

# Using AJAX to Develop a Web Interface to Run CO<sub>2</sub> Sequestration Simulations on Remote Servers

COMP 670 Final Report  
Ronald M. Caplan

December 18<sup>th</sup>, 2009

## Introduction

In the past few years there has been an increasing desire to reduce carbon emissions produced from industrial plants. Many techniques have been proposed and implemented. One such technique is the capture and storage of used carbon called CO<sub>2</sub> Sequestration. The success of this process is highly dependent on the material compounds within the ground, and how CO<sub>2</sub> flows through them.

In order to determine a good location for CO<sub>2</sub> sequestration, large-scale simulations are performed on a computer to simulate the CO<sub>2</sub> in the materials. These simulations are done frequently and have many different parameters. One such software package is called Sim.8 General Water-rock Interaction and Reactive-Transport Simulation. Since Sim.8 is a very complicated code to run, a simple GUI interface for plant personal to run the code is desired. Often the computations are required while the users are at remote locations (such as a mine, or on a ship) where computer resources needed to run the computations efficiently are not readily available. Due to the widespread availability of wireless internet access and cheap netbook computers, a web-based solution to the problem can be developed.

The basic idea is to create a web-based interface in which a person in a remote location (with internet access) can input parameters into a webpage and submit the information, which then runs Sim.8 on a remote server housed elsewhere. The interface would then receive the results from the computation and display them to the user. In this way, someone with a tiny, low-power netbook computer (or iPhone) can run large Sim.8 simulations on a powerful mainframe. Since such systems can compute results for typical problems many times faster than a mobile PC, having an efficient web interface could greatly improve efficiency.

In this report we investigate the use of AJAX to create such an interface for Sim.8.

## Sim.8

The Sim.8 General Water-rock Interaction and Reactive-Transport Simulation was developed by Sienna Geodynamics (SGC). The algorithm of the code is described by them as

“Using kinetic and thermodynamic reactions over discretized spatial domain to model compositional and textural evolution of sediments in geologic and engineering time scales. “

The code takes in many parameters including rock composition data in the form of an input file given a ``.sdb`` file extension. The code then runs the simulation and outputs the results in a ``.dout`` file. The results are outputted as the simulation runs so one can read partial outputs even before the entire simulation is completed. The output file contains many different types of data which require parsing. To do this, a program called “view1d” was written to select specific output data and parse it out of the

.dout file into a two-column ascii file. This file can then be used to plot the results using any plotting program desired.

## **AJAX, NetBeans, and FireBug**

There are several ways to develop a web-interface. One of the more recent developments is called AJAX (Asynchronous Javascript And XML). AJAX has a very large advantage over other web-interfaces because when information is submitted on AJAX, only the minimum information required is sent to the server, and the minimum output is received. This is done through the sending and receiving of SOAP messages. The web page itself is never reloaded, unlike when using form-based web-interfaces. This makes AJAX interfaces very responsive, mimicking the responsiveness of a desktop application.

AJAX applications can be developed with freely available software IDE packages. The IDE package we use is called NetBeans. NetBeans allows a user-friendly application development environment which combines the web-interface JavaScript code, the java application code (which can be programmed to call system commands on the server machine), and CSS style sheets. It also manages servers, and with one command can compile and combine a whole application project and deploy it to a server.

Another free tool that is useful in development is a FireFox plug-in called FireBug. FireBug allows the user to alter style sheets and see immediate results, as well as debug JavaScript code, see SOAP messages, and find errors and bugs in the application.

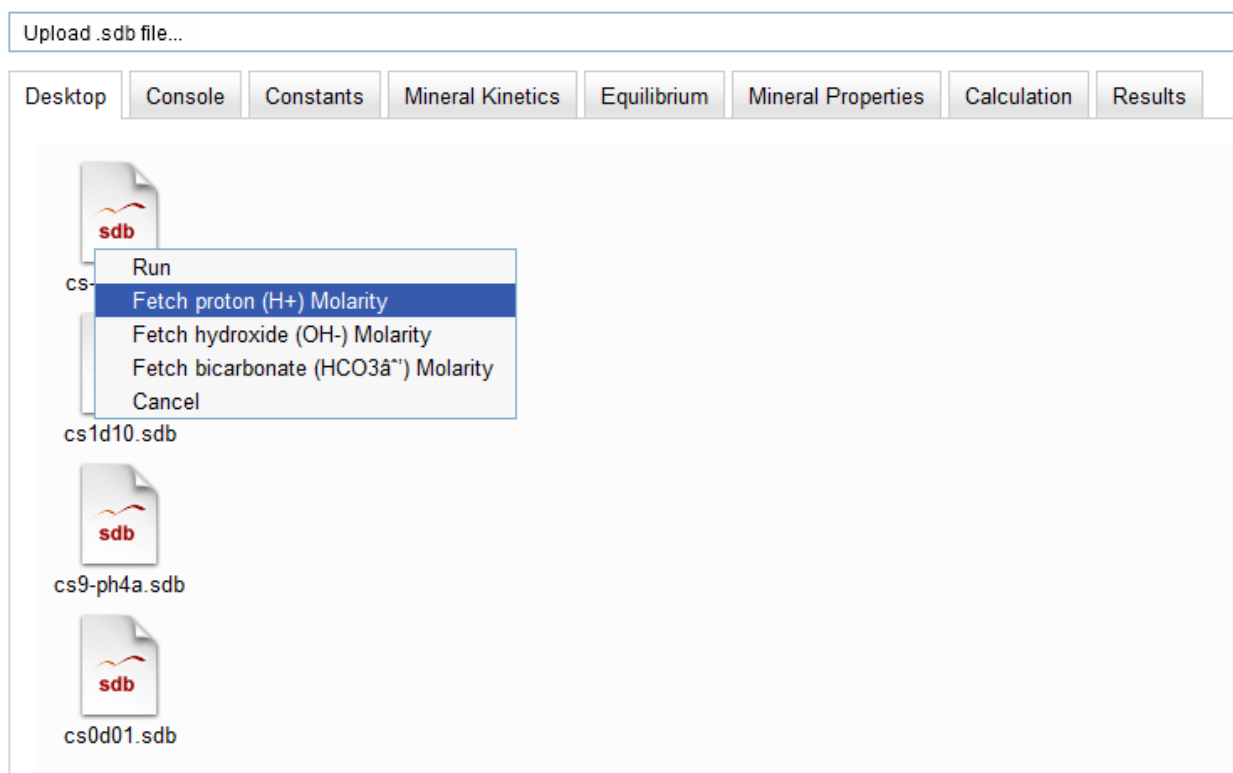
## **Sim.8 Web Interface and Implementation Details**

Our main task in building a web-interface for Sim.8 is to be able to upload an sdb input file, run Sim.8 and plot the desired results. The status of the interface before we worked on the project was that it could upload sdb files and run Sim.8 but did not return the results. Therefore, our task was to be able to retrieve and plot the results in the web interface.

There are three main components to our AJAX Sim.8 web interface. There is the javascript/HTML code that runs on the user's web browser, then there is the JAVA code which runs on the server and is invoked by SOAP messages (this JAVA code is called a webservice), and finally there is the Sim.8 code executables on the server.

A Sim.8 simulation is run by uploading an sdb file, and selecting the icon on the main interface and clicking on "run" in the pop-up menu. This action sends a SOAP request to a JAVA webservice which contains code to execute Sim.8 on the server with the input file specified. This webservice does not return anything (although returning the process ID would be very useful for future additional functionality). As soon as the Sim.8 code is running, an output file is created.

In order to plot result data from the run, we needed to add javascript code to request a plot/data for the selected run. As mentioned before, the program view1D is required to parse the output file to get specifically desired information. Which information is gathered depends on a menu number in the options for view1D. These menu numbers must be sent by the webpage to the server for retrieval. This is done in the input file's icon menu as seen here:



The javascript code that is run when a “fetch” option is selected can be represented by the following pseudo code:

```
create SOAP request string using input filename, selection #, timestep
SOAP action is set to invoke "getPlotData" webservice
```

The getPlotData webservice refers to JAVA code which retrieves the output data and generates a plot. It does this by executing the exterior program view1D. Unfortunately, view1D only runs in a command line interactive mode, so in order to automatically run it with the desired parameters, we must write a shell script with the command sequence to accomplish our task. This shell script has a .sh extension.

The view1D program only gets us a two column ascii file of the desired output. There are a few different strategies to be able to show this data as a plot on the web interface. The approach we use here is to generate a plot image and then reference the image on the web interface, causing the new plot to appear. To do this we add commands in the .sh script file to run GNUplot on the output of view1D which generates a 1D plot in jpeg format and stores the image in the OUTPUT directory within the web interface files folder. Our script thus takes the form:

```

#!/bin/bash
cd /opt/glassfish/domains/domain1/applications/j2ee-modules/rmc_Sim8/sdb/
BASENAME=`/bin/basename $1 .sdb`
FILE="OUTPUT/$BASENAME"
if [ ! -f $FILE.dout ]; then
    echo "file $BASENAME.dout not found in the OUTPUT directory"
    exit -1
fi
/usr/local/bin/view1d <<!
$FILE
0
1
$2
0
3
$3
0
-1
0
!
XFILE="xsp2.*d.t3.*dxy3.*dz3.*d.xgraph"
if [ -f $XFILE ]; then
    /bin/mv $XFILE ${FILE}_${2}_${3}.xgraph
else
    echo "file xsp2.*d.t3.*dxy3.*dz3.*d.xgraph not found"
    exit -2
fi
PLOTFILEIN="\${FILE}_${2}_${3}.xgraph\"
PLOTFILEOUT="\${FILE}_${2}_${3}.jpg\"
#PLOTFILE="\OUTPUT/testdata\"
#PLOTFILE="\OUTPUT/cs1d10_60_1.xgraph\"
/usr/bin/gnuplot <<!
set terminal jpeg
set output $PLOTFILEOUT
set size ratio .63662
plot $PLOTFILEIN title "Data" w lines
quit
!

```

Many more options are available for GNUplot, and these options should be exploited in future updates. The JAVA code to execute this script is:

```

String[] cmdarray = {"/bin/bash",
"/opt/glassfish/domains/domain1/applications/j2ee-
modules/rmc_Sim8/view1d.sh", filename, Integer.toString(parameter),
Integer.toString(timestep)};

process = runTime.exec(cmdarray, [], co2seqDir1);
int i = process.waitFor();

```

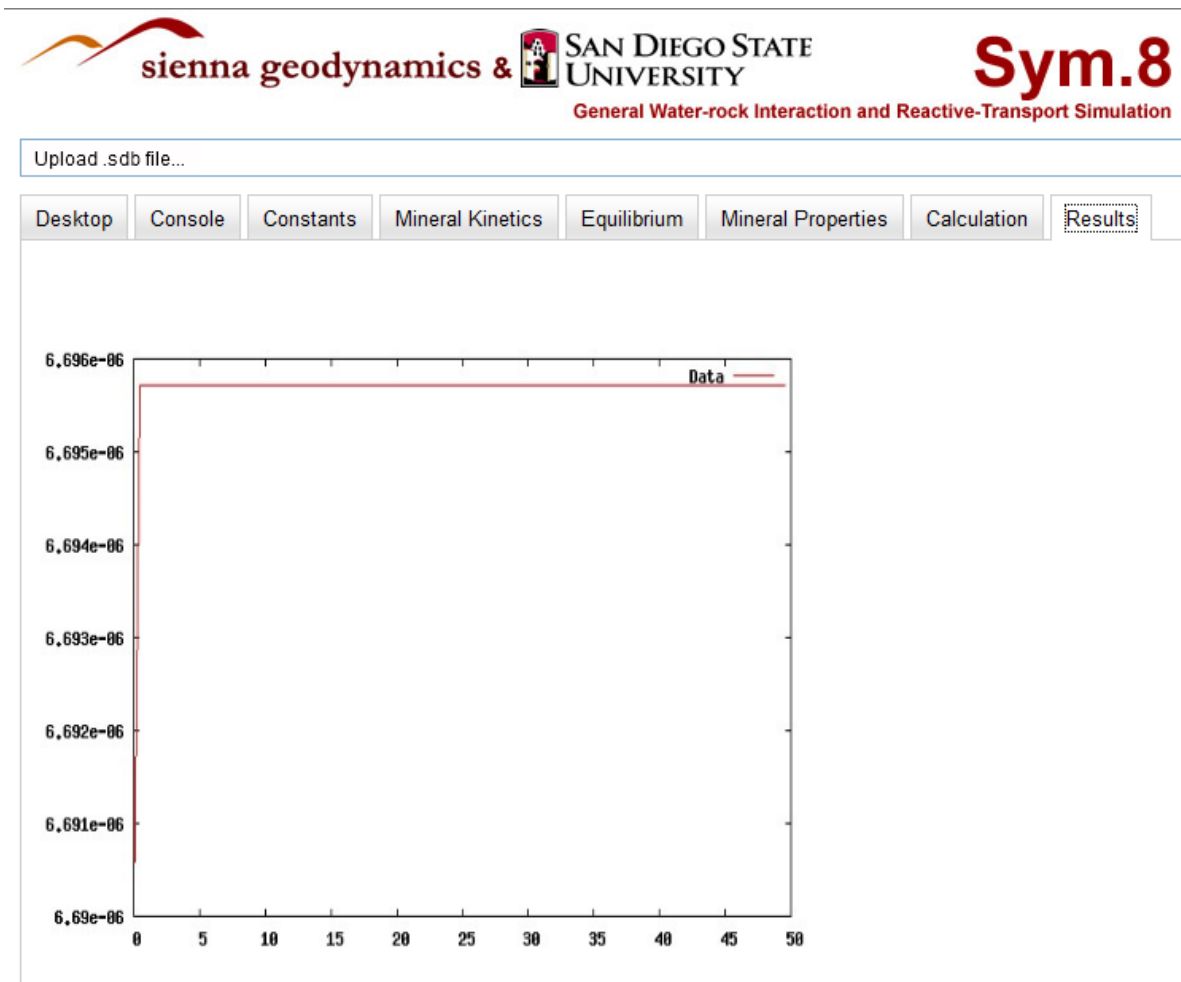
The waitFor() command makes sure that the process finishes before more JAVA code is executed. Once the process does finish (and hence there is a view1D ascii file and a jpeg image), we use a file reader to store the data for the view1D file in a string and send that string back to the web interface.

In the web interface, special javascript code is used to detect a SOAP response (when the data is sent back) and displays the jpeg image and the data as follows:

```
load: function(response) {
  var returnTag = response.getElementsByTagName("return");
  dojo.forEach(returnTag, function(oneEntry, index, array) {
    children = oneEntry.childNodes;
    returnData = '';
    for(var i=0; i<children.length; i++) {
      returnData += children[i].nodeValue;
    }
    textbox = document.getElementById("ResultData");
    textbox.innerHTML = returnData;
    var imagesrc =
      'http://co2seq.sdsu.edu/rmc_Sim8/sdb/OUTPUT/'+filename.substring(0,
        filename.length - 4)+'_'+parameter+'_'+timestep+'.jpg';
    oThis.refreshImage("MainImage",imagesrc);
  }
}
```

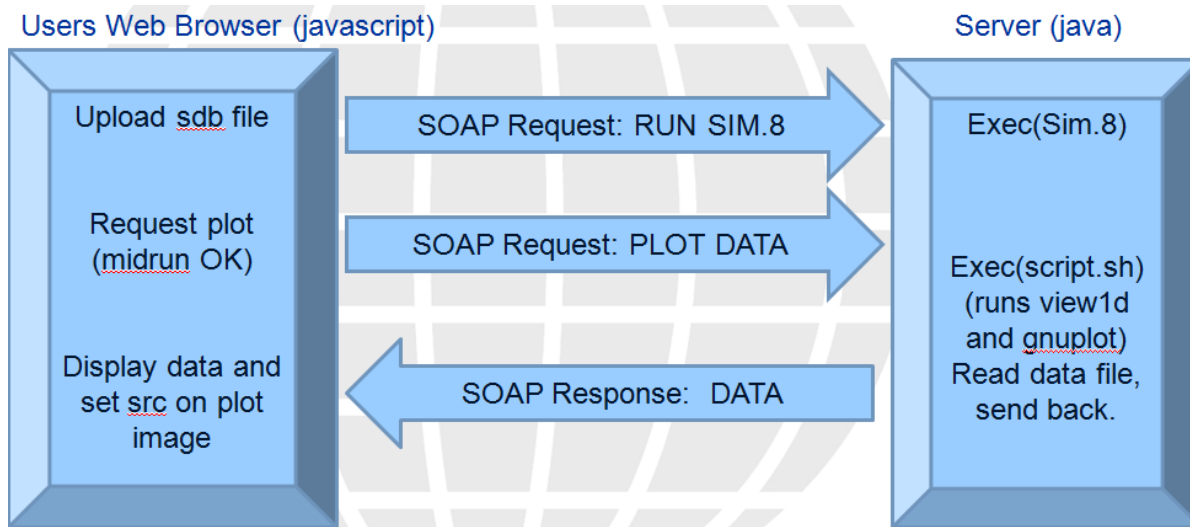
where “refreshImage” simply sets the “src” member of the “MainImage” to the new image source string.

An example of a resulting plot is given here:



The data is printed below the plot image (not seen here because one needs to scroll down to see it). It should be mentioned once again that since SOAP messages are asynchronous, one can retrieve data and plot it periodically even though the Sim.8 simulation has not completely finished.

This entire sequence of events to run Sim.8 and plot output can be depicted as:



### Conclusion and Future Work

The goal of this report was to implement data retrieval and plotting for the Sim.8 web interface under development at SDSU. This goal has been achieved in a basic way. Further work on this aspect of the web interface would include using more GNUplot commands and input to create proper titles and axis for the plots, as well as producing publishable eps or pdf plots accessible from a generated download link. Also, a lot of work needs to be done in error-checking, as well as file cleanup so that the output files and images do not clutter the server. Further enhancements could include the ability to plot multiple different plots in a single web window, refreshing as needed. Many other enhancements to the overall web interface are in the planning stages but outside the scope of this report.