WSS08.8

Design and Architecture

Overview

This document will show the issues that arose in designing the projected called WSS08, a working title for Waikato Semantic Searcher 2008. The document is split into two key areas, the Graphical User Interface, and the Back End System. The two areas have their own priorities that were taken into consideration when the decisions were made. Each of the key areas are also split further into different areas that need considering. Each area may have a number of issues, and each issue has a number of alternative options that could be implemented, and then the decision that was made based on these options.

Index

The Front End

The purpose of this section is to show the decision making process that went on in designing the front end of the program. This section has a further three specialized sub sections, the GUI, the Search System and the Communications System.

Priorities	
1	The program needs to be usable.
2	The program needs to be reliable.
3	The program needs to be aesthetically pleasing.
4	The program needs to be as responsive as possible.
5	The learning curve for using the GUI should be minimal.

The Front End: Graphical User Interface

This section is about the GUI without including information about how the results will be displayed (that is to be discussed in a different section).

Issue 1: *When a user has selected a result that they want to view, how will that result be displayed?*

Option 1.1: Display the result in an external browser.	
Pros	***Cons***
Easier to let external browser deal with the processing of the pages.	Having multiple windows is something that would be nice to avoid.
The user will be familiar with using that browser	This will interrupt the users use of the program by taking them away to an external application.

Option 1.2: Display the results internally (using Gecko or something similar).	
Pros	***Cons***
Continuous experience for the user.	May be difficult to get working properly.
May look and feel better.	

Decision: *Option 1.2*
This has been chosen because we want the user to have a continuous experience without external programs being opened. This way the user will not have any distractions from the core program.

Issue 2: *How will the user use the GUI to provide feedback on the search? They need some way to tell the program about results which are positive, negative, or not interested in.*

Option 2.1: Attach a traffic light type bar to each node so the user can click green for positive, red for negative, and yellow for not interested to give feedback on that node.	
Pros	***Cons***
Should be reasonably easy to use.	Screen will be cluttered and won't look good (against priority 3)
Should be reasonably easy to make.	

Option 2.1B: After thinking about the above the one, we thought that this system could work but only display the "traffic lights" when a node is selected.	
Pros	***Cons***
Less cluttered (in line with priority 3).	Learning curve will be larger (against priority 5).

Option 2.2: Dragging the nodes into regions which are positive or negative.	
Pros	***Cons***
Dragging will work easily with the nodes.	It will be easy to see what has been classified positive and negative.
	Users may not want excessive amounts of dragging.

Option 2.3: Combination of the above. The user will be able to drag nodes into regions. The user will also be able to right click on a node and be able to select a traffic light type option that will send the node to a region, or they will be able to select the "not interested" option.	
Pros	***Cons***
Less cluttered (in line with priority 3).	Slightly larger learning curve.
Still have the dragging option pros.	
Have the advantages of 3.1B.	

Decision: *Option 2.3*
This seems to get the best of all pros with small amounts of cons. It will probably be slightly more difficult to implement but it will work the best for the program.

Details of Design: Graphical User Interface

The Graphical User Interface will be created using Java Swing components such as JPanels, JButtons and JToolBars, e.t.c. This will allow the program to be flexible with its display on different operating systems.

The GUI will create seeding nodes by allowing the user to input text and then selecting the appropriate node which is brought up as a list of nodes that are in the database which is accessed through the communications. This will be done so that the user is only selecting from nodes that are going to be included in the database.

The GUI will determine whether the seeding node is “good” or “bad” depending on the area that the seeding nodes are put into. This set will be passed to the communications part of the program for processing.

So, the GUI interacts with the communications part of the program to send queries, and then the result display part of the program will be used to display the results on the GUI (both of these areas are discussed in a different part of this document).

When a result is selected to be viewed, the GUI will display this using a page renderer.

The Front End: Result Display System

This part of the program is responsible for displaying the results on the GUI.

Issue 1: *When the user has submitted a query, how will the results be displayed?*

Option 1.1: Display the results in a list (so that it would look similar to a basic google search).	
Pros	***Cons***
This would be simple for us to program (in comparison with other options).	Hard to show the clustering easily.
The user should find it very simple to understand (in line with Priority 5)	It would not look as good (against Priority 3).
It would be quicker as only text would need to be displayed.	It would not show the relationships between the results.

Option 1.2: Display the results as nodes shown as circles, with the nodes displayed in groups with links according to various search parameters and relationships.	
Pros	***Cons***
This would be aesthetically pleasing (in line with Priority 3).	The learning curve for using the program would be larger than alternative options (against Priority 5).
It would work better with the dragging system used for feedback (see further down in this document).	It will be harder to program.
It would show the relationships between the nodes which would be nice for the user to see.	It will be slower (against Priority 4).
It would some what reflect what actually is going on behind the scenes.	It may not be that easy to use with ART.

Option 1.3: Display the results in a combination of the both, with the general idea that it will look like an fi (like Windows Explorer) and nodes will be folders and the folders will contain lists of nodes/results.	
Pros	***Cons***
It would be better than just showing lists.	May not be as good for browsing the results
It would be organized and categorized.	Harder to program than simple lists.
Users will be familiar with the browsing system.	
The learning curve will be small (in line with Priority 5).	

Decision: *Option 1.2.*
With the general idea that the system will be easily adaptable to support an arbitrary number of different views and search techniques through the use of a plug-in system. This seems like the best way for the user to use the program, whilst getting the best user experience.

Issue 2: *How should the "uninteresting" result case be processed?*

Option 2.1: Remove the results from the display without remembering that result.	
Pros	***Cons***
Simple.	If the search is modified, the program will not remember that this has been displayed, and this result may be displayed again.
Uses less memory.	

Option 2.2: Put the uninteresting results into their own database, and do not bring them up again into the display.	
Pros	***Cons***
Program will remember these uninteresting results.	Will use a third database.
	Will use more memory.

Decision: *Option 2.2.*
This will be used as it is important that these uninteresting results do not come back up. The cons that come with this will be out weighed by the benefits of this feature.

Details of Design: Result Display System

This part of the program will deal with how the results will be displayed on the GUI. This will be implemented so that a plug-in type system can be used, and the way that the results will be displayed will be up to the user to decide on.

It will be passed the results from the communications part of the front end, and then will have to render these on the GUI in the way that is selected.

The two initial ways that will be used will be a list form or a node display.

The node display will rely on node classes that will contain information (such as the images) for how the nodes themselves should be displayed. The back end will have to return the results in a form that represents the clusters.

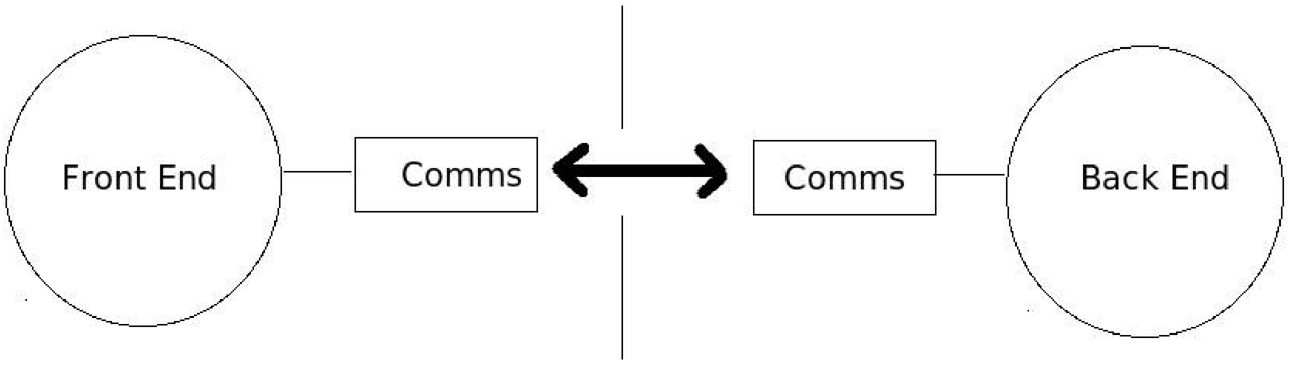
The list display simply shows the top results that are returned by the Art search.

The Front End: Communications

Details of Design

The front end communications will rely on the protocol that are listed in the back end communications section.

The image below shows the area of the program this is dealing with.



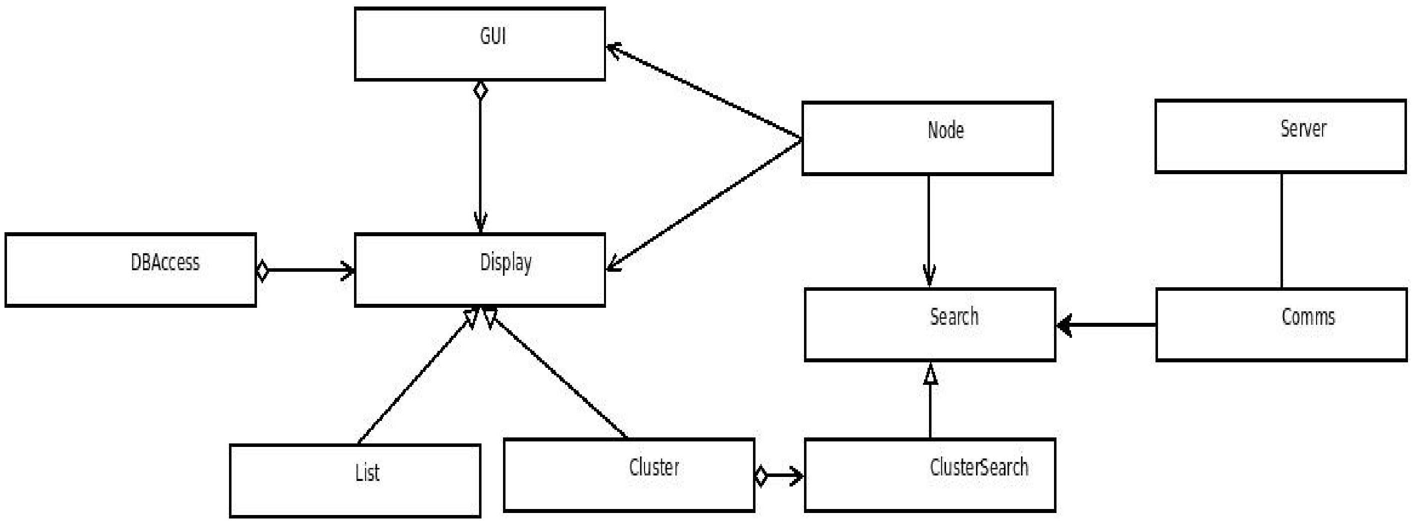
This part of the program will have to take the queries that are entered into the GUI, and send these to the server. It will then have to take the results that are returned by the server, and pass these to result display part of the front end.

Details of Design for the Front End: General

The front end will be responsible for getting and building the queries from the user and sending these to the server, and then displaying the results that the server returns. The front end will have to be able to communicate with the server.

“Uninteresting” result cases will be stored in a database that contains a link between query nodes and the nodes that are the uninteresting results. When the displaying is being done the nodes that are returned will be checked against this database and won’t be displayed if the uninteresting connections come up.

Below is a basic UML diagram for the front end [needs updating]:



The Back End System

The purpose of this section is to show the decision making process that went on in designing the back end of the program. This is split into two sub sections, the server, and the pre processing tools.

Priorities	
1	Needs to comply with use cases and requirements.
2	Needs to be able to be achieved in the time given.
3	Needs to not be overly difficult to implement.

The Back End: General

Issue 1: *Should we use a client/server model or should the back end system be rolled together as part of the front end.*

Option 1.1: Incorporate the back end mechanics as part of the front end	
Pros	***Cons***
Less reliant on a network connection	More complicated software requiring a higher degree of threading.
	User will need to install a large database of information
	Harder to ship updates to users.

Option 1.2: Split the system into a server/client model, where the server does all the searching.	
Pros	***Cons***
Less complicated threading required.	Reliant on a network connection.
User does not require much space on their hard drive.	
Easy to install database updates onto the server without having to deploy them to clients.	
Clear separation between GUI and back end system, making team development of the two systems concurrently easier.	

Decision: *Option 1.2*
This option seems to be the best as there only has to be one copy of wikipedia on the server, saving a lot of space for clients. It will also be better for the client as it will not have to do the pre processing of the wikipedia pages.

Issue 2: *Should we calculate the semantic relationships on the fly dynamically, or should we pre-process them into a database?*

Option 2.1: Calculate semantic relationships on the fly dynamically from source material such as Wikipedia and WordNet.	
Pros	***Cons***
Does not require a database system.	Could be extremely slow processing all the information.
Does not require the large amounts of space needed to store the database.	May not be able to capture all the relationships within an acceptable amount of time.

Option 2.2: Calculate the semantic relationships ahead of time using our own "in-house" tools designed for the job, and store them in a database.	
Pros	***Cons***
Easy to update the semantic relationship calculation technology used without having to update the rest of the software.	Reliant on a database system.
No waiting for software to compute the relationships dynamically.	Requires a large amount of space to store the semantic relationships.
Can analyse the source information for a much longer period of time, and in many passes if need be.	
Gives a list of nodes (documents/words) for which the user can use to seed the query.	

Decision: *Option 2.2*
This should speed up the process of performing a search quite dramatically. The links that are created should be able to be more complex due to having more time to process the information.

Issue 3: *In what format should we store the semantic relationship information, given that we are pre-computing it ahead of time?*

Option 3.1: Use GraphXML	
Pros	***Cons***
A standard format for storing graph structures, which can be read from other applications.	Would be extremely slow processing all the information. Would have to scan the entire file.
Easy to parse XML files.	File will be very large; slow to read all.
Simple structure.	

Option 3.2: Use a SQLite database	
Pros	***Cons***
Some of the Power of SQL but without relying on an SQL server which is external to the software.	Does not support all of the functionality of a enterprise grade SQL server.
Indexed records means fast lookup times within the database.	Possible issues with thread/process concurrency. SQLite does not appear to have good locking mechanics to protect against multiple processes modifying the database.

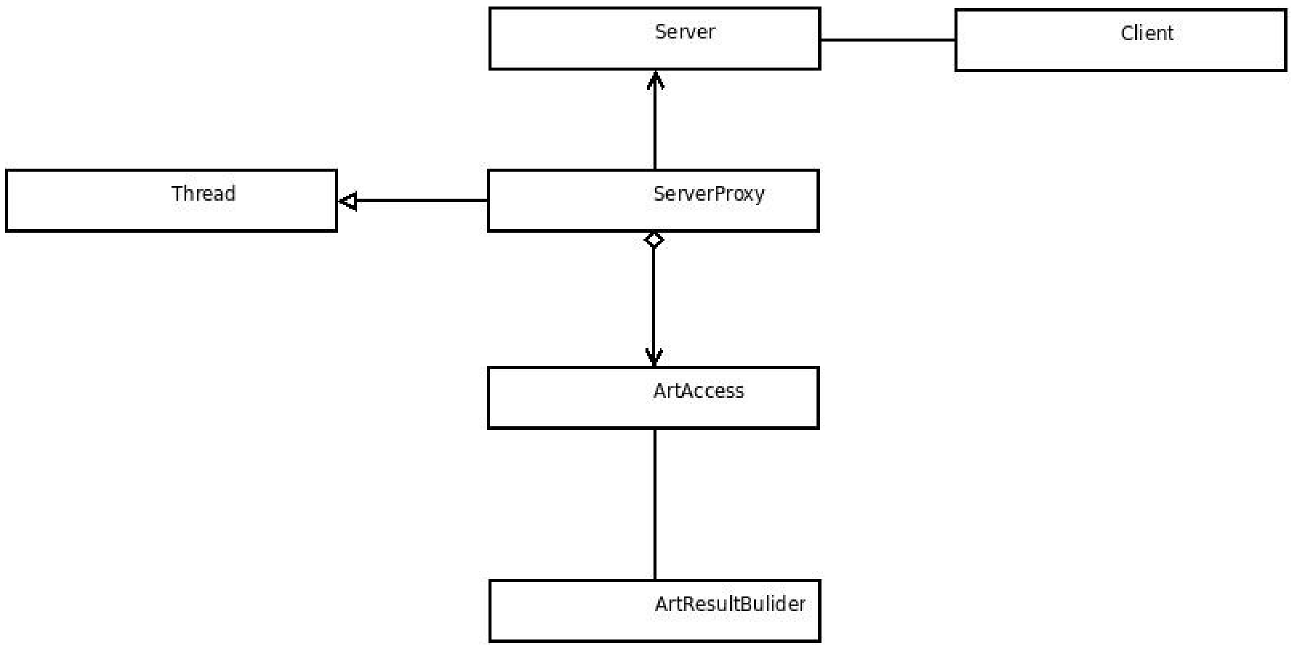
Option 3.3: Use a MySQL database.	
Pros	***Cons***
More powerful SQL functionality than SQLite	Requires an external SQL server.
Fast lookup times	
No concurrency issues which cannot be easily solved using mechanics provided by MySQL.	

Decision: *Option 3.3*
This is because it will work the best for what we are trying to achieve. It has the potential to be more flexible.

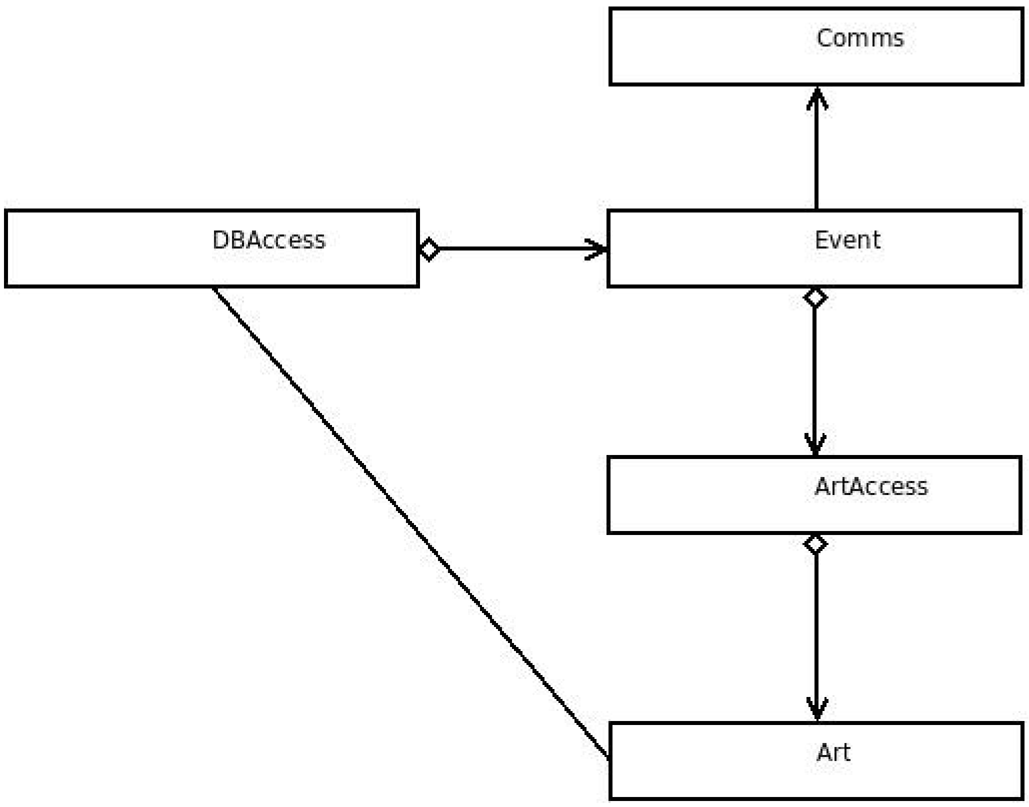
Details of Design for the Back End: Server

For the back end, Art will perform a crucial rule in being used to link nodes and returning results based on these links. The server will rely on databases being built for it, and will have to be able to communicate with the client.

The following shows a diagram that illustrates how the server will interact with the client and the part of the program that will actually build the results [diagram needs to be updated; it is incorrect].



The following shows a UML diagram for the part of the program that will build the results using the databases that are created from elsewhere in the program.



This page shows the protocols that will be used with the communications parts of the programs.

Client/Server Protocol – General

Code	Description
1XX	Client commands
2XX	Server responses
3XX	Misc. commands
4XX	Error commands

Client/Server Protocol – Client commands

Code	Format	Description	Example	Valid Responses
100	100 <Type>\<...>	Searches for nodes	100 Wikipedia\com	200, 300, 301
101	101 [<NodeKey>\<Weight>]+	Preforms a query	101 19456\1.0 23567\0.0	302, 400
102	102 NEXT	Request for next query result	102 NEXT	300, 301
103	103 “Name” [<NodeKey>\<Weight>]+	Save a query.	103 “Fluffy Cats” 19456\1.0 23567\0.0	302, 401

Client/Server Protocol – Server responses

Code	Format	Description	Example	Valid Responses
200	200 <# results>	Begin list of nodes	200 73	

Client/Server Protocol – Misc.

Code	Format	Description	Example	Valid Responses
300	300 <Key>\<Type>\<Title>	Specifies a node	300 19456\Wikipedia\Short_Haired	
301	301 DONE	Last operation is complete	301 DONE	
302	302 OK	Last operation is accepted		

Client/Server Protocol – Errors

Code	Format	Description	Example	Valid Responses
400	400 <Key>	Specifies a node does not exist	300 19456	N/A
401	401 <Name>	Name specified is used	401 “Fluffy Cats”	N/A
410	410 <Peers version>	Client/Server version incompat.	410 WSS08.8C	Disconnect
411	411	Last command was invalid	411	N/A

The Back End: Pre Processing Tools

For the database builder there will be a JWPL Parser and Jaws wrappers for getting access to Wikipedia and WordNet respectively. These will help to build the databases that contain all the semantic relationships.

There will be two databases, one table being the node table which will contain:
- primary key
- type of node
- title
- access date
- creation date

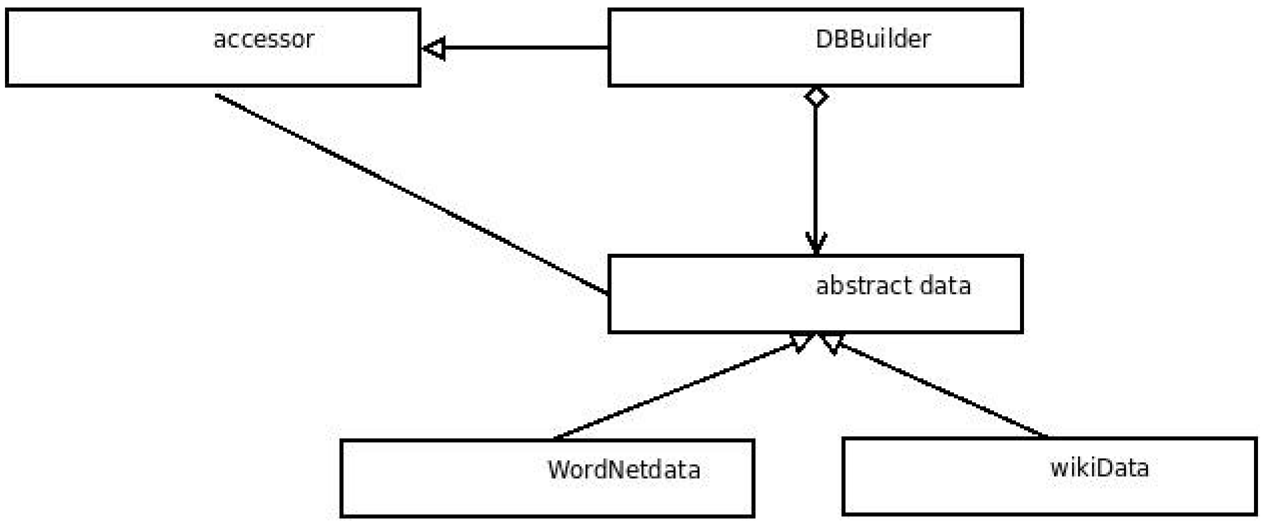
The other being the link table which will contain:
- primary key
- node 1
- node 2
- weight of link

For words, links will be created by looking for synonyms and other words that are related. This can be done using Jaws.

For wikipedia, links will be created by examining the outgoing links, the incoming links, and the content of the articles.

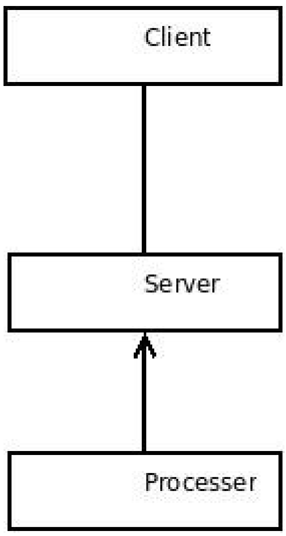
Details of Design for the Back End: Pre Processing Tools

The following diagram shows a fairly basic diagram for the part of the program that will build the databases that will be used for constructing the results from.



Details of Design: General

The basic model of our program will be a Client-Server model. This will allow the program to be split up into several key areas. These are the front end (GUI) as one area, the back end, and the database builder. As shown in the UML diagram blow.


Client
Server
Processer