Software Design within Comet: Manipulating Database Tables with JavaScript and PHP

Nathan Thomas

University of Notre Dame, Department of Computer Science and Engineering, Notre Dame, Indiana

Abstract. Ajax and Comet are a data sending library/toolkit They establish channels and allow for both the client to request data, and the server to directly send the data to another client without request. This is useful for simulation programs as some events are not directly controlled by the client. Most simulation data is saved on servers in the form of databases. There needs to be a way for the client to pull data from the server and send it over a Comet channel to the clients. Unfortunately, the standard server language, PHP, does not work very well within Comet, so the data must be pulled from the databases and manipulated into javascript where it can then be sent to the client. Keeping this process fast and efficient requires it to be processed on the server before being sent. In this paper, we will discuss the use of Comet and how it can be applied with data from a database. [9, 12]

Keywords: Comet, JavaScript, PHP, Array, Reverse Ajax

1 Introduction

Computer technology has grown by leaps and bounds over the past 50 years. Gordon Moore hypothized the growth of computing power, stating that technology would grow exponentially over the years. While Moore was not exactly correct, he was on the right track, computers did advance at near exponential speeds, with a few speed bumps along the way. Today, we have machines that can do almost anything at a speed faster than humanly possible. Yet with all this technological power at our fingertips we still have emergencies. No matter how smart our computers are, they are not capable of dealing with spur of the moment, unanticipated emergencies throughout the world. Whether it be natural like a hurricane or earthquake, or man made like an oil spill or economic collapse, nothing beats man's ability to improvise. Computers only can have so many responses and each needs to be individually programmed by an individual and, as a result, every single situation cannot be considered. [3, 13]

Yet, this does not mean that technology cannot be used in the event of a catastrophe. Emergency Operation Centers (EOCs) exist throughout the country so that emergency management officials can gather and respond quickly and effectively to national, state, or local crises. They use computers to help facilitate in the communication and organization of their response teams. Not only are these computers used for the real event, however, they are also used for training purposes as teams meet throughout the year to ensure they will be able to respond as best as possible to a real event. The problem with these excersises is that they are expensive and time-consuming, they require everyone involved to take a day off work to meet and spend it in training. [4, 5]

The virtual Emergency Operations Center (vEOC) is a web based EOC that aims to allows people to connect to the training excersises from the comfort of their offices. This enables them to continue to respond to pressing work issues, while at the same time participate in training activities. Not only does the vEOC create an easier way to train emergency managers, it also facilitates research into emergency management and critical decision making. While there exists plenty of material

analyzing emergency response after the fact, there has yet to be a way to get accurate data in large quantities highlighting the decision making process in crisis style events. The webbased vEOC will change this, as all actions made by the players can be recorded to a server and then analyzed by researchers, hopefully opening up a new level of understanding of human emergency response. This field could have the potential to drastically change response procedures and as a result create better more effective response plans. Essentially, this research area could save thousands, or potentially millions, of lives. [5, 7, 8]

The vEOC needs to do a lot of things to be successful. It needs to place a "controller" in charge of the simulation, essentially a module that mandates the order of events, and ensures all players are on the same continuous timeline. This module needs to be able to send data to all players, without the need for them to request that information. In other words, it needs to be able to issue an alert notice from the controllers module, and have the intended recipients receive that alert autonomously. Enter Comet, a form of Reverse Ajax, that allows for the pushing of data along user subscribed channels. Using Comet, all users can subscribe to a series of channels and then the controller can send alerts along those channels that the users all receive automatically. With this sending protocol the controller module is able to function as required. [6]

2 Previous Work

There exists much work elaborating on the inner workings of MySQL, and how to manipulate it within PHP and JavaScript. For example, forums like PHPFreaks have multiple topics discussing how to pull MySQL data off the database. Utilizing Comet is also a current topic with lots of articles being published as to its basic use. For example, Nilson provides readers with a basic overview of Comet and its basic implementation. There is a lack of work though as to combining the two fields of information. While its been done before, it has yet to be thoroughly explored and documented. [1, 2, 10, 11]

3.1 Design Considerations

In order to create a functional control module several problems needed to be fixed. While fixing any errors that arose in development, several considerations needed to be kept in mind:

1. Flexiblility

As the vEOC needs to be able to run on multiple browsers as well as a variety of operating systems and hardware, all modules need to be flexible so that they can be sure to run in a variety of contexts. [5]

2. Server Based Processing

Not every user of the vEOC has access to top-of-the line hardware, one had to consider that older machines lacking in processing power may be in use. These users are at a distinct disadvantage from other users, as they may experience processing delays. While the vEOC is not a competition, slow computers can create people who are lagging behind all the other users, which could cause the training exercise to lose effectiveness. Keeping as much processing on the server as possible limits the effect of slower computers on the simulation, helping to accompany all users as best a possible, as servers can generally handle much more demand. [6]

3. Synchronization

Keeping all the users on the same time frame creates a greater cohesion between players. For if some players were off by even just a few minutes, their requests could get jumbled, they could miss important events, and in general chaos could ensue. Rather than having individual clients manage the vEOC, having a server ensures that everyone is on one continuous timeline. This ensures greater inner stability, allowing for a more effective simulation. [5, 7]

3.2 Design Challenges

While using Comet it the context of vEOC development, a number of design challenges arose and had to be addressed:

-Not sending data

Initially any data that was sent by the controller module was unable to be received by any of the clients. When data was pulled from the MySQL database it was pulled into a temporary PHP table, where it then was sent out to users with Comet. This failed to work because Comet cannot handle PHP code. By switching the PHP into JavaScript form, the data proceeded to send properly to the clients.

-Jumbling of data

Once the data was received on the client side, it was processed and then displayed to the user. Unfortunately, the table values were being jumbled, and as a result were displayed out of order. The cause of this problem was that the client was looking for a specific column within the table, one that determines where and how it displays the table it just received. This was causing the client to essentially wait, accepting each piece of the table and checking it for the specific data type, where it would overwrite any data recieved prior to the specific data column, causing data loss and a frame-shift mutation of the table. This problem was fixed not by changing the client behavior, but rather addressing the form the data was being sent in. As a table, the data was being sent in array form, each piece being sent one at a time. By converting the entire array into a string, and sending the entire string at once, the client was able to break it back into array form and then look at the entire array at once. This allowed it to keep all the data stored, determine how to display it, and then have all the data on hand to display.

-Sending to Individual Users

During initial design, the controller module sent any data to all connected clients. During a simulation however, this is impractical as certain information is meant for specific clients. By utilizing the solution to the jumbling problem, there was a suitable array of data that contained all the information, including the intended recipients. By creating client specific Comet channels, the control module was able to check against the intended recipients column of the array, and then direct the data over the specified clients channel.

-Overwriting of Injects

There are a number of different simulated communication mediums that data can be received in the vEOC, for example: cellphone or radio. The intent of this is to allow multiple sources of information to be coming in at the same time without overwriting each other. However, whenever the controller module was sending data, it overwrote the data for all of the simulated mediums. By creating a sub-level of the client specific Comet channels for each of the simulated mediums, multiple

inputs of data no longer overwrote all of the simulated mediums. It limited data to only being overwritten when it was two sets of data over the same simulated medium, which was the intended design of the vEOC.

-Keeping Processing on Server

To meet the design consideration of server based processing, the vast majority of the data manipulation was done within the controller module with PHP. It was in this module that it was pulled from the database, turned into JavaScript, morphed into an array, and then converted into a string. When the clients received this data they only had to break the string back into an array for it to be used. The controller module also created the client specific channels and sub-levels. The clients however only had to make a few channel subscriptions, based on their own client tag. While one set of data being sent likely would not tax the client; however when amplified into an entire simulation with potentially hundreds of sets of data being received quickly the server could very quickly overwhelm an older client's processor.

-Synchronization

Keeping data processing on the server helps to ensure synchronization as slower clients have less opportunity to lag behind others due to the processing of data. The controller module also does as much MySQL manipulation at once as it can, by pulling the entire table into cache, rather than polling the database for each individual data set. This allows the converting and sending to happen faster, creating less downtime between data sets being sent.

-Flexibility

By folding as much of the processing into simple loops that can be quickly repeated with a multitude of different tables, using the data sets own internal parameters and controls for the loop, the controller module is very flexible. This allows the controller module to be used in a variety of settings, as some simulations may require larger databases than others. By keeping everything folded into small loops it makes the conversion to other browsers simpler, as the various browser specific tags only needed to be added a few times versus many.

During the implementation Comet into the vEOC context, these design challenges were all addressed. However, while they were being developed several bugs arose and the program had to be constantly modified and tweaked in order to get a full working product. These were minor bugs, usually involving syntax, and were not listed here as they only needed basic debugging to fix.

5 Discussion

As each piece of the Comet implementation within the vEOC came together, and each design challenge was overcome, there was knowledge that was gained. During a simulation like the vEOC, most of the information that gets sent to the clients, has already been written and stored within a MySQL database. While, as clients, they could easily send a request, and have the server send them back any information necessary. This is an example of pulling data, unfortunately it requires there to be a conscience action by the client. With Comet, that data can be pushed to the clients without them requesting it, creating a much more realistic simulation within the context of the vEOC. However, pushing MySQL through Comet is not nearly as simple as pulling the data. The MySQL database needs to be loaded into PHP where it than must be converted into JavaScript. At this point Comet can

send it, but it gets sent each cell at a time. To send larger blocks of data, it must be placed into an array, and then converted into a string. At this point it can be pushed by Comet to any number of clients, where it can than be split back into an array and used as necessary. Essentially sending MySQL databases over Comet in a few steps: MySQL into PHP table, PHP table into JavaScript table, JavaScript table into array to be sent, array into string, string sent through Comet, string then split back into array.

Implementing the Comet toolkit within the vEOC involved developing around multiple design considerations and challenges. Yet, upon completion there was a working controller module that could pull data from a specified MySQL database, and send it to individual clients, all without a single client side request. Without this module the vEOC would have been crippled as it would not have been able to autonomously send out its alerts to all of its users. Had each user needed to send a request for every new piece of information, the immersion of the simulation would have been broken. Comet allowed the module to work as it does. The vEOC is not the only area where Comet can be implemented however. Comet can be used in any situation that you need autonomous data sending to clients without them requesting it. With the steps earlier described, this data can not only come from basic HTML or JavaScript, but can also origin from a MySQL database. This opens up the possibility for Comet to be used in a much larger scale, as MySQL can handle significantly larger amounts of data than just PHP and HTML. Comet may be a new toolkit but when used properly it can open up many new possibilities in web programming.

6 Conclusion

Comet is a toolkit that enables a push data sending infastructure. It enables data to be sent from the server to the clients without the clients sending an initial request. This is usefull for real time simulation, as data can come freely without the need for repeated requests. For large simulations, the data needing to be sent is usually stored in some form of database for easy access. MySQL is a popular database, and is the one in use for the vEOC. Pushing data from MySQL through Comet is not as straight forward as one may think. It is not as simple as getting the data from the database and then sending it. It needs to be converted into a JavaScript string before it will send with Comet. Using this principle the Controller Module for the vEOC was developed, and the whole of this paper was based. More work may be required in this area to fully develop all of the potentials of MySQL working with Comet. The ideas outlined in this paper should help facilitate any further research.

7 Acknowledgments. I would like to thank Dr. Gregory Madey and Cynthia Nikolai for their mentorship in this research. I also would like to thank anyone else who helped. I also would like to thank Troy Johnson of the Miami-Dade EOC for his assistance in conducting this research.

8 References

[1] "Help w/ pulling data from 2 tables and displaying the results." *MySQL Forums*. N.p., n.d. Web. 2010. <http://forums.mysql.com/read.php?116,131529,131529 >.

[2] "Using PHP to pull data from MySQL and put into HTML - Novice." *PHPFreaks Forums*. N.p., n.d. Web. 2010. <http://www.phpfreaks.com/forums/index.php?topic=210697.0>.

[3] Ekman, Magnus, Fredrik Warg, and Jim Nilsson. "An In-Depth Look at Computer Performance Growth." (2004).

[4] "WebEOC 7.0 User Manual." Print.

[5] Becerra-Fernandez , Irma, Greg Madey, Michael Prietula , and Domingo Rodriguez . "Project ENSAYO: A Virtual Emergency Operations Center for Disaster Management Research, Training and Discovery ." Print.

[6] "vEOC Concept Map." Web. 2010.

[7] Becerra-Fernandez , Irma, Greg Madey, Michael Prietula, Domingo Rodriguez, Ricardo Valerdi, and Timothy Wright. "Design and Development of a Virtual Emergency Operations Center for Disaster Management Research, Training, and Discovery" Print. 2008.

[8] Wright, Timothy and Greg Madey. "A Prototype Virtual Emergency Operations Center using a Collaborative Virtual Environment ." (2008): Print.

[9] Nilson, Kevin. "Pushing Data to the Browser with Comet."*developer.com* 2008: n. pag. Web. <http://www.developer.com/tech/article.php/3756841/Pushing-Data-to-the-Browser-with-Comet.htm>.

[10] "comet/server-push servlet." *Caucho*. N.p., n.d. Web. 2010. <http://caucho.com/resin-3.1/doc/resin-comet.xtp>.

[11] Gravelle, Rob. "Comet Programming: Using Ajax to Simulate Server Push." *WebReference.* Web. 2010 <http://www.webreference.com/programming/javascript/rg28/>.

[12] Crane, Dave, Eric Pascarello, and Darren James. *Ajax in Action*. 2005. Print.

[13] Tuomi, Ilkka. "The Lives and the Death of Moore's Law ." (2002): Print.