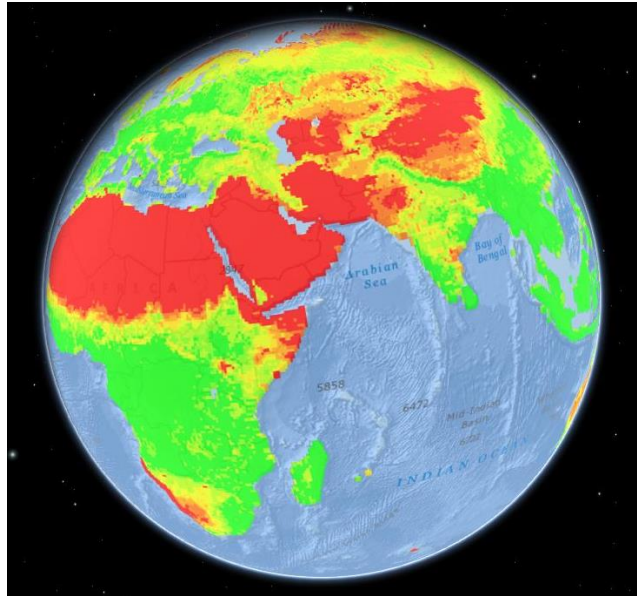


Visualization of Earth Modeling Systems (VEMS)

May1701



Client: Chaoqun (Crystal) Lu

Faculty Advisor: Johnny Wong

Team

Team Leader – Anish Kunduru

Team Communications Leader – Kellen Johnson

Webmaster – Eli Devine

Concept Holder – Julio Salinas

Table of Contents

| | |
|--|----|
| Section 1: Project Design | 3 |
| 1.1 Useful Definitions | 3 |
| 1.2 Project Idea | 4 |
| 1.3 Proposed Solution | 4 |
| 1.4 System Diagram | 4 |
| 1.5 Deliverables | 5 |
| 1.6 System Standards | 5 |
| 1.6.1 Functional Requirements | 5 |
| 1.6.2 Non-Functional Requirements | 5 |
| 1.6.3 System Requirements | 6 |
| 1.7 Proposed Testing Methods | 6 |
| 1.8 Team Collaboration / Proposed Scheduling | 7 |
| 1.9 Expected Challenges | 7 |
| Section 2: Product Internals | 8 |
| 2.1 Software Support | 8 |
| 2.2 Implementation Details | 10 |
| 2.2.1 Daemon / Server-Side | 10 |
| 2.2.2 Parsers | 12 |
| 2.2.3 Client GUI | 14 |
| 2.2.4 End-User Website | 17 |
| Section 3: Testing and Evaluation | 19 |
| 3.1 Parser (.txt to .csv) Speed | 19 |
| 3.2 Map Creation Speed | 19 |
| 3.3 End-User Reactivity | 19 |
| 3.4 Resource Management | 20 |
| 3.5 Input Data Validity | 21 |
| Appendix 1: Operation Manual | 22 |
| Appendix 2: Alternative / Other Initial Designs | 26 |
| Appendix 3: Considerations | 28 |
| Appendix 4: Future Implementations | 29 |
| Appendix 5: Creating Template Maps | 29 |

Section 1: Project Design

1.1 Useful Definitions

- ArcGIS: ESRI software suite
- ArcMap: Desktop ArcGIS Software (GUI Based) that allows for manual creation of map documents.
- Arcpy: ArcGIS Python library
- Basemap: The underlying map on which a map service's layers are projected onto.
- CDN: Content Delivery Network
- CPU: Central Processing Unit
- ESRI: Environmental Systems Research Institute (the company that creates ArcGIS software)
- FIFO: First in First Out
- GIS: Geographic Information System
- GUI: Graphical User Interface
- HDD: Hard Disk Drive
- JS: JavaScript
- Layer: Stackable visual representation of map data.
- Map Service: A published/hosted map.
- MVC: Model View Controller
- RAM: Random Access Memory
- Spatial geodatabase: A collection of tables or other datasets which allows for the efficient recall of information stored in a vector space.
- Symbology: Modifiable map field which contains coloring and legend breakpoints.
- Table: A collection of geographic information that is stored as cells in a matrix.
- TLS: A cryptographic protocol that allows secure connections over the Internet.
- VCS: Version Control System
- VEMS: Visualization of Earth Modeling Systems (the project)
- VM: Virtual Machine

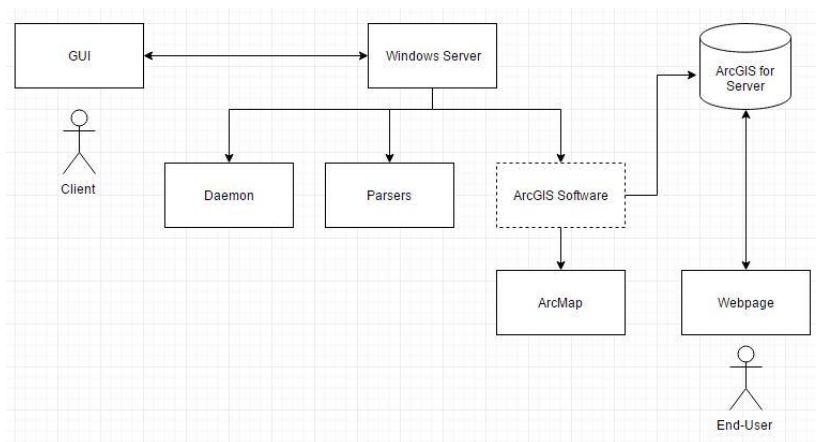
1.2 Project Idea

Professor Chaoqun (Crystal) Lu creates historical and projected estimates of ecosystem functions through an Earth Modeling System. In order to build a bridge directly between stakeholders and modelers, our client wishes to visually display time-series spatially explicit model output data across regional to global scales in decadal or centurial time periods. Dr. Lu would like an interactive method to display Earth system models in lieu of standard, non-interactive, image files.

1.3 Proposed Solution

Create an automated system that will allow our client to upload map datasets to be converted to a map that can be viewed on a 3D representation of Earth. Our research on methods to create such a map brought us to ESRI, as they are a leader in GIS products. With the help of the GIS Department at ISU, we obtained an educational license to utilize ESRI's ArcGIS platform. The solution that is outlined in this document shows the use of ArcMap for creation of map documents and map services, which a University server ultimately hosts. These, in turn, are accessible to the end user as interactive 3D maps that can be accessed from any web browser.

1.4 System Diagram



1.5 Deliverables

Create a server daemon to handle the map generation scripts to be called and host the final map services. Additional deliverables include a GUI to ensure easy use of the headless daemon and an interactive website, which can be accessed by final end-users.

1.6 System Standards

1.6.1 Functional Requirements

- The product shall allow the client to upload raw ASCII data to the server.
- The product shall parse all uploaded data from an ASCII file into a format readable by ArcGIS.
- The product shall automate the steps traditionally taken to create a map-service (typically done via ArcMap) by utilizing Python to take the CSV files and create a map, layers, and finally, publish a map service.
- The product shall allow end-users to select the compound on the relevant timeline they would like to view. These datasets will be ordered by year or both year and month.
- The product shall be viewable from an Internet browser when accessed using the Iowa State VPN.

1.6.2 Non-Functional Requirements

- The parsing of uploaded ASCII data should take less than 15 seconds.
- The viewing (rendering) of selected data shall be generated in less than 30 seconds; this means the datasets will be viewable and loaded by the product in less than 30 seconds.
- Once selected for creation, a dataset should be automatically created (from a CSV file) into a map service (using our automated Python scripting) in less than 10 minutes.
- When values are toggled, the values should be appropriately sized to the point that they will not cascade, or billboard, toward the user. The user should be able to differentiate between various points through color and/or size.
- The system will be able to seamlessly support 8 simultaneous users, 2 of whom will be able to simultaneously browse the same map (same compound for the same month).

1.6.3 System Requirements

- Our client will submit her data as a single ASCII format. The data will be formatted in a way such that the plotted data of several maps can be represented on a matrix, with each compound making up similar sparsity on the matrix.
- The website will be accessible by the client and any end-user, provided they are connected to the Iowa State VPN. Making the website available outside of the ISU VPN would be trivial on our end, but would require the proper vetting/red-tape from IT services.
- We assume that the client will entirely run this product off a University owned and provided server, and has no plans to make money off the product (as agreed to in our ArcGIS for Server license). This is important primarily because we are using ArcGIS as an educational license, which is provided to us for no cost as long as it is solely utilized for educational and informational use.
- Web server usage will take priority over the processing of newly imported compounds. In other words, if 8 users are trying to browse maps, then processing of any additional ASCII files will slow down until web usage diminishes.

1.7 Proposed Testing Methods

In an attempt to ensure the correctness of our client's transformed data, we were provided the original 2D images to compare against our generated 3D maps.

Testing will primarily focus on the speed of the automated map creation system and user-reactivity of the web page. In an attempt to provide end-users with an interactive system, the system should not lag significantly when zooming, panning, or loading maps.

Finally, we need to carefully manage our server's hardware usage (CPU, RAM, and HDD). This is important because our budget requirements place us on a shared University server that hosts multiple VMs.

1.8 Team Collaboration / Proposed Scheduling

The team will use GitHub as a VCS and Hangouts for general team collaboration. Additionally, the team will work in short sprints to maximize efficiency and increase accountability, utilizing the project management tool Asana to keep track of pending issues and proposed improvements in the next cycle.

1.9 Expected Challenges

Major expected challenges for the team come in two of the following areas: data size and software stability. In terms of data volume, the size of the datasets given by the client can cause rendering issues due to the sheer amount of items being generated by any single map. Additionally, unfamiliarity with the ArcGIS software suite might cause delays during a sprint cycle as team members continue to learn the platform.

Section 2: Product Internals

2.1 Software Support

Like many software projects today, our end product was created using a mixture of various existing software. The software we were able to utilize were as follows:

ArcGIS for Server

The GIS software that is utilized by our automated daemon in the creation of map services. These maps are then hosted by the ArcGIS for Server platform, which allows them to be accessible via a RESTful API.

* Only available with proper licensing

Link: <http://server.arcgis.com/en/>

ArcMap

The desktop software which allows for the manual creation of maps and their services. This product will ultimately be used by the client and her associates to create template files upon the addition of a new map region.

* Only available with proper licensing

Link: <http://desktop.arcgis.com/en/arcmap/>

WinSCP

File transfer application that securely transmits the newly generated JavaScript file upon the creation or deletion of a map service.

Link: <https://winscp.net/eng/download.php>

Procrun

An application used to run our Daemon as a Windows Service.

Link: <https://commons.apache.org/proper/commons-daemon/procrun.html>

Gluon Scene-Builder

GUI builder tool which aided in the creation of the client-side application designed to act as an interface between our client and the headless daemon.

Link: <http://gluonhq.com/products/scene-builder/>

Closure Compiler

A tool for minifying JavaScript code. This allows us to convert our automatically generated, human readable, JS into something that is as small as possible (to minimize the amount of data our web server has to transfer).

Link: <https://developers.google.com/closure/compiler/>

tinylog

A logging framework that we used to easily output error messages and other important information to a log file that is stored on the local disk. These logs are also accessible to the client via the GUI tool.

Link: <http://www.tinylog.org/>

2.2 Implementation Details

2.2.1 Daemon / Server-Side

The daemon is responsible for getting (and checking) information from the client and interacting with all the individual components of the map generation process. The architecture/flow diagram shown in the “Python Scripting” section illustrates the linking of this process in greater detail. Additionally, some of these critical components are outlined below:

ConvertedSet.java

Class that keeps track of all the ASCII files that have been converted by the daemon. An object of this class is used in key daemon operations (to ensure that the client cannot break the server by accidentally attempting to overwrite an existing map), so it is important to carefully consider implementation changes on this object. Making a change that requires an increase in the serialVersionUID nature of this object will cause inconsistencies and/or failures in server operations unless all previously stored values are converted to the new Serializable. Also used to get map information for the generation of JavaScript.

EarthModellingDaemon.java

Class that represents the main server thread that handles all generation of maps and related elements by calling the relevant helper classes, parsers, executables. Public methods are synchronized to prevent deadlock issues.

ClientServer.java

Server-side class that handles client connections and interactions between a ClientThread and the main daemon.

ClientThread.java

Server-side class that parses and handles the network streams between the server and an individual client.

JavaScriptGenerator.java

Class that generates a new JavaScript file that handles loading our map's HTML.

CompoundDescriptions.java

Utility class that will store compound descriptions for automatic generation of JS, and return the appropriate reference scale given a MapCompoundType. Defined via Reflection to make it easy for non-technical folks to add or change reference scales later.

ReferenceScales.java

Utility class that will store reference scales for Python Scripts, and return the appropriate reference scale given a MapRegionType. Defined via Reflection to make it easy for non-technical folks to add or change reference scales later.

FileLocations.java

Utility class whose sole purpose is to store directory locations among common classes. It will make it easy to change drive parameters on the disk.

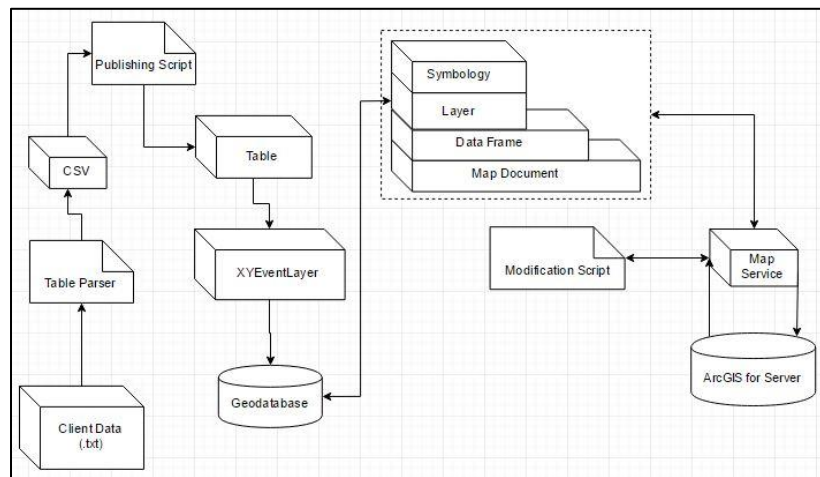
2.2.2 Parsers

Java Parser

| | | | | |
|----|---------------------------|-----------|-----------|-----------|
| 1 | <code>ncols</code> | 1404 | | |
| 2 | | | | |
| 3 | <code>nrows</code> | 924 | | |
| 4 | | | | |
| 5 | <code>xllcorner</code> | -169 | | |
| 6 | | | | |
| 7 | <code>yllcorner</code> | 7 | | |
| 8 | | | | |
| 9 | <code>cellsize</code> | 0.0833333 | | |
| 10 | | | | |
| 11 | <code>NODATA_value</code> | -9999 | | |
| 12 | | | | |
| 13 | | -9999.000 | -9999.000 | -9999.000 |
| | | -9999.000 | -9999.000 | -9999.000 |
| | | -9999.000 | -9999.000 | -9999.000 |
| | | -9999.000 | -9999.000 | -9999.000 |

Our Java parser serves as the component that takes our client's ASCII table (.txt) and translates it into an ArcGIS readable format (.csv). The parser begins by extracting the elements in the header of an input file (shown above). These items include the coordinates of the lower left corner of the table (in latitude and longitude), the number of rows/columns of the table, cell size (which will later affect reference scale), and the value of data to be ignored. After successfully parsing these header values, the computation of data within the table begins. By calculating the relative space of the upper-left corner of the table, we then calculate relative points downward and to the right using the cell size variable. We also continually store the min and max compound values and eventually output them as the first and second output values in the converted (.csv) file. This is good practice because, as we discovered during the creation of map symbology, ArcMap sets the minimum and maximum values for the visual classes based on polling the first few values stored in the dataset.

Python Scripting (ArcPy)



After successfully transforming the client's .txt file into a .csv file, we then use a pair of Python scripts that utilize the ArcPy library provided by the ArcGIS platform. The first script transforms our converted .csv into various ArcGIS datatypes. First, the CSV is turned into a table, which serves as the first ArcGIS datatype we actually work with. That table is then converted into an XY event layer that serves as a visual object that can be examined in ArcMap. At this point, the event layer does not have any of the applied symbology (it cannot be distinguished by color).

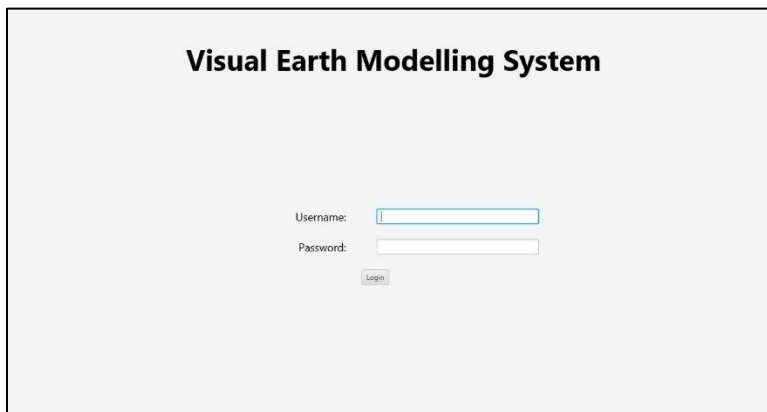
Finally, the event layer can be transformed into a geodatabase. Spatial geodatabases were chosen as the desired data source due to team testing; as our experience with ArcGIS for JavaScript showed geodatabases significantly outperforming other data sources during the real-time rendering map services. Once the geodatabase has been created, we are finally ready to swap in our data.

We begin by opening a template file and swapping the data source of the existing layer with the newly created geodatabase. The embedded layer of the template file can then be independently saved locally without saving permanent changes to the template file. It is important not to save changes to the template files, as they are reused throughout the system. Once the layer is saved locally, we open an empty map document and insert the modified layer. Finally, the symbology to the new document is applied from the template file. Once all of these transformations/edits have taken place, the map gets saved locally, and finally published to our ArcGIS Server. Upon successful publishing, we then call another script which performs small modifications to the published parameters. These parameters are utilized to minimize the strain end-users can place on the system.

2.2.3 Client GUI

The GUI was built using an MVC design paradigm. Each GUI page is represented by a controller that controls an FXML layout (the view). The model is represented as a singleton that stores a user's current session information.

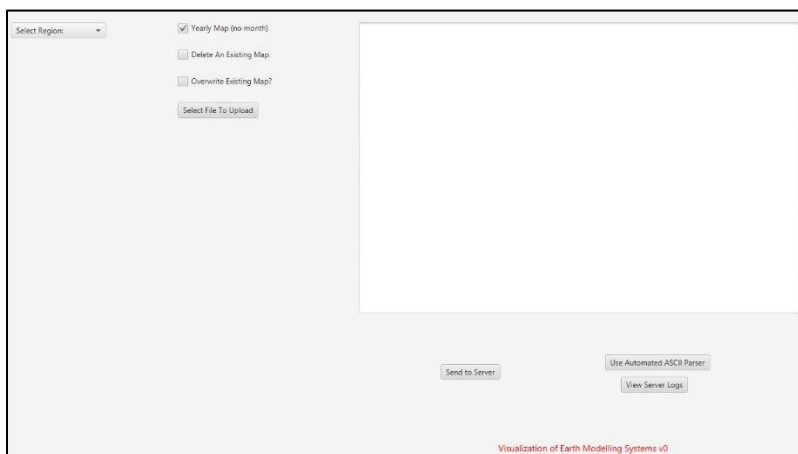
Login



The screenshot shows the login interface for the Visual Earth Modelling System. At the top center, the title "Visual Earth Modelling System" is displayed in bold black text. Below the title, there are two input fields: "Username:" followed by a text box, and "Password:" followed by a text box. A "Login" button is positioned below the password field. The entire interface is set against a light gray background.

The login Screen of the GUI. After entering the master username and password, the user is prompted to enter the keystore file, which is a collection of certification files that allows for a secure connection to the server (protected via TLS). If the server authenticates these details, the user is then brought to the upload map screen.

Upload Map



The screenshot displays the "Upload Map" screen. On the left side, there is a "Select Region:" dropdown menu. To its right, there are three checkboxes: "Yearly Map (no month)" which is checked, "Delete An Existing Map", and "Overwrite Existing Map?". Below these checkboxes is a "Select File To Upload" button. At the bottom of the screen, there are three buttons: "Send to Server", "Use Automated ASCII Parser", and "View Server Logs". The footer of the screen reads "Visualization of Earth Modelling Systems v0".

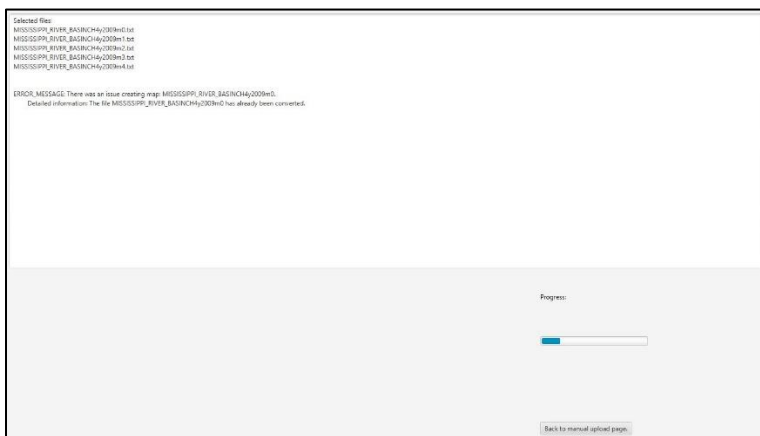
The key feature of the client application revolves around the ability to upload a map. After a successful login, the application allows the client to select regions, compounds, and enter year/month on the left hand-side. The user will then select the dataset to upload and send to the

server. If all selected parameters are correct, the daemon then permits the dataset to be sent through the map creation process. Should any error arise, the user is notified of the error with a popup.

Delete / Overwrite Map

Should the client ever feel the need to delete or modify a map service from the website, checkboxes have been included to allow that functionality. The delete will remove the entire map service and all related files from the server. An overwrite will delete the existing map and create a new map service with the dataset provided by the client.

Upload Multiple



Since the client's workload often causes maps to be updated and processed in batches, she requested the ability to upload multiple datasets at one time. As a result, we provided a standardized method to upload multiple file at the same time, with the information parameters of a map dataset being pulled from a dataset's filename.

Due to the static names of the template files being used, it is crucial to standardize the naming convention of the input files. Therefore, to utilize a multiple upload, the client is required to modify file names to match the format <R><C><y><m>.txt where:

R = Region (example: GLOBAL)

C = Compound (example: CH4)

y = year (example: y2005)

m = month (example: m_1)

After this quick modification, the multiple upload feature can be utilized. As with the standard upload screen, the user is also notified of any errors or successes, as they happen. Additionally, this screen gets an upload status bar to notify the user of overall map processing status.

View Server Logs

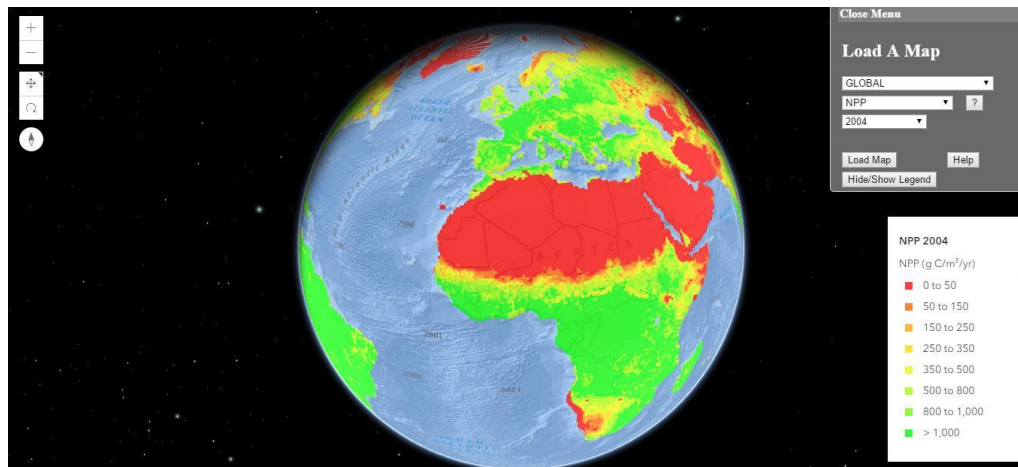
```
2017-03-29 16:22:28
2017-03-29 16:54:56
2017-03-30 15:28:20
2017-03-31 15:07:41
2017-04-03 16:10:47
2017-04-03 16:11:33
2017-04-04 09:42:54
2017-04-06 14:48:18
2017-04-06 15:00:42
2017-04-06 15:13:06
2017-04-06 15:49:00
2017-04-09 13:45:56
2017-04-09 16:34:43
2017-04-09 18:45:40
2017-04-09 22:40:12
2017-04-09 23:17:09
2017-04-10 00:02:00
2017-04-10 00:02:08
2017-04-10 01:02:04
2017-04-10 01:05:52
2017-04-10 01:08:11
2017-04-10 01:20:02

2017-04-06 15:43:26 [Thread-2] networking.ClientThread.initializeUser()
ERROR: java.lang.NullPointerException
    at networking.ClientThread.initializeUser(ClientThread.java:70)
    at networking.ClientThread.run(ClientThread.java:11)
2017-04-06 15:43:26 [Thread-2] networking.ClientThread.initializeUser()
ERROR: java.lang.NullPointerException
    at networking.ClientThread.initializeUser(ClientThread.java:70)
    at networking.ClientThread.run(ClientThread.java:11)
2017-04-06 15:43:26 [Thread-2] networking.ClientThread.initializeUser()
ERROR: java.lang.NullPointerException
    at networking.ClientThread.initializeUser(ClientThread.java:70)
    at networking.ClientThread.run(ClientThread.java:11)
2017-04-06 15:43:26 [Thread-2] networking.ClientThread.run()
INFO: null failed attempted connection too many times.
2017-04-06 15:43:26 [Thread-2] networking.ClientThread.run()
INFO: null successfully disconnected from the server
2017-04-06 15:43:26 [Thread-2] networking.ClientThread.run()
INFO: null successfully disconnected from the server
2017-04-06 15:44:29 [Thread-3] main.EarthModelingDaemon.stop()
INFO: Shutting down server
2017-04-10 01:26:03 [main] main.EarthModelingDaemon.start()
INFO: Server daemon is starting up.
2017-04-10 01:26:03 [main] main.EarthModelingDaemon.start()
INFO: Starting VMMS ClientServer
2017-04-10 01:26:03 [Thread-1] networking.ClientServer.run()
INFO: Waiting for clients on port: 1443
2017-04-10 21:18:06 [Thread-2] networking.ClientServer.validateUser()
INFO: test successfully connected to server
2017-04-10 21:53:40 [Thread-2] main.EarthModelingDaemon.convertKMLtoCSV()
INFO: Converting file: C:\EarthModelingDaemon\Temp_Working_Files\MIDWESTERN_US\NUPTRAKY\9994m-1.kml to CSV
2017-04-10 21:53:45 [Thread-2] main.EarthModelingDaemon.convertKMLtoCSV()
INFO: File converted to CSV.
2017-04-10 21:53:45 [Thread-2] main.EarthModelingDaemon.runExecutable()
```

One critical page of this system is enabling the client to be able to look at our server logs and attempt to discern errors and issues on her own. Since all critical actions performed by the daemon are logged, the client will be able to monitor login/logout activity, as well as script success/errors/warnings. If a failure warning pushed by the server (and ultimately displayed by the GUI) doesn't contain specific enough information to describe what went wrong, the logs can provide key insight into resolving the problem.

2.2.4 End-User Website

The website serves as the main desired component of the system; the 3D visualization of our client's work. By taking advantage of some ArcGIS for JavaScript libraries, we were able to load our hosted map services using a RESTful API. Our webpage simply requests from our hosted server the hosted map service (by URL) and that is returned to the user in the form of a map layer containing its associated legend symbology.



Auto Generated JS

Using Java, we were able to auto generate our JavaScript in order to keep the drop down selection updated as our client generates more maps. The creation of a new map causes the regeneration of this JS to happen automatically.

Load Map

The REST URLs are created by the values the end-user selects in the region, compound, year, and if supported, the month dropdowns. The load map button then calls the listener to create the layer on the main view of the webpage. The values of each selection are appended to create a string URL that is then used to create the layer with the functions in the ArcGIS JS.

Help / Explanation of Compounds

As we entered the deployment stage of this project, we did some usability testing to try and see what end users think of the project. One of the comments we consistently heard goes something along the lines of "What does the name of this compound mean?" Since one of the main goals of this project is to broaden the ability of interested (and potentially non-technical) parties, the team

and Dr. Lu felt it might be beneficial to provide a link or explanation of what the compound phrases might mean. This was implemented by providing a button that causes a popup to display compound information. When the end user is selecting a compound to view, they are given the ability to select the explanation “?” button present next to the drop down list to view details and description about that selected compound. Additionally, a help button is also present to explain to users how to navigate through the webpage, and this help information is also displayed upon initial load of our webpage.

Close Menu / Toggle Legend

While ESRI’s ArcGIS for JavaScript does not currently support mobile browsers for 3D mapping, we found that many mobile browsers could use our website’s functionality without issues. As we developed the web UI, we found that mobile devices could no longer view the map as the map menu and legend covered the majority of screen real estate on mobile devices. Our solution was to allow users to be able to toggle the menu, the map legend, or both. Ultimately, this will allow users to view the data on most mobile devices without an obstructed view.

Section 3: Testing and Evaluation

3.1 Parser (.txt to .csv) Speed

According to the system logs that are generated by the daemon, our Java parser processes the largest maps (1, 297,269 points) we are given at a consistent five-second runtime from start to finish. The smallest maps we are given (254,880 points) are parsed in less than one second. In our opinion, this run-time could be enhanced furthermore if the Java Parser did not require storing the min and max as the first two values of the output .csv file. This operation results in several additional operations instead of a simple calculated output.

3.2 Map Creation Speed

On our server, map creation & publication runs variably within 20 to 30 seconds. The modification script runs in variable times from 5 to 10 seconds. In terms of total time from initial parsing to the final push of the updated JavaScript, this process is typically done within a 45 second window, with a maximum observed time of 1 minute. We feel as if this is a result of network latency and a system that has to actively handle incoming map service requests (from the webpage). ESRI allows the GIS server to be decoupled from the processing of maps through the use of server clusters, and we would recommend future developers to consider such an implementation should performance requirements illicit it.

3.3 End-User Reactivity

Smooth operation of the website was stressed as an important facet of the product. While this is not easily quantifiable in terms of unit or integration testing, we did our best to minimize load times and reduce the stresses users can place on the website.

In our testing, a large part of end user usability is determined by the machine that is requesting the information. For example, a modern desktop might have very fluid interaction and panning of a map, while an older mobile device might have unacceptably slow response times. We believe that part of this can be the reason for ESRI's official declaration that mobile devices aren't supported.

Load Time

In our experience, an initial map load generally performs a complete rendering within 10 seconds. Once spun up by the server, however, the map service requested can typically be loaded by any user in less than a few seconds.

Interaction Time

Interaction time, while stressed in the requirements of the product, ultimately also becomes dependent on the end-users hardware.

3.4 Resource Management

In loading many maps, we came across an interesting issue: we could not cache map services due to the resources required to do so. As mentioned earlier in the document, we are working with a shared VM and have limited disk space, as well as limited CPU and RAM resources. Typically, when displaying visual data, a map service will pre-render the viewable areas of a map a series of images, called tiles. For example, the global basemap that we are using in our final application requires close to 2TB of storage (this is provided to us, at no cost, by the ESRI CDN). While doing so is programmatically easy, the issue with an interactive 3D map is such tiles need to be viewable at multiple scales. In other words, each time a user zooms in, the number of tiles that need to be generated to represent the same area increases exponentially. This is a concept that we've found hard to explain, so think of it this way: you have a window, or a screen size, that can be represented in 100 tiles. Mathematically, this would be 10×10 . If you were to make one of those tiles equal to the entire window size, the new number of tiles that would represent one tile would now be 100. As such, increasing the scale level by this factor would require the new number of tiles to be generated to be equal to $100 * (10 * 10)$, or 10,000 tiles. As one can see, this makes pre-rendered tiles on a large number of interactive datasets unfeasible in terms of disk usage.

Our solution to this issue was to create map service definitions that would display and render maps dynamically at runtime. The constraint we had here was controlling memory and CPU usage. In our testing, hosting a dynamically generated map service requires a couple hundred MB of RAM per user that is connected (and about 100 MB for an idle service that doesn't have a user connected). Since our client has hundreds, if not thousands, of datasets this wouldn't be feasible from a cost perspective. The answer to this was map processes that would be spun up only as requested by the user. Additionally, we limited the number of threads each map service could run on and the total draw area (resolution) that an individual user can request to be rendered at any given time. In order to do this, multiple requests on the same instance are simply queued and answered in a FIFO pattern.

3.5 Input Data Validity

While the correctness of a supplied dataset rests in the hands of our client, we were able to visually compare our generated maps with the original image files of these same datasets. Using this comparison, we were able to determine that our Java parser works successfully as planned.

Appendix 1: Operation Manual

Appendix 1.1 Cloning the Repo:

1. Obtain Windows Server 2012 R2 Standard (VM)
2. Install the Most Recent JDK from <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>
3. Download the Eclipse IDE from <https://www.eclipse.org/downloads/>
4. After downloading the installation files, first run the JDK installer. Default installation settings are fine.
5. Once the JDK has finished installing, run the Eclipse Installer. The desired install is the Eclipse IDE for Java developers.
6. Once all have been successfully installed, run the Eclipse IDE and select the desired place for the java workspace.
7. Next, we will clone the git repository. The url for the repository can be found at https://github.com/kj20207/SD_may1701_EM/
8. Navigate to the link, and click the download button.
9. Next, click the copy to clipboard button to copy the repository URL.
10. Navigate your way back to Eclipse, and select File->Import.
11. On the popup window, expand Git and select "Projects from Git". Click Next.
12. Select "Clone URI" and click next.
13. The URI field should be auto-filled, but in case it wasn't, paste the value copied from step 9.
14. Authenticate yourself using your own Git Username and Password if you are going to be modifying the Git Repository. Otherwise, skip this step and hit next.
15. Clone the master branch. The default destination directory is fine. Clicking next will begin the cloning of the repo.
16. Once options are selectable again select "Project using the New Project wizard". Click Finish.
17. Then expand the "Java" folder, and select "Java Project".
18. Once the "Create a Java Project" window pops up, select "Use Default Location" and select the default destination directory from step 15. Once the directory is selected, go ahead and click finish. Congrats, you've setup the IDE and cloned the repo!

Appendix 1.2: Fresh Install of Web Server & Daemon

1. Install ArcGIS for Desktop (with plugins: Data Interoperability for Desktop, Web Adaptor (IIS), ArcGIS Coordinate Systems Data) * Requires Licensing Agreement
2. Install ArcGIS for Server and configure for personal use (you'll have to use your own information). * Requires Licensing Agreement
3. Open ports 6080 and 6443 for ArcGIS for Server access, as well as a desired port for GUI access (this port will be explicitly set later).
4. Within the eclipse repository, navigate to the install folder. Copy the folders "EarthModellingDaemon" and "procrun" to the C: drive.
5. Download procrun from <http://www.apache.org/dist/commons/daemon/binaries/windows/>

The desired version is commons-daemon-1.0.15-bin-windows.zip

6. After downloading and unzipping, copy all of the files within the extracted folder, and paste them into the “procrun” folder on the C: Drive.
7. Additionally, copy the prunsrv.exe from the procrun folder, and the tinylog.properties file from the repo “resources” folder into the “C:/procrun/amd64” folder.
8. Install WinSCP.
9. Within the cloned Eclipse repo, navigate to src-shared/networking/ServerInformation.java

Edits will have to be made for the VM addresses. The host names will need to be changed to reflect your own system specifications. Additionally, change the other port opened to allow for GUI Networking access (from step 3).

10. Obtain a separate Linux web server.
11. Now that the files are stored locally on the VM, navigate to the C:/EarthModellingDaemon/resources folder.
12. The folder “End-User Website” houses all of the crucial website information.
13. Using WinSCP, these files can just be copied to the desired folder within the web server
14. Within the cloned Eclipse repo, navigate to src-shared/networking/ServerInformation.java
15. Edits will have to be made for the web server URL address (where the website is hosted). They will need to be changed to reflect your own system specifications.

To Generate a Keystore:

16. Open the Command Prompt (Admin), and direct yourself to the keytool.exe file located at “C:/Program Files/Java/jre1.8.0_121/bin/keytool.exe”
17. Once you’ve navigated to the correct directory using the Command Prompt, enter the following command:

```
keytool -genkey -keyalg RSA -alias selfsigned -keystore keystore.jks -storepass password
-validity 3650 -keysize 2048
```

*Password should be the desired Password.

18. After entering your information and finishing this process, the keystore file will then be created in the same bin folder as the keytool.exe.
19. Copy this file into the directory C:/EarthModellingDaemon/resources/
20. Next, update the approvedClient.txt file located in C:/EarthModellingDaemon/resources/
21. To enter a username and password on a line enter:

```
username <tab> password
```

22. Copy the keystore file AGAIN to a secure location, as you will need it to login to the GUI.

Finally, to install the Daemon:

23. Next, you will need to copy the file located at C:/EarthModellingDaemon/resources/installService.bat to the C:/procrun/amd64 folder. Next, open the file in a text editor of your choice.
24. Line 21 will have to be modified to match your keystore password, arcgis server information, and webserver information. The value “startOrstop” should also be changed to “start”.
25. Navigate through Package Explorer to src-server->main->EarthModellingDaemon.java.
26. Right-click on EarthModellingDaemon.java and select “Run As”-> Java Application.

This should default our run-arguments to the correct ones. The program will fail, as it is not programmed to run in this environment.

27. Next, right-click EarthModellingDaemon.java and select Export.
28. Navigate to Java->Runnable JAR File.
29. Under Launch Configuration, select “EarthModellingDaemon”, and select your desired export destination.
30. Name the output file daemon.jar
31. Select the button “Extract Required Libraries into generated JAR”.
32. Select Finish. Copy the created daemon file into the folder C:/procrun/amd64.
33. Open command prompt (admin) and navigate to the C:/Procrun/amd64 directory.
34. Assuming you’ve completed the steps and made the necessary modifications, run the following two commands:

```
installService.bat
```

```
prunsrv.exe //ES//EarthModellingDaemonMaster
```

35. Last but not least, copy the files within the repo from src-server/Python_Scripts to the C:/EarthModellingDaemon/Python_Scripts directory.

The Daemon and Website should now be functionally running on their separate servers.

Appendix 1.3: Using our Website

1. For the end-user 3D experience, the only user requirement is that the user must be connected to the Iowa State University VPN. To connect to the VPN, a user must simply download the VPN Client from (<https://www.it.iastate.edu/howtos/vpn>), and enter their login information.
2. After the VPN Connection has been established, a user can navigate to <http://may1701.sd.ece.iastate.edu/VEMS/VEMS.html> using a browser of their choice (we have had extensive successful testing on Chrome, Firefox, Internet Explorer, Microsoft Edge and Safari).

Appendix 1.4: Using the GUI

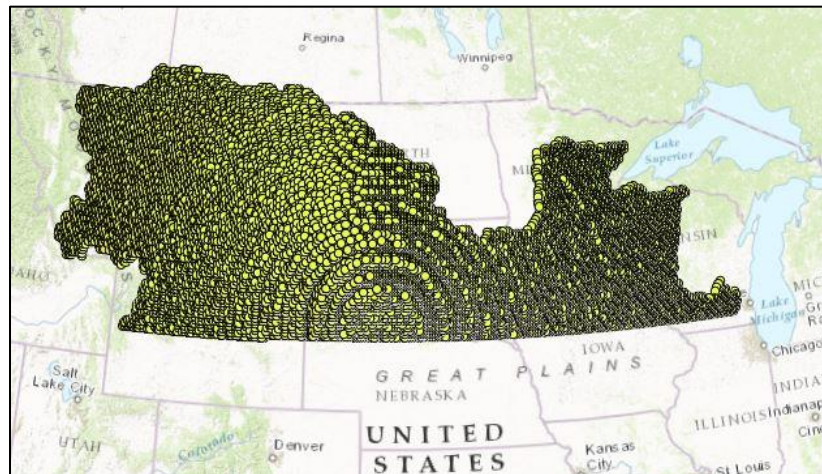
1. In Eclipse, Navigate through Package Explorer to src-client->view->MainView.java.
2. Right-click on MainView.java and select “Run As”-> Java Application.
 - a. This should default our run-arguments to the correct ones.
 - b. The login screen of the GUI should popup on your screen.
3. Next, close the popup window, and right-click MainView.java and select Export.
4. Navigate to Java->Runnable JAR File.
5. Under Launch Configuration, select “MainView”, and select your desired export destination.
6. We generally used C:\tmp\VEMS.jar
7. Finally, select the button “Extract Required Libraries into generated JAR”.
8. Select Finish.
9. The Client application can now be run by navigating to your Export Destination and double-clicking the created .jar file!

*Note: due to the security ensured to the client, external users will not be able to login to the client GUI without the authentication information and our generated keystore. For this, you will have to have your own system, and generate your own keystore. This can be demonstrated in the above how to documentation.

Appendix 2: Alternative / Other Initial Designs

Dynamically Render all Data Client-Side

One of the earliest prototypes of the system consisted of the java parser and the ArcGIS JS library. ArcGIS allows for dynamic placement of data through the usage of input files (in our case a csv) which can be completely rendered client-side, resulting in a system which does not incur the burden of simultaneous users. After parsing the file and attempting to render the file, we experienced extremely slow load times (upward of two minutes) and the following display:



What we found was that the client-side rendering maxes out at roughly 17,000 individual points. Due to the size of our client's data files, this approach quickly became unsuitable for our requirements. In an effort to solve this challenge, this is when we began to implore into the possibility of taking advantage of other ArcGIS Software (other than just rendering libraries) to lessen the load on end-users and increase performance/stability throughout the system.

Cached Map Services vs Dynamic Map Services

One of the largest problems we faced was the decision to go with cached map services or dynamic map services. Due to the size of the data sets that we are rendering (plus having hundreds to thousands of maps to eventually generate), this became a very crucial decision. Dynamic map services would result in less performance for a virtually unnoticeable size (a couple megabytes) per geodatabase. Cached map services would have resulted in a product that would have almost seamless user interaction times, but would cost our client roughly 2 terabytes per map generated. Since the maps have to be individually rendered at each zoom-level, this is what causes the balloon in size. Additionally, each additional zoom-in causes an additional factor of two in images generated.

What aided us in our decision came as a result of fine-tuning our map service properties. After requesting an increase in hardware, we limited each service to only be active for a limited amount of time, and restrict the services to be accessed by a maximum amount of users at once. These additions plus our server simultaneously handling some of the burden requested by users, resulted in a product which would give our client suitable end-user reactivity and not cost her additionally in storage space.

Appendix 3: Considerations

Cost

Due to the fact that Iowa State has a license with ESRI to utilize their ArcGIS Software, we were able to create this entire system with no additional cost to the university or our client. Should she feel the need to move the hosted server to a private server (not university-owned) or collect money from this product, she will have to consult with ESRI for independent use pricing.

Google Earth

One of the earliest considerations for this project was to use the Google Earth API as the basis of this solution. What we found, however, is that the Google Earth API had become deprecated and was being completely shut down. This caused the team to explore for viable solutions, and how we ended up being ushered into the ArcGIS software.

Unpredictable ArcPy Functionality

First drafts of the system had no explicit definition of a GUI to be used by our client. Due to the idea of the Daemon, we had expected that the automation process would be simple enough to create that we could continue onto other aspects of the system which revolve around extensive end-user interaction. What we found, however, is that when scripts failed, it was often fairly difficult to comprehend what exactly was going wrong.

Read-Only Values in ArcGIS Map Documents

Once ArcGIS was agreed upon as the suitable replacement to Google Earth, the team explored through the different possibilities available using ESRI Software. After much experimentation through ArcMap and the discovery of the Arcpy library, we finally thought we had found our completely automated solution. Later on, however, we came to the realization that the arcpy API (through extensive research and to our own knowledge) does not allow for the complete drafting of Map Documents from scratch via Python scripts. Due to this feature, we knew we either had to let some other process within our system be manually created (the creation of template files), or find another system entirely. The decision was made that the team would continue with ArcGIS given the usage of template maps, as this will allow the client to not only become more familiar with the ArcGIS products, it will allow her to define files which will be exactly to her liking. Additionally, since the creation of a template map only takes a couple of minutes (once the process is known), the template can be re-used infinitely throughout our system.

Appendix 4: Future Implementations

After speaking with the Client and our Advisor, we've come to the conclusion that the majority of future development upon this product would be in creating animated timelines, and statistical analysis reports from the queued data. In an initial presentation by our Client, it was mentioned that these features could be added to aid users in understanding the change of hotspots over time (in animation); and extensive analysis by providing more in-depth reports of the data which is being seen. Additionally, since our client creates all of the historical datasets, projection reports could be added as well.

We feel, as a team, that we have created a product that can be improved upon greatly due to our initial struggles in learning how the ArcGIS platform works together. The struggle in grasping the initial concepts of the platform resulted in creating a system that has many components that ensure system stability. Future modifications could have been created had we been able to figure out automation earlier on in the process, and if the GUI did not come as a result of unpredictable behavior in scripting.

Appendix 5: Creating Template Maps

Due to the fact that our system relies on Template Map files for the creation of symbology, we thought it would be necessary to describe the steps which revolve around this process. This also would describe the traditional way of creating a map service (the problem we attempted to fully solve).

- First, a user opens the ArcMap program and creates a blank document.
- Next, they select File->Add Data -> Add XY Data.
- At this point, they are prompted for a csv file.
- After selecting the file, they must apply a coordinate system to the layer.
- Now, an XY Event Layer has been effectively added to the map document.
- Next, they would right-click the layer, select Properties, and make their way to Symbology.
- Within this screen, they can add Graduated Colors (the style we chose) or any other desired stylings. Furthermore, they can change the name of the value that the color depends on, the colors associated with the classes, the number of classes, and the symbol to be generated at each position if they were to choose Graduated Colors.
- After applying this Symbology, they select the Data Frame and modify the Reference Scale (scale size at which values are drawn).
- Finally, they would Publish this map as a Service to their ArcGIS for Server where it can be accessed by others.

At this point, due to the restrictions of the software, our client will have to follow this process (minus the Publishing) should she feel the need to add additional compounds.