Visualizing Networked Writing Activity

An Honors Thesis (HONRS 499)

by

John Holden Hill

Thesis Advisor

Paul Gestwicki

Ball State University
Muncie, Indiana

June 2011

Expected Date of Graduation

May 2011
Abstract

In conjunction with the Honors Fellow program and two faculty advisors from both the English and Computer Science departments, another student and I have written software to visualize how participants collaborate on networked writing projects. Using Google Docs as a way to allow students to instantaneously interact with a document in real-time, this software captures data from Google’s cloud service and displays it in a pair of visualizations. We used agile methods of software development to devise a way to implement their ideas in an appealing way. This document contains detailed instructions on where the latest iteration of the software can be located. It also details the process of making the system operational on a new machine, stating how the software works and where it should be placed in the file system. The document also explains how one can use the system to visualize writing collaboration. Finally, many failed iterations of the software have led to meaningful reflections on software development practices. The document serves as a technical report for the software, but also elaborates on the hardships of development, as well as provides insight on how this software may evolve toward richer experiences. Also included is an Author’s Statement which reveals many of the learning experiences that arose throughout the development of this project.
Acknowledgements

There are three very important people that I would like to thank first and foremost. First off, I would like to thank Phil Parli-Horne for being such a wonderful partner for the entire run of this project. His skills were always a well-needed foil to my own, and our combined ways of thinking allowed us to best develop this software using the limited skills in our repertoires. Next, I would like to thank Dr. Brian McNely of the English department, and one of our advisors for the Uatu project. His view of what the software needed to be was a constant guidance for us, and seeing this problem through the eyes of an English professor allowed us to better visualize how we needed to tackle our ideas. Also, the man is incredible company, and it saddens me greatly that I did not get to take his professional writing class during my stay here. Most importantly, however, I would like to thank Dr. Paul Gestwicki, my advisor for the Uatu project and its related thesis document. Dr. Gestwicki is the reason I have had such a wonderful time at Ball State, and his vast knowledge, teaching methods, and unparalleled approachability have allowed me to achieve heights I never imagined possible from a simple four-year college experience. Without a doubt, the many projects I have tackled under Dr. Gestwicki's supervision have been some of the most important steps I have taken to becoming a well-rounded software developer. Along with those related to the project, I would also like to thank my fellow Computer Science majors for the wonderful time I have had here at Ball State. It is without a doubt that the people I encountered here have reaffirmed my love of computers.
Author's Statement

The Uatu project was the most realistic job experience I feel that I had attained throughout my experience at Ball State University. It was not only a complex, multi-tiered project with a multitude of different technologies to work with, but it was also the most frustrating work I have ever had to deal with. I considered Uatu to be somewhat of my pet project throughout the year I devoted to it, mainly because it dealt with some of my favorite software architectures and development strategies. Web technologies have always been my favorite canvases to work with, and Uatu was no exception. In spite of this, even working with a very intelligent partner and an excellent mentor throughout the process, this particular project was, perhaps, the most rage-inducing, yet fulfilling experience I have yet to encounter.

Most of my work for the project can be condensed down into two separate focuses, one for each semester that I worked on the software. During the first semester, I focused primarily on extracting the user's information through the Google's implementation of a three-legged OAuth system. An OAuth system revolves around an authorization system that requires tokens to authenticate a user with a remote server. In the case of a three-legged system, this usually involves generating a dynamic request token from the software which is then used to authenticate against user information stored on the remote system. The user, the software generating the request token, and Google's authorization service represent the three separate legs of the system. This would allow users to access their document information by logging in directly to Google's authorization service to securely retrieve their document list. It required the better part of a month for me to accurately implement this system in a way that would be non-intrusive to the user experience, and the end result was workable with what the rest of the team was developing parallel to me. It was mildly disappointing, however, to see that what I had built was not what I had originally intended for the process to work through. It achieved the same basic goals, but exchanging client information with Google's servers turned out to be a multi-step process that required the user's direct authorization, even when signed into their Google accounts. This made it highly difficult to develop a streamlined log-in process, since the user would be required to agree to terms and conditions from outside the actual application itself. It also required strange ways of returning itself to the application that I never fully understood the process for, though I did eventually manage to reference the right URL into the proper callback function in the end.

The second semester was mainly devoted to tightening up the backend of the program by tinkering with the CSUatu server, an Ubuntu server housed within the Ball State network. This server houses testable versions of the Uatu software, but it can only be accessed from within the Ball State firewall. My first contribution was to establish a MySQL database on this server to manage all of the document information that the rest of the team was pulling down through Google Docs. To do this, I installed all of the relevant software on the server, as well as installed
a database management tool to allow me to visually configure all of the parts of the database, as well as delete excessive entries caused by software bugs. From here on out, I developed and debugged the transfer of document information to and from the database. Once we had established a stable system using our document revisions, I also developed the basic layouts of the client-side utilities and incorporated the document revision software into the main client pages.

Perhaps the most interesting struggle for me throughout this experience was facing legitimate failure at implementing an idea. Using the Google Docs API, at first, was a promising endeavor. It was the first time I was truly forced to sit down and study documentation as to how a service works and could be implemented. For several weeks, I sat down with my colleagues and sifted through pages and pages of online documentation of how to use the different features of the document API to securely log a user into his or her account and attempt to very cleanly tear apart each document into the pieces we needed to establish an accurate representation of the data for our visualizations.

When my work with authorization was virtually complete, I switched gears into helping the others come up with an elegant solution to retrieving the documents. We attempted to use every feasible way of attaining what we needed without having to build an entire system devoted to document retrieval, but it would not cooperate, no matter which methods or techniques we used to access the file. We attempted to track down any leads as to how these documents could be worked with, only to find a single reference through a Google search that told us everything we were attempting to do had yet to be implemented in Google's code, even in spite of it being well documented as functional in the API. Once this was determined, I felt a huge sense of disappointment with the project. We had spent the better part of our first semester chasing a dead end. All of the code I had written to authorize the users was virtually useless to the project (the OAuth knowledge is still very handy to know, on the other hand), and all of the code we had written to pull down a document would have to be entirely overhauled to adapt to a new system that we had yet to establish. This same occurrence continued to plague us through the remainder of our development whenever we thought up a brilliant idea for a software implementation, only to have it unravelled by an incomplete method in the API. This is certainly a problem that I will continue to run into as a software developer in the future, however, making this experience very worthwhile merely because it awakened me to the inconsistencies that even large development companies such as Google may have in their documentation.

In spite of these roadblocks in the development cycle, we eventually implemented a system that did what we needed it to do, and much of the frustration of these challenges is immediately dismissed upon seeing visualizations come to life the way one had originally envisioned. The act of fumbling through the early stages of this project was most certainly a painful experience, but without it, I would not have picked up on technologies such as OAuth or Hibernate, nor
would I have been able to further my database skills. It is also interesting to note that throughout each major revision of the software, we relied less and less on outside resources and began developing entire chunks of functionality by hand. The entire document revision system using MySQL and Hibernate was developed as a result of Google’s inconsistent API. Much of what we did in this project was thinking outside of the box and building from scratch. This allowed us to fine-tune exactly what we needed from the very beginning of the final implementation, and, even in spite of minor technical difficulties along the way, the final revision of the software was developed much more smoothly (from my perspective, at least) than past attempts at solving the problem. Working on a project with so many brick walls involved fosters deeper understanding of what the software really needs to do to function properly and promotes greater creative thinking to break down the wall.

As a result of all of this, Uatu was an incredible experience to have had at the collegiate level. Even though it was full of headaches through every step, the fact that we were able to step back, reevaluate, and come out with a working product has given me skills as a developer that I would have never achieved through a traditional Computer Science offering at Ball State. To me, this project was a real job experience. We set out to develop software that a pair of client partners wanted to have finished before we graduated. Throughout development, we were forced to adhere to outside restrictions, build an entire piece of software that was quickly dismantled and rebuilt, all the while using industry-endorsed practices that led to a functional program that was used within a group of peers on campus to learn about collaborative writing. Because of this, headaches and all, I am truly proud of what me and Mr. Parli-Horne have developed under the supervision of Dr. Gestwicki and Dr. McNely.
Visualizing Networked Writing Activity

J. Holden Hill, Phillip Parli-Horne, Paul Gestwicki
Computer Science Department
jhhill@bsu.edu, pparlihorne@bsu.edu, pvgestwicki@bsu.edu

Brian McNely
English Department
bjmcnely@bsu.edu

June 3, 2011
Ball State University
Computer Science Department
Technical Report 2011-01

Abstract

Over the course of an academic year, two students and a pair of faculty advisors have written software to visualize how participants collaborate on networked writing projects. Using Google Docs as a way to allow students to instantaneously interact with a document in real-time, this software captures data from Google’s cloud service and displays it in a pair of visualizations. We used agile methods of software development to devise a way to implement their ideas in an appealing way. This document contains detailed instructions on where the latest iteration of the software can be located. It also details the process of making the system operational on a new machine, stating how the software works and where it should be placed in the file system. The document also explains how one can use the system to visualize writing collaboration. Finally, many failed iterations of the software have led to meaningful reflections on software development practices. The document elaborates on the hardships of development, as well as provides insight on how this software may evolve toward richer experiences.

1Primary contact
1. Introduction

The Vatu project is designed to visualize collaborations in a digital workspace. The name comes from the Marvel Comics character, Uatu the Watcher, who is tasked with watching over the universe; in the same way, this software watches over revisions of Google Documents and stores their information for later access. This project was designed to create a visual representation of how participants work together; the system was tested with Ball State University Computer Science undergraduate students during the spring semester of 2011, exploring their writing about various topics in advanced programming. For this particular example, Computer Science students have been asked to work with Google Docs to allow them to collect data for the project.

The current implementation of this project evolved from Google’s cancellation of Google Wave, requiring the team to begin work on new software using Google’s Document application instead. The most recent version of the software has been in development since August of 2010, concluding at the end of the spring semester of 2011. This software has been tested with the aforementioned group of Computer Science students who have been using Google Docs to collaboratively edit documents directly relating to their experiences in CS222. Over the course of the spring semester, many interviews with both those directly involved with development and those students taking part in the study have been conducted to complement the findings of this study.

Accessing the project

To obtain the code for the Uatu project, install a Mercurial plugin for Eclipse. Installation of this plugin is quite simple. Go to http://javaforge.com/project/HGE, scroll down to the “How Can I Download the Plugin?” section and follow the instructions. A comprehensive resource with instructions for using Mercurial may be found at http://hginit.com/.

Uatu’s file structure is broken down into several folders based on the type of resources being used by the program; for the most part, the project can be separated into three distinct sections based on the functionality of the program. One section is devoted to the Watcher script classes, another focuses on storing and retrieving document revisions, and the last deals with the JSP interface that shows the visualizations.

2. Project Architecture

The Uatu visualization system is divided into three modules, which are defined in the subsequent sections. A rough outline of the overall process can be seen in the component diagram shown in Figure 1. There are five major components that drive Uatu. The first is the Google Docs service, which is where all of the collaboration takes place. Google provides
many ways of accessing document information through the extensive GData API, which allows the Uatu system to capture all of the relevant data for the visualizations.\textsuperscript{2} The next major component is the Watcher program, which runs on a Linux server and continuously talks to Google's servers to retrieve document revisions through the GData API. The user sees the Vis component of the software, which contains the web front-end and all of the visualizations for each document. This requires the final component, the Data Table Servlet, to turn document data into the annotated timeline and bar chart displayed on vis.jsp. This set of modules works in conjunction to power Uatu's software.

**Watcher**

The first segment of the project deals with the script that runs on the server and collects information about documents as they are revised. This is called the *Watcher* service, and sits within the Java Resources directory under `src/watcherdaemon`. When running, Watcher does a number of things: it is designed to run at all times and will start automatically upon boot, and it runs as a single process within the server, pinging Google's document server every minute to compare and identify documents associated with a single user. This user is defined in the UatuDocsServiceFactory class's static initializer. After it has made a connection, the Watcher class will call the DocumentGrabber module, which is described in greater detail in its corresponding section. This generates the entire backend for the site.

This can be started up in one of two ways, depending on how the software is to be used. The first is to simply run it through Eclipse by running Watcher as a dedicated Java application and the Uatu frontend as a web application. This would normally require that Apache Tomcat 6.0\textsuperscript{3} be installed on that particular computer; however, the J2EE version of Eclipse will allow one to establish a virtual Tomcat server directly through the IDE itself. However one chooses

---

\textsuperscript{2}http://code.google.com/apis/documents/
\textsuperscript{3}http://tomcat.apache.org/
to configure their server, it is important that version 6 of Tomcat be used. Tomcat is currently on version 7, but project facet settings, a type of configuration setup Eclipse uses for J2EE projects, causes conflicts when attempting to run it on the newer version. As a result, this also means that version 6 is running on the CSUatu server. Running it this way causes the system to detect that the application is not running on its release machine, and defaults to creating a temporary database on the host machine via HSQLDb.4

In order to start this system on its release platform, one must log into the server where Uatu is hosted through the use of a secure shell client. The software suite is running atop an Apache Tomcat web service that has been installed on a physical machine running an Ubuntu distribution of Linux. Once logged into the server, in order to begin running the Watcher service, one must run the `watcher_start` script. This script can be run anywhere within the server.

```bash
#!/bin/sh
#
# Start Uatu the Watcher.
#
# The Watcher will periodically check with google docs to pull down
# document changes and store them in mysql on this machine.
#
# The Java class that is The Watcher.
WATCHER=edu.bsu.uatu.watcherdaemon.Watcher

# The ant script that runs watcher
WATCHER_ANT_SCRIPT=/var/lib/tomcat6/webapps/UatuWEB-INF/watcher.xml

# Let's see if watcher is already running.
WATCHER_PROCESS=`ps -ef | grep $WATCHER | grep -v grep`
if [ -n "$WATCHER_PROCESS" ];
then
  echo "It seems watcher is already running. I am giving up."
  exit
fi

# Otherwise, let's start 'er up.
  echo "Starting the Watcher."
  ant -f $WATCHER_ANT_SCRIPT
```

After running this script, Watcher begins its run every minute. This wait period may be edited by manipulating the constants in the Watcher class. Using this frequency, it checks the Google Docs server for updates to any documents associated with the proper user id (in this case, bsu.test.research) and stores any new revisions within a MySQL5 database located on the CSUatu server. This database must be created prior to running the application’s first run. The most simple way of doing this is to log into phpmyadmin on the server (http://

4http://hsqldb.org
5http://www.mysql.com/
csuatu.dhcp.bsu.edu:8080/phpmyadmin/) and manually create a new Uatu database with all of
the necessary fields required for an InMemoryDocRevision. Once Watcher is properly running,
any update to a related document that changes its content size will add a new entry to the
database with all relevant information for the web front-end.

Perhaps the most useful part of Watcher’s design is the generation of logs. These log files store
every step of the process for simple perusal of events should an exception arise that causes a
service interruption. The logs are started automatically through the Watcher’s start script, and
are created to be easily accessible through the use of a configuration file located with the script.

Pulling document revisions from Google’s servers required a new system to be built in order to
capture all of the data needed to generate accurate visualizations. Using Hibernate as a tool
to map the proper data into a MySQL document, the Uatu system maintains a local database
of all document revisions of any Google Doc shared with the proper Google account. Figure
2 (above) shows how the different pieces of the Watcher program work together to generate
proper revisions in the database.

This whole process begins every time Watcher completes a cycle on the server. This triggers
DocumentGrabberTask—located in the base source directory of the system—to run, which in
turn triggers Uatu to generate a client containing Google account information for the shared
account, using that to take a snapshot of each of this account’s documents on Google’s server at that particular minute by turning the document into an InMemoryDocRevision object. This class proceeds to break down the document into the different resources used to compare entries. The system then compares the timestamp of Google’s last revision and the content size of the document against the last revision in the system, should one exist. If one does not exist or the revisions are not identical, Hibernate then takes all of the data required for the visualizations and maps it as a new entry in the local Uatu database. This information will then be used as the next reference to compare against during Watcher’s next cycle.

While the vast majority of the software’s code lies in its Java resources in the first two segments, any user of the software will only see the segment of the code dedicated to its user interface located in the WebContent folder in the project. Within this folder are two directories, one containing a dummy manifest file, and the other containing a copy of all of the external libraries referenced in the code. The crucial components of this directory, however, are all of the HTML, CSS, and JSP6 files, as detailed in Figure 3, that work in unison to allow the client’s browser to communicate with the server.

http://java.sun.com/products/jsp/docs.html
Welcome to Uatu!

Please enter your username below:

username

For additional help please click help.

Disclaimer: We promise we are not collecting any information about you. Your login is only used to identify the documents you have shared with your system.

Figure 4

When a user logs in to the Uatu website (see Figure 4), he or she is presented with index.jsp, which serves as the login page for the application. From here one can access either help.html, which will explain how to use Uatu, or enter the Gmail account that is linked with his or her collaborative documents into the form to log in to the system. After logging in, the user is directed to vis.jsp, where he or she will be able to select from a drop down list any document he or she would like to have visualized. Should an invalid email or one with no associated documents be entered, the program will instead redirect to loginfail.html, which will give the user tips on how to properly log in to the system and direct him or her back to the index file.

vis.jsp is the heart of the visualization front-end for the project. Once it is opened, it checks to see if a document id was passed through post as the 'uid' parameter. If so, the JSP begins pulling the data for that particular document from the database. Otherwise, vis.jsp skips the visualization process and creates a dummy page with only the available listing of documents to choose from. This listing is generated by comparing the email of the user against the Access Control List of every document that is shared with bsu.test.research. If the user's email matches, the document is added to a list that populates a drop-down menu the user can select from. Once a document is selected, the program will generate a query for the Google visualization API. By interfacing with the visualization API, Uatu generates a pair of visualizations that are situated within a predetermined position on a basic JSP page.
The top visualization as shown in Figure 5 includes document revisions as they occurred over time. They denote who made the revision, when the revision was saved (notated by the horizontal axis of the graph) and the size of the visualization (notated by the vertical axis of the graph). Users can adjust the list by the side of the graph to highlight specific revisions. Below the graph is a horizontal bar chart displaying each user who has contributed a revision and the number of revisions they have made. These statistics are not pulled from Google’s servers; rather, they are generated based on the entries in the Watcher database. Therefore, these values are not truly representative of the entire document, but rather a very close approximation based on the frequent polling of the real document.
These visualizations are generated dynamically by populating Google's chart API methods using JavaScript calls to a pair of classes located in the 'dt' folder of the source directory. These classes—AnnotatedTimelineDataTableFactory for the annotated timeline and UserRevisionBarChartDataTableFactory for the bar chart—are given the document's id as a parameter via the DataTableType class. This class is designed to direct revision information to the right table factory by passing the document id and table type as parameters in a URL. This eventually allows the DataTableServlet class to accurately generate tables by pulling all of the relevant data for that particular document into new rows for each data table. This whole process is illustrated in Figure 6.

Tests and Resources

Aside from these three major segments of the program, the project also contains several additional components that are required to properly run and test the project. This project uses JUnit to actively test any components of the code that are written in Java. These are broken into two separate folders, one for the basic tests for displaying document information, and another
for testing Watcher's functionality. Additionally, the WebContent folder contains a subdirectory titled lib where all of the libraries required for Uatu's server-side portion to properly do its job. In particular, Uatu uses the Apache Tomcat libraries, connectors for MySQL and HSQLDB, and JUnit. This folder also contains the jars for each of the libraries used by the web application itself, including Ant, Apache Commons Lang and Logging, Google's GData API, Hibernate, Joda Time, SLF4J, and Google's Visualization API.

3. Development

Uatu was originally conceived as a Google Wave extension. Wave's system, which consisted of a series of threaded 'blips,' was highly conducive to collaborative thinking. Because of these affordances this development began on a program during the summer of 2010 that would skim through wave blips and log a multitude of data about each separate entry. In the wake of Google's cancellation of the Wave project, however, the researchers' focus had to be shifted toward Google Docs, which offered the most likely alternative for pursuing this line of inquiry and development. This move necessitated an entire reworking of the existing software in order to take advantage of the new API requirements. This constituted the work completed during the course of the 2010-2011 academic year.

This software was developed using a form of spiral development model, which was created by Barry Boehm in an article titled "A spiral model of development and enhancement" in 1986. This iterative model promotes the use of four distinct phases of development, as seen in Figure 7. It begins with a design phase where the key feature set for that particular iteration are planned out. From there, the spiral continues into a risk-management phase, where the team takes inventory of possible risks and other issues that could possibly arise from the design decisions generated in the first phase. This allows the team to put together mitigation strategies and contingency plans for when development halts due to any number of possible issues.

http://ant.apache.org/
http://commons.apache.org
http://www.hibernate.org/
http://joda-time.sourceforge.net
http://www.slf4j.com
http://code.google.com/apis/chart/
http://portal.acm.org/citation.cfm?doid=12944.12948
Once risks are ironed out, coding begins. For this process, the team established a set of prioritized tasks to be completed. Members of the team would then pick out which tasks they felt comfortable developing. Depending on the task, the team would sometimes assign multiple people to particular feature. Throughout development, tests would also be written through JUnit to capture various inputs and conditions of the software that could be run without being forced to run the actual software. This allowed the team to quickly and efficiently track down bugs that could have otherwise been difficult to isolate. Once all of the features of that particular iteration are completed, the team moves on to an evaluation phase where the last iteration is dissected and reflected upon. This allows the team to move seamlessly into a new development cycle. The current implementation of the Uatu software is the second iteration using this cycle. The first revolved around a direct use of Google’s API for most of the features, but inconsistent documentation forced the team to move into its contingency plan and begin a new software iteration.

4. Evaluation

Preliminary research with student participants during the spring of 2011 indicates that participants found the visualization of their revisions generally unhelpful. There are many factors that could have lead to this evaluation that do not necessarily reflect upon the software itself, most notably that the group initially testing the software ended up performing most of their work in the same place (rather than as distributed, real-time participants), thereby circumventing
one of the primary purposes of the software. Because of these user localization practices, many participants never even logged into the system at all, since in many scenarios one team member managed the writing work of the entire group. This leads the researchers to believe that more testing needs to be conducted using a more finely assembled and directed test group.

There are some obvious shortcomings to the current implementation of the system, as well. The most obvious issue is that the server only looks for updates every minute, and discards any incremental revision that occurs between cycles. While this may not have a major impact on the document's statistics, it can create faulty values if the document is being constantly updated by two different people at the exact same time.

Many of these inconsistencies could be mitigated by using the proposed next version of the software. This would pull in information based on Google's independent HTML files for each document revision. Once these are pulled down, the system would use a clever tool to scrape information directly out of this HTML file and populate the database much more frequently with much finer detail than the current implementation. Should Google ever implement its system to pull down document revisions, the need to use a database will disappear. This lightens the system and will allow future development to move more freely and to focus on more creative ways to visualize and engage participants. The system also requires additional methods to display contributions. Using content size changes is largely meaningless and cannot properly represent the quality of a contribution.

5. Conclusions

Through the Uatu software, professors and students alike are able to visualize how technical writing can be accomplished when distance may separate its contributors. The software, which is comprised of the Watcher script, the DocumentGrabber, and the Web interface, provides viewers with two visualizations to track how thoughts and ideas are conveyed without direct, person-to-person contact by working with the Google Docs service and establishing a database of objects holding document revision data. A full academic year of development has led to many different attempts to accomplish this goal, but through iterative practices, the team has produced software capable of not only delivering the product originally conceived, but software that can be easily expanded through continued development. The Uatu software is a first step into analyzing how people collaboratively edit written documents on the Web, but there are many additional ways of viewing this data that have yet to be implemented. It will be interesting to see how these visualizations continue to evolve as collaborative work environments become more and more common, especially in the field of education.

Acknowledgements

This work was supported by the Indiana Space Grant Consortium, a Ball State University Emerging Media Initiative Innovation Grant, and the Ball State University Honors College.