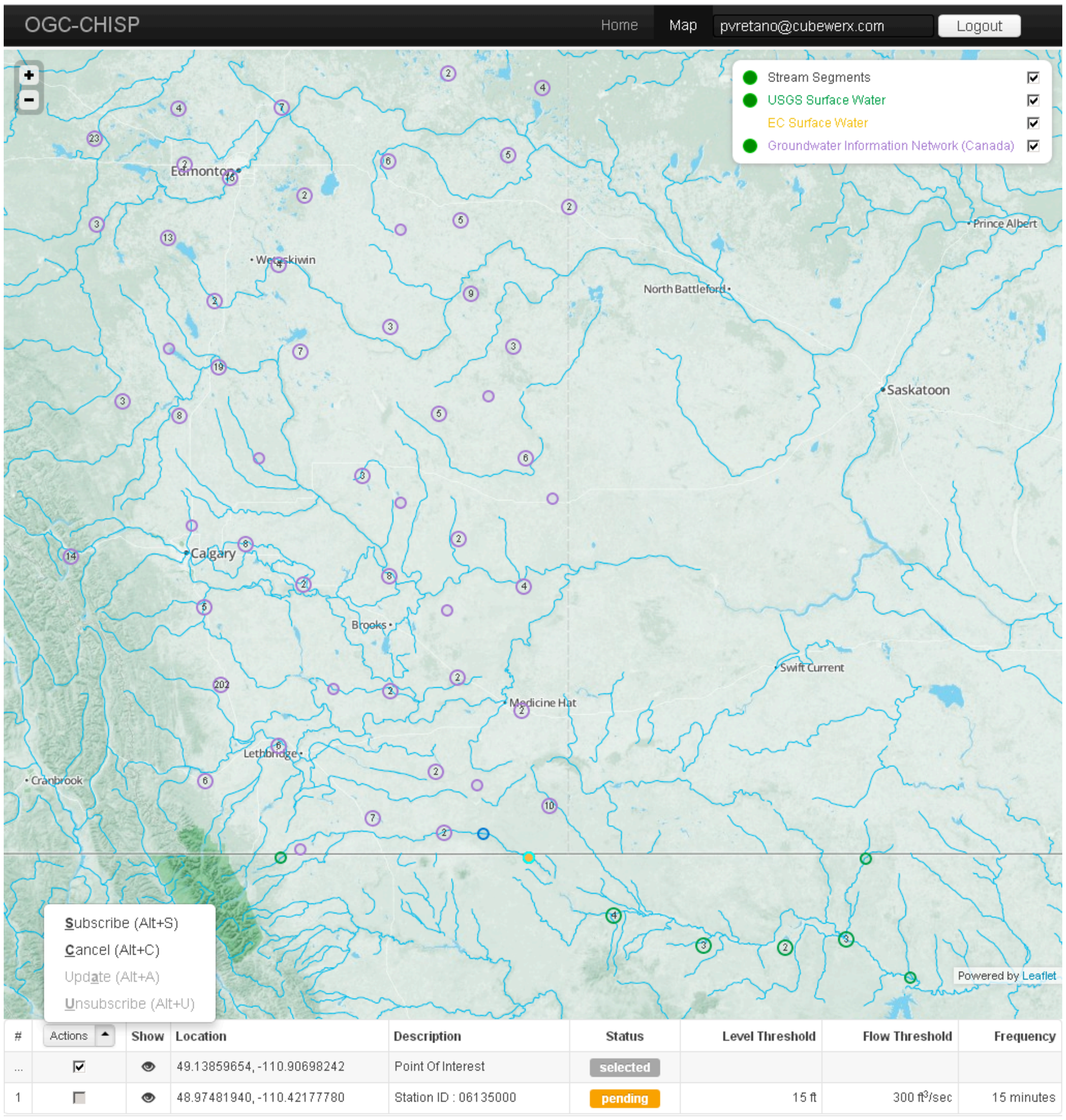
    ii. Pending – indicates that a subscription request has been made for the point of interest and the web client is waiting to get verification that the subscription has been registered

    iii. Subscribed – indicates that there is a registered subscription for the point of interest

e. Level threshold – if there is a registered subscription for the point of interest this column shows the water level threshold specified with the subscription

f. Flow threshold – if there is a registered subscription for the point of interest, this column shows the flow threshold specified with the subscription

g. Frequency – if there is a registered subscription for the point of interest, this column shows the notification frequency specified with the subscription



#	Actions	Show	Location	Description	Status	Level Threshold	Flow Threshold	Frequency
...	☑	👁	49.13859654, -110.90698242	Point Of Interest	selected			
1	☐	👁	48.97481940, -110.42177780	Station ID : 06135000	pending	15 ft	300 ft^3/sec	15 minutes

Figure 17 – Web subscription client

Having chosen a point of interest on the map, an EM analyst can click the “Subscribe” action in the “Actions” column (see Figure 17) to subscribe to that point. This action causes the web subscription client to present the EM analyst with the dialogue box shown in Figure 18.

The “New Subscription” dialogue allows the EM analyst to choose their preferred units of measure. The EM analyst can then enter threshold values for surface water level and surface water flow that will trigger the event notification system to send a notification message. Finally, the EM analyst can enter a frequency value that represents how often the EM analyst will receive notifications, with updated information, once an event has been triggered. So, for example, entering a value of 15 minutes will cause the event notification system to send the analyst an email every 15 minutes, after the flood event has been detected, with the lasted water level and flow data.

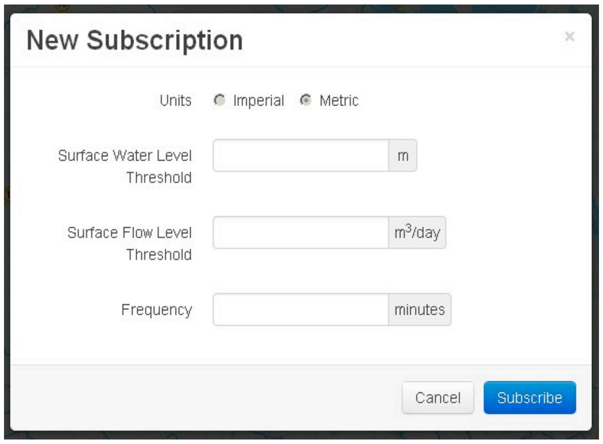


Figure 18 – Subscription dialogue

6.6 Water Quality SOS

6.6.1 Introduction

The water quality service layers an OGC compliant SOS API on top of a couple of non-OGC water quality data sources; one from Canada and one from the United States. Putting an SOS façade on top of these non-OGC sources allows their information to be easily accessed by other components in the CHISP-1 project that are implemented to OGC standards.

6.6.2 Service end point

The service endpoint for the water quality SOS can be found at:

http://sos.chisp1.asascience.com/sos?service=SOS&request=GetCapabilities

6.6.3 Implementation details

The water quality SOS was built in Python using the following libraries:

- Web development framework: Flask (http://flask.pocoo.org/)
- Geometric objects, predicates and operations: Shapely (https://pypi.python.org/pypi/Shapely)
- For making HTTP requests: requests (http://docs.python-requests.org)

- World time zone database in Python: pytz (http://pytz.sourceforge.net/)
- Extended date/time manipulation in Python: python-dateutil (https://pypi.python.org/pypi/python-dateutil)
- OGC Web Service Utility library: OWSLib (http://geopython.github.io/OWSLib/)
- A Python library for collecting Met/Ocean observations: pyoos (https://github.com/asascience-open/pyoos)
- XML and HTML processing in Python: lxml (http://lxml.de/)
- A python-based HTTP server: gunicorn used as a WSGI server for the service (http://gunicorn.org/)

6.6.4 Operational details

6.6.4.1 Introduction

The water quality SOS implements the following operations: GetCapabilities, DescribeSensor and GetObservation.

The water quality SOS does not implement all the operations required by the NRCan profile because the discussion about the profile during the CHISP-1 project was not completed before the server's implementation was completed. In addition, the nutrient load calculation use case does not use the other, unimplemented, operations from the profile.

6.6.4.2 Architecture

Figure 19 illustrates the architecture of the water quality SOS. One side the service implements an OGC compliant SOS v2.0 API (see OGC 12-006). On the other side, the service implements the necessary access methods to read information from the two water quality sources, convert the information into a compatible format – taking into account differences in units of measure, semantics, etc. -- and then serve it through the SOS API.

When the water quality server receives a GetObservation request it makes dynamic calls to waterqualitydata.us (for the US data) and a local database (for CAN Ontario Provence data). It then processes the responses and serves the information through the SOS API in IOOS SWE xml format. The server also offers the legacy formats txt/tsv and txt/csv.

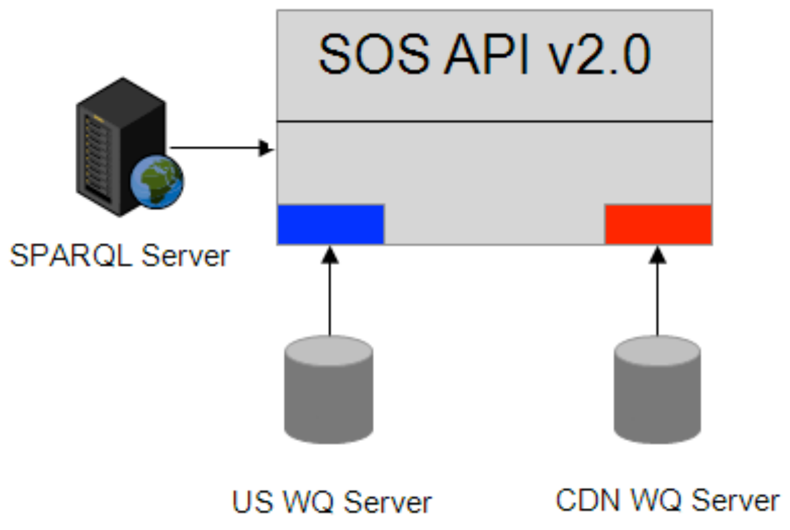


Figure 19 – Water Quality SOS Architecture

The WQ SOS architecture diagram (see Figure 19), also shows a SPARQL service. The intended function of this server was to provide semantic mediation services for analyte names. For example, the name for Nitrogen for a US station is "Nitrogen". The same name for a Canadian station would be "NNTKUR". The idea was to use the SPARQL server to determine the semantic equivalence between "Nitrogen" and "NNTKUR" this allowing a client to use either nutrient parameter name in a request to the server. Resource limitations within the CHISP-1 project, however, prevented the SPARQL server from being integrated into the system.

6.6.5 Examples

- Capabilities document:
  - http://sos.chisp1.asascience.com/sos?service=SOS&request=GetCapabilities
- DescribeSensor examples:
  - PWQMN station 04001309202
    - http://sos.chisp1.asascience.com/sos?service=SOS&request=DescribeSensor&version=1.0.0&outputformat=text/xml;subtype=%22sensorML/1.0.1%22&procedure=04001309202
  - WaterQualityData.us stations WIDNR_WQX-10001133

    - http://sos.chisp1.asascience.com/sos?service=SOS&request=DescribeSensor&version=1.0.0&outputformat=text/xml;subtype=%22sensorML/1.0.1%22&procedure=WIDNR_WQX-10001133

  - WaterQualityData.us station ISGS-04085427

    - http://sos.chisp1.asascience.com/sos?service=SOS&request=DescribeSensor&version=1.0.0&outputformat=text/xml;subtype=%22sensorML/1.0.1%22&procedure=USGS-04085427

- GetObservation examples:

  - All 'Mercury' data from WaterQualityData.us station USGS-04085427

    - http://sos.chisp1.asascience.com/sos?service=SOS&request=GetObservation&version=1.0.0&responseformat=text/xml;subtype=%22om/1.0.0%22&offering=network-all&observedProperty=Mercury&procedure=USGS-04085427

  - Mercury' data from WaterQualityData.us station USGS-04085427 between 1980 and 1990

    - http://sos.chisp1.asascience.com/sos?service=SOS&request=GetObservation&version=1.0.0&responseformat=text/xml;subtype=%22om/1.0.0%22&offering=network-all&observedProperty=Mercury&procedure=USGS-04085427&eventtime=1980-01-01T00:00:00Z/1990-01-01T00:00:00Z&

  - All Phosphorus and Nitrite data from PWQMN station 04001309202

    - http://sos.chisp1.asascience.com/sos?service=SOS&request=GetObservation&version=1.0.0&responseformat=text/xml;subtype=%22om/1.0.0%22&offering=network-all&observedProperty=PPUT,NNO2UR&procedure=04001309202

  - Phosphorus and Nitrite data from PWQMN station 04001309202 between 2005 and 2010

    - http://sos.chisp1.asascience.com/sos?service=SOS&request=GetObservation&version=1.0.0&responseformat=text/xml;subtype=%22om/1.0.0%22&offering=network-all&observedProperty=PPUT,NNO2UR&procedure=04001309202&eventtime=2005-01-01/2010-01-01

6.6.6 Issues

The SOS specification needs to be reconsidered for these kinds of use cases: enormous networks of sensors aren't sufficiently supported. Listing all procedures and features of interest is often not manageable - particularly in large dynamic SOS systems

6.7 Nutrient Load Calculation WPS

6.7.1 Introduction

This clause describes the details of the components developed for the nutrient load calculation use case. Specifically it discusses the web client built to allow a water quality analyst run a nutrient load calculation and the server implemented to make the calculation model web accessible via the WPS (see OGC 05-005r1) API.

As described in clause 5.4, the use case also uses the water quality SOS developed during the CHISP-1 project which is describe in Clause 6.6.

6.7.2 Client

6.7.2.1 Client endpoint

The client endpoint for the NLCS client can be found at:
http://client.chisp1.asascience.com/

6.7.2.2 Implementation details

The client was built as a browser application in javascript using the following libraries:

- HTML manipulation: jquery (http://jquery.com/)
- UI components: jquery-ui (http://jqueryui.com/)
- Vector graphics: jquery-svg (http://archive.plugins.jquery.com/project/svg)
- HTML data library: d3js (http://d3js.org/)
- CSS activity indicator: spin (http://www.myjqueryplugins.com/jquery-plugin/spinjs)
- Statistics module: geostats (https://github.com/simogeo/geostats)

6.7.2.3 Operational details

Figure 20 illustrates the client build for the nutrient load calculation use case. The client allows a user to select a Great Lake of interest, a nutrient of interest and a time period of interest and then invokes the nutrient load calculation WPS with those arguments.

The NCLS returns a jsonp type response with its typical XML response as a payload. This allows the client to communicate directly with server avoiding the need to proxy the NLCS WPS response.

No technical challenges or recommendations resulted from implementing the client.

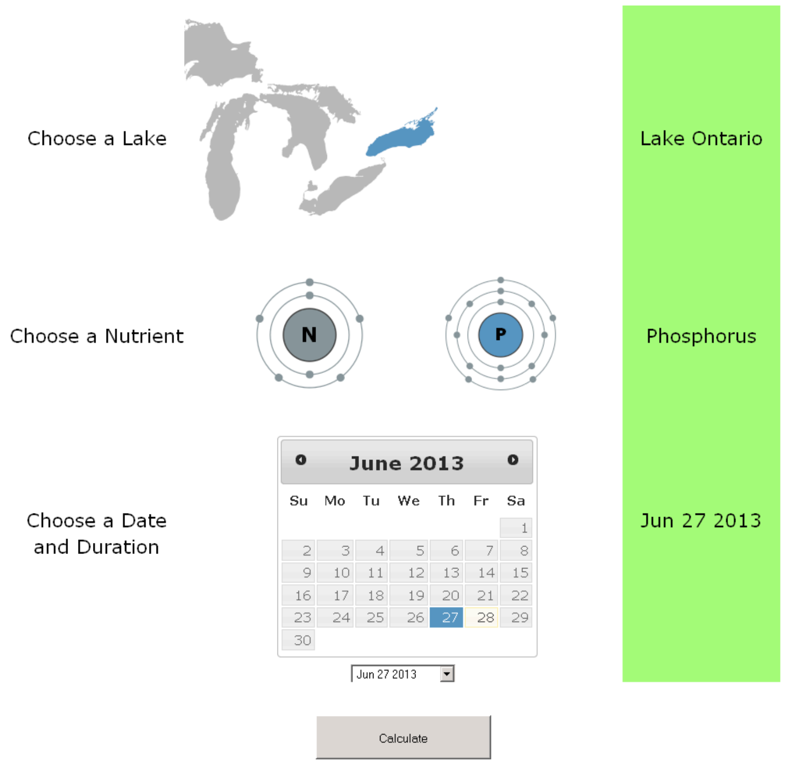


Figure 20 – Nutrient Load Calculation Client

6.7.3 Server

6.7.3.1 Service endpoint

The service endpoint for the nutrient load calculation service can be found at:

http://64.72.74.103:8080/nlcs/?request=GetCapabilities&version=1.0.0

6.7.3.2 Implementation details

The nutrient load calculation WPS was built in Python using a custom developed WPS framework for flexibility and depends on the following libraries:

- For scientific computing: numpy (http://www.numpy.org)
- Web framework: django (http://www.djangoproject.com)
- For making HTTP requests: requests (http://docs.python-requests.org)
- A python-based HTTP server: gunicorn (http://www.gunicorn.org)
    - was used as a WSGI server for the service

Due to a number of implementation and data issues (see 6.7.3.5) a simplified nutrient load calculation model was developed that could be used for both U.S. and Canadian data. The simplified model does the following steps to calculate the nutrient load for a given lake:

- Query a local catalogue for tributaries on the lake of interest that have both a stream gauge and water quality samples available for the nutrient of interest.
- Access data from the water quality and stream flow SOS services for the stations returned by the catalogue query.
- Union the time series of water quality samples if more than 1 water quality station is found.
- Interpolate the water quality and stream flow measurements of the period specified by the WQ analyst's request.
- Calculates nutrient flux from the interpolated measurements and numerically integrate the results to determine the total load over the period of interest for each tributary
- Sums all of the tributaries' contributions for the total load of the lake, converting units between standard and SI where appropriate

6.7.3.3 Operational details

6.7.3.3.1 Introduction

The nutrient load calculation WPS implements the following operations: GetCapabilities, DescribeProcess and Execute.

The nutrient load calculation WPS offers the following methods or processed: calc_nutrient_load

6.7.3.3.2 Process calc_nutrient_load

The calc_nutrient_load method computes the nutrient load on a Great Lake using a simplified calculation model (see 6.7.3.2).

The process accepts as input the name of a Great Lake, a date interest, a nutrient of interest, and a duration of interest (one of "year", "month" or "day").

The response is either an XML document containing the results of calculation or an exception message is a problem was encountered.

6.7.3.4 Examples

- Capabilities document:
  - http://64.72.74.103:8080/nlcs/?service=WPS&version=1.0.0&request=GetCapabilities
- Process descriptions:
  - http://64.72.74.103:8080/nlcs/?service=WPS&version=1.0.0&request=DescribeProcess&identifier=all

6.7.3.5 Issues

The original intent was to use the "rpy" (http://rpy.sourceforge.net/) or "rpy2" (http://rpy.sourceforge.net/rpy2.html) libraries to interface with the USGS EGRET WRTDS Nutrient Load model written in R (see https://github.com/USGS-CIDA/WRTDS). However a couple of problems were encountered that ultimately lead to abandoning the use of the WRTDS model.

First, a lot of effort was expended writing middle wear to help move the correct kinds of objects between R and Python and ultimately some bugs and limitations in both the rpy libraries and some c libraries that the python service was depending on made it impossible within the given time to track down all of the problems and resolve them.

Second, the Canadian data was not suitable to drive the WRTDS model, due to its sparseness, and so a simplified calculation model would be required.

Taking both these issues into account it was decided to implement a simplified computation model for both countries' data (see 6.7.3.2).

The following additional challenges, not including the R model implementation, were encountered:

- using Python's urllib2 for requests to distributed services was hard to debug, switching to the *requests* library made the code easier to write, read and debug
- using *gunicorn* workers that depended on *libevent* had intermittent problems making requests to distributed services
    - this appears to be a known bug in *libevent* or the python bindings
    - switching to using tornado workers and the requests library seems to help stability
- sometimes requests that don't timeout the *gunicorn* WSGI server would timeout the nginx http server (see http://wiki.nginx.org), causing hung workers, defunct processes and slow performance as a whole
- moved the NLCS model work to an asynchronous style WPS and urge people to use the ip/port URL instead of the subdomain URL for both the NLCS WPS and the upstream service WPS (see 6.1)
- implemented a *jsonp* style response that wraps the typical xml WPS response so that web clients can communicate directly with the NLCS service without having to proxy