BEAR Advisor
March 10, 2000 Overall Project Design

1 Document Overview

1.1 Introduction

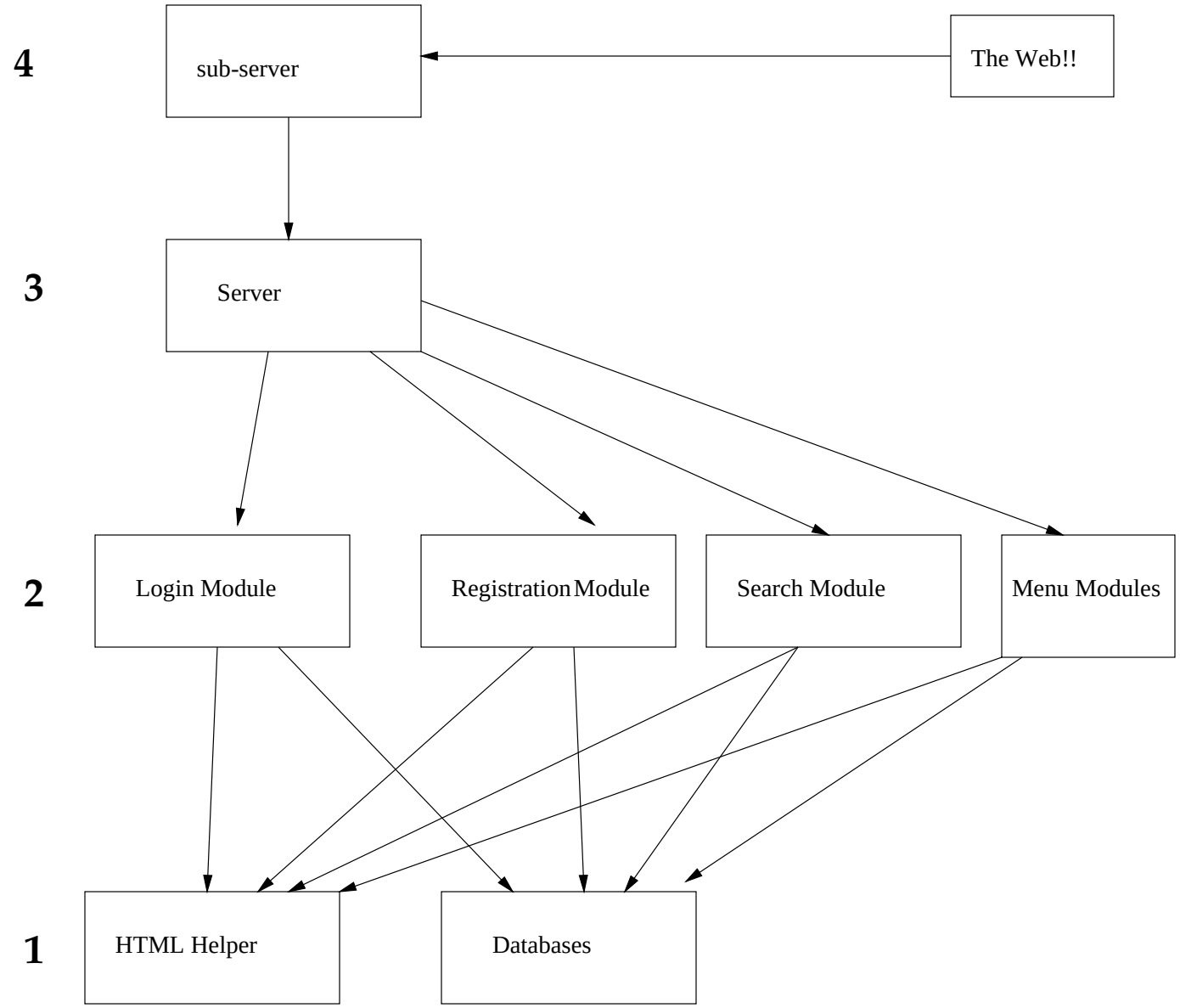
This document is the Overall Design for the BEAR project. BEAR Advisor which stands for **B**rown **E**lectronic **A**dvisor and access to the textbfRegister. This project aims to make the course registration process easier by providing students with "smart" tools for searching for classes, as well as allowing them to register for classes online. The following document will outline the top-level design for this project. It will explain the modules that make up the design. Then it will discuss the team members involved in the project and their roles and tasks. This document will outline a schedule and critical path for the project with milestones.

1.2 Disclaimer

This document is a more detailed design of BEAR then the top-level designs individually turned in previously. It is important to note that the team has only just begun to work together so that the design is just begining to be more in depth than the last document. We are begining to start thinking about our interfaces so the method names you see below and the estimated needs from other users are only begininers. The schedule on this document may need to be changed slightly next week with changes in design, but after that it should be frozen.

2 Components Overview

Levels



* Note * On the level 2 modules are the main modules that must be implemented. There are more modules on this level included in the design their interaction with the server, HTML Helper and the database will be the same as the above.

2.1 User Interfacee/Server

The user interface is all - through html and the input through CGI scripts which is parsed in the server and makes the appropriate methods calls with data to the specific modules. The sub-server will handle all the input. So that we can have multiple clients, it is almost

acting as a gateway to a persistent server. This will prevent us from having to create a new instance of the server for each user logged on.

2.2 Login Module

The login module handles all maters related to security and authentication. It verifies user credentials through the kerberos server, and sets up a secure connection to the user. Apon login the user will be given a random one-time session id. The module will send back the appropriate html page based on the succesfullness of the users login and the users status. This module will be designed to use kerberos, but if by March 24th we have found that kerberos will be too difficult to implement a simplier security system must be used instead because of time constrants.

- login(NetId, Password)
  I am assuming that this function gets called once the NetId and password get retrieved from the Html page via some CGI scripting.
  This function then corresponds with the Kerberos Server. It sends the user information i.e the login and password to the Server.
  Return Values
  - If the server is not responsing.
  - If the user information is incorrect
  - If the user login is sucessful
- In all cases, it needs to make some function call to the HTML Helper

- pingServer(), This function calls will ping the Kerberos Server to see if it up and running
  Return Values: If server is alive, If server is down.
  Following are a list of interfaces that other modules might need to call the login module for.
  - int activeLogins(), This function returns the number of users currently logged into the system.
  - int getUserInfo

2.3 Registration/Written Permission Module

The Course Registration Component allows the user to view and change course registration information. This includes courses, grade options and status of written permission requests.

- showRegistrationFor(studentID, semester)
  - DB query: all classes with a given student ID and semester

  - figure out whether this info can be modified (e.g. whether it's before the last day to drop classes for this semester)
  - construct appropriate HTML page (either read-only for past semesters or with fields to add/drop/change grade option for current semester) addClasses(studentID, semester, list of class nums, list of grade options corresponding to those classes)
  - for each class in the list, add to the database
  - DB write: student ID, semester, course number, grade option, written permission flag

- dropClasses(studentID, semester, list of class nums)
  - DB query: all classes with given student ID and semester, to verify that student is enrolled in the classes they're trying to drop
  - DB write: delete rows from database

- changeGradeOption(studentID, semester, list of class nums, list of grade options)
  - DB query: all classes with given student ID and semester, to verify that student is enrolled in the class they're trying to change grade option of
  - DB write: change appropriate row in DB to reflect new grade option

- showProfClasses(professorID, semester)
  - DB query: all classes taught by that prof in that semester
  - return HTML page that links to showPermissionRequests(.)

- showPermissionRequests(class number, semester)
  - DB query: all rows of the registration DB that match class num and semester AND have "permission requested" flag set and "permission filed" flag NOT set
  - return HTML page that has an entry for each student returned

- setPermission(class num, semester, boolean for granted/denied, list of names) for each name in the list:
  - DB query: get the row corresponding to this class, semester, student combination
  - DB write: set permission filed flag as appropriate

2.4 Search Module

The following is a list of priorities for this component:

2.4.1 Top Priorities

- Basic search by user commands (like a file cabinet)
- SearchCourse(department, course, time, prof, overallRating)
- SearchRequirementsForConcentration(concentration)

2.4.2 Mid Priorities

Priority search: allow user to prioritize top 3 departments, times, etc.

2.4.3 Low Priorities

- boolean SearchAvailableTimesOnly -¿ need to know current registration

  Graduation search: take into account date of graduation and concentration need to know current registration, courses passed, concentration requirements

2.4.4 Course Review

- AddStudentReview(studentID, course, year, concentrator?, overallRating,review)
- ModifyProfReview(profID, course, review)
- ViewReview(department, course, prof, overallRating)

2.5 Grades Module

The Grades module allows the user to view grade reports from previous semesters and order transcripts. This component has database access and grade information will be read from the database by this component. If a student requests a transcript a email will be sent to the register(eventually it would be nice if it automatically deducted the fee from the students account and sent the transcript in the mail to the appropriate location).
Subfunctions performed by the Grades module:

- View Grades for student X, from semester Y.
- View Grades for student X, from all semesters at Brown.
- View transcript for student X.
- Order transcript for student X.
- View Grades for student X, from "last semester". (shortcut)

2.6 HTML Helper

The job HTML Helper is to handle the building of the HTML pages so that it doesn't have to be done by the modules. The HTML Helper recieves data from a module and a HTML template and returns a complete HTML page ready for the server to display.

The following is a general sketch of the functions that will be available in the html parsing language. This is not a finalized spec, but this is close to how it will look.

[IF <data>] <Html block> [ENDIF]
If ¡data¿ exists, insert the given html block, otherwise don't.

[*<data>]
Insert the requested data from the data block.

The data block is organized as follows:

class Data {
string Name;
string Info;
list<Data> Children;
}

Data blocks can be organized in any form, allowing accesses like this from within html templates:

D.studentname
D.homeaddress.street
D.classes.1.professor

The interface to the module will consist of a single function call, which take an html template and a data block, and returns a finished html page.

2.7 Declare Concentration

This module allows the user to declare their concentration. The module will have the forms for the user to fill in. When they complete their forms a flag will be tagged in the database. When their advisor logs in the database will look through all the advisors advissees and check to see if any need the Concentration accepted.

2.8 Admin - add/remove

This module will add/remove users and add/remove courses. This module would only be used by admin. This module is not part of the basic requirements, but will be picked-up by a programmer if there is time.

2.9 myRegister

Way off in the distant future we would like to make it so the user can change the look and feel of the user interface. Most simply we would like the user to choose their own colors

for the UI. This would require another section in the database to be saved with the users information.

2.10 Database

As we know by now, everything we want to do will result in a database query of some sort. When we communicate with the database we want to do three things:
1) Retrieve existing data
2) Change existing data
3) Enter new data

In order to do this it's important that the correct data is modified, hence some kind of unique identifiers (KEYS) must be agreed upon. For people it's easy – all students, faculty and staff have a nine character ID number (which for students begin with SIS). In the case of courses it is important to note that a unique course requires a course number and a section number. Since I suggest that we use SIS numbers for identifying users it will be important that whenever a query require student information that number is provided by the modules. Upon successful login the database will return the SIS number, which then will be used by the system to retrieve other information.

In the case of student information (including their concentration filing, and course registrations) the modules will provide the student's SIS Number and a list of fields (see list below) that they want from the database. For instance, say that you want to know a student's grades and majors you'd do something like: getSystemUserInfo(SISD99999, stuff); where stuff maybe is an array containing GRADES, MAJOR_I, MAJOR_II or something like that. Retrieving course information would be done in a similar manner. I hope you get the rough idea.

Changing information is a bit more complicated. I imagine that you'd pass in a key with some kind of structs which have fields matching the fileds in the relevant table an update needs to be done. Care must be taken for null fields or fields changing from containing data to no data.

New data I think would be done similar to changing data, but there's no problem worrying about corrupting existing fields.

Here are some functions that I anticipate should be in the interface. There are probably many functions missing at this point, but these are some that I can think of right off the bat. Note that these are just very rough sketches and will be changed a lot after your input, more thought, etc.

- getSystemUserInfo(SISId, fields)
- getCourseInfo(Course#, Section#, fields)
- searchCourses(user specified fields)

- getCourseReviews(Course#)
- addSystemUser(struct containing the fields defined below)
- addStudent(another struct)
- add....etc.
- editSystemUser(some struct)
- edit...etc.

I've tried to come up with a exhaustive list of required data for the whole project. I'm sure I might have forgotten something, so please read through the list carefully. The sections are some rough drafts on tables in the database, which is the reason why I for instance have left out students' names in the Student information section – it can be found in the System User Information Section. Once the table design have been finalized they can be created easily and very quickly.

SYSTEM USER INFORMATION

- Name (First, Middle, Last)
- SIS Number
- Net_Id
- Permission level (student, faculty, or admin)
- Password

COURSE INFORMATION

- Number Section number
- Name
- Description
- Instructor
- Instructor's SIS Number
- Department
- Meeting time
- Final examination time
- Written permission required
- S/NC required

STUDENT INFORMATION

- SIS number
- Student's e-mail address
- Student account number
- Home address (street, city, state, zip, country)

- Campus address (box #)
- Parents' address (street, city, state, zip, country)
- Home phone
- Campus phone
- Parents' phone
- Major I
- Major II
- Degree (A.B., Sc.B., or both)
- Expected graduation date

COURSE REVIEWS

- Course number
- Reviewer's SIS Number
- Course ratings
- Student review

COURSE REGISTRATION

- SIS number
- Course number
- Course section
- Grade option
- Grade
- Permission granted/denied
- Semseter course taken

CONCENTRATION INFORMATION

- SIS Number
- Concentration advisor
- Concentration advisor's e-mail address
- Courses student intend to take
- Essays motivating the concentration
- Pending approval (yes/no)
- Approved/denied/talk to dean

HTML PAGE REPOSITORY

- Page ID
- Page

3 External Dependencies

- Database Information, database server
- Critical review information
- Kerbros Server
- Web Support
- Outside Help : tstaff, knowledgable security people

4 Testing Strategy

4.1 Module

Make testing as automated as possible.

- Code Checks
- Test with dummy data
- Test as part of each milestone, not just when module is finished.
- No integration until you have completely tested and have gone through your testing procedures with Emily and declared ready for integration.

4.2 Intergration

- Break testing down into subparts as much as possible.
- Integrate only one module at a time.
- At each integration point go through testing procedures again and have Emily(tester) test it before another module can be integrated and it can be declared working.

4.3 User

- **Testing accuracy of information**. The information provided to the user should be compared to the data in the system's database and, if possible, similar data available from other sources. For example, the grades given to a user via the online system could be compared to the user's actual grades as recorded on their official transcript.
- **Testing reliability.** A large number of users can try to access the system simultaneously, to see how well it performs under a heavy load.
- **Testing ease of use.** Ask a group of potential target users to try to complete a series of tasks (example: pre-register for classes w, x, y, and z, then drop class z and add CS190, then change the grade option of class x to S/NC, etc.) and see if they can do it without needing assistance. Ask the users for feedback on how the system could be easier to use.

- **Testing speed.** Try to connect to the system with a 28.8 modem to see how quickly pages load.
- **Testing compatibility.** Connect to the system with a variety of different Web browsers to ensure that pages look right in each of them.

5 Culture Section

5.1 Keeping the Spirit Up!

To make sure the spirit of the group is kept up and everyone is aware what is going on we will do the following things:

- When a programmer has reached their milestone they will email the group.
- When a programmer checks something in they will email the group.
- Each team member must take at least one night a week away from this project. - most likely you reach the friday milestone and take friday night off.

5.2 Outside Activities

To keep group morale up and make sure we have connections together outside cs190 we are planning a couple of outside activities. Since Lisa is on the lacrosse team and Joe is on crew we plan to go to their home matches. Lisa's first one is March 18th and Joe's is in early April. This way we can really get into both the cs190 spirit by rooting for our team members and Brown spirit!

6 Task Breakdown

- **Project Leader/Spirit Leader**, Joe Fuqua: The Project leader is the final word on all inter-component decisions. The purpose of this is not to stifle democratic group decision making but instead to make sure the group effectively makes decisions rather than spends too much time in the decision making process. Being sensitive to all group members' needs, strengths, and weaknesses, the leader is able to keep the group on schedule, provide motivation, and keep up group morale. The leader is also responsible for providing resources (e.g. himself, someone else, online documents, etc.) and handling/directing group members' questions. In addition, the leader is responsible for administrative needs and recruiting user testers.ss
- **Architect/Lead Tester**, Emily Leventhal: Responsible for the design of whole components. The architect will be in close communication with the leader, especially discussing progress or asking for help. This person will update the design when their are changes and work with the project leader for delegation for roles. When new designs are suggested by team members the design master will review them and decide whether they can be included. At times the design master may need to call a team meeting or a meeting of small groups to go over changes in design. Before coding this person

should make sure that everyone in the group understands the overall design and how things flow. Emily will also be in charge of testing for the whole program. The tester will help programmers step throught their code and check for bugs as well as test code at both the module and integration stage. The tester is also responsible for testing the final program on users.

- **Lead Programmer**, Lisa Cozzens: Lisa will intially be setting up the enviorment for coding. She will become familiar with and set up CVS. As well as create the intially make files. Her job will include working with the architect to come up with the interfaces. Finally she will also code the registration module. Lisa is also in charge of the professor admin module(entering grades, entering course descriptions, and professor information).
- **Database Manager**, Rikard Grafstoem: Rikard will set up the database and the database manager that allows the modules to interact with the database.
- **Documentation**, David Yun: Responsible for program documentation. Each programmer will keep an updated documentation of their piece of the program and design decisions they have made. The editor will review these pieces of documentation and give feedback to the team members. Finally the editor will make one united document from all the team members documentation. Also the editor will write the user manual and help sections of the program, because they should understand from the team members documentation how to use each component.
- **User Interface**, Brian Chuck: Creating the user interfaces in html and the templates. Also will create the cgi scripts to get the data from the page and put the data on the page - basically the server that mediates between the web and the modules.
- **Programmer**, Melissa Cheng: Melissa is in charge of the entire search module. Above in the modules section is a priority list of what parts of the stages she will write the module in. Since the number of possible additions is so large, Melissa will work on it and add functionality as time permits. She will also be creating a place for students to add new course reviews and search past course reviews.
- **Programmer** , Brett Heath-Wlaz: Brett will create the specifications for the parsing language and the class that will create help create the html pages. As time permits Brett will also work on the Grades module and the module myRegister.
- **Programmer**, Neelu Bedi: The login module and security of the whole program. Neelu is also in charge of creating a basic menus, adding and deleteing users and adding and deleting courses.

7 What Can I do to keep the project moving?

The following is a list of tasks group members are currently working on before we start interfaces and coding.

- Joe Fuqua: Keeping the project moving!

- Emily Leventhal: Finalizing the design. Working with the programmers to create the interfaces. Starting to test the UI and coming up with testing strategies to be used later.
- Lisa Cozzens: Setting up cvs and the makefiles. Working with the programmers to create the interfaces.
- Rikard Grafstoem: Researching databases, setting up dummy data and creating a server(getting the space) for the data.
- David Yun: Keeping up with the documentation. Attending meetings and keeping the current information up to date on the webpage.
- Brian Chuck: Creating drawings of the UI design and testing preliminaries on users. Researching cgi and learning anything that he is not familiar with yet.
- Melissa Cheng:Creating a priority list and researching how we can get the current information in BOCA.
- Brett Heath-Wlaz: Defining information types and setting up a language file. Will work with davey to create this file and emily(tester) to make sure everything is covered.
- Neelu Bedi: Researching security. How we are going to pass information back and forth. Talking to tstaff and other outsiders about encryption/decryption methods.

8 Development Schedule

2/25(f): Top-level Design Proposals
3/3(f): Initial Group Meeting in class
3/5(s): Group Meeting to finalize design
3/8(w): Group Meeting to finalize group roles + schedule
Milestone 3/10(f):
Final Top-level Design
3/13(m): Initial Interface Proposals and HTML specs finished
3/14(t): Basic UI Drawings ready to test.
3/15(w): Interface Proposals
3/20(m): Testing for intial UI completed, Interface Comments,

Milestone -3/24(f):

- Detailed Designs finished.
- Header files written.
- Database - set up with dummy data
- UI tested - visually
- Good docs on interfaces
- HTML Helper coded, testing began on one template.

: Final Interface Definition, HTML Helper finished.
4/5(w): Detailed Designs

Milestone 4/7(f)

- Login - Given a username and password can validate without a security, started security, and can encode session ids.
- Server/UI - All UI designed, CGI designed, Done initial visual UI testing and gotten feedback. Started on CGI for login and registrations. Can output registration.
- Database - Database up and running with all tables, interface done, .H done and partial implementation done of .C - making a actual connection to database. Can work with login data and do a search for class by number.
- Search - can search by one basic criteria
- HTML Helper - robust testing, works for registration template.
- Documentation - Good internal documents of the UI, good docs on the detailed designs.

Milestone 4/14(f):

- Login -Security Implemented, Kerberos if possible.
- Server/UI - All HTML templates and CGI scripts coded. Be able to check session Id storage.
- Database - All types of quieries should be supported.
- Search - All Top prorities completed.
- HTML Helper - Shoul be complete, tested and ready to add new templates.
- Documentation - Begin user Manual support: Login, Registration, serach, database structure - requirements to change.
- Testing - All module testing done and checked with Emily.

4/17(m): Initial System Integration

Milestone 4/21:
Most integration should have happened by this point. During this week if modules are completed programmers can move on to next module in their list.

- Login - Should be fully functional, integrated and working.
- Server/UI - Complete and currently being tested - integration and module.
- Database - Major Testing!!
- Search - Integrated with database.
- HTML Helper - Integrated and should be working.
- Documentation - Add sections to user manual.
- Testing - Test all integrations.

Milestone 4/28(f):
Functionality Freeze, Full System Implementation
Begin user testing early in the week and try to make changes immeadiately.
In week between 4/21-4/28 All new features added must be approved by both the architect and project lead.

Milestone 5/12(f):
During this week we will finish user testing. Documentation - All documentation completed - internal and user.
All requirements met by this time. 9am: Final Documentation/System Submission