Schedream: A Consultant's Scheduling Dream

by

Christopher G. Cotton

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Computer Science and Engineering

and Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 1994

Author _____

Department of Electrical Engineering and Computer Science
May 16, 1994

Certified by _____

Professor James D. Bruce
Thesis Supervisor

Accepted by _____

F. R. Morgenthaler
Chairman, Department Committee on Graduate Theses

Schedream: A Consultant's Scheduling Dream

by

Christopher G. Cotton

Submitted to the
Department of Electrical Engineering and Computer Science

May 16, 1994

In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

Many consulting and support services need to create Master Schedules to allocate efficiently their consultants' time. Creating these schedules manually entails many problems, including inefficient use of human resources and difficulty in maintenance. The Schedream system addresses these issues with an electronic scheduling system that automatically generates schedules and allows for easy schedule maintenance. The system is implemented as a Macintosh client and UNIX server, written in C++. Schedream also incorporates an extensible architecture for future enhancements. Testing of the preliminary system has shown that it successfully generates schedules using a simple scheduling algorithm.

Thesis Supervisor: James D. Bruce
Title: Vice President for Information Systems and Professor of Electrical Engineering

List of Figures

1. Introduction

When you cannot figure out how to get your word processor to spell check your thesis, or when your computer does not even turn on, what do you do? Hopefully, after thinking about tossing everything out the window, you call your local computer expert. Many organizations and companies hire a group of these experts, called consultants, to support the entire company. The companies put these consultants in a single location where the rest of the employees contact them and get support via telephones, on-site, on-line, email, other electronic means, and even by walking in. The groups are referred to as "Help Lines" or Help Desks.

To provide quality support, each Help Desk needs a Master Schedule which lists, "which consultants work when and how long." Generating this Master Schedule and keeping it current is a job of the Help Desk administrators. The complexity of this process rapidly increases as the number of consultants increases. My thesis concentrates in the area of creating an electronic system to support this scheduling process.

In order to generate a Master Schedule the administrators must accomplish a couple of tasks. First, they must determine how many consultants are needed for each block of time during the week. They must balance each hour between experienced and new consultants, in order to train the new consultants, and yet keep the quality of support high. If an hour needs a certain expertise of a consultant, they must make sure those needs get fulfilled. The administrators must take into account the scheduling availability and preferences of the students and then use those to generate the master schedule. After the final version of the schedule has been generated, they must be able to make changes throughout the term.

In M.I.T. Information Systems Computing Support Services Group there are two Help Desks that serve the M.I.T. Community. These are the Micro Computing Resources (MCR) Help Line and the Athena On-Line Consulting (OLC) Help Line. Together these Help Desks employ 4-8 full-time consultants and around 35-50 student consultants, who could work for both Help

Desks. Having students as consultants complicates the scheduling process. They have a limited number of hours, and the job must match the student's studying and class schedule. Also with students, the Master Schedule must change every semester, and may also fluctuate during the semester due to adding/dropping classes or even exams.

How do you accomplish this massive task of generating the Master Schedule? Currently, the M.I.T. Help Desks have the dreaded "Scheduling Meeting" once a term. In this two hour meeting, the available slots of the Help Desks are filled by student consultants. Each consultant in turn takes a PostIt™ Note with his or her name and places it on a huge paper version of the weekly schedule. The consultants are ranked by experience to give first choice of hours to the senior consultants. This actual scheduling process is highly inefficient since people spend most of their time waiting for their turn. It is also expensive since each consultant gets paid for the full two hours.

The goal of my thesis is to design an electronic tool that can be used in the scheduling process: setting up the Help Desk requirements, collecting the consultants' preferences, generating the Master Schedule, and allowing changes. The goal is to have the software take away the burden of current administrative tasks.

I researched previous development on a scheduling system and also interviewed the full-time and student consultants. After defining the problem, I designed a system for the scheduling process. The resulting design is presented in Section 5. Implementation, Section 6, covers the specific part of the system I chose to implement as a proof of concept. Designing and building the proof of concept brought out many previously unanticipated issues which need to be addressed to make the current system professional. These issues are covered in the Conclusions and Future Directions section.

2. Background

In order to understand the motivation for this project, it helps to look first at the two MIT help desks, and at previous work which has been done regarding the scheduling process.

Students can work for one or both of the Help Desks. The two Help Desks are tied together though joint training meetings and shared administrative functions (time cards, hiring, etc.) Although tied together they provide different areas of support. Micro Computing Resources (MCR) meets the needs of clients using personal computers, including giving information on a collection of software packages and limited hardware support on DOS, Windows, and Macintosh platforms. Athena On-Line Consulting (OLC) provides on-line consulting on the use of the Athena system (mainly consisting of UNIX workstations with email, news, word processing, file management, and other network applications).

The scheduling process over a semester's time can be seen in Figure 1. There is an initial scheduling meeting to create the master schedule. That schedule is subject to modification during a two week period (due to students' changing classes), and then that schedule governs the rest of the term. Any other changes to the schedule (maintenance period) are handled by a student consultant who spends about two hours each week creating a current schedule for the upcoming week.

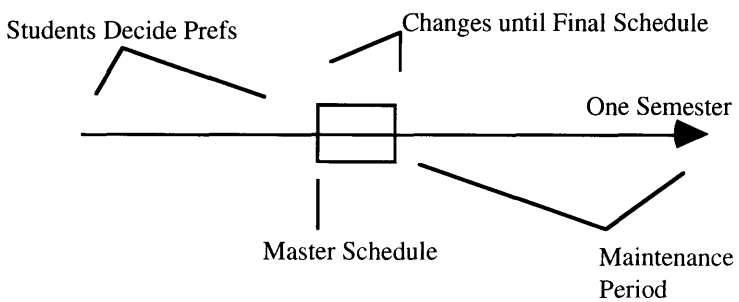

Figure 1: Diagram of the scheduling process with divisions over the time of one semester.

There has been some previous development done on the scheduling problem for the MIT Help Desks. Thomas Palka designed and implemented a scheduling program as part of an MIT independent study class. His program focused on the scheduling process from the beginning to the generation of the "First Schedule" (see Figure 1). The students used a Motif program on Athena to select their preferred hours. The administrators then used the program to generate a final schedule.

Since his program was a limited solution to the general scheduling problem, there are several issues left unaddressed. First, the program was not designed to handle changes to the final schedule or the maintenance period. It operated solely via a simple algorithm that generated the schedule with no options for how students were selected for each hour. Besides the limited algorithm, there were many parameters that could not be changed without recompiling the program, e.g. there was a predetermined list of abilities assigned to the consultants which could not modified. The current implementation of the program was limited to one platform, UNIX. Also, the MCR Help Desk administrators would not use the program because the algorithm lacked necessary constraints, e.g. preference could not be given to the consultants with a larger contiguous block of hours. Palka's development gave me a starting point for my design, and my design seeks to address some of those limitations.

3. Motivation

I have worked for both of the Help Desks for over four years, and have seen a growing need for a new scheduling system. Every time I suffered through one of the scheduling meetings, I agonized knowing software tools should exist that would allow the process to be less painful. As part of the Consulting Group, I have looked for tools which could do this scheduling. There have been many tools, but each only solved a small part of the problem and caused more trouble than it was worth. Therefore, only an original design would cover all the necessary aspects.

The scheduling problem lends itself to a electronic solution, since computers are great for executing tedious algorithms. When the number of consultants reaches 20 or more, it becomes

difficult and time consuming for the manual scheduling process. The manual system requires all the consultants to be in the same room at the same time. In contrast, the proposed electronic solution would allow consultant to enter in their scheduling preferences over a longer period of time. Because the system is distributed, the consultants can schedule from any location as long as they are connected to the network.

Besides being a great technical problem, I have personal motivations for this project. Being part of the Athena Computing Environment, I have seen the advantages of a distributed computing environment and now I can actually implement such a system. This system also gives me a chance to do what I love, designing tools and systems. Finally, throughout my four years working at the Help Desks I have always looked for ways I can give something back, especially when it supports the jobs of student consultants. Instead of dealing with the busywork of scheduling, they can spend more time learning and serving customers.

4. Defining the Scheduling Problem

Before attempting to solve any problem, it is always an excellent idea to determine the actual problem. I interviewed almost all of the full time consultants [Full-Time interviews, 10] and also received many comments from the current student consultants. The objective of the interviews was to distinguish the important information to track and the unsaid rules used in scheduling. I asked the full-time consultants what the important rules are, and what their wishes for the perfect scheduling system were.

4.1 What is the problem?

The problem reduces to a complex resource allocation problem with constraints. There are many consultants with available hours that need to be assigned to the various Help Desks. The problem occurs in deciding which consultants to assign to a given slot, and what information to use in making the decision. This leads to various constraints that either need to be met, or that determine the assignment order.

These constraints and the inherent structure of the information determines how the scheduling system manages the data. They also impact how the scheduling algorithm actually works.

4.2 What are the constraints?

There exist a couple of underlying rules for the scheduling process. These fall into several categories: information about hours on the help desk, the consultant's personal choices, and how the hours are distributed.

4.2.1 Help Desk Template

Most help desks are open during normal business hours (9am-5pm), but may have consultants working other hours. Each hour also has a certain number of slots, or positions that consultants fill. The open hours and the number of slots create a "Template" for a specific help desk. Each of the consultants who fill one of these "slots" has a specific level of experience (newly hired, senior consultant, etc.). These levels need to be balanced during each hour in order to maintain quality support and yet train the inexperienced consultants. Since, students can take hours on more than one help desk, it is important they are not assigned to more than one Help Desk during a single hour. Finally, particular hours should not be overloaded while leaving others empty.

4.2.2 Consultant's Choices

After the Template has been set up, the next important aspect is what hours the consultants want. For each hour on a Template the consultant needs to specify their availability — whether they could possibly take a given hour. In order to give the consultants more choice, they get to rank their hours: 1st Choice, 2nd Choice, etc. The algorithm should try to assign the consultant all of the hours ranked as 1st Choice, before moving on to the list of lower ranking hours. Consultants can also create a group of hours, which means they either want all of the hours in the group or none of them. Besides which hours they want, the consultants need to specify the total number of hours they want to work. The consultants should be able to set their

own minimum and maximum hours inside the limits the administrators set. From the student interviews, I found that there are an incredible number of ways the students determine their hours. Sometimes students would want to work with a buddy, or else they have a particular person they cannot be with in the same room at the same time. A Student's preferences can also have many dependencies — "only give me this hour if I also get this other hour, but only if on Sunday if I get at least 5 hours, otherwise give me hours on Saturday." [8] Many of these choices were determined to be either impossible to implement, or were too arbitrary and only applied to one consultant.

4.2.3 Hour Distribution

The next set of constraints deals with the order in which the consultants are assigned hours. The administrators assign each of the students a seniority ranking, determined by the number of full semesters' worked. Seniority levels can be created by assigning multiple consultants the same rank. Selection of hours is given preference to the higher ranked seniority levels. If allowed, the higher ranked consultants would take all of the "good" hours, leaving the less experienced consultants with scattered less preferable choices. There needs to be some way to make sure the hours are distributed fairly. If consultants have the same ranking and conflicting choices, other methods need to determine who gets assigned hours first. From the interviews, the full-time consultants said consultants selecting contiguous hours should get first choice. This decision is based on the observation that consultants are more productive with a larger block of time, since it takes a certain amount of time to get started each shift. The final method for giving preferences is determined by the specialties. This one is last since it is more important to get consultants scheduled to hours, and the MIT Help Desks have a history of unassigned hours. Each consultant has a couple areas of knowledge they are considered experts on. There are certain times during which these specialties would be useful based on hour the help desk operates. For example, if clients continually come in on Fridays with questions about Microsoft Word it is important to have a couple of consultants with that specialty at that time.

5. Design

5.1 Electronic System

There are three possible approaches the system could take in assisting during the scheduling process. In the first approach, the schedule system would be used as a real-time tool during the scheduling meeting. Students would still take turns, but would record the results directly into the program. The problem with this model is that it still takes a large amount of time during the meeting and does not use the computer's capabilities of processing the scheduling algorithm.

With second approach, the consultants would input their choices using a program. After all the preferences had been entered, the program would generate a single master schedule. This is the method that Palka's program took [2]. The problem here is that in order to get an acceptable schedule (one in which the students get some choices), the program must support a large number of options the students can use to tell the system what hours to give them.

The final approach follows the second approach, but allows updates via electronic means. I chose this method because it gives the students the opportunity to change their hours electronically, with the system handling the administrative overhead of updating the schedule. Thus, the birth of "Schedream."

The goal of Schedream is to take administrative duties from the administrators and off-load them to the student consultants and to the server itself. The operation of the system also has some more intangible requirements. This includes goals such as "Be Quick!" and "Save Time And Money!" Specific goals include working with multiple help desks, and working on multiple platforms. The overall design goal is to be extensible. This means that instead of having fixed lists (e.g. the number of Help Desks), they should be variable.

5.2 Information to Track

There are many pieces of information that need to be tracked in order to provide the algorithm enough information to do its job — to generate the Master Schedule.

The information can be broken down into various divisions and the interface of the client reflects those divisions. The Help Desk Template covers any information that is specific to one of the help desks covered by the system, including open hours. Administrative information is kept about each of the consultants, including their names, their experience levels, and what specialties they have. Only the administrators can set that information. Next is information or choices that each consultant inputs. This includes the consultant's personal ranking of hours and minimum and maximum number of hours. The final group is any information relating to the Master Schedule. A Master Schedule contains listings of consultants scheduled to work during each time slot.

5.3 Client/Server Division

The system is based on the client/server model, in this case a single server and multiple clients. One of the benefits of this architecture is the ability to change the server implementation without changing the client implementation.

The server takes on the function of storing and controlling access to the information that the client uses. The server also runs the scheduling algorithm. On the other side, the client software provides the interface a user sees when modifying or inputting information.

5.4 Client

This section describes one possible client interface. Other interfaces are possible, but they would need to provide the same functionality. Each sub-section describes one of the major scheduling activities.

5.4.1 Help Desk Template

The first scheduling task is for the administrator to setup a Template for each Help Desk. The design of the window to edit each Help Desk can be seen in Figure 2. The Help Desks at MIT are scheduled at a granularity level of one hour, and the schedule repeats every week. Other Help Desks could possibly be based on half-hours. For each Help Desk, the name field, *Help Desk Name*, must be unique. The *Largest Hour Block* is the largest number of consecutive hours in a day that a consultant can work. To edit the list of *Specialties* or *Experiences* the user clicks on the respective buttons.

The hours for the week are displayed using a table-like method. This is the most common method that people use for editing or viewing a weekly schedule. For each hour, the table displays parameters that the administrator can set. These parameters determine the number of consultants, and which consultants can sign up for a specific hour. To set the parameters for a given hour, the administrator clicks on the desired hour. The hour's parameters can then set in the panel labeled *Current*. Another approach would have been to allow in place editing, but screen space limits the number of editable fields.

In the Current panel there are two scrolling list controls, labeled Experiences and Specialties. These determine which and how many experiences of each type are necessary to have, and the specialties consultants needs to get preference for this hour. A simple click on one of the names in the list increases that number by one, and an option click will decrease it.

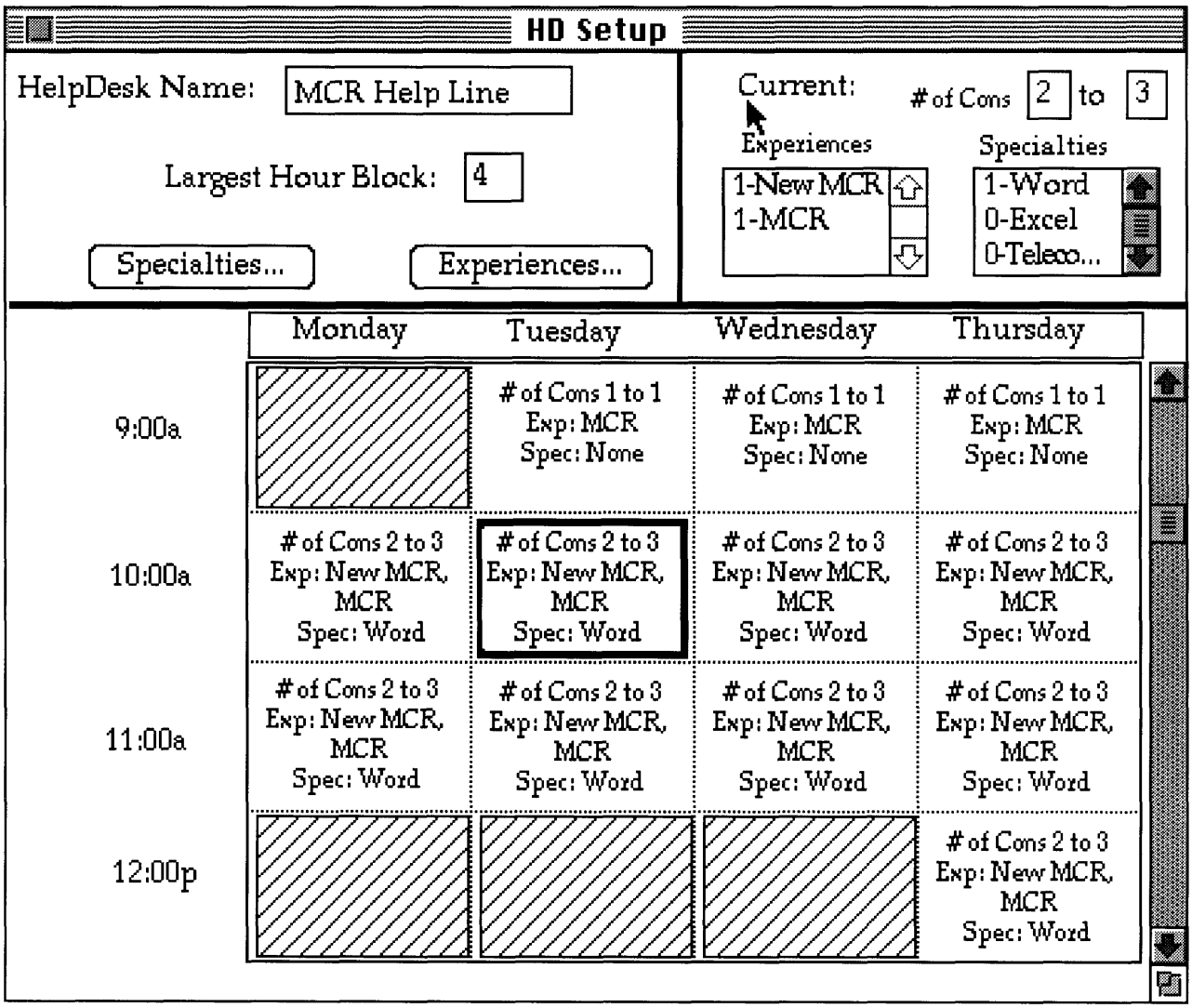

Figure 2: Help Desk Template Window. This shows a possible design of an interface to modify the Template.

5.4.2 The Consultants

In order to allow consultants to edit their preferences, the administrator needs to add them to the list of consultants. The entire list is edited at one time and displayed in its own window. (See Figure 3).

Username	Full Name	Help Desk Name	Min	Max	Rank	Experience	Specialties	Turned in Prefs
regis	Chris Cotton	OLC HelpLine	0	7	5	OLC	Unix, Emacs	Not Yet
		MCR HelpLine	4	15	1	MCR	Networks	3/10/94
wchuang	William Chuang	OLC HelpLine	4	30	1	OLC	Networks, Unix, Emacs, C	3/10/94
		MCR HelpLine	3	10	2	New MCR	None	3/10/94

Figure 3: A Sample Interface to Edit Consultants.

The administrator has the ability to edit any of the information about each consultant. Administrators can also add or delete consultants.

Each consultant has the same information tracked. First, *Username* is used for authentication purposes. For each help desk the consultant, the *Rank* is a seniority rank, where lower numbers mean a higher ranking. The *Min/Max* columns refer to the minimum and maximum number of hours a consultant can take on a specific Help Desk. The *Experience* column contains the level of the consultant's experience. The *Specialties* column includes the specialties the consultant has (e.g.. Word, LaTeX, etc.) The last column allows the administrator to determine who has or has not turned in preferences for the specified Help Desks, which helps in terms of deciding when to run the scheduling algorithm.

5.4.3 Consultant's Choices

Figure 4 shows the interface for consultants to input their choices. A separate window is used for each user. Using the popup menu, the consultant selects which Help Desk he wants to set preferences for. Only Help Desks that the consultant is eligible to schedule for will show up

in the popup menu. The consultant sets the minimum/maximum number of hours they want for each Help Desk. These numbers are constrained by the min/max set by the administrators.

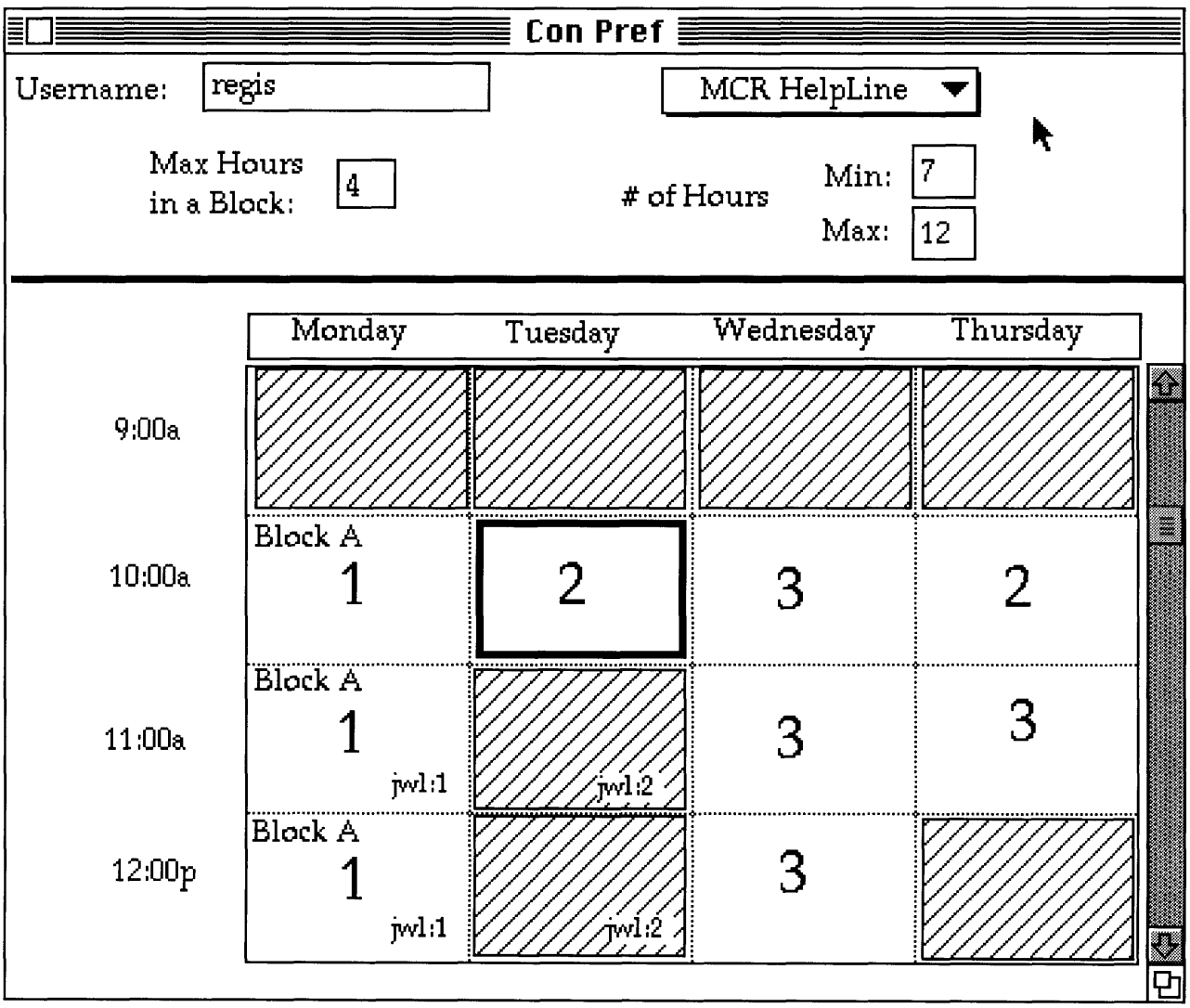

Figure 4: A Sample Interface for Consultant Choices.

The table in the bottom half of the screen represents the availability and ranking of the hours for that particular Help Desk. This ranking determines the order in which the algorithm attempts to schedule the hours. After highlighting the hours (by using the mouse), the hours can

be set a ranking from 1-9 (1 being the most desirable) by pressing the appropriate key. Pressing the space bar will make all the selected hours non-available.

Consultants can also group hours together creating a block of hours, with the same ranking. This group is treated as a single assignment, since the consultant will receive the entire block of hours, or none of them. The word, "Block," with a letter, determines to which block the hours belong.

The consultant also can show the preferences for another consultant. This allows them to see when a friend (or foe) of theirs had selected hours. The overlapping hours are displayed using the other consultant's username and what rank the consultant gave that hour (e.g., jwl:2).

5.4.4 The Master Schedule

A Master Schedule is the schedule that has been created using an algorithm to assign consultants to a specific Help Desk Template. For each hour of each day, it contains a listing of the consultants scheduled to work during that hour (See Figure 5).

	Monday	Tuesday	Wednesday	Thursday
9am	regis wchuang	regis	wchuang	wchuang
10am				
11am				
12pm				

Figure 5: A Sample Interface for the Master Schedule

The administrators can modify the Master Schedule by swapping, adding, and deleting consultants.

This window also allows the consultants to modify their own hours. They can make permanent changes — saying that they will no longer work a particular hour for the rest of the

term. They also have the ability, instead of just giving up hours, to swap them with another consultant.

The server must log all the changes to the schedule. This allows administrators to determine what changes have occurred.

Finally, the client can export the Master List in a variety of formats. This allows the information to be used in other applications or sent via email in a readable format.

5.4.5 Using the Algorithm

In order to generate a Master Schedule, the administrator uses the window (See Figure 6) to select the Help Desk Template, and the algorithm with options. The pop up menu contains an entry for each algorithm the server has. The window then lists the specific options for that algorithm. The administrator supplies the name of the Master Schedule to be created.

The algorithm shown has the name "Default." The check box determines whether the algorithm gives preferences to consultants with larger blocks. This algorithm cycles through the consultants in order of their seniority ranking. The text box, with the label "Hours Assigned Each Cycle," determines in each cycle, how many hours should be assigned to the current consultant before moving on to the next available one.

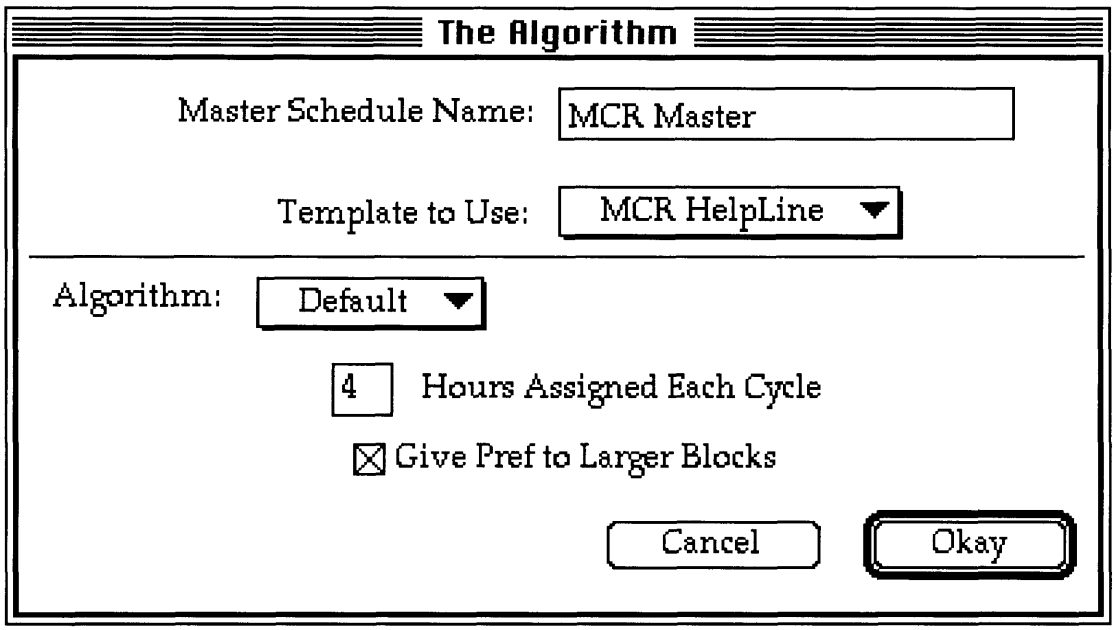

Figure 6: A Sample Interface for Using the Algorithm

5.5 Server

The server has a two functions. First, the server stores all of the data, including the list of consultants, the preferences for each consultant, the hours of the Help Desk, and the Master Schedule. Second, the server generates and maintains the final schedule.

An authentication method is needed since the user of the client application must prove their identity to the server. Every user of Schedream is assigned on of the possible authorization levels, currently "Consultant" and "Administrator." This level determines the available commands, e.g. only a user with "Administrator" can modify a Help Desk Template.

Since multiple clients can access the server at the same time, the server should provide methods which allow the client to "LOCK" various parts in the database. When a client locks a part, another client cannot modify this information until the client releases the lock. This avoids the problem of users making conflicting changes to the database. The locking applies to both administrators and consultants.

5.5.1 Algorithm

Since the chosen approach employs object oriented technology, one of the goals is to create a easily replaced or modified general algorithm object. By a general algorithm object, I mean an object class that always takes the same inputs and has one simple function call to generate the schedule. Since the object uses a standard interface, it allows newer and more efficient algorithms to be written and inserted without modifying the rest of the system. Options would allow the administrator each semester to change their decisions about the specifics of each scheduling algorithm. The final reason is this is the first electronic system in use, and there are some cases which cannot be determined until actual students have been run through the scheduling process many times. With a general object and options, this possible breakdown can be avoided.

The algorithm is initialized with a Help Desk Template, a Master Schedule to fill, and all the consultants and their selection of hours. It takes all of these and turn them into a Master Schedule (see Figure 7).

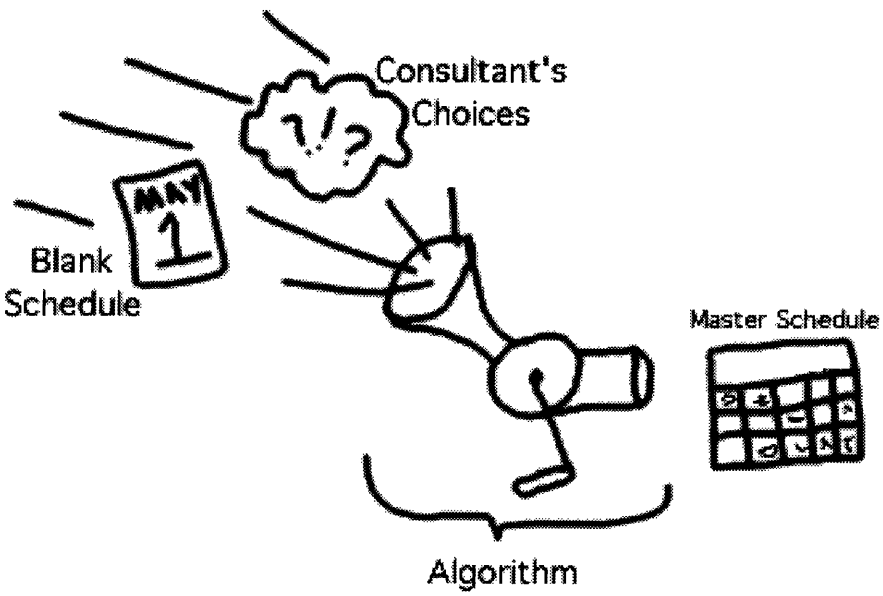

Figure 7: The Inner Workings of the Algorithm

A simple version of the algorithm works in the following manner. Consultants are ordered by their seniority ranking. Each consultant is taken in turn and assigned a number of slots. This process continues until all the slots have been assigned. In the case of a tie, preference is first given to consultants in the same rank who can take larger blocks of hours. If there is still a tie, preference is given to the consultant who matches the abilities of the slot.

A more complex version could start to add other features, such as varying the order of assignment instead of a strict assignment by rank. This would allow the algorithm to try to give all the consultants a certain percentage of their first choice hours. It could also first try to give everyone the minimum number of hours, before assigning additional hours to consultants.

The algorithm allows the administrators to receive feedback about the performance of how well the algorithm performed given the input data. It shows what percentage of people received their first choice. Other statistics include the total number of hours scheduled, and number of slots that were not filled.

One of the features of having multiple algorithms with a plethora of options is that you can generate "what ifs." This allows you to use the same consultant preferences and generate many different Master Schedules. The administrators can then select the one that is best.

5.1.2 Formal Algorithm

This is a formal explanation of the simple algorithm implemented in the server. The method *RunIt*, which handles the actual scheduling process, follows.

```
void CAlgo::RunIt(char *theCommand)
{
    maxCount = DEFAULT_MAX;
    goAgain = 1;
```

RunIt's parameter, *theCommand*, contains the options to the algorithm. The *maxCount* variable determines the maximum number of hours that are assigned to each consultant during each iteration of the main loop. DEFAULT_MAX is arbitrarily set to 4.

```
while (goAgain)
{
    goAgain = 0;
```

This main loop continues until an iteration runs without a single hour being assigned to a consultant.

```
SetUpUsers();
while( (user = GetNextUser()) != -1)  // NO_MORE_USERS = 1
{
```

After calling *SetUpUsers*, which sorts the consultants by their seniority ranking, *RunIt* starts a loop which cycles through each consultant.

```
count = 0;
if (OverLimit(user))
    continue;
while ((hisHour = GetNextHour(user)) && (count < maxCount))
```

It resets the *count* variable (the number of hours that have been assigned to the current consultant). If the consultant has reached the maximum number of hours, then *RunIt* continues for the next consultant. Otherwise, it enters a loop that examines each of the consultants' hours. Every hour is examined in the order of the ranking assigned by the consultant. This continues until either there are no more hours or *count* has exceeded the maximum number of hours for this cycle.

```
{
    when = hisHour->Number();

    if (CanSchedule(user, when))
    {
        Schedule(user, hisHour);
        count++;   // since we just added another hour
        goAgain = 1;  // there still may be people left
        hisHour->hourStatus = H_SCHEDULED;
    }
    else
        hisHour->hourStatus = H_BUSY;
```

Then, *RunIt* calls *CanSchedule* to check if the current consultant can be assigned to that hour. It then either assigns the consultant to that hour, or it marks that hour as H_BUSY. The process is finished when no hours are assigned during a cycle through the consultants.

6. Implementation

6.1 Overview

Initially I set out to design a complete system that could stand the test of time and brave the battle of a real-live scheduling meeting. The problem occurred in realizing what it actually would take to implement a "sound" server. This section describes what I have implemented, and also covers many of the problems and details that need to be attended to in order to make the system a full version instead of just a proof of concept.

For the actual implementation of the programming languages I chose Symantec C++ with TCL (Think Class Libraries) for the Macintosh, and GNU g++ compiler.

I chose to implement the server side on a UNIX platform, specifically the DECStation 5000. For the streaming implementation I chose TCP because it is available on all platforms and is widely used.

6.1.1 C++

I chose to write the server and client in C++. Besides its being one of the hottest trends today, there are two main reasons for choosing C++ as the base: the underlying data structures and the need for a GUI (Graphical User Interface).

The data structures that compose the Help Desk are very redundant—everything is based on hours. These data structures lend themselves to an easy design and implementation in C++, which has the advantage of being able to easily reuse code.

The other main issue is designing a application with a GUI. With C++, there is a smaller overhead in writing a GUI, because the developer can draw on the reusable objects that others have written. The developer does not have to reinvent (and recode) the window. The language choice is a big design issue since the framework will be need to be designed with an object oriented methodology in mind.

6.2 Data objects

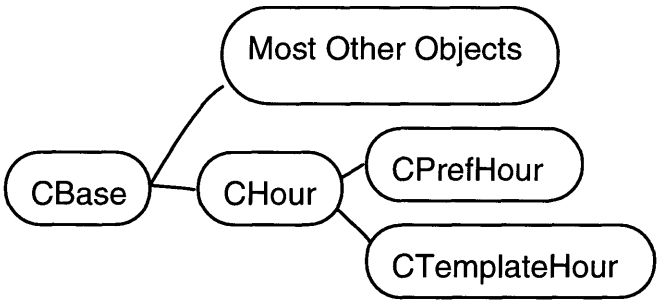

Figure 8: The Object Hierarchy

6.2.1 Overview

In the high-level design, a object class exists for each of the major parts of the scheduling system: *CConList* — listing of consultants, *CConChoices* — a consultant's choices, *CTemplate* — a Help Desk Template, and *CMaster* — a Master Schedule. Each high level object contains arrays of lower level objects and functions to modify them. The lower level objects include *CPrefHour* — a single hour of a consultant's choices, *CTemplateHour* — a single hour of a Help Desk Template, and *CCon* — a data structure. Other lower level objects provide a general data storage mechanism: *GenList* — a list of strings, *HDStruct* — specific information about a Help Desk, *String* — how strings are stored, and other objects that provide a data storage.

The object hierarchy (Figure 8), shows that the most important sub-classes are from *CBase*, the basic data object. There are also sub-classes of the basic hour object to deal with different needs for hours.

These same data objects are used on both the client and server side, and the same source code should compile on both platforms. Only the class representing the underlying data storage mechanism needs to be altered.

The following sections give a high level description of the objects for each data structure, with information about which other objects they use.

6.2.2 GenList

I needed a general storage object in order to implement many of the data structures. *GenList* provides a list storage facility for many of the objects. The *GenList* object deals with elements of a string of the form: "element, element, element", and can translate between that form and its own internal representation. It provides all the standard operations for lists: addition of elements, deletion of elements, and checking whether the list contains certain members.

6.2.3 CBase

Since many of the objects are based on data which has been read in from a file, I needed a way to check whether an error occurred. I wrote a base object, *CBase*, to fulfill this need. Any object which is a sub-class of the *CBase* object gets the functional equivalent of a large GOOD/BAD switch. Other objects can easily test whether the data contained in the object is valid using standard methods.

6.2.4 CHour

An instance of this object is a specific hour. In this case, it is a single hour of one week from Monday to Sunday, e.g. "m10" refers to Monday at 10 AM. Since all references to hour are through this one object, it would be a simple task to change the granularity of the entire scheduling system by changing this one object. *CHour* objects also know how to read and write themselves to and from a string.

6.2.5 CTemplate

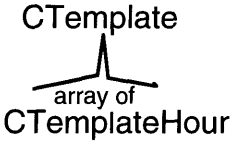

Figure 9: Structure of *CTemplate*

For each Help Desk a *CTemplate* is created, it includes an array of *CTemplateHours* which are the open hours (See Figure 9). It also includes information specific to the Help Desk. Currently, the only information kept besides the name is the largest number of hours a consultant can take in a row and a listing of the possible specialties and experiences on this Help Desk. See Appendix A.1 for the file structure.

6.2.6 CTemplateHour

In order to keep track of every hour during the week, *CTemplate* keeps an array of *CTemplateHour* objects. *CTemplateHour* objects are a sub-class of *CHour*. The additional attributes the object has are the minimum/maximum number of consultants for each hour. It also has two GenLists to keep track of the specialties and experiences needed for each hour.

The format of the template hour is "m10:max:min:expr,expr,expr:spec,spec,spec". An example of an hour without any lists would be "m10:3:1::" I chose colons as separators for the tokens because names of specialties/experiences do not contain a colon as part of there name.

6.2.7 CAuth

This object contains all the authorization code. It keeps track of usernames and there corresponding authorization levels. It can read and write itself.

See Appendix A.2 for the format for the authorization file.

6.2.8 CConList

This object keeps track of the administrative information in the consultant listing file, by using an array of *CCon*s (See Figure 10). The object knows how to read/write the entire list from a file.

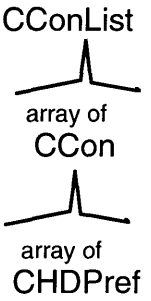

Figure 10: Structure of *CConList*

6.2.9 CCon

CCon is the object used to represent a user in the scheduling system. It has two strings: username and fullname. The object also keeps a list of *HDPrefs*, one for each Help Desk that the consultant can schedule for (See Figure 10).

6.2.10 HDPrefs

The *HDPrefs* object keeps the minimum/maximum number of hours that the consultant can take. There are two general lists, which are the experiences and specialties that the consultant has for the particular Help Desk. There is also an integer, which is the rank assigned by the administrative consultants.

6.2.11 CConChoices

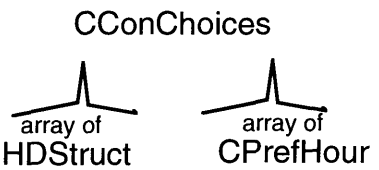

Figure 11: Structure of *CConChoices*

An instance of the *CConChoices* object keeps track of a consultant's selection of hours and Help Desk specific options (See Figure 11, and See Appendix A.3 for the file structure). To keep track of the hours it uses an array of *CPrefHours*, one for each hour of the week, which keeps a rank for the specific hour. The *CConChoices* object has a number of *HDStructs*, one for each Help

Desk since the consultant can set his own personal minimum and maximum number of hours for each Help Desk.

6.2.12 CMaster

Figure 12: Structure of *CMaster*

In the master schedule, for each hour there is a listing of consultants. *CMaster* uses a *GenList* for each hour to store the usernames of the consultants (See Figure 12). *CMaster* also contains the name of the *CTemplate* that it was based upon. This allows the administrator to later determine the original *CTemplate* for each the Master Schedule.

6.2.13 iofuncs

Almost all of the data objects have the ability to read and write their data using the methods *readit* and *writeit*. These methods accept a function of type *iofunc* (for Input/Output functions). I based everything on a line format, reading in a line of text, and writing out lines of text one line at a time.

By having the I/O set up using this type of function, I can easily have the data objects read or write using any new method I find, without having to modify the data objects themselves. Currently there are *iofuncs* to read and write from the TCP stream, and from a file on the server.

6.3 The Protocol

The connections between the client and server is accomplished using the TCP/IP streaming protocol. I chose to use straight text because protocols based on straight text are easier to debug. Straight text also makes writing and debugging new clients easier since you can read

the raw stream without having to interpret 1's and 0's. The command format is shown in the Appendix B.

6.4 Server

Out of the many different types of servers that I could have implemented, I considered two possibilities. The differences between the two are in terms of how many processes they have. Processes on the UNIX platform act like separate programs.

The first design option for a server has a single process which would accept requests from multiple clients. The main advantage would be that memory could be easily shared. If one of the clients modified an object then the server would only have to change a single copy in memory because it uses one copy for all of the clients. This would also save on disk accesses. The challenge is to make sure that each client receives an equal amount of time. This makes coding the server difficult.

I chose a second design option. This server has one main process that waits until a client connects, at which time it starts up a separate process to handle the client. A possible problem with this approach is that each process would have to communicate with each other process, in order to maintain data coherency. One way to get around this is to use a file locking protocol, and to have each process re-read the data objects when they need them.

The server has a main directory on the local file server. From this directory the server stores all of its files (see Figure 13).

Directory	**Description**
~/	The Parent Directory Set by the Administrator
~/conlisting	The Listing of all the Consultants
~/cons/	The Directory that contains the consultant's choices
~/templates/	Contains all the templates for the Help Desks
~/masters/	Contains all the master schedules

Figure 13: Directories on the server's file system with corresponding description.

6.5 Client

6.5.1 How it Works

The client I wrote used the Symantec THINK Class Library's (TCL) Application Framework. The TCL is organized into three distinct, interacting structures: the class hierarchy, the visual hierarchy, and the chain of command. The class hierarchy is the set of all the classes that make up the TCL. The visual hierarchy describes the organization of all visible entities. The chain of command specifies which objects get to handle commands.

The TCL converts Macintosh events, such as mouse clicks, activate events, and update events, into visual messages and direct commands. A visual message is an event that affects the visual hierarchy. A direct command is a request that an object perform an action. Direct commands are usually the result of menu commands [9].

TCL provides standard classes which implement a text field, buttons, windows, scroll bars, popup menus, and tables. TCL also contains a base class for a application (object *CApplication*). Each application displays and manages each window type by creating an instance of *CDocument*.

One of the *CDocument* sub-classes I created was *CChoiceDoc*, which implements **Consultant Preferences** (Section 5.4.3). It creates a window the controls that belong to the window: text box fields for the minimum and maximum number of hours, popup menus to select the Help Desk, and the table matrix to show the ranking of the hours. It does not have to deal directly with the operation of any specific control, since each control handles its own key presses and mouse clicks. *CChoiceDoc* only handles the transfer of information to and from the server.

The other windows were created in the same manner. For further detail on the specific implementation, view the source code in the Athena file system directory /afs/net.mit.edu/users/regis/thesis/client/src/.

6.5.2 The Data Objects

For the client application, I used the same data object source code from the server with two simple changes. The *GenList* object, which stores a modifiable listing of strings, was based on a storage mechanism that was found only on the UNIX side. I changed it to a corresponding object available in Symantec C++. The other change modified how the data objects dealt with strings, since the UNIX side deals with strings a bit differently than the Macintosh side.

6.5.3 Networking

On the client side I used MacTCP as the device driver to send and receive information from the server via the TCP protocol. Instead of using lower level device driver calls, I borrowed the *net_stuff* library from MIT Distributed Computing and Network Services. This library provided a higher level set of functions similar to the UNIX side (which made the porting of the code easier). I modified the functions by adding some better error reporting. I also took out spurious global and local variables. I added additional **iofuncs** (see Section 6.2.13), which give the Macintosh client the same functionality as the server: reading/writing from a file and from a TCP stream.

7. Results

At the time of this writing, the major functions of the server and client have been implemented and tested.

The server works according to the protocol defined earlier, and responds to all the commands except commands related to locking specific data objects, such as "LOCK TEMPLATE MCR" or "RELEASE TEMPLATE MCR." The server can send and receive all the data objects (template, consultants, consultant listing, and master schedules), though it only does simple error checking on the data it receives. The server has been successfully tested by sending and receiving several sample data files for each function of the server. At this time, the server uses the username only, rather than Kerberos, for authentication.

The client implements a subset of the original design. The client can send and receive information from the server. In the "Consultant Preferences" window, the ability to display another consultant's choices is not implemented. Some windows for auxiliary server functions are not implemented. For instance, even though the server has the functions to allow clients to modify the access control list, currently the client does not have that functionality. Also, many of the windows have a basic interface than the interface in the design; e.g., to show unavailable hours the "Consultant Preferences" window uses "N/A" instead of greyed-out boxes.

The server has been tested with two algorithms. First, the "simple" algorithm worked as follows: cycle through the consultants by rank, and assign 4 hours each cycle, using the consultant's ranked hours. This algorithm deals with multiple Help Desk templates, but does not give preference to consultants with larger blocks, nor does it deal with experiences and specialties. The second algorithm addressed these missing constraints, but was unable to keep proper track of the experience list for each hour.

8. Conclusions

8.1 What I Learned

I think the most significant lesson I learned was how complicated it is to take a process normally done without electronic means and move it to an electronic system. One must discover what the actual process entails and be able to determine which parts of the algorithm are most important. Besides the technical aspect, electronic systems encounter political situations as well. In this case, because a computer algorithm is replacing human judgment, students must give up some of the control they had when the process took place in the room. Before, a student could argue with other students, bribe them, or try to influence the administrators. The administrators, in order to keep the Help Desk balanced and take care of details, would sometimes influence students' decision. This cannot be done with the electronic version.

Looking back after completing the project, I realized that I had made some design decisions that I would now change. When I implemented the algorithm and then the client, it became apparent that it was too difficult to access information from certain objects. Many objects could be simplified and re-written with the hindsight knowledge about the entire process. For example, in order to extract the values assigned to a consultant, such as the experiences for each Help Desk, the method that must be used is:

```
currentCon = theConList->getCon(username);
hdprefs = currentCon->getHDPref(theTemplateName);
hdprefs->expr;  //  this is a list of the experiences
```

This process could be simplified if both the consultant object, *CCon*, and the consultants list object, *CConList*, were rewritten. The consultant's choices for every Help Desk are currently combined into one file and one object, *CConChoices*.. I would change the objects to store only one Help Desk choice. This would simplify the algorithm code, since at any time only the choices from a single Help Desk are used.

8.2 Future Directions

8.2.1 Server Robustness

There are many directions and improvements that could be made to the scheduling system. One of the ways that the system can be improved is in terms of the robustness of the server, *i.e.* making it a professional system. One the issues that I have not addressed fully is what happens when the server crashes. After a crash, it is important to ensure that all data files were left in a state that permits them to be recovered and verified when the database comes up the next time. One of the ways of doing this would be to maintain lists of transactions. For example, the client would say, "Add this user," and when the server actually responds, "USER ADDED," then the client knows for sure that whatever happens that the user would have been added to the server. That interaction would be one transaction.

One of the other aspects of the software that could use improvement is the trust level of the server. Currently the server trusts that the client will behave in a normal fashion, and

therefore it does not validate the data. It does not check for spaces or extraneous characters in the file names, nor does the server check that the users choice of hours are actually open. The server does not cross reference the validity between different data objects. For example, if a consultant was assigned a specialty that did not exist, the server would not catch the error. The server should be written so that it does not trust any information the client sends, and does thorough error checking.

Also, the speed of the server could be improved. One of the easiest ways to improve it would be to rewrite the general list, *GenList*. Rewriting this single object would improve many of the functions of the server, since many other objects rely on *GenList*. Currently, *GenList* performs a linear search for a string through its list of strings. This means it has to look at most strings every time that it does a search. One way to improve this would be to use a hash table such that for a given key, the program would immediately be able to find, with a high success factor, the desired string . Another way to improve the overall speed would be to do profiling on the server to determine the functions in which the server spends the most time.

8.2.2 Client Improvements

A future direction for the client would be to port it to hardware platforms other the Macintosh. This could include a Motif application for the Athena Computing Environment, a Windows client, and possibly even a text based interface for UNIX. The largest problem with such ports would be finding general objects to handle the windowing, matrix display, and text editing functionality. The actual user interface code is not very portable, because I took advantage of the Macintosh operating system and the Symantec C++ GUI libraries. I believe that ported applications should always look like a native application instead of a rough port.

The client could also use many of the features found in professional programs. This could include implementation of cutting, pasting, and printing in every window. The user interface could be given a general overhaul, possibly taking advantage of the drag and drop

feature of the Macintosh operating system. Many of these features were left out because of time constraints.

Ideally, all the network operations would be asynchronous. Currently the user must wait while there is activity between the client and server. Asynchronous operation would, e.g., allow one to modify the Help Desk Template while the listing of consultants was being transferred across the network. Asynchronous code was not implemented because the difficulty level increases by many orders of magnitude. The easiest way to implement the asynchronous operation would be to use *threading*. With threading, an application can have more than one loop of code executing independently. So, one thread could wait to receive information from the server while others handled the user's other requests.

8.2.3 Algorithm Enhancements

The algorithm is one of the more interesting aspects of the scheduling system. Unfortunately, I needed the rest of the scheduling system to experiment with the algorithm. The architecture is designed so that new algorithm objects could easily be created and added. If I had more time, this would be where I would spend the majority of my time.

One of the other algorithms I researched was linear programming. [11] This involves setting up constraint functions and then using a minimize function. Rather than performing a simple linear assignment of hours, the minimize function searches through the space of possible schedules to arrive at one with the least possible deviation from the given constraints. The result is a nearly-optimal schedule. One problem with this approach is that it would cause the consultants to lose their guaranteed right to seniority, since the consultants would not be assigned hours in the strict ordering, but rather in a best fit ordering. Since that seniority has been such an important part of the process, changing it would require a major political change. [10] Nevertheless, there is a linear programming C++ library that could be incorporated into Schedream in the future.

8.2.4 Other Design Improvements

Since this was my first attempt at an entire architecture and system based on C++, and also my first real experience with C++, there are many aspects of C++ I did not take advantage of. Many of the objects could be better designed.

In looking back, many of the functions I wrote are similar in functionality to a database program. Using a general database library would probably provide more reliability, and would also permit easier extension of the system.

9. Acknowledgments

First off, I would like to thank the people who kept me going when I was not moving: Tim O'Malley, Mike Wiese, and Mike Kobb. Thanks to everyone who made comments: Carla Fermann, Debi Fuchs, and Joanne Larrabee. Of course thanks to the actual organization of Consulting and the MCR/OLC Groups. Thanks to Jim Bruce for actually advising my thesis, and for being in charge of such a cool organization as Information Systems. Thanks to Mark Bonchek and Alex Jackl for all of the coaching around my thesis and graduation, and thanks to Landmark Education Corporation for the technology which made the coaching possible. Special thanks to those who helped me save my wrists by typing for me: Chris Horton, Jeff Gold, Brian Judd, Paul Kirby, Dan Sheldon, Josie Ammer, and Richard Perkins.

Appendix

A. File Structures

Notes on the file structures:

\n (Carriage Return) is assumed at the end of a line
... stands for repeating
% means that it is not part of the file, just a comment about the file

A.1 CTemplate File Structure.

```
HD Name   % The name of the Template
MAXBLOCK  % integer for the maximum block size allowed
exper,exper,exper  % a list of experiences for this Template
spec,spec,spec   % a list of the specialties
{hourname}:#low:#high:exper,exper:spec,spec  % this is a CTemplateHour
...
END
```

A.2 CAuth File Structure

```
username {authlevel}
...     % the authlevel is an integer where,
...     %       1 = Normal Access, 2 = Admin Access
...
END
```

A.3 CConChoices File Structure

```
username
HD Name:max:min      % The Help Desk Template name, and the min/max hours
{hourname}:#block:#rank % the hours consultant has a rank for,
...                   %        and the number of hours in that block
...
ENDHD   % no more hours for that Help Desk
HD Name:max:min     % repeats once for each HelpDesk
...
ENDHD
END
```

B. Client/Server Protocol

The server sends back errors in the following manner.

-ERROR {ERR #} {TEXT}

Commands:

GENSCHED - Used to generate a schedule for a particular HD w/ a certain algorithm with the given options

+GENSCHED {MASTERNAME} {TEMPLATENAME} {ALGO #} {OPTION1} {OPTION2} ...
-STARTING
-OKAY

-ERROR ?? Not Authorized to Generate a Schedule
-ERROR ?? HD does not exist

GETMASTER - Used to get the most recent master scheduled that has been generated/updated for a particular HD.

+GETMASTER {HD}
-SCHED {DATE} {username} % when and who last touched the master
-{SCHEDULE STRUCTURE}
-OKAY

-ERROR ?? No Master

SETMASTER - Used to set the master for a particular HD.

+SETMASTER {HD}
-SENDIT % server could send an error here
+{SCHEDULE STRUCTURE}
-OKAY % server could send an error here

-ERROR ?? HD does not exist
-ERROR ?? Formatting problem with Structure
-ERROR ?? Not authorized.

DELMASTER - Deletes the previous Master that was there)

+DELMASTER {HD}
-OKAY

-ERROR ?? Not authorized.

GETSTATS - Used to get various statistics about a generated schedule.

+GETSTATS {HD}
-{STATS STRUCTURE}
-OKAY

-ERROR ?? HD does not exist
-ERROR ?? Not authorized.

GETHDHOURS - Gets the Template for the HD's hours (which hours are available and how many consultants are needed each hour)

+GETHDHOURS {HD}
-{HD Template}
-OKAY

-ERROR ?? HD does not exist

SETHDHOURS - Sets the Template for the HD's hours (which hours are available and how many consultants are needed each hour)

+SETHDHOURS {HD}
-SENDIT
+{HD Template}
-OKAY

-ERROR ?? HD does not exist % or should it just create the HD?
-ERROR ?? Bad formatting in HD Template
-ERROR ?? Not authorized.

GETCONCHOICES - Gets a consultant's choice of hours preferred

+GETCONCHOICES {username}
-{Con Choices}

-ERROR ?? Con does not have choices yet.
-ERROR ?? Not authorized.

SETCONCHOICES - Sets a consultant's choice of hours preferred
It deletes any previous preferences for that student.

+SETCONCHOICES
-SENDIT
-{Con Choices}
-OKAY

-ERROR ?? Con does not exist (doesn't currently check to see if con exists)
-ERROR ?? Con Choices has invalid format.
-ERROR ?? Problems in opening the file.
-ERROR ?? Not authorized.

GETHDLIST - Gets the listing of the known Helplines

+GETHDLIST
-{# of HD}
-{HD Name}
... repeat until done
-OKAY

GETCONLIST - Gets the listing of All the Students known by the system

+GETCONLIST
{# of Consultants}
{username and entire ConPref}
... repeat until done
END

- If there are no Cons in the List, it will just send a 0.

SETCONLIST - Gets the listing of All the Students known by the system

+SETCONLIST
-SENDIT
{# of Consultants}
{username and entire ConPref}
... repeat until done
END
-OKAY

-ERROR ?? Not authorized. (errNotAuthorized)
-ERROR ?? The Con List was invalid. (errInvalidConList)

GOODBYE - The client is finished with the connection.

+GOODBYE
-CYA!

% The server now closes the connection

Bibliography

[1] Thomas Cormen, Charles Leiserson, and Ronald Rivest. *Introduction to Algorithms.* MIT Press, 1992.

[2] Thomas Palka. 6.100 Independent Project Laboratory. September 20, 1993.

[3] G. H. Gonnet. *Handbook of Algorithms and Data Structures.* Addison-Wesley, 1984.

[4] Douglas E. Comer. *Internetworking with TCP/IP Volume I: Principles, Protocols, and Architecture.* Prentice-Hall, 1991.

[5] Kurt Mehlhorn. *Sorting and Searching,* volume 1 of *Data Structures and Algorithms.* Springer-Verlag, 1984.

[6] Donald E. Knuth. *Fundamental Algorithms,* volume 1 of *The Art of Computer Programming.* Addison-Wesley, 1968. Second edition, 1973.

[7] Bjarne Stroustrup. *The C++ Programming Language, Second Edition.*

[8] Interviews with the MIT OLC and MCR Help Desk Student Consultants, conducted via email by Christopher Cotton, 1994.

[9] Philip Borenstein, et al. *Think C for Macintosh: THINK Class Library Guide.* Symantec Corporation, 1993.

[10] Interview with the MIT Information System Computing Support Services Group Full Time Consultants, by Christopher Cotton, 1994.

[11] Gregory, John W. <jwg@cray.com> (1994) "Linear Programming FAQ", Usenet sci.answers. Available via anonymous ftp from rtfm.mit.edu in /pub/usenet/sci.answers/linear-programming-faq