2008 NSDI Cooperative Agreement Program

Category 2: Best Practices in Geospatial Service Oriented Architecture (SOA)

Documenting Best Practices in Geospatial SOA: Wetlands JD Analyzer

Design
Version 1.1

Image Matters LLC
201 Loudoun St, SW Leesburg, VA 20175
Internet Address: http://www.imagemattersllc.com

June 29, 2009

Table of Contents

1. Introduction

1.1. Identification

This document is identified as the *EPA Wetlands JD Analyzer Application Design* document. The production and maintenance of this document is the responsibility of Image Matters, LLC. Approval of this document is the responsibility of the persons listed in the approvals table in the front portion of this document.

1.2. Scope

This document covers the design of the Wetlands JD Analyzer (JDA) application developed by Image Matters, LLC.

1.3. Reference Documents

1. Documenting Best Practices in Geospatial SOA: Wetlands Permitting Solution Use Cases, Version 0.4 (Draft).

2. Application Design

2.1. Overview

The Wetlands JD Analyzer Application (JDA) allows users to identify National Wetlands Inventory (NWI) wetlands and then process those wetlands against National Hydrography Dataset (NHD) streams to determine which streams intersect the wetlands or which streams are closest to – within a maximum proximity – each of the wetlands.

2.2. Requirements

2.2.1. Functional

1. Application must connect to and query the NWI Web Feature Service (WFS) to identify wetland features
2. Application must display wetland query results on a map
3. Application must connect to and query the NHD Web Feature Service (WFS) to identify stream features
4. Application must connect to and invoke a Web Processing Service (WPS) to perform intersection-based and proximity-based analysis of wetlands and streams
5. Application must display analysis results in tabular form and on a map
6. Application should provide the user the ability to select wetlands and streams from the map and display information about them
7. Application should allow the user to browse the wetland and stream features used in the analysis within a report detailing the results of the analysis

8. Application should allow the user to specify annotations on the map which are persisted with the report

2.2.2. System

1. Application must provide an easy-to-use user interface (UI) for identifying the wetland and stream features as well as acesssing and displaying analysis results

2.3. Architecture

Figure 1 illustrates the relationship of the components and external services used by the JDA application.

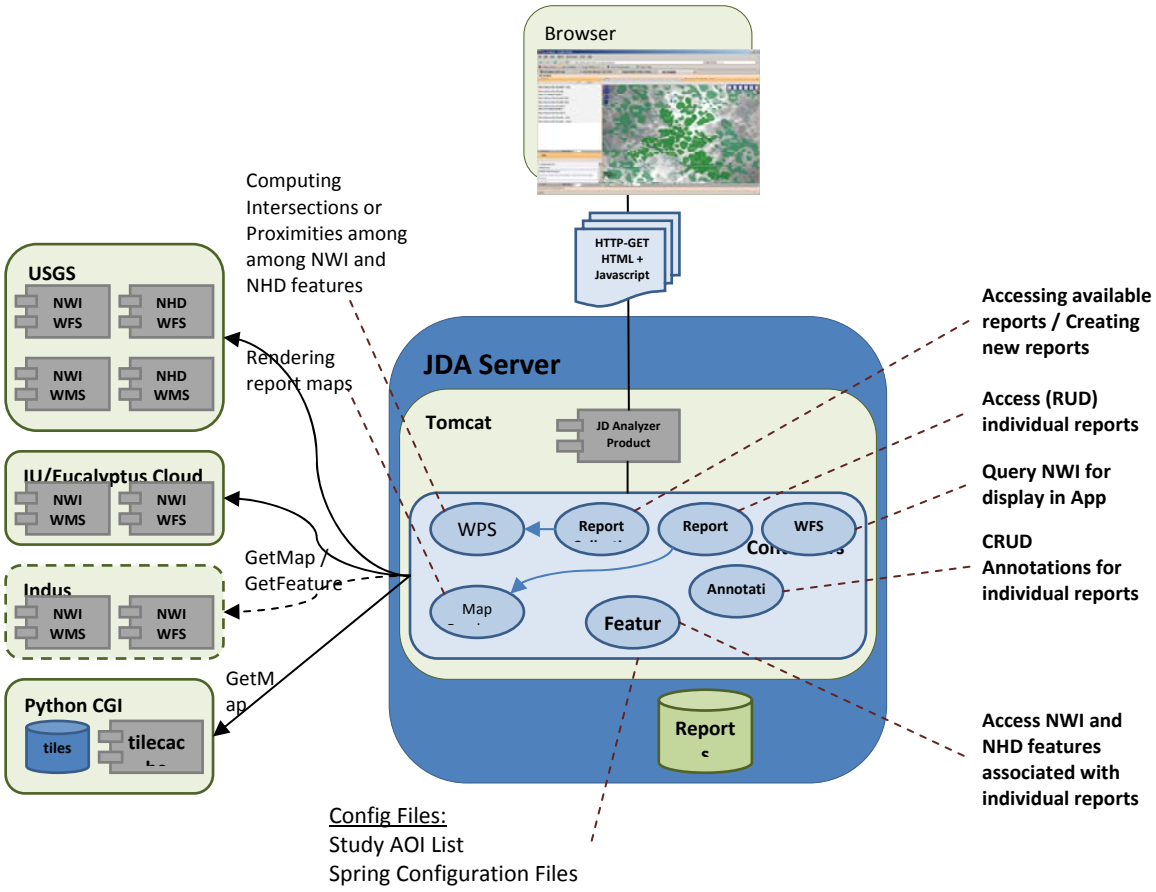


Figure 1: JDA Architecture Diagram

2.3.1. Server

The server-side components contain the necessary Java libraries to:

- Query and parse results from WFS and WPS

- Store analysis results as a “report”
- Present features and analysis results as Atom feeds in JSON format

2.3.2. Client

The client-side components, based on userSmarts™ Gx rich-client platform (RCP) libraries, provides a graphical user interface with interactive controls necessary to:

- Select and display wetland polygons on a map
- Invoke intersection and proximity analysis using previously-selected wetland polygons
- Display the results of analysis in tabular and graphical (on the map) form
- Annotate analysis results

2.4. Design

The following sections describe the design of the Wetlands JD Analyzer application, including critical decisions made regarding implementation.

2.4.1. Use Cases

Refer to the Use Case document (1.3.1).

2.4.2. Decisions

2.4.2.1 JD Reports

The usage of “JD reports” is absent from this design for simplicity of the application. Future versions may include support for these reports as a means for defining the analysis context, such as areas of interest and map layers.

2.4.2.2 Analysis Input

The inputs to the WPS include stream and wetland features which are made available through OGC WFS.

The WFS service endpoint for the NHD (streams) is:
http://frameworkwfs.usgs.gov/framework/wfs/wfs.cgi?

The WFS service endpoint for the NWI (wetlands), being used for demonstration purposes only, is:
http://indusspatial.induscorp.com/ArcGIS/services/INDUS/NWI/MapServer/WFSServer?

2.4.2.3 Analysis Results Format

The results of the intersection and proximity analysis, as reported by the WPS, is being represented as OGC features, conforming to the response generated by a WFS to retrieve a set of features. This decision was made to ensure that all responses originating from the server-side components regarding wetlands and streams would be represented in the same way.

2.4.3. Web Feature Service (WFS)

Wetlands JD Analyzer application connects to two WFS implementations to locate appropriate features (wetlands and streams) for performing analysis. The application makes use of the userSmarts™ connection libraries for communication with OGC web services, such as WFS. Below is sample Java code illustrating how the application issues a query to WFS and handles the response.

```
//instantiate the wfs client
WfsClient client = new WfsClient(
      "http://frameworkwfs.usgs.gov/framework/wfs/wfs.cgi?");

//create an empty query ("get all")
Expression query = new OrExpr();

//send query to the wfs for the "streams" feature type
Object result = client.getFeatures(query, "hyd:HydroElementFLHI");

//cast the response to a GFR object
GetFeatureResponse response = (GetFeatureResponse)result;

//get the list of features from the response
List<Feature> features = response.getFeatures();

//walk the list of features and write them out to the client
//...
```

2.4.4. Web Processing Service (WPS)

A Web Processing Service is used to perform the analysis of the wetland and stream features to determine intersections and proximities. WPS is a service which accepts one or more data inputs and provides some well-defined processing operations which can be performed either on the data inputs or on some other source of data using constraints defined by the input data. In the case of Geo-Analysis, the WPS is configured at run-time to communicate with both the NWI and NHD WFS services automatically, so the only data inputs necessary are those defining the processing constraints.

The WPS implementation used by the Wetlands JD Analyzer application defines two “processors” – Intersection and Proximity – which may be configured at deployment time

within the application's .WAR file using standard Spring Framework (http://www.springframework.org) conventions. Here is an example of how these processors are configured:

```
<bean id="wpsController"
            class="com.usersmarts.cx.wps.spi.WPSController">
      <property name="processors">
            <map>
                  <entry
key="gov.epa.geoanalysis.controller.ProximityProcessor">
                        <bean
class="gov.epa.geoanalysis.controller.ProximityProcessor">
                        </bean>
                  </entry>
                  <entry
key="gov.epa.geoanalysis.controller.IntersectionProcessor">
                        <bean
class="gov.epa.geoanalysis.controller.IntersectionProcessor">
                        </bean>
                  </entry>
            </map>
      </property>
</bean>
```

The application also makes use of the userSmarts™ connection libraries for communication with the WPS. Below is sample pseudocode which issues a proximity query to the WPS.

```
//instantiate the wps client
WpsClient wps = new WpsClient(
      "http://localhost:8080/epa-analysis/api/wps?");

//build new execute request
ExecuteRequest wpsRequest = new ExecuteRequest();

//specify the processor to use
wpsRequest.setIdentifier(ProximityProcessor.class.getName());

//specify the maximum proximity distance of 1000 meters
wpsRequest.getDataInputs().add(
      new Input("distance", new LiteralData("1000.0")));

//specify the bounding box of the wetlands query
// the WPS will use this to query for wetlands
Envelope envelope = new Envelope(-78.0, -76.0, 38.0, 39.0);
wpsRequest.getDataInputs().add(
      new Input("bbox", new BoundingBoxData(envelope)));

//limit the number of results to 50
wpsRequest.getDataInputs().add(
      new Input("maxWetlands", new LiteralData("50")));
```

```
//issue the request to the wps
Object results = wps.execute(wpsRequest);

//cast the response
GetFeatureResponse response = (GetFeatureResponse)results;

//get the list of features (results) from the response
List<Feature> features = response.getFeatures();

//walk the list of features and write them out to the client
//...
```

2.4.5. Atom + JSON

The Atom Syndication Format (ATOM) is used as the representation of data being sent between the client and server-side components of the Wetlands JD Analyzer application. The data is serialized as JSON (JavaScript Object Notation) to minimize costs associated with parsing XML in the web browser. The following is a snippet of a JSON-formatted Atom feed document (with GeoRSS extensions) containing one entry:

```
{
      "title" : "Feed Title",
      "updated" : "20080806T12:00:00-0400",
      "box" : "-180.0 -90.0 180.0 90.0",
      "entries" : [
            {
                  "id" : "some_unique_id",
                  "title" : "Some Title",
                  "summary" : "Some Summary of this entry",
                  "published" : "20080806T12:00:00-0400",
                  "updated" : "20080806T12:00:00-0400",
                  "links" : [
                        {
                              "rel" : "alternate",
                              "href" : "entries/some_unique_id"
                        }
                  ],
                  "box" : "-180.0 -90.0 180.0 90.0"
            }
      ]
}
```

The Atom Publishing Protocol uses standard HTTP methods to facilitate access to or modification of resources provided by an APP-supporting server.

- GET is used to retrieve a specific representation of a resource
- POST is used to create a new resource through submission of representations of the to-be-created resource to the collection to which the new resource will be added
- PUT is used to update a resource using a modified representation of the resource
- DELETE is used to delete a resource

This technique of using HTTP methods to access and modify resources is known as being "RESTful".

2.4.6. RESTful Resources

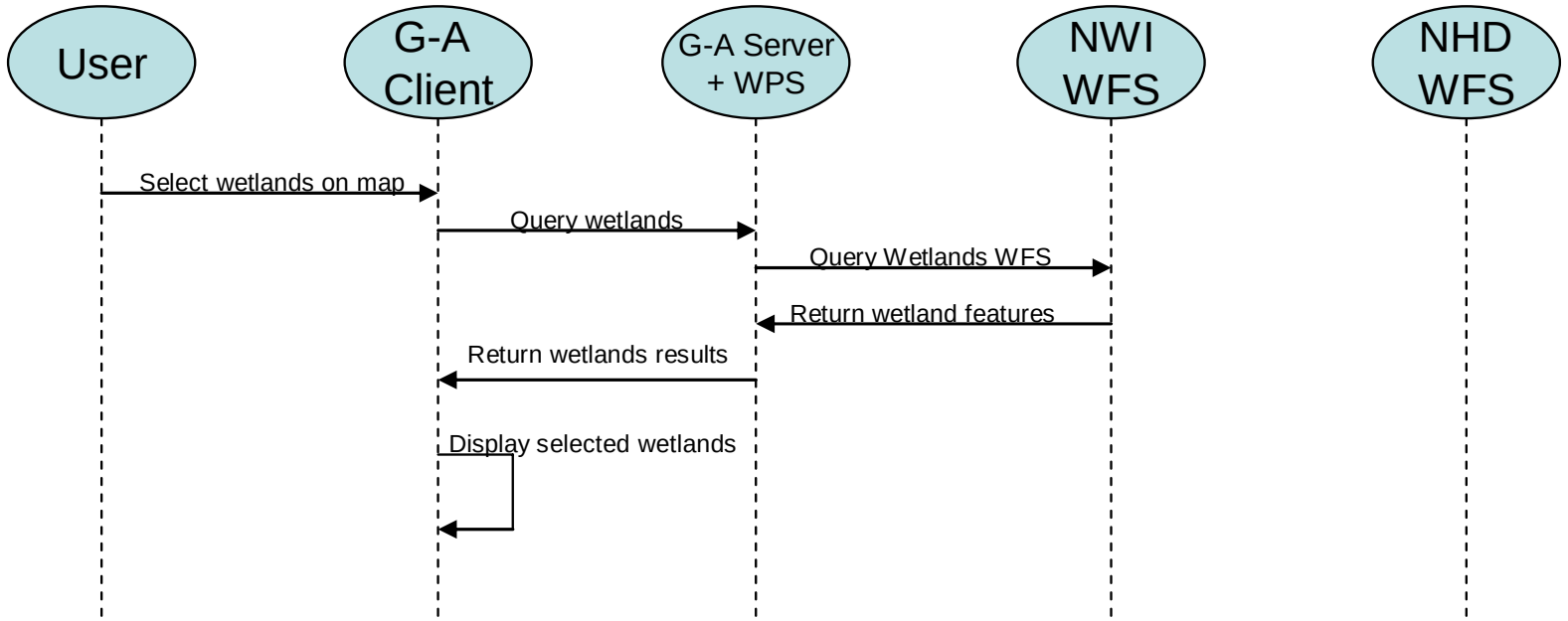
The server-side components of Wetlands JD Analyzer application provide several Java classes which act as RESTful servlets using Atom to process requests for the various types of objects withing the application.

- WfsResource – provides access to the configured WFS – the NWI wetlands WFS in the default configuration
    - HTTP-GET – querying the WFS using the supplied querystring parameters
- ReportResource – provides access for retrieving, updating, and deleting analysis reports
    - HTTP-GET – retrieving a single report
    - HTTP-PUT – updating a single report
    - HTTP-DELETE – deleting a single report
- ReportsCollectionResource – provides access for retrieving lists of analysis reports and creating new analysis reports
    - HTTP-GET – retrieving a list of reports
    - HTTP-POST – creating a new single report
- AnnotationResource – provides access for retrieving, updating, and deleting report annotations
    - HTTP-GET – retrieving a single annotation
    - HTTP-PUT – updating a single annotation
    - HTTP-DELETE – deleting a single annotation
- AnnotationsCollectionResource – provides access for retrieving lists of report annotations and creating new annotations
    - HTTP-GET – retrieving a list of annotations
    - HTTP-POST – creating a new single annotation
- FeatureResource – provides access for retrieving lists of features (wetlands and streams) associated with a report
    - HTTP-GET – retrieving a list of features

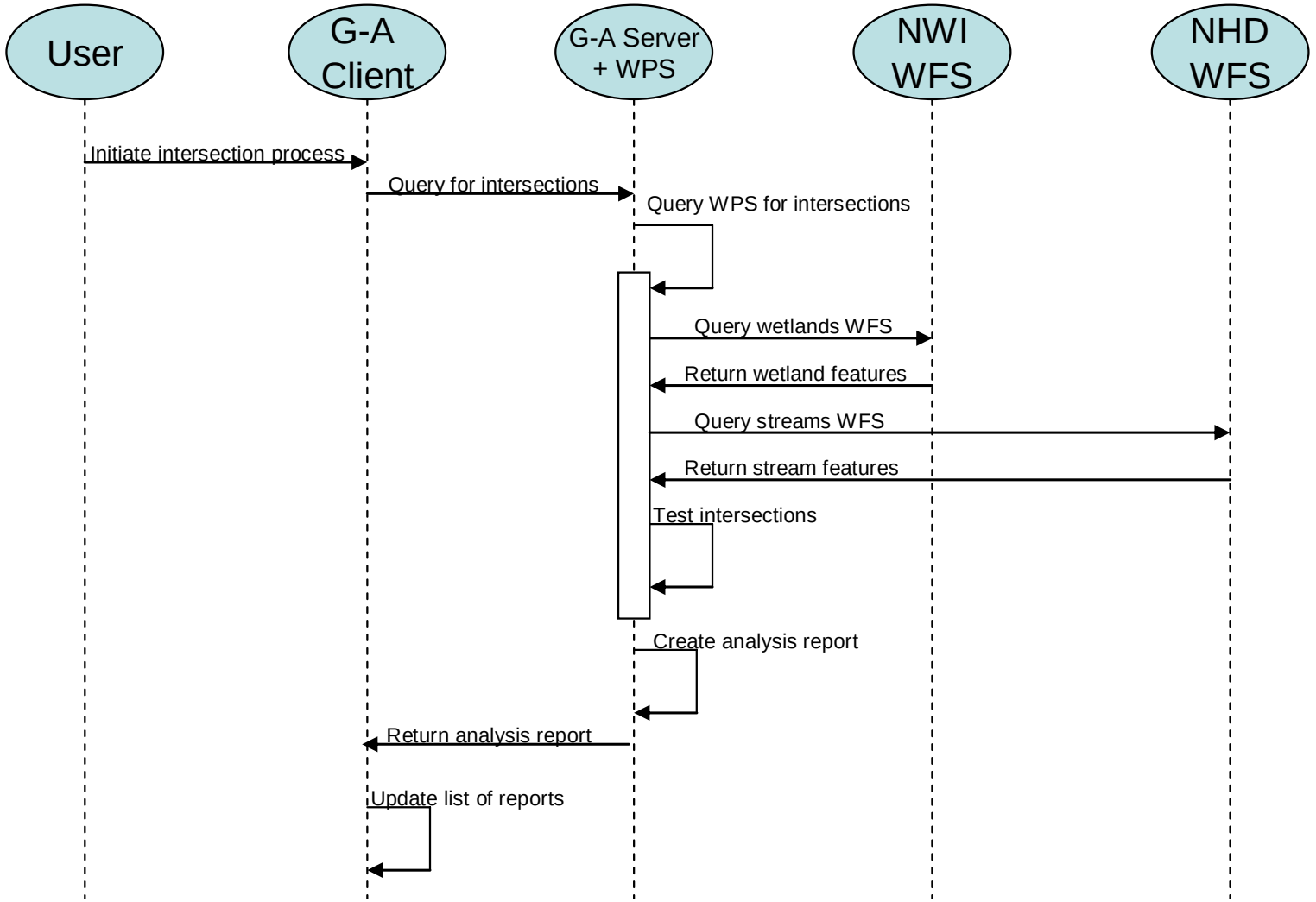
Appendix A. Diagrams

A.1 Sequence Diagrams

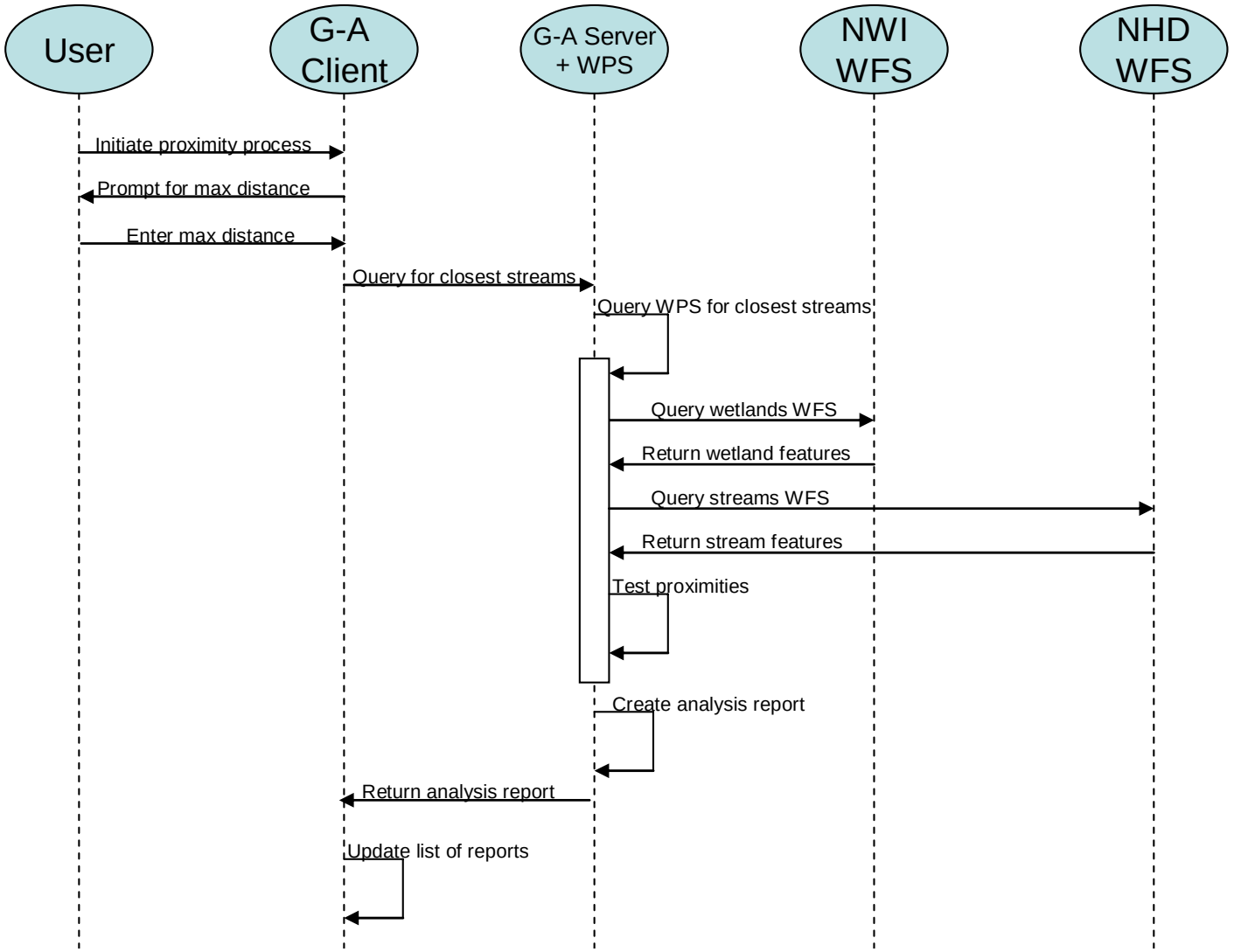
A.1.1 Selecting Wetlands



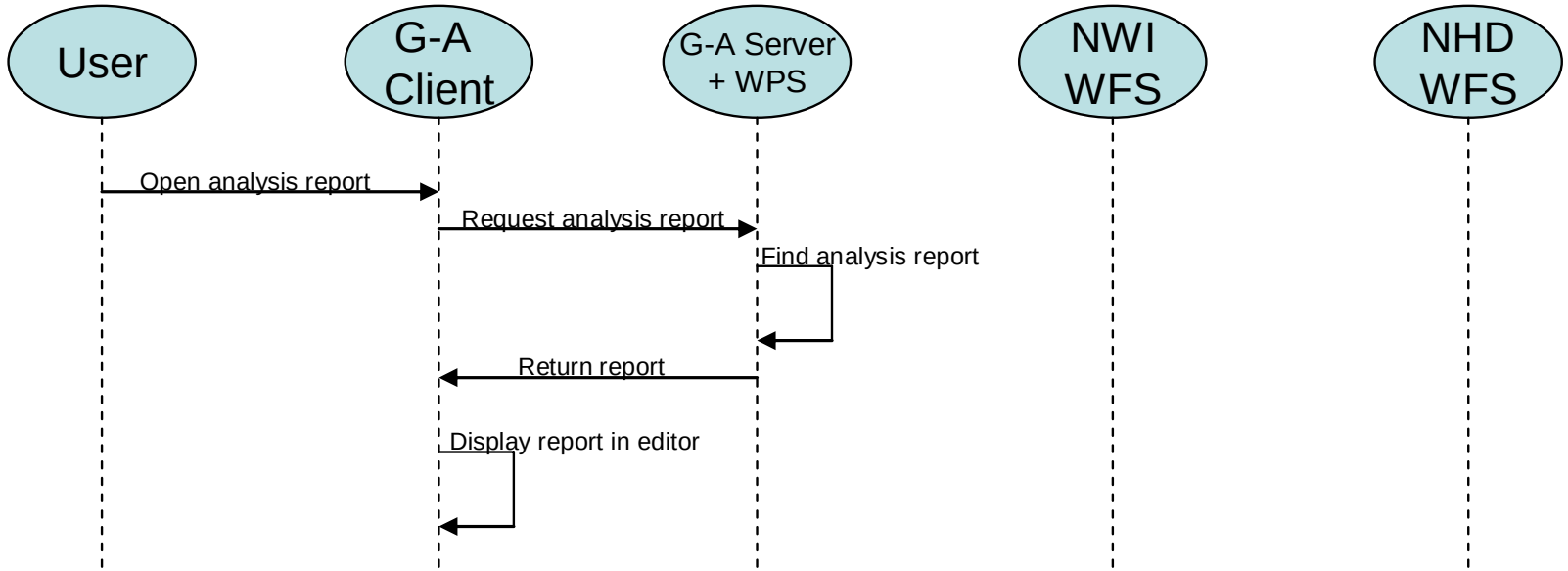
A.1.2 Testing for Intersections



A.1.3 Testing for Closest Stream (Proximity)



A.1.4 Viewing an Analysis Report



A.1.5 Annotating a Report