CS 395 – Summer Internship

Jake Feasel

I am applying my job toward this internship class. I work at an Anchorage based software development company called GeoNorth, LLC. GeoNorth has several means of doing business – they sell "boxed product" software, provide training, and do per-hour development and analysis work. They speciallize in GIS (Geographic Information Systems) with ESRI products and web development with Macromedia's ColdFusion. Clients often include governmental agencies or corporations.

GeoNorth hired me in August of 2002 as a full-time "Programmer / Analyst 2" position. For the purposes of this paper, I will focus on my employment during the summer of 2003 and the work I did on the "Watershed Management Tool Viewer" (WMTV) project. I was the main developer for this project.

The project consisted of taking a product that GeoNorth markets and extending that product to work with another company's system. Our product is called MapOptix. It is a ColdFusion based frontend to ESRI's ArcIMS server. It is designed to ease the deployment of web-based mapping systems that ordinarily require programmers to build. The system that we were building atop of is called the "Watershed Management Tool". I don't actually know much detail about what the WMT is supposed to be used for or how it gets used – our concern was presenting the data that it made available within our application.

Our application's base functionality was to be extended in several ways. The client needed to be able to add new shapes to a map layer from within the application, and add associated tabular data to that shape. The client also needed to be

able to choose from any arbitrary MXD (map file for ESRI's products) and be able to overlay the contents of that MXD with the base service in ArcIMS. The result of this overlay needed to be shown in the interface. Manipulating these dynamic services and making the ordinary GIS features available to these services was the bulk of the work for this project.

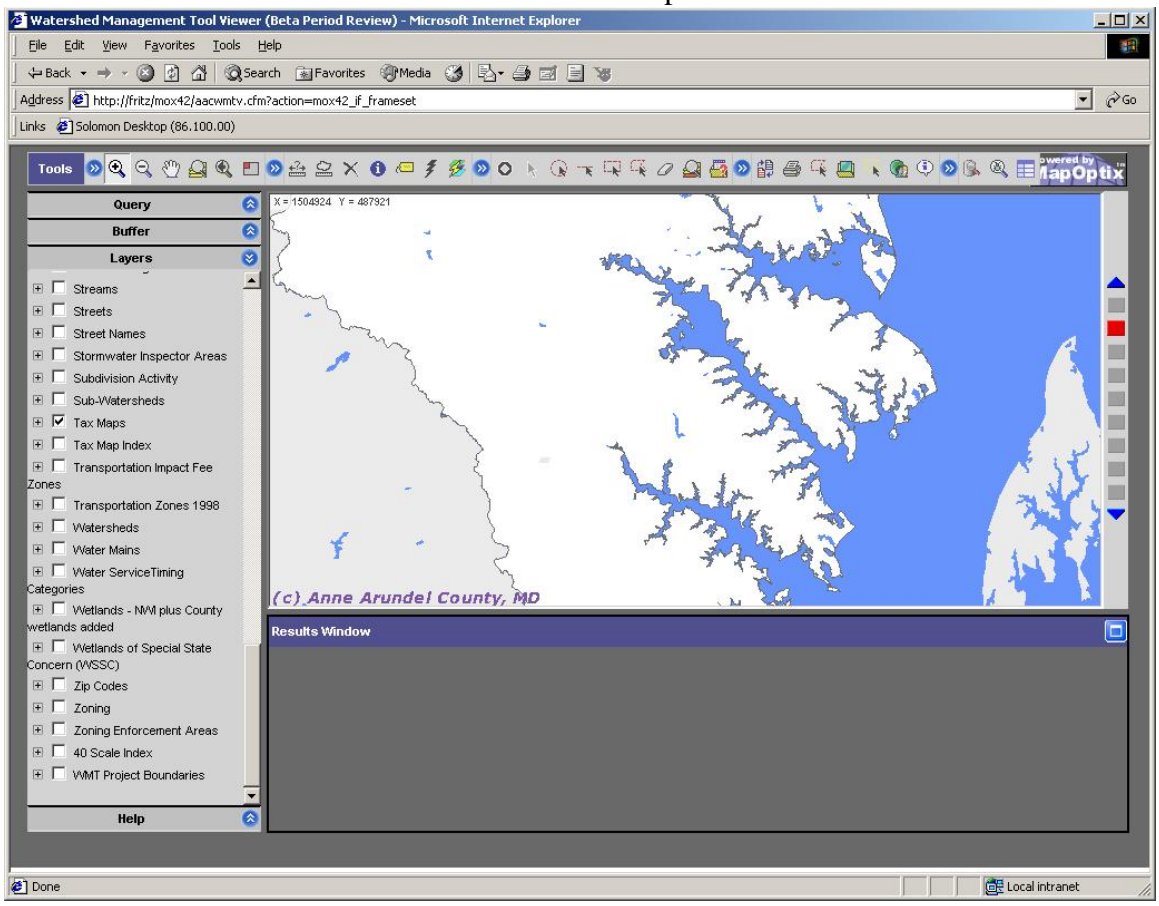
The specific technical hurdles were as follows:

1. Use the Java API for ArcIMS to create / delete services from an MXD
2. Build an extension to ColdFusion with Java to overlay two images that exist only in memory (not on the disk). This extension makes use of the ImageMagick API via JNI called JMagick.
3. Use the Java API for ArcSDE to add shapes to map layers
4. Use ColdFusion and the API we built with MapOptix to extend MapOptix to leverage the new Java API calls.

The planning process was outlined by a scope of work / design document that identified all of these tasks and the expected way they were going to be implemented (at a high level). I was not involved in this process – I was basically given detailed technical tasks that could be focused on without needing to know the big picture.

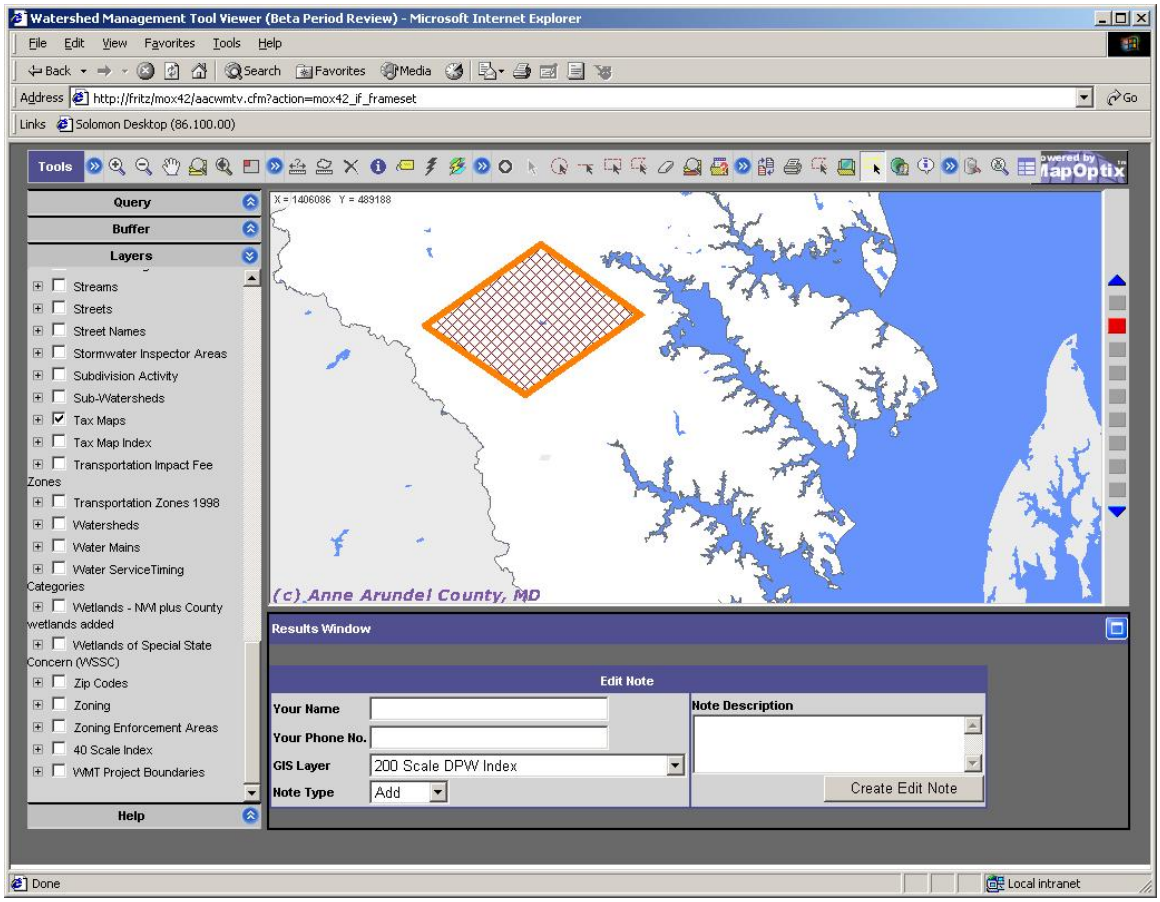
Here are some screen shots from the final application:

This shows the basic map interface.



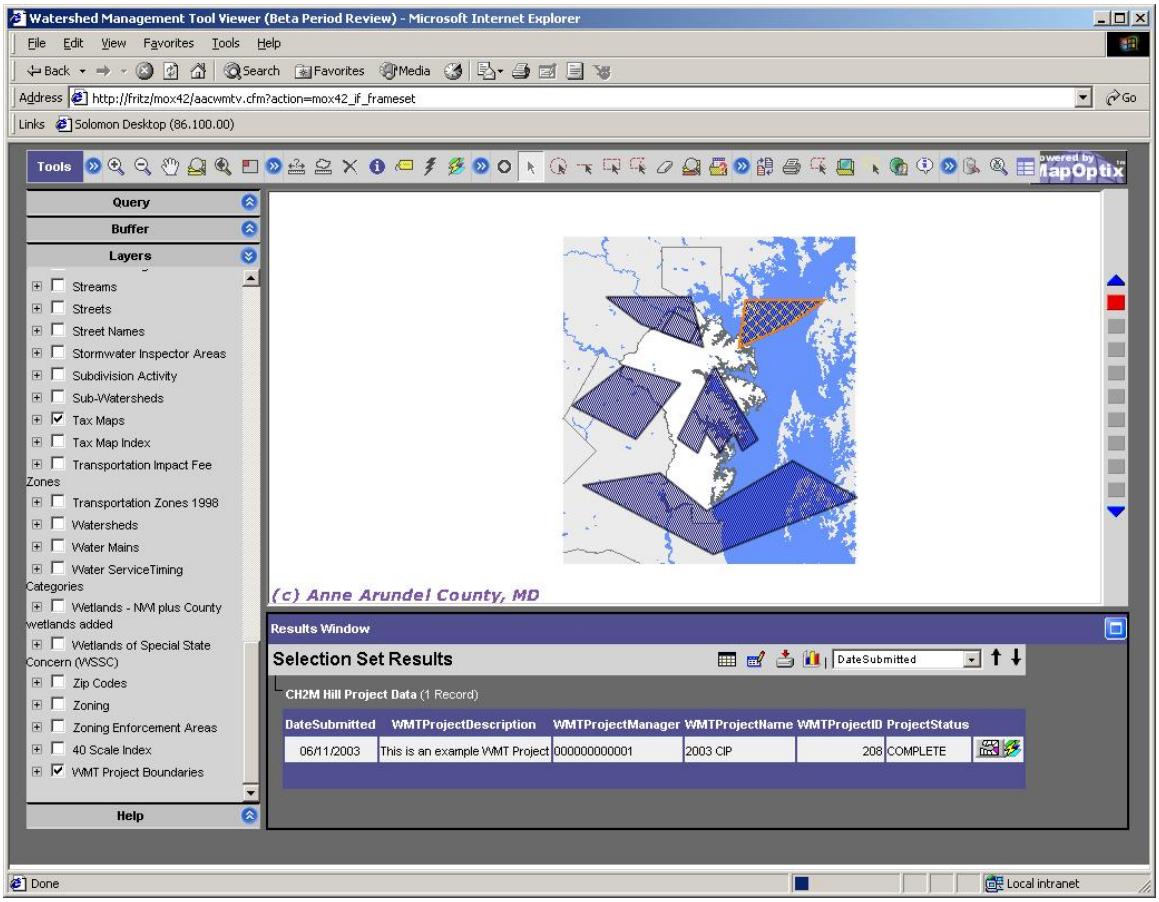
Here you can see the main map image, the layer sidebar, results panel, and tool bar. This is standard MapOptix without any visible customizations.

This screen shows the shape creation utility.



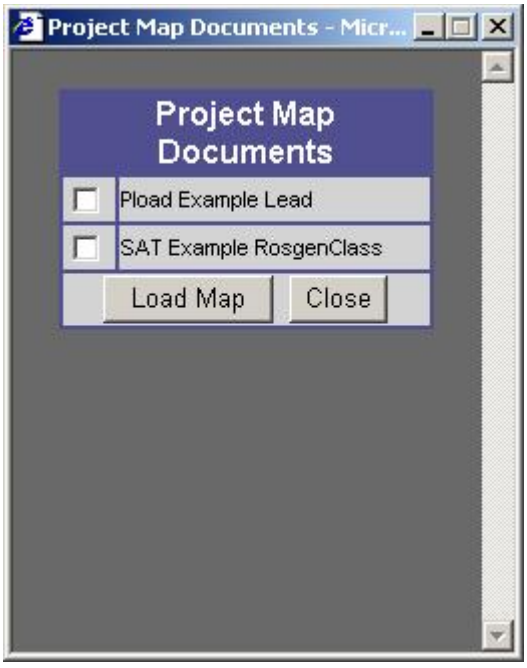
The orange polygon represents the new shape before it is committed to the layer. The results window shows the tabular data fields that will be associated to this shape.

Getting data about a feature



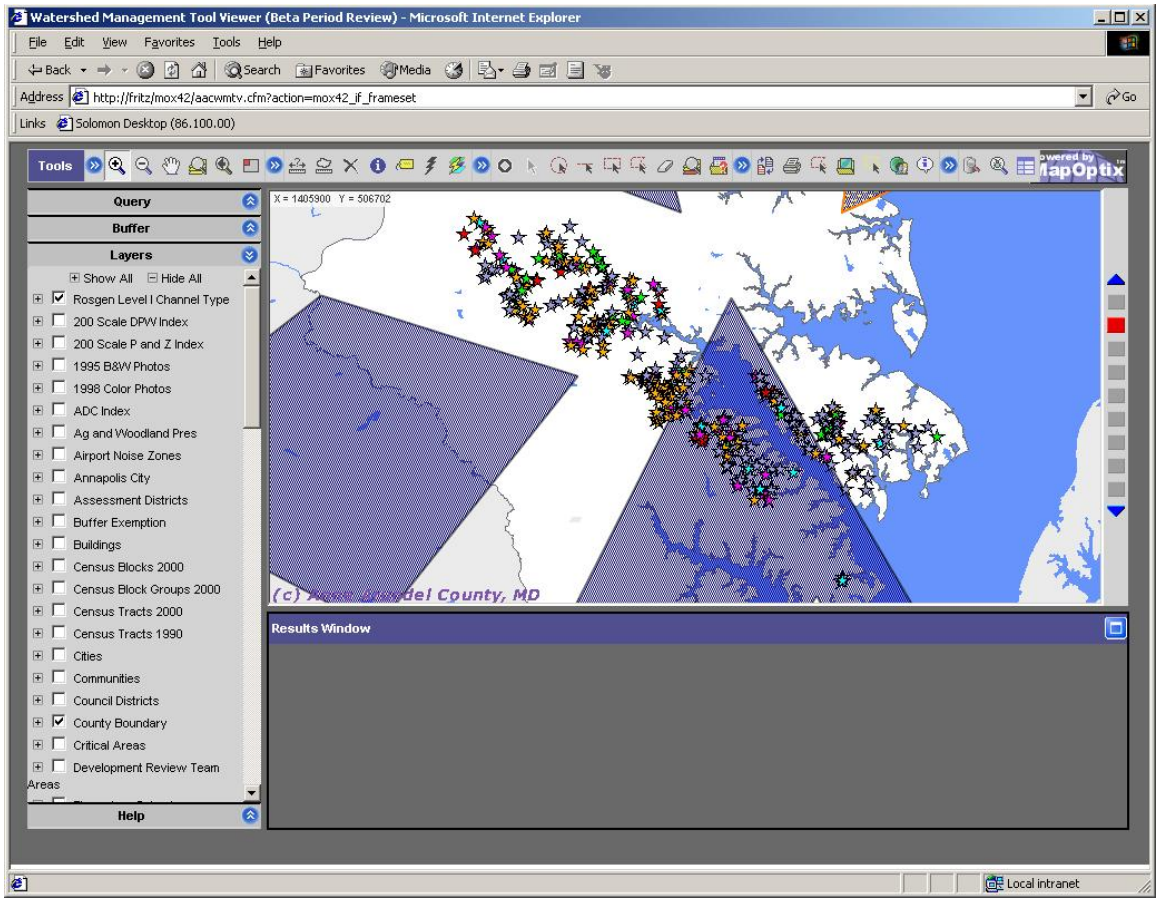
Here we see 5 shapes on the map – these were created with the polygon creation tool built for this project. This shows one of them "highlighted" and the results frame shows the data associated to that highlighted shape.

A popup window that displays map document choices



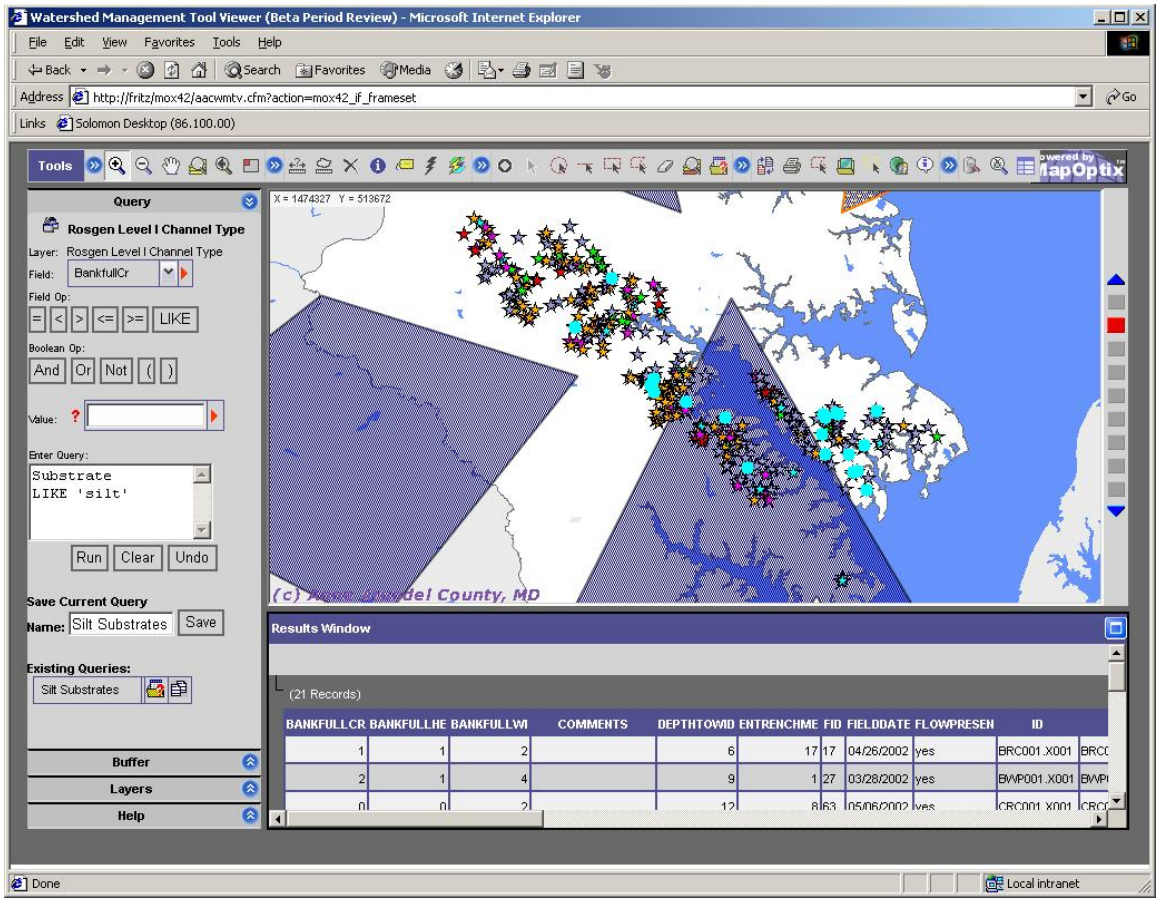
This screen scans a specially define directory structure that contains any number of MXDs, or “map documents” that are related to various projects. These map documents are used to create new ArcIMS services on-the-fly, which will later be overlayed on top of the main map image.

The main map being overlayed with a map from a dynamic service



This shows how the new services seem to act just like normal layers in the main map. The new layer (the stars) is being overlain atop of the other map. All this involves is making multiple requests to ArcIMS, once per image, and then combinding them with some Java code. The Java code sets the white background of the higher images to be transparent so that they don't block out the other images. The reason this is useful for a user is that they could potentially have hundreds of map documents that they would like to be able to incorporate, but without having to reconfigure the site or have them all available at once.

Quering data against the dynamic layer



This screen shows the query capablities of the system. Here we are building a query with some simple “Ad-Hoc Query” tools. These queries search the map document and highlight the features that match the criteria. It also shows the tabular data associated to each matching feature in the results frame. The query panel also allows administrators to store queries for use against specific map documents so other people don't have to try and build a query dynamically.

The project took most of the summer to complete. I started the very first tasks in the middle of May, and we deployed the finished application in the beginning of September. After deployment we trained a relatively large group of people on how to use the application, which was a good stress test for it. Fortunately, the system held and everything worked.

I feel this project was a perfect fit for the "internship" process. It fit the timeframe correctly and the workload involved plenty of programming. I am thankful I had the fortune to be able to apply my full time job towards this class.

Along with this document there are two .java files. SDE_Edit.java is the code that will add polygons to a layer in ArcSDE. It is intended to be executed from a ColdFusion custom tag call. WMTVOverlay.java is code that will take in a list (really a ColdFusion query object) of base 64 encoded image data and overlay them, returning a new base 64 encoded string that contains the new image data. This is also expected to be called as a ColdFusion custom tag. I feel these code samples are the most appropriate for inclusion here – they are fairly discretely defined and are a good example of the kind of work being produced. I also assume Java would be a more familiar language to readers of this document than ColdFusion, so these samples are better suited.