BEAR + Virtual Academic Planner

Design Overview

I will give a quick outline of the key points in my design. It is in these highlighted places I imagine there are several design choices, each of which will affect the application to a large degree.

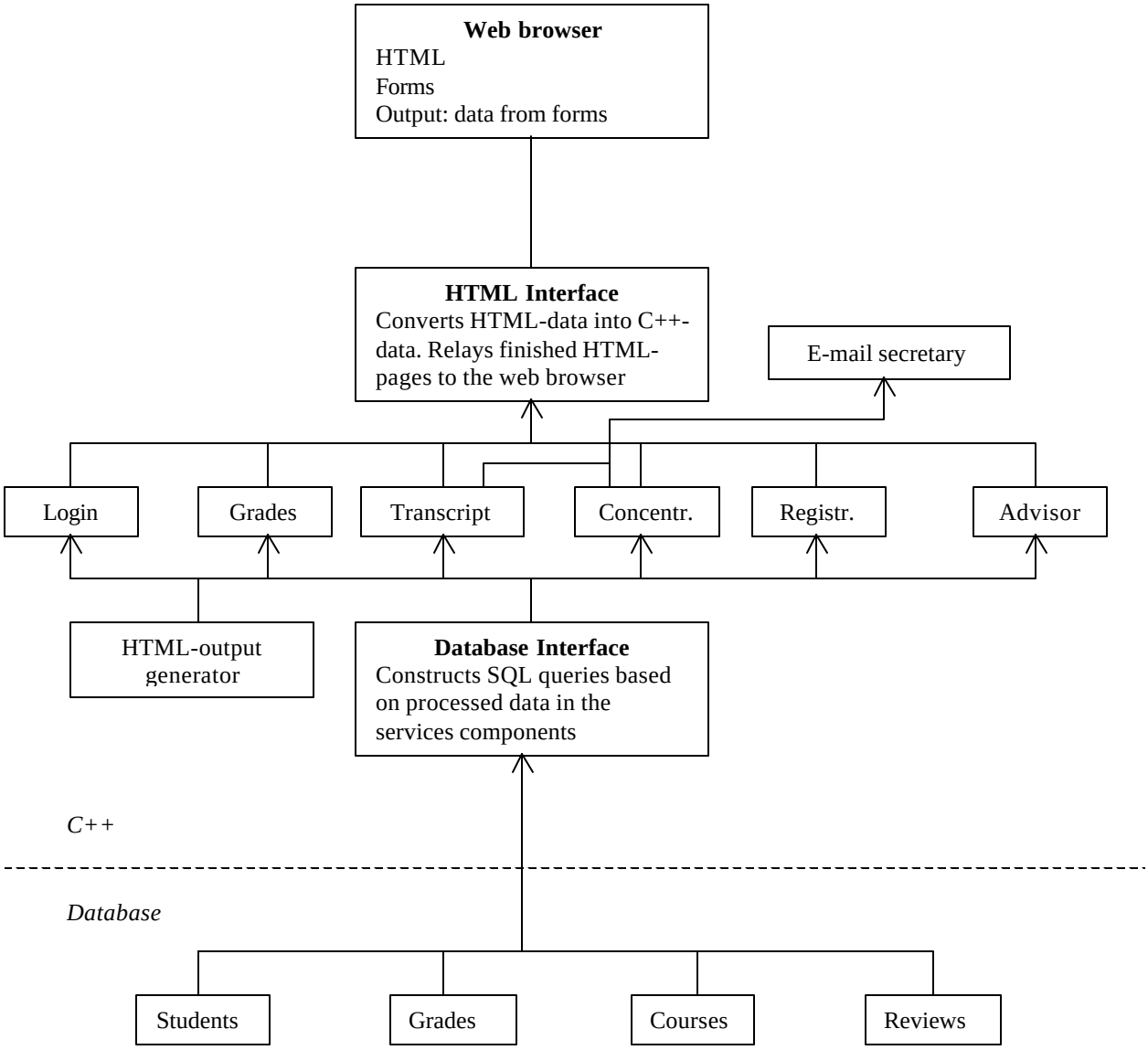
The BEAR and Virtual Academic Planner (BEAR, for short) should be designed as a database application. All requests that are made through the system will require database access and/or manipulation. Hence the reliability and extensibility of the system is heavily dependent on wise database design. Tables should be constructed such that redundant data can be avoided without loss of speed. Since there is a clear trade-off between the two, some redundancy might have to be accepted to allow fast access to the database over a slow network connection. The database system will use SQL, which implies that we will have to construct SQL queries from the input we receive from the user, to properly access the database tables.

Having said that, BEAR also has a large and dynamic user interface. In particular, it depends to a large degree on the output from the database queries. The input forms are fairly static, which serves as parameters for proper access of the database tables. What both input and output UI have in common is that they are in HTML. Some kind of translation of HTML data forms into C++ parameters and vice versa will be necessary.

I imagine the Virtual Academic Planner is a module of BEAR just like the ones originally set out. From now on I will call the modules “services”, which is a better representation of what they actually are. It will therefore be a selection from the user main menu, just like all other services.

High-Level Component Diagram

The diagram gives an intuitive overview of how the system parts are related to each other. Each component will require a higher level of detail, but it shouldn’t affect the top-level design, since each component is very isolated from one another. The services box represents each and everyone of the modules outlined in the specs. The purpose of the parser (I couldn’t come up with a better name for it), is to pass the information of the HTML forms from the user level to the correct service, which process the information, and validates it, before passing it to the SQL-interface which constructs appropriate SQL queries for the database. The databases’ output will be parsed by the HTML-output generator and then proper HTML pages will be constructed for the user to view. In addition there’s a Secretary which works on the side to notify relevant party about action that needs to be manually taken: Produce transcripts, approval of concentration, etc.


Web browser
HTML
Forms
Output: data from forms
HTML Interface
Converts HTML-data into C++-data. Relays finished HTML-pages to the web browser
E-mail secretary
Login
Grades
Transcript
Concentr.
Registr.
Advisor
HTML-output generator
Database Interface
Constructs SQL queries based on processed data in the services components
C++
Database
Students
Grades
Courses
Reviews

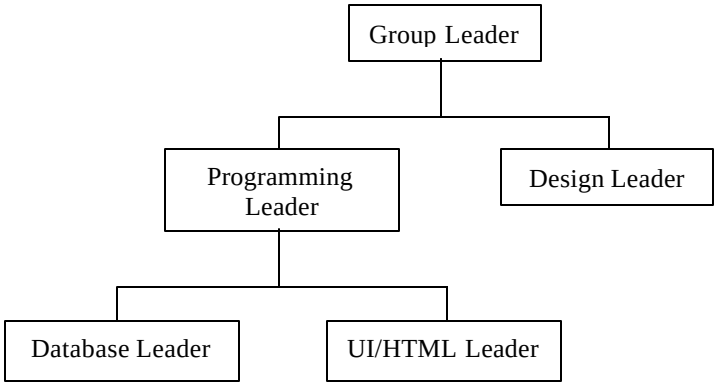
Task Breakdown

The projects breaks down into three distinct parts, which is illustrated in the diagram above. There will be work needed on the user level – HTML, form creation and a logical system for navigating the pages. A large chunk will be in C++ where data is analyzed, validated and processed. Finally, database design will be of crucial importance for the performance of BEAR.

Each of these three components can be worked on individually once low-level interfaces has been agreed upon. The two key interfaces ought to be the HTML and database interfaces. They are responsible for converting data from different languages. The modules can be worked on independently. User documentation for this application will probably not be extensive, as the application should be very easy for a user to use, and the web browser already come with a rich documentation on how to use it, and I anticipate that the biggest learning curve is getting familiar with the web browser. Presumably most users of BEAR have already used a web browser, hence it reduces the need for documentation further.

Group Organization

The primary organization of the group will be as follows. It is intentionally left small at this point to allow for flexible allocation of resources where they will be needed. The group leader will mostly be concerned with inter-personal issues, make sure that the project progresses in a timely manner. He or she will also get involved in simpler programming tasks. The design leader has the over-all responsibility for creating a solid design, and specifying the interfaces. Sine they are crucial for the entire project, it is important that a single person has a great understanding of how the data will flow from one part of the program to another. The programming leader is overseeing the coding part of the project. He or she will put programmers on specific tasks, and the programming leader understands where the challenging parts implementing-wise of the project is. The database leader will, in conjunction with any assistant database programmers be responsible for building efficient database tables. The HTML leader will be designing an efficient page hierarchy.



Schedule

3/10: Final design overview including specific top-level interfaces defined.
3/17: Low-level interfaces agreed upon with specific functions, other low-level interfaces complete
3/24: Interfaces complete, programming in progress of all parts of the application
4/17: Integration of all components. Bugs of individual parts should to a large degree be completed, bugs in interfaces will be worked on.
4/24: Users can start test the application and report any bugs found. No new major functionality
4/28: Integration complete. Any bugs hereafter found are fixed cautiously
5/3: Presentation

New Assumptions

The specs were pretty clear on most points and I can't think of any additional assumptions I made.