CS CAPSTONE TECHNOLOGY REVIEW

NOVEMBER 21, 2017

COVERAGEJSON RESPONSE HANDLER FOR OPENDAP

PREPARED FOR

NASA JET PROPULSION LABORATORY

LEWIS JOHN MCGIBBNEY

PREPARED BY

GROUP55

RILEY RIMER

Abstract

This document reviews the possible technologies choices that can be used for the CoverageJSON Response Handler for OPeNDAP project.

1 Project Overview

One way that satellite data is currently handled by NASA is through OPeNDAP, the Open-source Project for a Network Data Access Protocol. OPeNDAP is a data transport protocol that supplies its users a way to provide and request data across the web. One of the main functions of the protocol is the ability to pull data in multiple different formats including ASCII, netCDF3, netCDF4, binary (DAP2), and several other object serializations. OPeNDAP does not currently have a response handler in place to serve data in the CoverageJSON data format, which is a JavaScript Object Notation (JSON) data format for describing coverages. CoverageJSON would be a very useful format to have available from OPeNDAP due to it encoding data values based upon a spatial temporal domain similar to how satellite data is collected from NASA's satellites. The integration of the CoverageJSON format into OPeNDAP would help support the creation of coverage data driven web applications by NASA JPL(Jet Propulsion Lab) and any other users of OPeNDAP.

2 Project Role

My roles will consist of working on general development across the entire project along with specific work pertaining to the technologies described in this document. The technologies I will be focusing on are the testing frameworks, web application frameworks and the UI user interaction.

3 Technology: Testing Framework

3.1 Overview

C++ doesn't have a clear winner in terms of which Unit Testing framework to use. This section will cover a few of them and compare their pros and cons.

3.2 Criteria

1) Amount of work needed to add new tests.
2) Crash and exception handling.
3) Level of assert functionality.

3.3 Potential Choices

3.3.1 CppUnit

CppUnit is one of the most used unit testing frameworks for C++. It has been around for a while and has had years of fine tuning to try to make it fully featured, but in turn it has become a bit obfuscated.

1) It takes quite a bit of work to create new tests in CppUnit. Simple tests end up needing a rather large amount of setup from within the test.
2) It handles crashes and exceptions really well. It uses wrappers around tests to catch all exceptions making the handling of the rather exceptions simple.
3) The assert functionality is decent. It is missing some assert functionality for comparative operations, but conveniently variables used in asserts are printed out when there is an assertion failure.

3.3.2 CppUnitLite

CppUnitLite is naturally just a lighter version of CppUnit that was made by the original CppUnit developer. It is definitely very ”lite”, but that allows for a lot of customization to make it fit your own needs.

1) It is really easy to add new tests in CppUnitLite. It only requires the test declaration itself and nothing else.
2) CppUnitLite doesn't handle exceptions at all by default, but it can be added rather simply and be as robust as the user wants to make it.
3) The assert functionality is rather bad for CppUnitLite compared to the other frameworks discussed in this paper. It natively only checks equality and it does not print out variable values on assertion failure.

3.3.3 Boost.test

Boost.test is a newer testing framework than CppUnit. It has solid documentation and Boost itself includes other various functionalities.

1) It requires a rather minimal amount of work to create new unit tests. If the tests are part of a suite though it requires quite a bit more setup and the registration of the tests with the suite.
2) Boost.test seems to have the best exception handling out of the main C++ unit testing frameworks. It prints out information relevant to exceptions and also allows for exception handling to be disabled, allowing the exceptions to be caught in a debugger.
3) Boost.test has really fleshed out assert functionality. You can assert for all basic operations and it prints out variable values on assertion failure.

3.4 Discussion

Each of the frameworks available for unit testing in C++ have their own pros and cons. CppUnit is probably the most tried and tested frameworks but it seems rather dated in terms of functionality compared to Boost.test. CppUnitLite seems like it could be a very good unit testing framework once the setup and customization is done, but the required setup may not make it any better than boost.test.

3.5 Conclusion

Boost.test seems to be the clear winner in terms of setup, exception handling and assert functionality. It will be up to discussion whether this will be what will be used in this project considering our work will be part of an already existing framework and what they use will need to be considered.

4 Technology: Web Application Framework

4.1 Overview

There are multiple different web application frameworks for application development all with their own strengths and weaknesses. I will be comparing the strengths and weaknesses of a few of these frameworks in the following section.

4.2 Criteria

1) Performance
2) Usability
3) Application turnaround time

4.3 Potential Choices

4.3.1 Rails

Rails, or Ruby on Rails is a web application framework written in Ruby. It is a MVC style framework that focuses on convention over configuration and reducing repetition in code.

1) Rails has rather average performance and can be rather resource heavy.
2) Rails has good usability due to it being flexible and IDE friendly, having a simple structure and due to Ruby itself being a very expressive language.
3) Rails does fall short in turnaround time due to its complexity, which becomes very apparent in larger projects

4.3.2 Node.js

Node.js is a web application framework that uses JavaScript as its primary programming language at all levels. It is praised for its speed and scalability.

1) The performance is great in Node.js, and it has very powerful underlying libraries that support non-blocking I/O which is essential in UI.
2) Node.js is very usable partly to it using the same language on both the server and client side. One problem with Node.js is the upkeep required to make sure code doesn't become incompatible with newer Node.js versions.
3) Since it is somewhat simpler to create applications in Node.js, its turn around time is rather good.

4.3.3 Django

Django is a web application framework that is written in python and commonly seen in academics and science.

1) Django has rather average performance, though it can be considered to be quite poor compared to Node.js
2) Django has quite good usability with Python being a very easy to read language. Django is rather simple and easy to learn compared to Node.js and rails.
3) Django development is rather quick, with rapid development being one of its main principles.

4.4 Discussion

Each framework seems to excel in one or two of the criteria but not all three. Rails has a pretty good development environment and is widely used, Node.js has very good performance and conveniently uses Javascript as its sole language for the server side and client side and Django is simple and quick to develop in. Node.js itself isn't exactly a "framework" though and will need to be used alongside a light framework like express.

4.5 Conclusion

In the end since this project is a continuation on a preexisting application, this project will be using the framework already implemented in Hyrax, which is Ruby on Rails. Rails may not be a clear winner among the choices but there isn't anything that makes it a poor choice either, since Ruby is a very strong and expressive language.

5 TECHNOLOGY:UI INTERACTION

5.1 Overview

There are many different ways user interaction can be setup to allow the user to subset the amount of data they are requesting. Some implementations are much easier to do, but are harder to use and vise versa.

5.2 Criteria

1) Usability
2) Development Time
3) Reliability

5.3 Potential Choices

5.3.1 Text Input Bounds

Currently the Hyrax front end allows for the user to input values as bounds for what data they want to pull from a data set, along with the variables they want to be shown.

1) Usability for this implementation gets worse the larger the amount of bounds and details the user needs to set.
2) Development time is short since this is a rather basic thing to implement, but this is already done so we wouldn't have any development on our end.
3) Considering the simplicity of this setup it should be more reliable than other means.

5.3.2 Interactive

A more convenient way for users to interface with the data would be to make the data selection be interactive through incorporating drag and drop functionality and more visual queues to what the user would be queuing.

1) This would be much more user friendly for users, especially if not all data options weren't shown all at once like how the current configuration is.
2) The development time for something like this would be much greater, which is a problem for a project of this scale.
3) This would be much less reliable and need much greater testing.

5.3.3 Trimmed Interface

A good balance between improving the current UI and keeping the old UI would be just making some adjustments to the current setup. This would primarily include not showing the users all the data options at once, and maybe creating tabs or something to separate the user controls.

1) This would have better usability when there are lots of different fields to apply bounds to, but wouldn't help much if there weren't very many bounds originally.
2) This would require more development than the basic setup that is already done, but not nearly as much as a full interactive UI.
3) This would be rather reliable as long as it doesn't get to obfuscated.

5.4 Discussion

Considering the time frame for this project it would be rather hard to spend too much time on UI. This puts making more advanced UI changes out of our possible solutions. Considering how other parts of the project go it may be ideal to not try to make any UI changes at all. Despite the current setup being somewhat overwhelming it does work very well.

5.5 Conclusion

In conclusion, for this project we will plan on not making any changes to the UI setup already implemented. Currently UI changes seem to be beyond the scope of this project, this may change in the future though depending on how the project progresses once work actually begins on it.