9A.2 SOFTWARE ARCHITECTURE PRINCIPLES FOR WEB-APPLICATION INFRASTRUCTURE – HOW EARTH SCIENCE SOFTWARE SYSTEMS SUCH AS THE LIVE ACCESS SERVER FIT IN

Roland H. Schweitzer*
Weathertop Consulting, LLC, College Station, Texas

Steve Hankin, Ansley Manke
NOAA/Pacific Marine Environmental Laboratory, Seattle, Washington

Jonathan Callahan, Kevin O'Brien
JISAO, University of Washington, Seattle, Washington

Steve Du, Jing Li
Macrostaff, Seattle, Washington

1. INTRODUCTION

The Live Access Server (LAS); Hankin (1998) is a well-established Web-application software system for display and analysis of geo-science data sets. The software, which can be downloaded and installed by anyone, gives data providers an easy way to establish services for their on-line data holdings so their users can make plots, create and download sub-sets in a variety of formats, and compare and analyze data.

One of the most fundamental decisions that software architects must make is how much of their software infrastructure they must write themselves and how much can they leverage from existing software frameworks. Often, software engineers working in areas such as the earth sciences feel as though their requirements are too specialized to take advantage of software frameworks designed primarily for business-to-business and on-line commerce transactions and user interactions.

In this paper we discuss our experiences using off-the-self Web-application frameworks to build the next generation of the Live Access Server. The proper design software components to isolate the specialized processing needed for working with geo-science data from the application frameworks will be a special emphasis. In conjunction with the discussion of software infrastructure issues related to Live Access Server development, we will provide an overview of the latest advances in the capabilities and configurability of the Live Access Server itself.

Specifically the new architecture makes it easier to add new output products to our core system based on the Ferret analysis and visualization package. By carefully factoring the tasks needed to create a product we will be able to create new products simply by adding a description of the product into the configuration and by writing the Ferret script needed to create the product. No code needs to be added to LAS to bring the new product on-line. Equally important, these same design principles make it easier to add other analysis and visualization packages (such as CDAT and GrADS) to LAS. The new architecture is faster at extracting and processing configuration information needed to address each request. Finally, the new architecture makes easier to pass specialized configuration information through the system to deal with unanticipated special data structures or processing requirements.

*Corresponding author address: Roland H. Schweitzer, Weathertop Consulting, LLC, 2802 Cimarron Ct, College Station, TX 77845. E-mail: weathertop.consulting@gmail.com

2. WEB APPLICATION TECHNOLOGY

A Google search on any aspect of web application technology will quickly overwhelm you with buzzwords (like AJAX), commercial software (like ASP.NET and ColdFusion) not to mention many open source possibilities ranging from content management systems like Plone to programming frameworks like Ruby on Rails, PHP and Java. Within each of these categories you'll find many competing frameworks and scaffolds that intend to factor out the common parts of building Web applications software into reusable and configurable frameworks.

Once our group decided to convert our Web application (the Live Access Server) from its current technology (Perl CGI) to a new technology we found ourselves adrift in this seemingly endless sea of possibilities. Our first task was to narrow down the choices.

Our product is not a Web site, but rather a collection of software which we distribute so that others can provide scientists and other users access to the geo-referenced scientific data the installer wishes to make available. Therefore, we cannot just buy a license to a commercial product and use it to develop our own killer Web site. Any software we choose we must have the right to redistribute or must be available for anyone to download and install themselves.

We also have to consider the skills and prejudices of the people that will be doing the development work. We all like to learn new skills, but it makes no sense from the project perspective to expect the developers to spin up on an entirely new technology if good solutions exist in the areas where the developers have experience.

Finally, we want to be able to leverage existing software that is outside of the actual Web application technology we choose to help us with the specialized tasks associated with our core mission – providing access to scientific data.

All of these considerations lead us to using Java and Java Servlets as the basis of our implementation. Unidata has invested heavily in Java development. The THREDDS data server; Caron (2006), the Integrated Data Viewer; Murray (2006) and the netCDF Java libraries; Davis (1998) are all examples of Java software which we want to leverage in our development. We have experienced Java developers on our team. Java itself is available for download and there are many high quality servlet containers which are available for download on top of which our software can then be installed.

However, even after we have settled on Java as an implementation language and Servlets as the supporting technology there are still many shades of grey.

3. WEB APPLICATION FRAMEWORKS IN JAVA

Even after settling on the two basic implementation choices Java and Java servlets there are many choices left to make. Even without Google I can name several frameworks that purport to ease the burden of Web Application development in Java. Here goes: Spring, Struts 1.x, Struts 2.x, Barrachuda MVC, GlassFish and Shale. And the list goes on to include frameworks that were developed for specialized use in scientific data sharing. The Anagram framework from COLA is an example of this type of framework.

In fact, the first version of our new LAS product server was developed using Anagram. However, after careful study of the problem space we are working in, it was clear that we could use any of the popular Java Web-application frameworks equally well and would have the advantage of the support, active development and commercially available literature on the use of such a framework.

By design the LAS Product Server has a limited View component in the classic Model, View, Controller architecture. The Product Server can render results of a product request into an HTML page, but all of the complex user interactions are handled by other components in the LAS architecture. This fact made choosing a Web-application framework much easier. We knew at the outset that our problem could fit easily into any of the popular frameworks, so we were free to choose among them on that score.

The group also made a firm commitment to carefully design and implement our Web-application such that all of our "Model" code (the guts of the logic that is particular to our problem space) would be strictly separated from the Web-application framework. The extent that we are committed to succeed with that separation gives us the flexibility to choose any framework, because we would then be able to switch easily later since our code is not tangled up with the framework code.

4. OUR APPROACH IN LAS

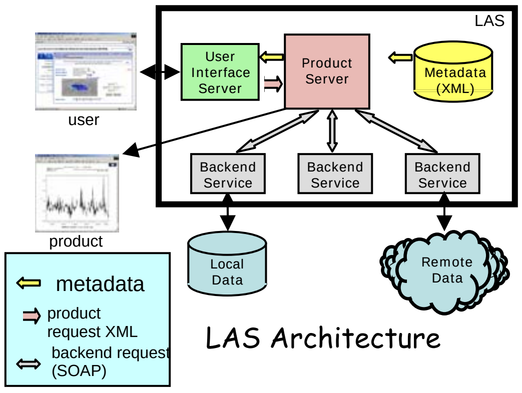

Figure 1. The Live Access Server Architecture

Given that our problem space would fit into any framework and given our commitment to separate our Model logic from the framework to allow us to switch later if we want, we chose to do our initial implementation using Struts 1.3. This is a popular and mature system with many resources on the Web and in bookstores to help new users. Even our Integrated Development Environment (Eclipse with the commercial MyEclipseIDE Enterprise Workbench installed) contains automation tools for use with Struts.

Separation and testing are the two mantras we tried to follow during implementation. To begin with, each major piece of business logic (the process of translating a request for a product into the sub-requests needed to run the services that will fulfill the request, the job of contacting the services and the collection of the responses and the job of delivering the results to the user) were each implemented as a Plain Ordinary Java Object (POJO) outside of the Web application framework. Along with these POJO objects to encapsulate the business logic we wrote Units tests to exercise each component. Once the business logic was finished, then and only then did we go to the framework and begin the tasking for marshalling those objects together into workflow that could be deployed on the Web using Struts.

So far so good, but we also had some goals for this development that directly involve the Controller portion of the architecture. These goals were to be able share product requests between users and to be able to report progress back the user if the process takes along time to complete. The logic to accomplish these two goals was done directly in the Controller component of the Struts framework that is exposed to the developer by default (the Struts Action class). When a request comes in for a product, before contacting the services that will make that product the Product Server checks to see if it already has a thread running that is working on that exact product request. If it does the new request simply joins the thread in progress and the user begins to see updates of the progress already completed on that request and new progress going forward from the moment they joined the thread.

A new user request being able to join an existing request is predicated on the fact that after a certain configurable amount of time has passed the Product Server stores the thread that is working on the request into the Servlet context and returns a progress report to the user. The browser is set to automatically refresh the request for the product and the original user rejoins the original thread working on their request to see further progress just as a new user with the same request would join. Because the capabilities to share requests and report progress are implemented directly in the Controller part of the Struts framework it will be more difficult to change these to a different framework, but not impossible.

5. FUTURE PLANS

In addition to continued improvements in the LAS Product Server we have another significant development task ahead of us to re-write the LAS User Interface Server. Though implemented using what was a state-of-the-art technology framework for its time, this software component which manages all of the user interactions with the LAS (except for the view of the final product) needs to be modernized. When evaluating solutions for re-writing the User Interface Server we will again be evaluating an open slate of techniques (AJAX implementations for some components) and scaffolding technology (the Shale or Spring frameworks come to mind), but we will likely begin our implementation in exactly the same way we did with the Product Server – separate the Model logic from the framework and test components outside the Web application framework before integrating into the Struts Action framework.

6. CONCLUSIONS

Least we sound too self-congratulatory for simply following good software engineering practices we should note that it is easy to be seduced by the notion that our application area (science data display and analysis) is only amenable to custom solutions. By exploring and eventually using a commonly use Web-application framework we have saved significant development time and effort and have a more robust product.

7. REFERENCES

Caron, J, E. Davis, Y. Ho, R. Kambic, 2006: Unidata's THREDDS Data Server, Preprints, 22nd International Conference for Interactive Information and Processing Systems for Meteorology, Oceanography and Hydrology, Atlanta, GA, AMS CD-ROM

Davis, G, D Fulker, 1998: Extending the netCDF Model to Java, Preprints, 15th In International Conference for Interactive Information and Processing Systems for Meteorology, Oceanography and Hydrology, Dallas, TX, AMS

Hankin, Steve, J. Davison, J. Callahan, D. E. Harrison and K. O'Brien, 1998: A Configurable Web Server for Gridded Data: A Framework for Collaboration, Preprints, Fourteenth International Conference for Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology, Phoenix, AZ., AMS, 417-418.

Murray, D, 2006: The Integrated Data Viewer (IDV) – A Discipline Agnostic Tool for Geoscience Exploration, Preprints, 15th Symposium on Education, Atlanta, GA, AMS CD-ROM.