# Employee Time Scheduling

**A Senior Project**
**presented to**
**the Faculty of the Computer Science Department**
**California Polytechnic State University, San Luis Obispo**

**In Partial Fulfillment**
**of the Requirements for the Degree**
**Bachelor of Science**

**by**

**Mark Peter Smith**
**June, 2010**

## ABSTRACT

Small business managers face the common problem of employee time scheduling. There is a solution to this problem in the form of an application called Lemming Scheduler. Lemming Scheduler is a Java based employee time scheduling program. Its features include a desktop based application that stores employee and business information as well as a web interface for employees to view schedules and update availability. The desktop application uses employee and shift information to automatically generate schedules. The generated schedules are viewable by employees outside of work by way of the web interface. Lemming Scheduler provides a light weight interface for small business owners to manage scheduling problems without the need for a great deal of technical knowledge.

## TABLE OF CONTENTS

## PROBLEM DESCRIPTION / MOTIVATION

The purpose of this project was to upgrade an existing student developed project called Lemming Scheduler. The original Lemming Scheduler set out to create an employee scheduling application for a small business. Created during CSC 308/309, Lemming Scheduler represented the work of Jeff Holden, Erik Carpenter, Ramya Jagarlamundi, Carl Taylor, Abhi Vaishnav, and Mark Smith.

Jeff Holden and I decided to enhance Lemming Scheduler with a new data storage model and a web client for employees to interface with scheduling data. Jeff worked on the project through one quarter of independent study and I worked on the project through both quarters of senior project.

Jeff and I were motivated to do this because we enjoyed working on the original program and felt that we could make positive changes. We also felt that by upgrading Lemming Scheduler we could create a project with real life business potential.

Upgrading Lemming Scheduler offered a way to work on a number of key skills for any computer scientist. Firstly, Jeff and I were offered the opportunity to work on an existing project with a large code base written by in large part by other people. We were faced with the challenge of reading and understand other peoples code as well as dealing with adding upgrades and handling general maintenance. Secondly, this project allowed us to work with a large number of different technologies. During 2 quarters of senior project, Jeff and I used the Google web tool kit, the Google Spreadsheet API for Java, a MYSQL database, and Java Applets.

Lastly, Jeff and I faced managing deciding design trade-offs. We balanced multiple factors including quality, development time, and technology choices.

# PREVIOUS WORK

The following section gives a brief description of the original Lemming Scheduler and investigates similar scheduling products on the market. The similar products were chosen on the basis of comparable features to those already offered by Lemming Scheduler as well the features Jeff and I planned on implementing.

## LEMMING SCHEDULER

The original Lemming Scheduler was completed by a team of students during the winter quarter of 2008 in CSC 309.

### LEMMINGS SCHEDULERS FEATURE SET:

- The ability to manage employees, shifts, and positions
- Generate schedules based on available employee, shift, and position data
- Format schedules into different reports including report by staff and report by individual
- Store all aforementioned scheduling data through serialization

The original application was lacking in the area of data management. Lemming Scheduler used serialization for data storage. Jeff and I proposed upgrading the data model to use Google Docs spreadsheets and then later a database. Serialization, while easy to implement, lacks the advantages typically found in a database, namely ACID: atomicity, consistency, isolation, and

durability (1). Additionally serialization suffers from performance issues in relation to memory management (2). Serialization works on small scale, while professional scale scheduling programs utilize a database as the Similar Work section will show.

Another area found lacking in the original Lemming Scheduler involved employee schedule viewing. The only means of viewing a schedule available to an employee involved receiving a hard copy from an employer. A problem arises in which a new week's schedule is generated and an employee is unavailable to receive a hard copy. Jeff and I attempted to rectify by making schedules available online.

## SIMILAR WORK

This section details the features of three employee time scheduling products on the market

I focused on scheduling applications that were either open source or offered free versions. I studied the following products: DRoster by Kappix, Schedulewall by Arb Design, and TimeCurve Scheduler by TimeCurve Software Inc.

DRoster and TimeCurve offered a myriad of services and were by the far the most robust of the three programs. They also required a great deal of overhead as far as database management. Schedulewall is a light weight alternative that is purely web based. Schedulewall takes data management out of the hands of the employer.

I think Lemming Scheduler compares favorably to these three scheduling applications. Lemming Scheduler offers automatic schedule generation, something missing from all three. I feel that

Lemming Scheduler stands as a middle ground between Schedulewall's purely web based nature and DRoster and TimeCurve's robust feature sets.

## DROSTER

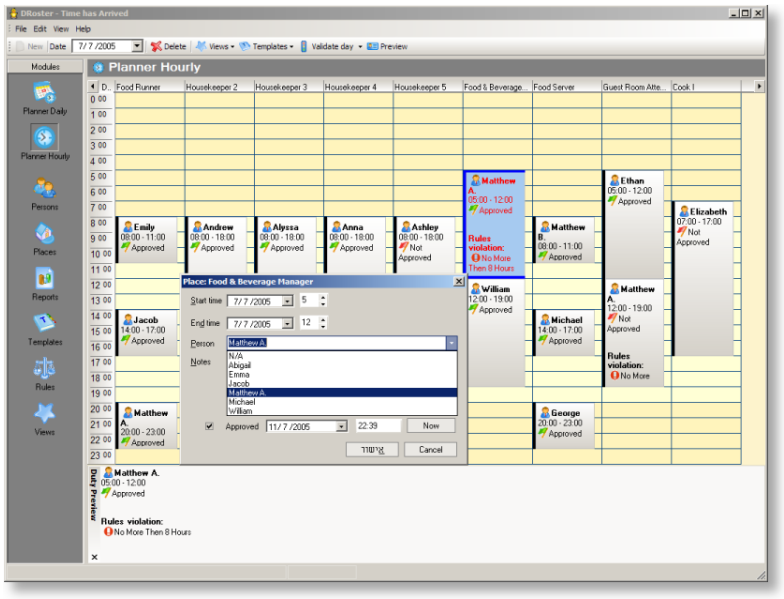


Figure 1 - View of DRosters hourly schedule view page

DRoster is a full featured desktop scheduling product. It offers a suite of tools including: daily and hourly planners, schedules based on location, templates, and multiple report options. The templates model the auto schedule generating algorithm used in lemming schedule. DRoster handles data storage using the open source Firebird database. Much like Lemming Scheduler's earlier Google Docs support, DRoster can export schedule information into a number of different forms: txt, pdf, htm, rtf, xls, tiff, gif, jpeg, and bmp.

DRoster seems like a more advanced version of Lemming Scheduler. It offers all of the features of Lemming Scheduler and then some. On the downside, it lacks a schedule generating

algorithm and requires a considerable amount overhead. The overhead results from an employer having to administer his or her own database. Lemming Scheduler eliminates this overhead by handling database management on our own server thereby eliminating the need for a user to install a local database.

## SCHEDULEWALL


**Figure 2 - Schedulewall schedule view**

Schedulewall is a simple scheduling tool focused on serving individuals and small businesses. It is purely web based; Arb Design hosts the product on its own web server. Beyond scheduling options, Schedulewall offers a personal day planner. Schedulewall lacks a scheduling algorithm; schedule generation is handled through a drag and drop interface. Additionally, Schedulewall uses shift templates to handle commonly occurring schedule configurations.

As far as employee interaction is concerned, Schedulewall uses a commenting system. A Manager can post on a message board and employees can comment on the resulting post. The product also offers a chat service, allowing employees to hold basic meetings online.

When compared to Lemming Scheduler, this product seems more like a suite of business tools. Schedulewall offers more in regards to inter business communication. Much like DRoster, Schedulewall lacks the ability to auto-generate schedules using a scheduling algorithm. Lemming Scheduler is purely dedicated to scheduling, while Schedulewall handles a range of operations with less focus on a single aspect. Lemming Scheduler and Schedulewall share a similarity in terms of the light weight nature of their design. Neither requires a client to install a local database

## TIMECURVE

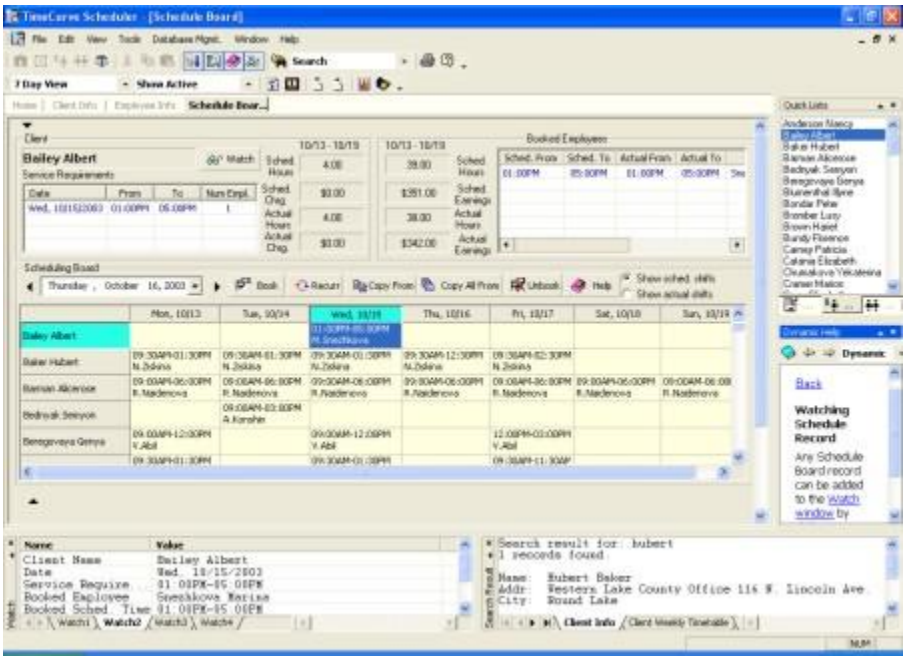

Figure 3 - Overall view of TimeCurve software

TimeCurve is probably the most robust of the three options. Unlike the other two products, TimeCurve features payroll support. Schedules can be integrated with QuickBooks to handle financial information. TimeCurve interfaces with a telephone service to handle punch clock operations; an employee dials a toll free number to punch in and out. Again, unlike Lemming Scheduler TimeCurve does not use a scheduling algorithm. Instead, TimeCurve uses a scheduling engine that notifies a manager of scheduling mistakes such as uncovered shifts, under/over booking, and overlapping schedules. The engine merely verifies an employer's scheduling choices, rather than handling actual schedule generation.

TimeCurve uses a database to manage employee data. It does not feature a web component although it does have email support. TimeCurve does feature an integrated web browser to use within the desktop application, but its focus is to provide helpful resources for managers.

Like DRoster, TimeCurve is more robust than Lemming Scheduler. TimeCurve requires a lot of overhead to manage and is more suitable for large businesses. Lemming Scheduler has the edge in terms of being lightweight and easy to install.

## REQUIREMENTS

Updates on the Lemming Scheduler necessitated the creation of a new set of requirements to cover additional features. Jeff and I followed the steps taken by PolyCron during 308/309. We updated the existed SRS to include the following new features.

- A web client for employee availability editing and schedule displaying

    - Allows Employees to view a week's schedule at any time, requiring only an internet connection
    - Employers obtain another vehicle for schedule delivery, limiting miscommunications
- Incorporate a database into the desktop applications data model
    - Remove parts of the serialization model to prepare Lemming Scheduler for remote data management
- Integrate with Google Calendars (not completed)
    - Employees could view schedules in Google Calendar form, allowing them to integrate work schedules with general day schedules
    - Extends access to mobile devices

We devised our feature set over a series of meetings. We developed a series of use cases that seemed applicable to our project, dividing them into two sections, one for employees and one for managers. (use cases and meeting minutes available in appendix)

We also constructed a number of prototypes to test the various technologies used on the project. Prototyping allowed us to gauge the viability of the various features. It also helped in development scheduling, as Jeff and I bettered our abilities to estimate the amount of time certain features would require.

I personally wrote a prototype using the Google web toolkit to test availability entry on the web client. I also developed a prototype to test the Google spreadsheet API's ability to connect to Google's servers and read information from a spreadsheet file. Similarly, Jeff created a

prototype to test document creation in Google Docs as well as a testing how to represent schedules in a spreadsheet.

## DESIGN

The design section looks at our ultimate design goal with this project and the process we used to achieve that goal.

### GOAL

The goal of this project is to end up with two components: a desktop application for use by a manager at a small business and web client for use by an employee at the same business.

The desktop application handles all of the schedule data management which includes: employee, shift, skill, and availability data. The desktop application uses said data to generate schedules by matching shifts with employee availabilities and skills. A user downloads the desktop application to manage employee scheduling for a small business. Managing data was originally to be done through Google Docs spreadsheets, but ultimately migrated to a database. Desktop applications distributed to multiple businesses all interface with the same database operated by Jeff and I. Each business receives a unique id number to insure security between businesses all storing information in the database. Minimal data storage was still handled locally, but all data involving positions, employees, and schedules would reside in the database.

The web client allows an employee limited access to data used in schedule generation. Specifically an employee can change their availability over a week and then save those changes to the database. Employees can also view any schedules that have been generated for the

current week. That being said, the desktop application is more than capable of handling employee availabilities and can stand alone from the web client.

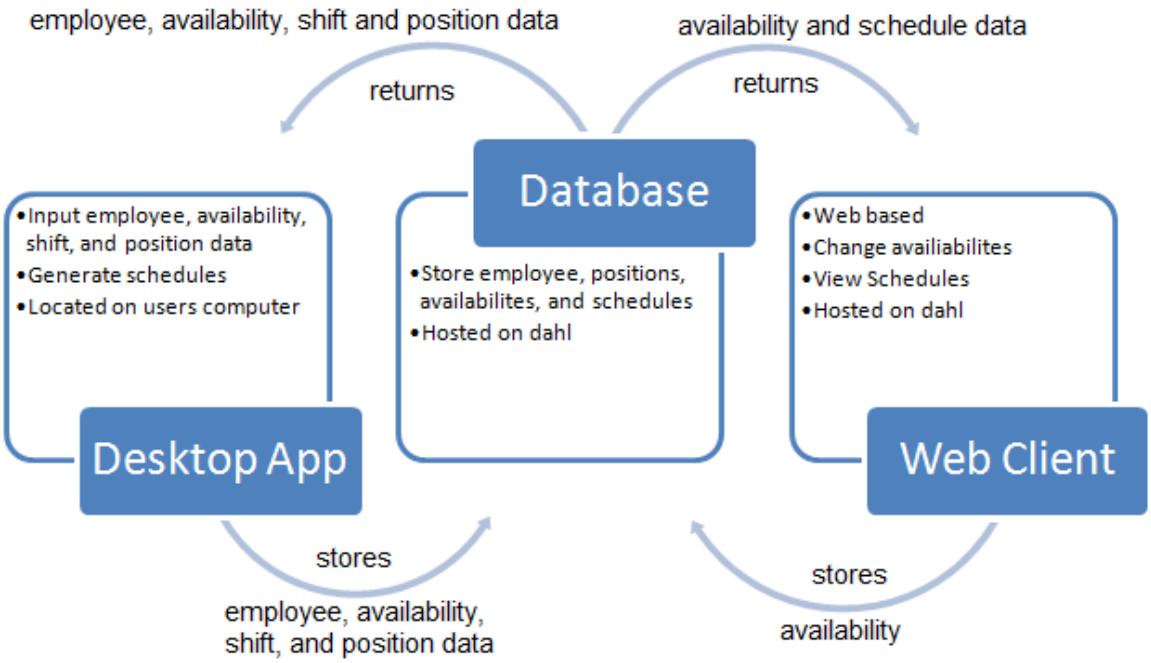


Figure 4 - Lemming Scheduler data flow diagram

The above diagram illustrates how our web client and desktop application communicate with one another. The database acts as an intermediary between the web client and desktop application by storing availability, employee, and schedule data. Employees use the web client to view schedules and alter availabilities. Managers use the desktop app to input employee information, shifts, and positions. The manager then generates schedules which are in turn stored in the database.

## PROCESS

Design decisions were made during meetings. Jeff and I divided the design into two parts: changes needed in the desktop app and the web client. The desktop changes involved fixing bugs left over from 308/309 as well as adding database/Google doc support. Design on the web client focused on user interface development.

## MY APPROACH

I approached the design phase through reading online documentation and taking notes in my developer journal. Technology selection necessitated reading available online documentation. Google provides a wealth of material in regards to its Data API and Web Toolkit, and by reading this documentation I grasped the general limitations of the individual technologies. Online research served an additional purpose of providing Jeff and I with the technical resources needed during implementation. I used my developer journal as a brainstorming mechanism. The journal proved especially helpful during meetings in which Jeff and I would suggest a wide range of design solutions, only to narrow them down as the meeting proceeded. I would document the various design solutions in order to keep my thinking clear on a particular issue.

## TWO COMPONENTS OF DESIGN: DESKTOP CHANGES AND WEB CLIENT

The desktop changes needed on the project dealt with bug fixes and Google Docs/database support. There were a couple of minor bugs left over from last quarter mainly dealing with null pointer exceptions. We looked through previous documentation from 308/309 as well as carrying out some basic testing on the original desktop app to discover any lurking bugs. Designing the new data management schemes proved a bit more difficult. We first laid out

what needed to be updated to account for the new data models. Employee information entry and schedule generation were the two major components that needed updating. After deciding what was going to change, we decided what new work was going to be needed to interact with Google Docs. We would need two classes: a class to handle schedule importing and exporting and a class to handle availability importing and exporting. With all those ideas in mind we set about creating class skeletons for the new work.

On the web side of the project we undertook similar steps. First we decided on a general look and feel for the web client. I documented some of our initial UI sketches in my journal. From there we went about deciding what classes we would need to handle the page layout and Google Doc communication. We used one central class to handle the page's layout, formatting items like drop lists and tables. Another class was designed to facilitate Google Doc communication. We then populated those classes with method skeletons and wrote some psuedocode.

## TIME LINE

The majority of design decisions were made during the first quarter of senior project. We spent the first four weeks designing the upgrades to Lemming Scheduler. Choices made on both the desktop and web sides occurred at roughly the same time. There was no design order, so as designed progressed on the desktop application the web client received an equal amount of progression. Design extended into the second quarter of senior project to account for problems during implementation

## SOLVING DESIGN PROBLEMS

When implementation issues forced Jeff and I to reselect technologies, we would undertake impromptu miniature design phases to rework Lemming Scheduler. These design phases lacked a general process and proceeded in rapid fashion to meet schedule deadlines. Generally, Jeff and I would each undertake a rapid research session to make a technology related decision. With the new technology in mind, we would then distribute the coding work to be done during implementation. Once the work was assigned, coding would begin and any design decisions would be noted in our journals. For the most part the design decisions were small enough to be handled individually. If a large decision choice was needed, we communicated through email or met in one of the labs on campus.

## TRADEOFFS

The design process raised a number of tradeoffs which I documented in my journal and are featured in some of the meeting minutes. Tradeoffs fall under three different categories: technology, features, and design.

## TECHNOLOGY

Technology tradeoffs involve the decision making process regarding which languages, API's, and data management paradigms were selected for Lemming Scheduler.

### WEB TECHNOLOGY SELECTION – JAVASCRIPT AND AJAX USING GWT VS JAVA APPLET

What technology should be used for the web client?

- Applets have long load times compared to dynamic html generation in JavaScript
- Applets allow us to reuse code
- We have limited JavaScript experience, compared to a wealth of experience with Java
- GWT lessens the learning curve involved with JavaScript and Ajax
- Ultimately selected GWT on the basis of speed

### GOOGLE WEB FORM INSTEAD OF WEB CLIENT

Would using a Google Web Form offer a better way for employees to update availability?

- Google Web Form would be easier
- Web client offers more control
- Technical Limitations in Google Web Form, namely selecting multiple cells in a table
- Decided on a web client because of the additional controls and extendibility

### PHP VS SERVLETS VS APPLET

How should the web client be handled based on design problems?

- I'm relatively familiar with PHP and database interaction is a breeze
- Servlets are a bit more complicated, but still offer easy database interaction
- Applets are easy, but major security risks are involved with database access through and applet
- Servlets are the best choice in the case, because I'm more familiar with Java than PHP and applets are a security risk.

## FEATURES

Feature tradeoffs focused on the selection process used when deciding new features to add to Lemming Scheduler.

### EMAIL SCHEDULES VS GOOGLE DOCS

Should schedules just be emailed or stored in Google Docs?

- Emailing schedules pretty much removes the need for a web client
- Google Docs are versatile, can be shared through email
- Decided on using Google Docs because of the added versatility

### AUTO UPDATE VS ON DEMAND UPDATES FOR AVAILABILITY

When an employer updates employee information, is the Google Doc updated automatically or does the employer have to click a button?

- Auto update could update when a manager actually doesn't want to
- On demand requires an extra step from a manager
- Manager could forget to update
- Decided on auto update for ease of use on the part of the manager

### WILL AVAILABILITY CHANGES BE REFLECTED IMMEDIATELY IN SCHEDULES

When an employee changes his availability does the current schedule change?

- Employees get instant feedback

- Schedules would not be concrete, meaning Employers could have a week's schedule change at the last minute
- Decided that they shouldn't because it could lead to major scheduling issues for employer

## DESIGN

Design tradeoffs centered on the deciding the best methods to use when implementing changes to Lemming Scheduler

### SHOULD AVAILABILITY APPLY ON A WEEK TO WEEK BASIS OR BE CONSTANT THROUGH TIME

Does and employee need to update his availability on a week to week basis?

- Week to week means additional work for the employees as they would need to update their availability online before every schedule generation
- Availability settings applying to every week can lead to inflexibility and requires a manager to set limits on when an altered availability will affect schedule generation
- Decided on one availability applying to all weeks on the basis of it being easier for employees and managers

### ENHANCE LEMMING SCHEDULER OR G1'S SCHEDULING APPLICATIONS

Lemming Scheduler has a few bugs where as G1's application was seemingly more stable. G1 worked on a separate scheduling application during the same 308/309 courses as PolyCron.

- Greater learning curve: while I didn't write every piece of code in Lemmings Scheduler, I'm still familiar with the design.
- Making changes would be more difficult: again because I'm unfamiliar with their design
- Their project is more stable
- Ultimately I choose to stick with my project because its familiar and for the most part isn't that buggy

### ONLY STORING AVAILABILITY DATA IN GOOGLE DOCS VS AVAILABILITY ALSO BEING STORED LOCALLY IN DESKTOP APP

Does the desktop app need to store availability locally?

- Storing locally wouldn't require an active internet connection
- Storing it just in Google Docs prevents and consistency errors from arising
- Decided on just storing in Google Docs because storing it locally as well was superfluous and not having an active internet connection shouldn't be a problem for a small business

### DATA ENTRY IN TABLE: ONCLICK OR MOUSE OVER

When selecting availability slots in a table should selection be handling by mousing over a cell or by clicking on cell?

- Mouse over is quicker than onclick
- Mouse over behavior could be erratic
- On click offers greater control

- Ultimately, the added control outweighs the slight speed increase

### WHAT HAPPENS TO A TABLE CELL WHEN SELECTED

How should table selection be handled: click and drag? click on every cell? click on one cell to mark the beginning and one cell to mark the end?

- Click and drag is a common behavior, but could proof difficult to code
- Click on every cell is easy to code, but unwieldy
- Click on cell to mark the beginning and one cell to mark the end would be easy to code, but could be confusing for a user
- The third option seems better because of the ease of coding combined with quick entry for a user. Any confusion could be cured with a tooltip on help dialog box.

### HOW TO REPRESENT AVAILABILITY IN DATABASE

Should availability be represented as a giant table or broken up into multiple tables?

- Representing availability as a giant table models the object representation
- Using a giant table could prove unwieldy as you would need to have 24 x 7 columns for individual hours
- Breaking it up over multiple tables could be equally complicated
- Best solution is to just have one large table, it models the availability object.

### HOW TO REPRESENT SCHEDULES IN DATABASE

Should a schedule also be stored in one table or should it be broken up over a couple of tables?

- Breaking it up in this case makes sense because of the dynamic nature of shifts and shift assignment: for example, 3 people can work over the same time period one day and the next day only two
- Decided to break it up for the reason stated above

## IMPLEMENTATION

This section is divided up into two main sections: Process and Implementation Stages. The Process section covers the general implementation philosophy I practiced at each stage. The Implementation Stages section covers the steps undertaken during development and details the unique work associated with each stage.

### PROCESS

My implementation philosophy followed a general pattern. I did not code using a particular known process, but rather I approached each stage with my own unique method.

The first step I usually undertook was to write out all the data structures I would be using. In many cases I would be using upwards of 10 different data structures for a given module. The web client used upwards of 15 different data structures to handle the various tables, list, and maps being used. This forced me to think ahead and plan out the type of interactions occurring in the program.

The next step would involve one of two processes. When dealing with GUI programming, I would handle visual layout and instantiate data structures with empty data. I am a very visual thinker and seeing a clear model of what I was aiming for helped with the logically process.

When dealing with non-GUI programming, mainly database and Google docs related work, I would pseudo code for whatever algorithms I was going to handle. Pseudo code was especially important when writing database statements to ensure that I completed the proper DML and DDL commands.

The third step involved the actual coding. My approach was straight forward. When pseudo code existed, I matched my preplanned logic; otherwise, I used my knowledge of the design to match the planned functionality of a section of code. When problems would arise I would either consult Jeff or use one of the myriad of resources available to me, including: Javadocs, sun Java tutorials, and reference sites like w3schools.

The forth step focused on testing previous work. My testing was limited to an ad hoc approach, I did not utilize unit tests or any other structured form of testing. Instead I would do things like test whether a database statement retrieved the appropriate data by executing commands and testing the effect on the database.

The fifth and final step dealt with integration. I would ensure that methods in a newly written class integrated with methods in another already implemented class. A good example of this was in the Desktop application. I would be reading and writing information to and from the database that needed to update a local object. If an object in the employee frame was updated, I would have to ensure that object remained updated when a schedule was created. Similarly, I would test the interactions between web client and desktop application. I had to ensure that availability changes in the web client were reflected in the desktop app.

Throughout my implementation process, when a problem appeared particularly difficult or I coded a significantly smart section of code I would document it in my project journal. I used my project journal throughout implementation to reference logic used in one section that could be matched in other.

## IMPLEMENTATION STAGES

Development proceeded along in three stages. Each stage accounts for a change made in design due to technical difficulties.

### STAGE 1: GWT WEB CLIENT + DESKTOP CHANGES

Changes to the desktop were completed rapidly and without difficulties. We fixed lingering bugs and integrated Google Doc support into key components. We made minor UI changes to account for new Google Docs related operations, but in general the UI remained untouched.

The GWT prototype was adapted to handle web client duties. Unfortunately, we ran into major integration issues at this point and had to stop development on the web client.

*See Issue #1 from the Problems Section for a detailed explanation*

### STAGE 2: APPLET AS WEB CLIENT

After discovering our problems with the GWT web client, we made a design decision to use an applet instead. This proved to work in our favor in some regards. The similarities between working with applets and general Java swing programming allowed us to reuse a great deal of code. The only critical thinking that was needed was deciding how to layout the applets UI.

Logic for Google Doc communication was already written as was the table cell selection algorithms we needed.

While everything worked well in our NetBeans test environment, we discover some crippling issues once we tested elsewhere. Google Doc support in applets was lacking and required us to rethink our data model. In response, we decided to go with a database.

Using a database did not solve our problems. In fact it unleashed a slew of new issues involving applet security and SQL support. It seemed that applets just couldn't be relied on in the capacity we needed them for this project. Knowing that Java and SQL played fine in servlets and that the GWT used servlets I decided to return to where I first began my web client work. While I lacked experience using servlets, I was able to use online resources to supplement.

*See Issues #2 and #3 from the Problems Section for additional details*

## STAGE 3: GWT WEB CLIENT + DATABASE INTEGRATION

The first step I underwent was to write the database schema. I wrote out everything that I would need stored in the database and began deciding the appropriate relations between tables. The schema was relatively simple and was mostly complete on the first try. I did revise it a number of times, but they were all relatively trivial revisions.

With that done I began writing the servlet classes I would need. Servlets require multiple modules to handle asynchronous access. I coded a database manager class that handled all of the database operations for the web client. The servlet class would interface with the database manager, forwarding any database operations needed by the client.

## TESTING

Testing was very limited. Problems encountered during implementation caused scheduling issues, limiting the time available for testing. Any testing that was done was Ad Hoc and undertaken by the developer during code implementation and after code completion. There was no organized system testing.

## PROBLEMS ENCOUNTERED

We suffered through numerous setbacks during implementation. Three issues in particular called for major revisions. The first problem involved Google Web Tool Kit and the way it translates Java code into AJAX and JavaScript. This problem caused us to abandon our work on the GWT. The next problem involved Google Doc support in Java applets. We switched to using a MYSQL database as a result. The third and final problem involved security concerns with applets and SQL. I decided to drop applet support and return to using the GWT.

### ISSUE 1: GWT – NEED SERVLETS TO ACCESS GOOGLE DOCS

After completing the various prototypes and beginning implementation, we ran into a major problem with the GWT. We discovered that the GWT does not support Google's Data API. The problem boiled down to the GWT not knowing how to translate the Java code using the Data API into JavaScript and AJAX. While we had prototypes testing both the GWT and Data API, we neglected to test them together, which was a major oversight. The only solution involving the GWT was to implement Java servlets to handle all Google Document operations on the server side.

In response to this issue, Jeff and I decided to scrap using the GWT. Neither of us had much experience with servlets, so for the sake of time we switched to using Applets. While this meant that we had to abandon the code done thus far on the web client, we were able to recycle code from the desktop app to use in the web client applet.

### ISSUE 2: APPLETS – GOOGLE SPREADSHEET API CANNOT CONNECT TO GOOGLE DOC SERVERS THROUGH APPLETS

After scraping the GWT work we set about adapting our project to use an applet for the web client. We were able to get everything up and running with relative ease, but discovered a problem once we tried to access our applet outside the confines of NetBeans. The applet would generate an access violation being thrown from the Data API. After doing some research we realized that the Data API featured major development obstacles when working with applets. While we could connect using our test environment in NetBeans, once we tried to access Google's servers elsewhere the errors occurred. The only way to fix the problem we discovered was digitally sign all of the jar files being included in the applet. This turned out to be a major hassle and we dropped Google Doc support.

The decision to drop Google Doc support was also influenced by the poor performance and slow load times we were experiencing when read and writing from a Google Spreadsheet. Saving changes was especially time consuming. We dropped doc support in favor of database. I've used databases on a number of projects and felt that they could make up for the faults found with the Data API.

### ISSUE 3: APPLETS – DATABASE ACCESS THROUGH AN APPLET IS UNSECURE

The switch to a database on the desktop side was handled quickly and with ease. I ran into a stumbling block when trying to use the mySQL jdbc connector in an applet. By nature, applets run on the client side, after doing some research I discovered that handling SQL commands in an applet was a highly questionable move. First of all, it cannot be done easily and requires a number of hacks to work properly. Second of all, it's highly insecure. In order to get it to work a port must be opened on the client side to the server to talk with the database, which is totally unreasonable and insecure. With these facts in mind, I choose to scrap the applet and return to using my work from the GWT.

I did this for a number of reasons beyond just security concerns. One of my biggest gripes with the applet was how slow the download process was on the client side. Additionally, the applet was not aesthetically pleasing. Using GWT solves both of these issues by using servlets to increase speed and by allowing a developer to design nice looking UI's through cascading style sheets.

# RESULTS

The following sections covers my feelings on the current state of Lemming Scheduler as well a concrete look at Lemming Scheduler's feature set.

## KNOWN ISSUES

A number of known bugs are present in the current form of Lemming Scheduler.

- Shift templates are not stored in the database and still require serialization.

- Updates to availability made from the web client aren't visible in the desktop app until the app is closed and reopened.
- The web client doesn't display employees based on company.
- Updating the availability on the web client can cause data lose issues in relation to employee email address, phone number, and positions.

## GENERAL FEELINGS

I generally feel that the final project works well and could offer a small business benefits. However, in its current form Lemming Scheduler is not ready for release. The following steps need to be undertaken to ensure a quality product:

- In depth system testing
- Individual unit testing
- Full database integration and a complete removal of serialized data
- Performance analysis of database algorithms and web client interaction

## CURRENT FEATURE SET

- Employee's can view their own weekly schedules and edit availabilities
    - Schedule viewing and availability editing can all be completed within 60 seconds of opening the web client.
- Employee, skill, and schedule data are all stored in a database
    - The database is remotely operated on the Dahl virtual machine.
- Managers can save schedules to the database

- Schedules are automatically stored to the database on generation.

## DID WE MEET REQUIREMENTS?

The final product met all of the requirements Jeff and I laid out during the first quarter of senior project. We did not manage to include Google Calendar support, but that was a bit of an ideal goal from the start.

## SAMPLE USAGE

The section displays the amount of time required to complete all of the actions associated with the desktop application and web client.

### DESKTOP APPLICATION

| **Action** | **Time (seconds)** |
|---|---|
| Enter one skill | 10 |
| Enter information for one employee | 45 |
| Enter one shift | 30 |
| Generate Schedule | 10 |
| **Total:** | 95 |

### WEB CLIENT

| **Action** | **Time (seconds)** |
|---|---|
| Select name from drop down list | 3 |

| | |
|---|---|
| Alter availability information | 30 |
| **Total:** | 33 |

## FUTURE WORK

The section details the business application of the Lemming Scheduler.

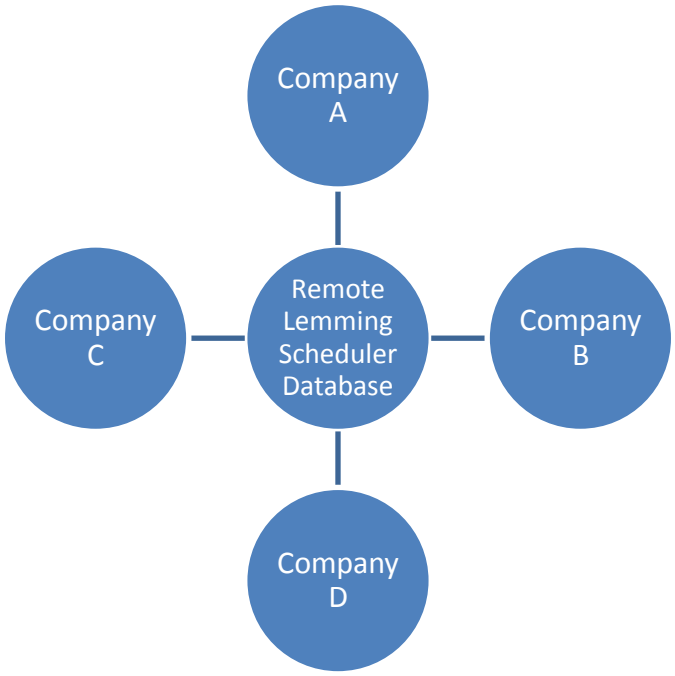


Figure 5 - Lemming Scheduler business model: each company recieves the desktop app that communicates with the remotely run database

### ADAPTING FOR BUSINESSES

One business model Professor Dalbey and I discussed for this project was to deliver the desktop app to small businesses and then manage all database operations on our end. This allows a business to avoid having to set up a database and would only require an active database connection. The advantages of this setup include:

- Rapid Deployment
- Ease of Use
- Data integrity

An employer could simply download a copy of desktop app via web, fill out some simple online registration forms with business information, and then begin scheduling. Some of the related work I researched required the user to download and set up a database. This requirement is well outside the technical skills of an average small business owner. Lemming Scheduler removes the database requirement and supplies an algorithm for automated schedule generation, another feature lacking from current products on the market.

### EMAIL SUPPORT & GOOGLE CALENDAR SUPPORT

There is room in this project for a lot of improvement in terms of schedule delivery to employees. I feel that it would be helpful if the desktop app emailed schedules once a week or updated them to Google calendar. I've worked with the Google Data API, so I wouldn't expect the Calendar API to be that much more difficult. This offers a lot of advantages for employees, especially if they use smart phones.

These two changes together could make the Lemming Scheduler a viable product in the small business arena

## LESSONS LEARNED

### PROTOTYPE AND RESEARCH EVERYTHING

The main lesson I learned was the importance of researching the technologies being used thoroughly. That means reading source documentation and developer impressions of working with the target technology. Along those lines, I also learned the importance of prototyping. In depth prototyping would have saved Jeff and I countless hours on wasted development by discovering technology compatibility issues early.

Research can stop a person from wasting time on ideas that won't work together. Prototypes can ensure that ideas that do work are feasible given various constrains like time and resources.

### STEPS I WILL TAKE IN THE FUTURE

1. Decide what technologies will work for a given problem
2. Research whether said technologies will integrate with one another
3. Prototype technologies working together
4. Begin design and development

## CONCLUSION

My senior project work impressed upon me the shear amount of technology required to run a large scale application. From beginning to end, I used the following technologies: Google Data API, Google Web Tool Kit, HTML, CSS, Java, MYSQL, SVN, NetBeans, and Eclipse. Often times in course work you focus on a single technology for an entire quarter, without seeing how that technology interfaces with something like a database.

Using multiple technologies hold many risks that adversely affected Jeff and I. We were forced to undergo multiple design changes to compensate for these errors. I've learned an important lesson on research and prototyping as a result. Work on the Lemming Scheduler has enhanced my ability to handle multiple technologies and shown me the dangerous involved in doing so.

Development on the Lemming Scheduler prepared me for a real time work environment. I was able to work with a large code base, much of which is written by other people. I dealt with numerous deadlines and worked under stress. I dealt with technical setbacks and managed to finish on schedule. Taking all of these factors into account, Jeff and I delivered a product with actual business potential.

## BIBLIOGRAPHY


1. **Seshadri, Govind.** What are the advantages and disadvantags of serialization? *jGuru.* [Online] Jan 15, 2000. [Cited: May 20, 2010.] http://jguru.com/faq/view.jsp?EID=5068.

2. **Chapple, Mike.** The ACID Model. *About.com.* [Online] [Cited: May 20, 2010.] http://databases.about.com/od/specificproducts/a/acid.htm.

3. **Arb Design.** Employee, Staff and Group Scheduling Software: Schedulewall. *ScheduleWall.* [Online] 2008. [Cited: May 22, 2010.] http://www.schedulewall.com/.

4. **TimeCurve Software.** TimeCurve Software, Employee Scheduling Software. *TimeCurve Software.* [Online] 2008. [Cited: May 22, 2010.] http://www.timecurvesoft.com/tcdescription.jsp.

5. **Kappix.** Kappix - It's About Time. *Kappix.* [Online] 2010. [Cited: May 22, 2010.] http://kappix.com/.

# APPENDICES

## INSTALLATION GUIDE

### DESKTOP APPLICATION

1. Ensure that Java 1.6 or greater is installed on your computer
2. SVN checkout a copy of the repository at

   https://wiki.csc.calpoly.edu/svn/PolyWebCron/Release/Desktop
3. Open up terminal(linux) or command line(windows)
4. Navigate to the directory you checked out Lemming Scheduler
5. Run the following command Java –jar LemmingScheduler.jar ‘companyid’
6. Click on the ‘To Do’ button of the desktop application
7. Follow the steps on screen

### WEB CLIENT

1. Open a web browser
2. Navigate to dahl.csc.calpoly.edu:8080/markSP
3. Follow on screen steps

### TOMCAT DEPLOYMENT

1. Retrieve latest war file for web client from SVN
2. Go to the tomcat homepage at dahl.csc.calpoly.edu:8080
3. Select the manager option
4. Enter user name and password
5. Navigate to the middle of the page
6. In the box label war file click and select the location of the war on your local machine
7. Click the deploy button

## DATABASE SETUP

1. Retrieve database scripts from PolyWebCron website
2. Open a terminal and navigate to the folder containing the scripts
3. Start mysql
4. Enter username and password
5. Navigate to PolyCron database
6. Run the following command \. sp-db-setup.sql
7. The database is now ready