Web Based Survey Development System

A Manuscript

Submitted to

The Department of Computer Science

And the Faculty of the

University of Wisconsin-La Crosse

La Crosse. Wisconsin

By

Kalon Eichman

In partial fulfillment of the

Requirements for the Degree of

Master of Software Engineering

August 2010

Web Based Survey Development System

By: Kalon S. Eichman

We recommend acceptance of this manuscript in partial fulfillment of this candidate's requirements for the degree of Master of Software Engineering in Computer Science. The candidate has completed the oral examination requirement of the capstone project for the degree.

_____	_____
Dr. Tom Gendreau Examination Committee Chairperson	Date
_____	_____
Dr. Kenny Hunt Examination Committee Member	Date
_____	_____
Dr. Kasi Periyasamy Examination Committee Member	Date

ABSTRACT

Eichman Kalon S., "Web Based Survey Development System," Master of Software Engineering, August of 2010, Advisor: Dr. Tom Gendreau.

The Web Based Survey Development System is an application for Saint Mary's University faculty and students of Winona MN. This application replaces the paper survey with a computerized process that allows authors the ability to create surveys, define target populations, distribute surveys, and analyze survey results. This application allows the target population the ability to remotely access and respond to the created surveys. The surveys and populations defined in this application may be reused; a survey can be made available to multiple populations; or the survey can be made available to one population multiple times.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to my advisor, Dr. Tom Gendreau, for his guidance and advice in developing my application. I would like to thank the project sponsor, Francis Speck, Computer Center Director for Saint Mary's University, for guiding me in setting up and organizing our project development meetings. I would like to thank the three Saint Mary's University teachers who volunteered and contributed so much to the design of my project. I would like to thank my family for all their support and for putting up with me while I developed this application. Finally, I would especially like to thank my Dad, who doesn't understand computer programming and doesn't want to, who, when I showed him the proposal for my project, said "that is science fiction", even though he believed that I would accomplish my goal.

TABLE OF CONTENTS

LIST OF FIGURES

APPENDIX A

APPENDIX B

GLOSSARY

Ajax

A programming technique that changes static web page(s), or web site into a dynamic web application

Author

A person who uses the application to create surveys, publish surveys, and retrieve and analyze survey responses

Advertising

A recording of the surveys which have been published and made available to a target population

Client

An overall description of the people who participate in using this application in one way or another such as “authors” and “users”

Content

The text and associated formatting for a given question in a given survey

CSS

Cascading Stile Sheets, a programming language that is used to change the look of HTML

GUI

Graphical User Interface, the part of the application where clients interact

HTML

Hyper Text Markup Language, the programming language that forms the elements of a given web page

Interface

A standardized boundary between the layers of an application which allows it to be more adaptable

JavaScript

A language that empowers web pages to be interactive and dynamic

Module

A subcomponent of an application that is self-contained

Option Group Format

Defines the way in which survey question response options are displayed

Oracle

The database server software used in this application

PHP

A powerful server side programming language that can also be used to preload web pages with data

Population Information

Known data which describes a target population

Response

A user's answer on a given survey question

Report

A record of survey response data that can be analyzed and displayed

Server

The application core which validates and filters all data transferred between the interface and data storage

Session

A data value that keeps track of a user's interactions with a survey

SMU

Saint Mary's University of Winona Minnesota

Survey

A container for a set of questions and response choices

Survey Page

A sub container used to break up long surveys for display purposes so that survey takers are not overwhelmed by the number of displayed questions

Survey Taker

A user who responds to the questions in a given survey

Target Population

A container used to group together users with common traits

Web Page

A portion of the application interface made up of HTML, JavaScript and PHP

Web Site

A collection of web pages which forms the interface for an application

XML

A programming language mainly used for data storage and transfer

1. INTRODUCTION

The purpose of this project is to create a Web Based Survey Development System that would be available to the entire population of SMU. This application will automate some of the uses that SMU has for surveys such as research projects and teacher evaluations. It will give students and faculty the ability to do such projects without consuming paper resources. SMU would like to reduce the amount of paper they use, they would rather do surveys electronically. The system will have three main parts: survey development (for creating surveys), survey access (for filling out surveys), and survey processing (for tabulating and displaying survey results).

SMU primarily uses the Windows operating system in their computer labs, although the dorm rooms are also wired for personal computers. Their Computer Center uses an Oracle database for their primary means of data storage, and this was the type of database to which the application was to connect. Because SMU has a lot of computers it was decided that this application be web based as it would be easier to distribute and update as the code would reside in one location and internet access and personal web browser settings would be the limiting factors.

2. LIFE CYCLE MODELS

The developer followed two life cycle models in producing this application, the prototyping model and the water fall model. Prototyping was used to gather the requirements for the application. After the requirements were agreed upon the waterfall model was followed during the design, implementation and testing phases of the application.

Prototyping is the process of creating a model of a target object. A prototype model isn't the complete application but it will help the development team get a sense of how the application will work. In each prototype revision more requirements are added. This process of developing and redeveloping an application model encourages brainstorming, the proposal of ideas and organizes the ultimate product. In other words, clients and developers converse and discover what is needed in the product, and explore ideas without committing resources and having to backtrack later.

The waterfall model describes the process of programming as a set of sequential steps, like cups overflowing with water, one into another. The process moves only one way, forward. The model defines the following steps: requirement gathering, design, implementation, testing, and maintenance. This model does allow for going back but only in a limited fashion if there is something that was missed at a previous level.

3. REQUIREMENTS

Over the course of three sessions, each spaced about two weeks apart the developer build up a model GUI. It was decided that the number of prototyping sessions would be three as that would be enough to get a good start on finding the requirements for the application without overloading the developer. The break between each session gave the developer time to rebuild the GUI model and refine the functionalities that had been previously discussed. During this process the types of activities which were desired in the application became understood. During these sessions possible additions were also discussed that could be added to the application once the first version was implemented. From the model the developer formed a good base understanding of what the project would entail. From the initial screen shots and meetings, the developer was able to produce the requirements document which contained 120 or so functionalities for the application. The following describes the requirements for this application.

There are two types of clients who will access this application, "authors," who create surveys and survey takers or "target population users," who respond to the surveys. Authors will be described first as they do more with the application. Once logged in, and registered with the system, an author can choose to work with one of the following: surveys, populations, advertising, or reports. The author might be reviewing, editing, creating or deleting their previous work as their chosen activity. The following is a brief description of how authors and users interact with this application.

To develop and use surveys is the goal of this application. In developing a survey an author has a large number of options for changing the look and feel of their surveys as well as being able to input survey content. In terms of formatting a survey an author can divide up questions into multiple screens or "pages". Authors can also set how the groupings of response choices are displayed and aligned to create a desired effect. The format options and response groupings that an author develops may be used for multiple surveys.

Besides survey development an author needs to be able to describe to whom a survey will be sent. To do this an author must define one or more target populations. A target population in this context is a structure used by the application to classify a set of potential survey takers. Depending on the use that the author wishes to make of the target population additional information may be provided, such as information describing the population, and possibly identifying the users to whom access to a survey will be granted. An author may also have an open population where survey takers are anonymous.

Once the surveys and populations are formed the author can publish the developed survey and make it available to a target population by specifying an advertising schema. Because both surveys and populations can be used multiple times it is necessary for the application to record the connection made by pairing a survey with a target population. Once an author has a survey developed he/she can keep the survey and use it repeatedly for researching thoughts and behaviors of one or more populations. For example an author may want to compare the response values of one target population with another or compare responses of a single population over time, such as at the start of a school term and then at the end of a school term. As for populations an author may need to evaluate their responses to a number of surveys so it makes sense for the author to have to define and identify the given population once.

After a survey has been published and user responses have been recorded the author may use the application to analyze and generate a survey report. This system organizes and displays response data based on the analysis type and response scope that the author has chosen. Survey taker responses are stored separately for each survey publication so that the data that is evaluated can be scaled (scoped) in multiple ways such as by survey, by survey and population, or by publication.

On the survey taker side of the application the survey taker has a login screen. After the survey taker has logged in the survey pops up and all the survey taker has to do is answer each question and navigate the survey pages.

4. DATABASE DEVELOPMENT

From the initial prototype screen shots and the requirement discussion meetings that the developer had with his sponsor one of the first designed components to the application was the database. The web based survey development system is a data oriented application, meaning the entire application is based on the data that it contains. The organization of the data in the application affects what functionalities are needed. In order to produce a firm base for the implementation of the application the developer decided to use Entity Relationship "ER" diagramming in order to understand the relationships between the entities, or tables, formed in the database. [See Figure 1]

Survey Development Database ER Diagram

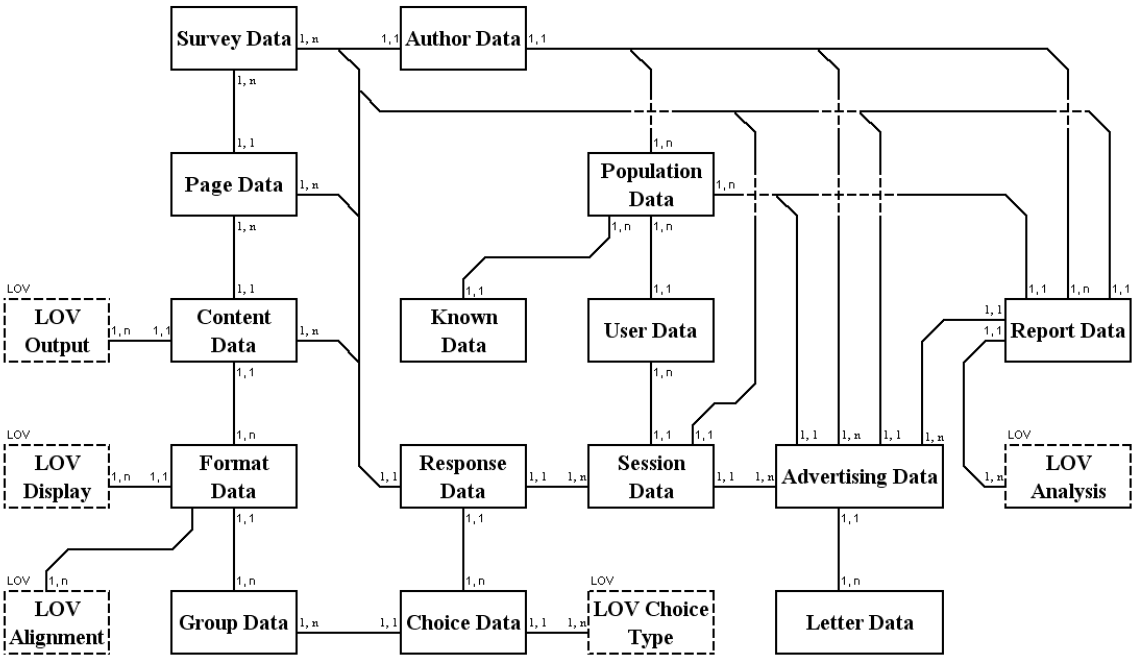

Figure 1: Simplified ER Diagram

What the developer learned from producing this diagram was that the resulting application would be complicated. There are six components to the application: author accounts, surveys, target populations, survey publishing, reports, and survey takers. Each part of this application will be discussed in turn following the hierarchical nature of which components must be defined first.

Author accounts [Author Data] are at the top of the hierarchy in this application. Once authors register with the system they will be able to form most of the entities that exist in this application: surveys [survey data], target populations [population data], survey publications [advertising], and reports [report data]. All entities the author creates will belong to that author and therefore will only be viable to that author when he/she logs into the application. This application allows for multiple authors each able to do independent work. If authors work together they would have to share one author account.

Next on the hierarchy of the application are surveys and populations. These two entities are independent of each other though they will be linked together later in the application. Surveys will be discussed first. An author may develop multiple surveys [survey data], one for each set of questions to which they need to research. Each survey is uniquely identified by the application. However for the author the identification is based on the name and purpose that the author gives. The application does not limit the author to having to specify unique names for each survey. This identification system is used so that authors may edit their work and rename things as needed. The structure of a survey contains: survey page [page data] and survey content [content data]. This data was organized in a hierarchical fashion which allows the author great flexibility in developing the look and feel of their survey. The option group formats [format data], option groups [group data] and choice values [choice data] of a survey are independent of the survey hierarchy so that they could be used on multiple surveys.

Target populations [population data] are the other independent type of component that an author can develop in the application. Target populations have the same identification structure as surveys (see above for details). When an author defines a target population they are identifying a grouping of potential survey takers [user data]

and may further describe the population [information data] by adding anecdotes about the population. When the author defines "user data" they are defining survey taker accounts through which survey takers will be able to access surveys that are sent to them.

Once survey(s) and target population(s) have been developed the author can define a survey publication [advertising data]. The separation of surveys and target populations is intentional so that data once entered into the system can be reused. This entity associates a given survey with a given population and makes the survey available to the population's survey takers. The initial design of the application would have an author send a cover letter [letter data] to the survey takers, in which a link to the survey would contain the survey individual information. Once the survey taker responses are retrieved the survey publication would somehow be disabled.

The last component that the author can define in the application is the survey report [report data]. This application is meant to automatically retrieve survey taker responses and then be able to display the response data to the author. The input that the author would have at this point would be to create the report in which the author specifies what data the application is expected to retrieve.

The one section of the application that the author can't control is the response retrieval mechanisms. This part of the application is meant to keep survey taker responses hidden from the survey author, such that the author would not be able to identify individual survey taker responses. The application is meant to allow survey takers [user data] the ability to access a survey one or more times [session data]. This number would be defined by the author. Each question response [response data] is individual recorded so that survey takers may change their responses if they access a survey multiple times or inputted the wrong initial response.

5. APPLICATION PRESENTATION

Before discussing how this application was built the developer first needs to describe how an author will use this application. This section focuses on the steps involved in producing a survey, defining a target population, making a survey available to that population, describing how a survey taker will access and respond to the survey, and how to form a report and view the results. The figure below [Figure 2] is the use case digram for the application. It describes the relationships between different parts of the application and how one navigates between each of the available services.

The ovals in this diagram represent the services or actions that a client (author or survey taker) may choose to do. The arrows indicate available choices. When two arrows come together this means that the entities created in each service must be defined before the new service can be fully utilized.

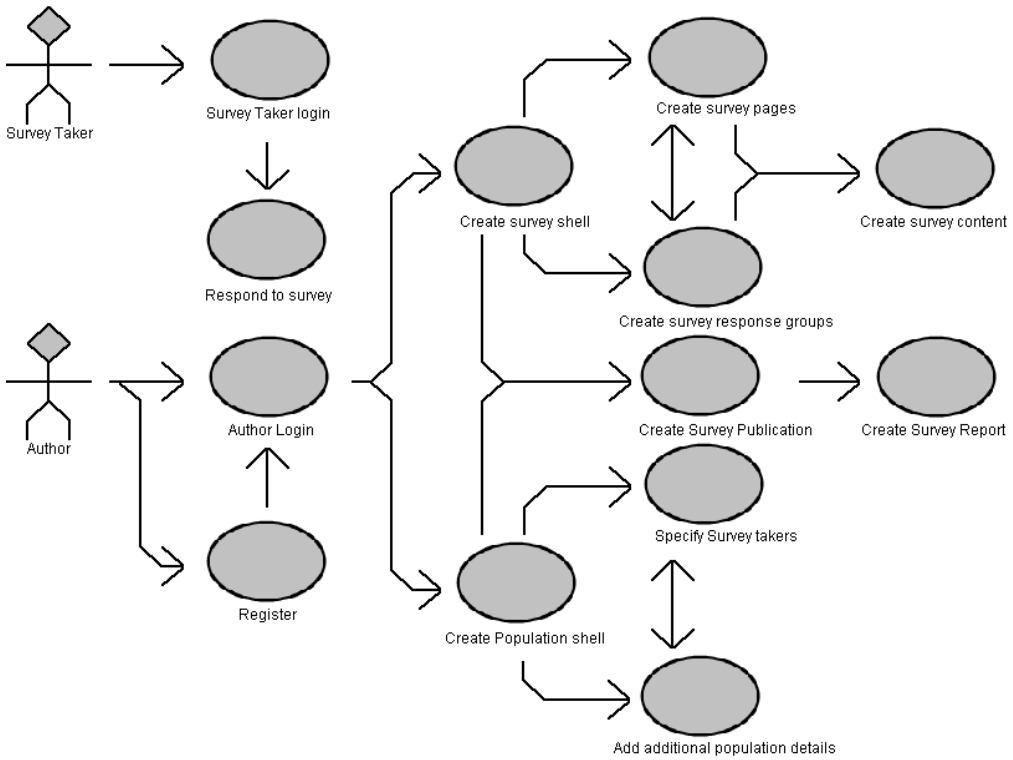

Figure 2: Use Case Diagram

In making surveys the author must first define a survey entity, provide it a name and a purpose to distinguish it from the other surveys that the author will produce. Once that's done the author needs to define the response options that will be chosen by the survey takers. Formats may be reused. To define the choices that a survey taker must choose from the author needs to create the following:

1. A choice group – a container for choice values. Each group is identified by the names provided.
2. Choices values – displayed options that can be chosen, such as "true", "false", 1, 10, "pepperoni", etc., depending on the type of group (a ranking, a toping list, etc.). Each choice requires a name and an index value. These values are ordered based on index values the author provides.
3. A format – describes how a group of choices is displayed. Provide a name, set the display type (select box, buttons, radio, checkbox), alignment type (horizontal or vertical), and choice group.

Once the choice options are created, the author needs to define the pages that will go into the survey. [Figure B.5] A survey can have multiple pages so that survey takers are not intimidated by the number of questions shown on each screen. Each page is ordered by the index the author provides and may contain a title if the author feels that the page deserves one. Only then can the author go on to work with survey content. [Figure B.9] All the author has to do is write out the questions, order them with index values, and set the appropriate format. Once the author has their questions imputed the survey is complete.

Next the author needs to define to whom the survey will be made available. [Figure B.10] This is a two step process where the author defines a target population entity (give it a name and a purpose) and then define the survey taker accounts for the users who will respond to the surveys. [Figure B.11] To define users' accounts (survey

taker accounts) the author must input a unique user ID (such as an email address) and a password. If at this point the author wants to further describe their target population they need to go to the known information page and input the questions and answers which describe the population.

After the survey(s) have been developed and the population(s) has been defined the author may proceed to the advertising portion of the application where surveys are made available to target populations. To publish a survey to a target population the author must input a name for the advertisement, provide a purpose and then set the pairing of survey and target population from which the author desires results.

In order to get results from the published surveys there has to be a point at which a survey taker (user) accesses the survey and responds to the questions proposed. On the survey taker login screen the survey taker selects the appropriate advertisement (by name) and then inputs his/her ID and password. The login screen will take the survey taker to the appropriate survey, and all the survey taker has to do is click on his/her response choices and the responses are recorded.

To view the survey results the author must define a report. A report is the application's way of determining which data values to analyze and return to the author. To create a report the author has to give the report a name so the author can identify it, and then specify one of the following types of reports [Figure 3] by selecting: a survey, or both a survey and a population, or an advertising schema as well as the analysis type. Once the report information has been recorded the author may then bring up the report analysis results.

Type:	Inputs:	Description:	Contains:
A	Survey	All responses ever made using a particular survey.	All user Input (all populations and all advertising sessions)
B	Survey and Population	A population is a school class or an age group that has been targeted. This type of report retrieves responses that the target population has made to a particular survey.	Population Inputs from all Advertising sessions.
C	Advertising	Advertising defines an occupation or reason for survey to be used. This type of report retrieves only results for that publication.	Advertising session data only (survey and population are paired by the advertisement)

Figure 3: Report types

Figure 3 shows the report types available in the application. Type "A" reports include every response ever made for a particular survey. Type "B" reports include all of the responses that a given population has made to the given survey. Type "C" reports include only those responses associated with a particular survey publication.

Suppose an author forms a student evaluation survey for Professor X and his CS 101 and CS 102 classes. Professor X is evaluated by both classes six times per year, which means there are twelve total publications of the survey, six to each class. When evaluating Professors X's total performance we produce a type "A" report which gives a composite evaluation of all input. To evaluate Professor X's performance for class 101 the author will produce a type "B" report. If it is necessary to view individual evaluations the author could produce type "C" reports.

6. DESIGN AND ARCHITECTURE

In order to produce the application components, the developer had to determine how data will flow between the database and the GUI. From the prototype model the developer was able to generate the initial database tables, initial class diagram [Figure A.3], and requirements document. The requirements document is a listing of functionalities that must be implemented in the application. The database tables describe the content of the application. In order to ensure that the process of implementing the application would go smoothly there were several design and structural issues that had to be addressed.

First, as a web based application it was important that it be responsive to client inputs and not be a series of static web pages. To solve this issue the developer decided to use Ajax. Ajax is not a programming language but rather a technique that makes web pages dynamic. Ajax improves the performance of web page interactions by changing the dynamics of message passing between the server and the GUI. An Ajax application has all of the same parts as a regular web application except that the web pages can pass data between the GUI and the server asynchronously. This ability to preload existing data and dynamically change the displayed data makes a website into something very similar to the feel of a desktop application. [See Figure 4 (below) for the structure of Ajax and figures A.1 and A.2 for images describing the effect of Ajax on web applications.]

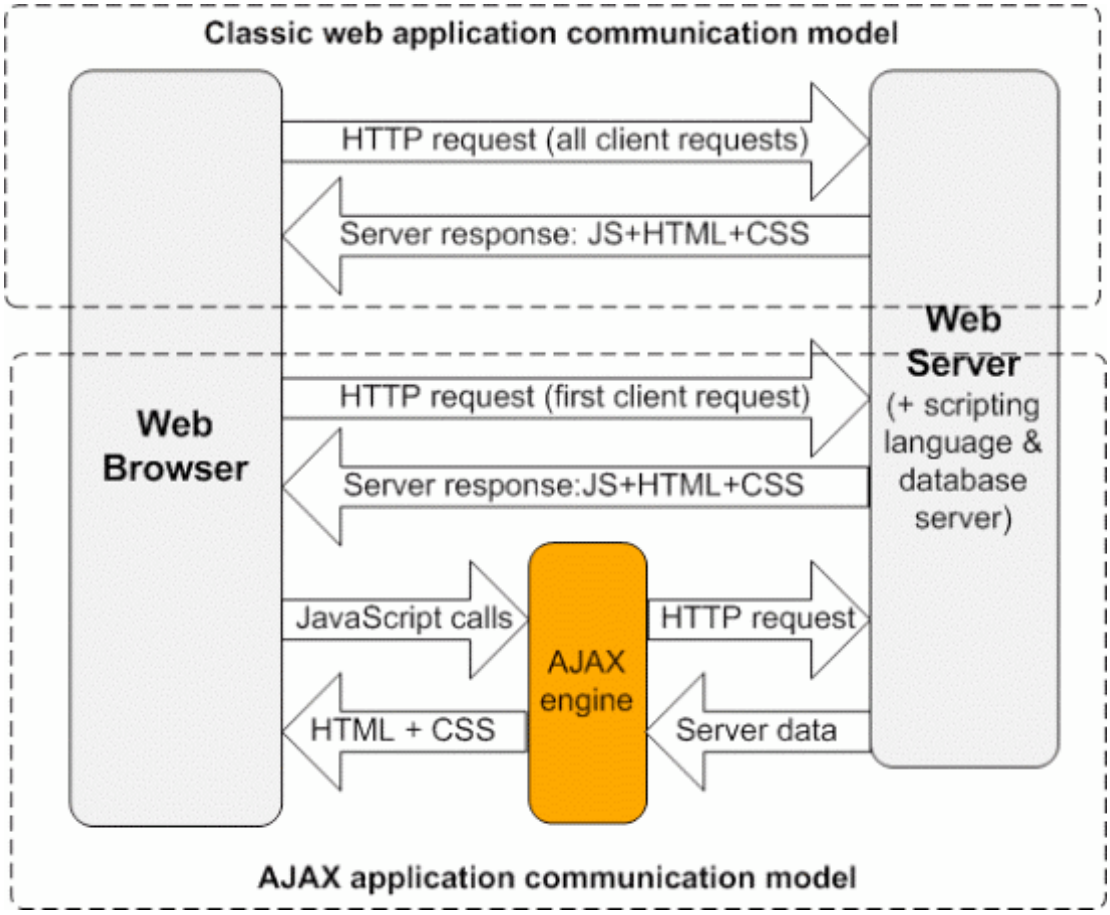

Figure 4: Ajax vs. Classic Web Model [17]

Secondly, this application needed to be both modular and adaptable. It should be modular so that the subcomponents of the application could be singled out and modified without having to rewrite large portions of the application. It should be adaptable so that the structure of the application would allow for both variations in content (the features that are available to clients) as well as environmental needs (such as changing the database that the application links to). These conditions were not defined requirements by the client but rather known expectations for any web-based product.

In order to fulfill the requirements of the application there were two ways in which the developer had to look at the application: the architecture and the Use Case Diagram [See Figure 2]. The architecture of the application is a top down description of

the layers of the project which describes how data is transferred between the database and the GUI. The Use Case Diagram describes the forms that the data takes as seen through the GUI. The developer had to produce the architecture of the application first as it provides the resources needed by the content of the application in the Use Case Diagram.

Each layer of the application architecture will be discussed in turn starting with the database. The database, Oracle, stores the data for the application and the relationships between the tables and defines what data is collected and associated in the application. Forming the database was the first task in making a working application. Once the prototype was broken down and formed into tables that data had to be reorganized and expanded to handle all of the relationships required by the application. The developer used ER diagram to represent his data organization. The developer also found it important to keep track of how data tables (objects) relate to each other, one to one relationships, one to many relationships, use of primary and foreign keys, and the need to add indexes for ordering data.

In order to accommodate the possibility that SMU might use a different database, the developer added a database interface to the application. This is an artificial interface because all of the code at this level is PHP, and can be considered part of the server. This interface does not have to deal with a variation in programming languages, but rather it is a variation in how the data is handled or represented. In this interface server requests come in as XML and are converted into SQL for sending to the database. The database responses are converted into XML before being sent back to the server. The XML and SQL classes are models of the languages they are named after; the former is used for data representation throughout the application, and the latter is strictly used to form the queries which are sent to the database but originate in the server.

In the architecture of the application the server layer is the largest and most complicated. The server in this type of project is the core of the application, all data must pass through here. The server must validate inputs, understand and respond to user requests, and be able to send and retrieve data from the database. What the server does is respond to user actions by sending data where it needs to go. The server knows how to

send and query for data, it doesn't store data. In a web based application the server is the layer of the application in which all of the logic of the application is kept. All of the components below the GUI down to the database are parts of the server. The core server code was designed to be standardized in that it has subcomponents which do varies tasks needed by the application. The server core retrieves messages sent by the GUI and contains the validation functions for retrieved data. Once data has been caught by the server core this piece of the application must interpret the header data in the retrieved message and then send the body of the retrieved message on to the server modules. Once the server is finished processing the message content a response message must be formed and sent back to the GUI.

The division between the core server functionality and the GUI functionality was the second interface that had to be dealt with. This gap is a natural interface as the server is in one language, PHP, and the GUI is in another set of languages, HTML and JavaScript. To bridge this interface there are two activities that must be performed: data must be preloaded onto the web pages and the application must have a messaging system to handle client inputs. Data is preloaded onto the web pages to make the over all application faster. The GUI pages retrieve data via PHP. The PHP data is converted into displayed HTML data and JavaScript variables. This conversion removes all of the PHP objects and server side resources from the web page. None of the PHP code is ever delivered to application clients. In this application XML is the form in which data is maintained so that large amounts of data can be transferred from one part of the application to another. To send messages from the GUI to the server the data must become JavaScript variables which are formed into an XML style message prior to being sent. When the server receives the XML text it has to convert it into an XML object before it can be analyzed or used. After the server side processing is complete a response message, again in an XML format, is sent back to the GUI. This XML server response must be retrieved and interpreted by a JavaScript handler function

The GUI in a web application displays the current application data to the client, handles various client requests, and must maintain client session data. In other words, in

the GUI, data is either pre-loaded onto the web pages, transferred live asynchronously to a web page from the server, or is transferred between web pages. This application's GUI for the most part was created during the prototyping process. The only redesigning the developer did was to make it functional and change the prototype into a usable application.

Throughout this application the developer used the PHP language to form and run the components that are needed here. PHP is a powerful server side language which can be made to perform a large number of tasks. The developer needed a programming language that could do all of the above described tasks; he found information that suggested PHP would work for this project and decided to go with it.

7. IMPLEMENTATION

This developer has found that creating an application is similar to putting together a puzzle, were software engineering is the process of creating the pieces. In order to direct the creation process the developer had to improvise and diagram the overall application until the patterns in the flow of information became clear. In order to produce this application the developer needed to understand the languages that were needed. The developer did some research on the required languages, XML, SQL, PHP, JavaScript, XHTML (a version of HTML) and CSS. (See references 1-7) In addition to the books that the developer read there were methods and bits of code which required additional research and tutorials in order to be worked out and produced. (See references 8-18)

As with any puzzle one needs a good plan of attack. The plan was to extract what the developer could from the prototype, form the database tables, build up the server core, and finally rework the prototype into a usable application. The developer surmised that producing the application in this manner would enable the developer to build up working components that would be used by the latter parts of the application to ensure the flow of data and deal with unresolved design issues as they were encountered. As the structure of the application was built up the developer created a layering of interfaces and base code which contained the high level functionalities, the server modules, which the system was to provide to clients. The following are the steps taken to turn the idea of the web based survey development system into a reality.

The first step was forming the database. In order to work at a practical location and work station the developer needed to load a database server onto his computer. The developer downloaded and used Oracle 10g.. The process of learning how to manage a database server and create database tables was new to the developer. It took time to get the database up and running and to be comfortable using it, but in the end it was good learning. The data tables that the developer initially used came from the GUI prototype,

and as an understanding of both the database software and the application grew, the tables became more elaborate and concrete.

The second step was building the application foundation. It struck the developer that the more he worked on the application the harder it became to visualize the processes encapsulated in the application, as each layer of the application excluding the database interaction class, this is the lowest level class, would rely on the previously developed components. In order to counteract this effect, the developer had to find ways in which he could organize the pieces of data being worked with. [See Figures 1, 2, and A.3 and A4 for example organizational diagrams] The actions the developer performed included: nomenclature (naming every component of the application so there was no ambiguity in identifying the parts of the application), indexing (the developer devised a numbering system for all developed classes and methods to help with code searches and testing), and added an error tracking interface (it allowed the developer to form execution stacks and report errors.

The third step was producing the application core: The server core filled the gap between the lower level functionalities (such as database interactions) and the higher level application components by providing services such as interpreting GUI request messages and returning requested data from the database. The following is a brief list of the classes that the developer produced in order to service the higher level functionalities. This list does not include the server modules, the testing classes, or GUI related classes.

- **00_constants.php** – contains the defined unchanging values in this application as well as the locator values used to connect classes
- basic (folder)
    - **01_helper.php** – code used to form HTML and simplify the testing of other server classes
    - **02_error_tracking.php** – A super class used to simplify the error reporting of the other classes in this project
- database (folder)
    - **03_equation.php** – a class that simulates an equation such as those used in SQL queries.
    - **04_sql.php** – A class that simulates the SQL language (a database query language)

    - **05_xml.php** – A class that simulates the XML language, used to contain data throughout the application
    - **06_database.php** – The class which communicates directly with the Oracle database
    - **07_db_interface.php** – The interface class through which the server must use to communicate with the database
- server (folder)
    - **01_server_core.php** – A class that contains the application's top level server code (this class will initiate the creation and use of all other classes in the "server" of this application)
    - **server.php** – Contains the script through which the GUI initiates contact with the server, also creates the server class object.

The forth step was the development of the server modulus. In order to service the GUI and make it adaptable the developer devised the server module system. In this system the objects and services described in the Use Case Diagram [Figure 2] of the application were given form and function. It became obvious to the developer that the prototyping meetings only defined what should be in the application not which parts were most important. This meant that some of the objects that the developer was creating were a necessity and others were peripheral. In order to produce the current working application it was decided that the nonessential components of the application be put on hold. The following is the list of fully developed modules:

- interface modules (folder)
    - **08_module_engin.php** – A class that makes it easier for modules to form data requests (contains functionalities required by all modules)
    - **09_module_interface.php** – the class through which the server class must work to operate the systems modules
    - **10_module_hub.php** – The class that allows the application to use multiple modules (knows which module objects to create based on user request data)
- GUI Modules (folder)
    - Community (folder)
        - **author_module.php** – contains the functionality for maintaining author "accounts" in the system
    - Distribution (folder)

        - **advertising_module.php** – contains the functionality for forming advertising schema, the object which records the publication of a survey to a target population
    - Population (folder)
        - **known_info_module.php** – contains the functionalities needed for recording acknowledged population information
        - **population_module.php** – the functionalities used to form target population objects
        - **user_module.php** – contains the functionalities used to create survey taker accounts
    - Report (folder)
        - **report_analysis.php** – a class that contains the algorithms needed to retrieve and analyze survey response data
        - **report_module.php** – the module which contains the functionalities used to create report objects
        - **response_module.php** – contains the functionalities used to capture user answers to survey questions
        - **session_module.php** – the functionalities needed to create and maintain objects which define the connections between a user, and a given publication of a survey
    - Survey (folder)
        - **choice_module.php** – the module which preserves the options in a survey that a user can choose from
        - **contant_module.php** – the module which records the questions and format options that an author develops
        - **format_module.php** – forms the objects which allow an author to display survey content with a desired effect
        - **group_module.php** – contains the functionality needed to define a collection of response choices
        - **page_module.php** – contains the functionality needed to form divisions in a survey
        - **survey_module.php** – a module that forms the shell which represents a survey in this system
        - **user_survey_module.php** – contains the functionalities needed to convert survey data into a viewable format

The final step was GUI redevelopment. Once the server, the Ajax engine of the application, was working it was time to revisit the prototype and to convert it into a working interface. The issue here was not which languages to use but rather data management, knowing what data one has and where it is going. In the GUI there were four types of data to piece together:

1. **Preloaded Page Data** – information that is loaded onto a page from the server at the time the page has been requested
2. **Page Session data** – the information that is passed from web page to web page in order to maintain consistency
3. **Client Inputs** – the gathering and responding to the information from the text fields and buttons so that data can be sent to the server
4. **Server Responses** – decoding the messages retrieved from the server so that the page data is fresh, and the client is kept informed as to the changes in his or her information

Type one of the above was performed by PHP and the rest were handled by JavaScript. The conversion process was straight forward: place all of the necessary functionalities on a given screen and find ways to maximize efficiency and screen space while hiding unused or seldom used components.

For further details on the classes and content described in this section please refer to diagrams A-F for details on the class structure of this application. Diagrams A–E contain the internal data for the different parts of the application, the attributes and methods for each class, and diagram F shows the relationships between the classes. The module classes are encircled as they are as a group linked to the other classes as the diagram indicates. This short hand notation is used to simplify the diagram.

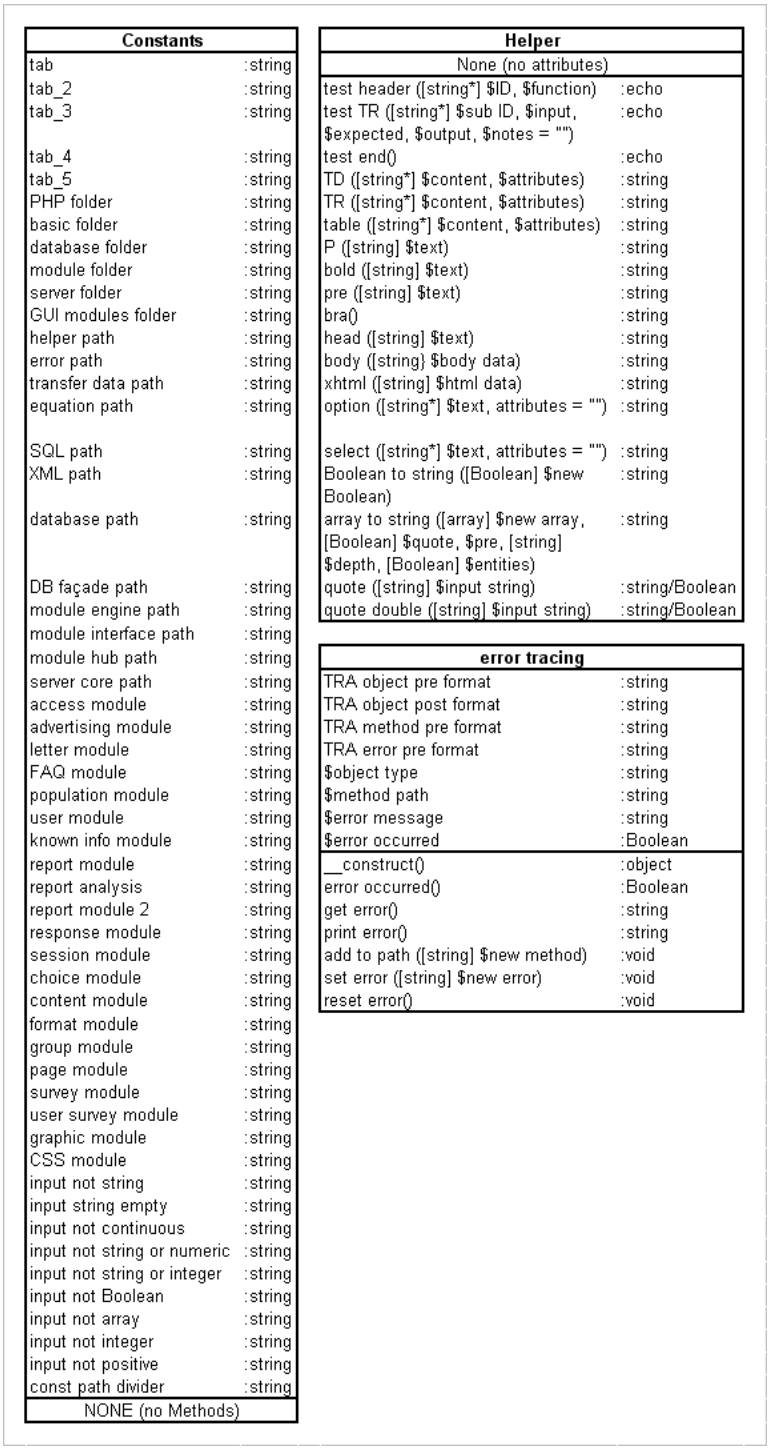

Figure 5.A: General Server Classes

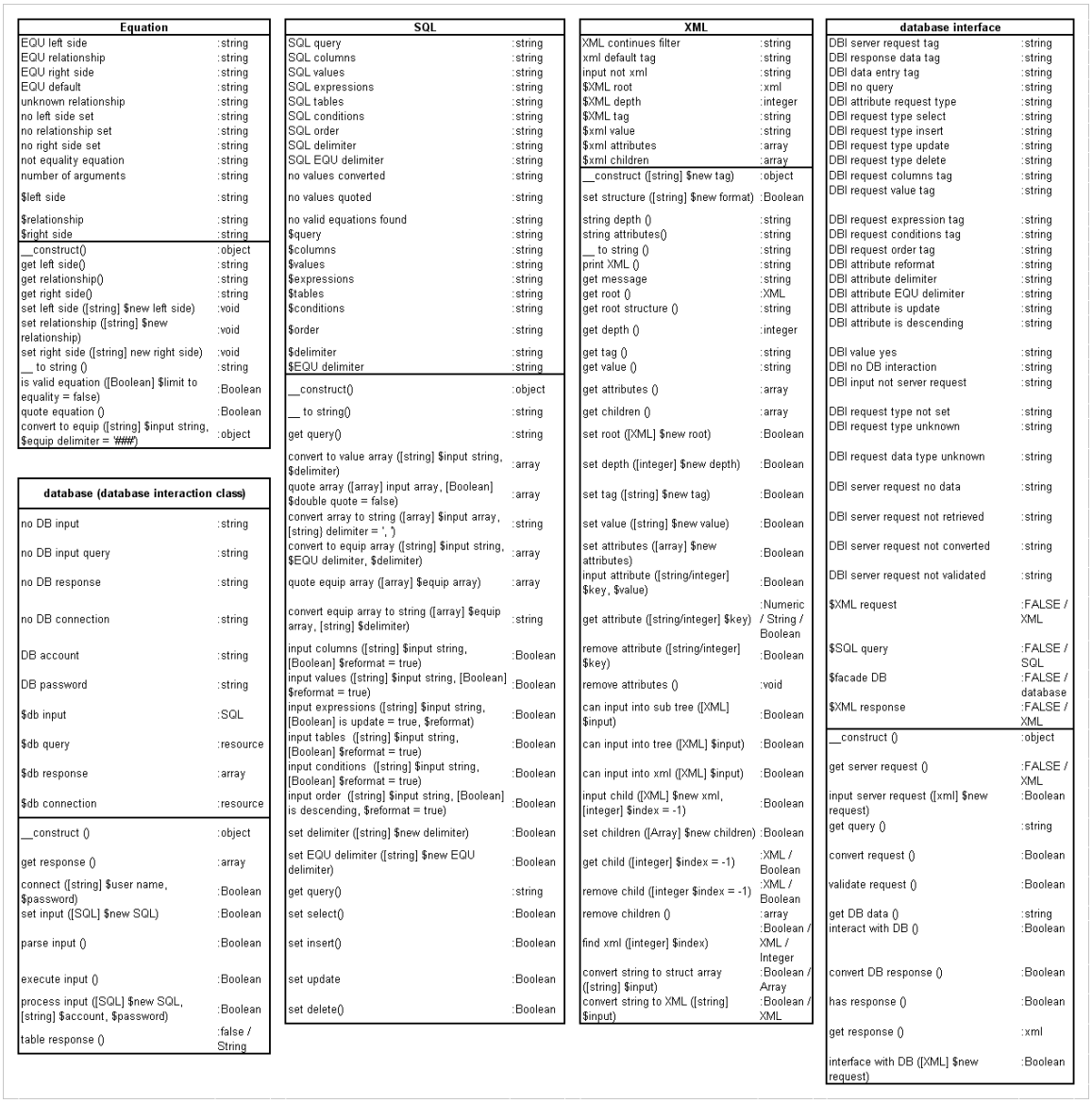

Figure 5.B: Database Classes

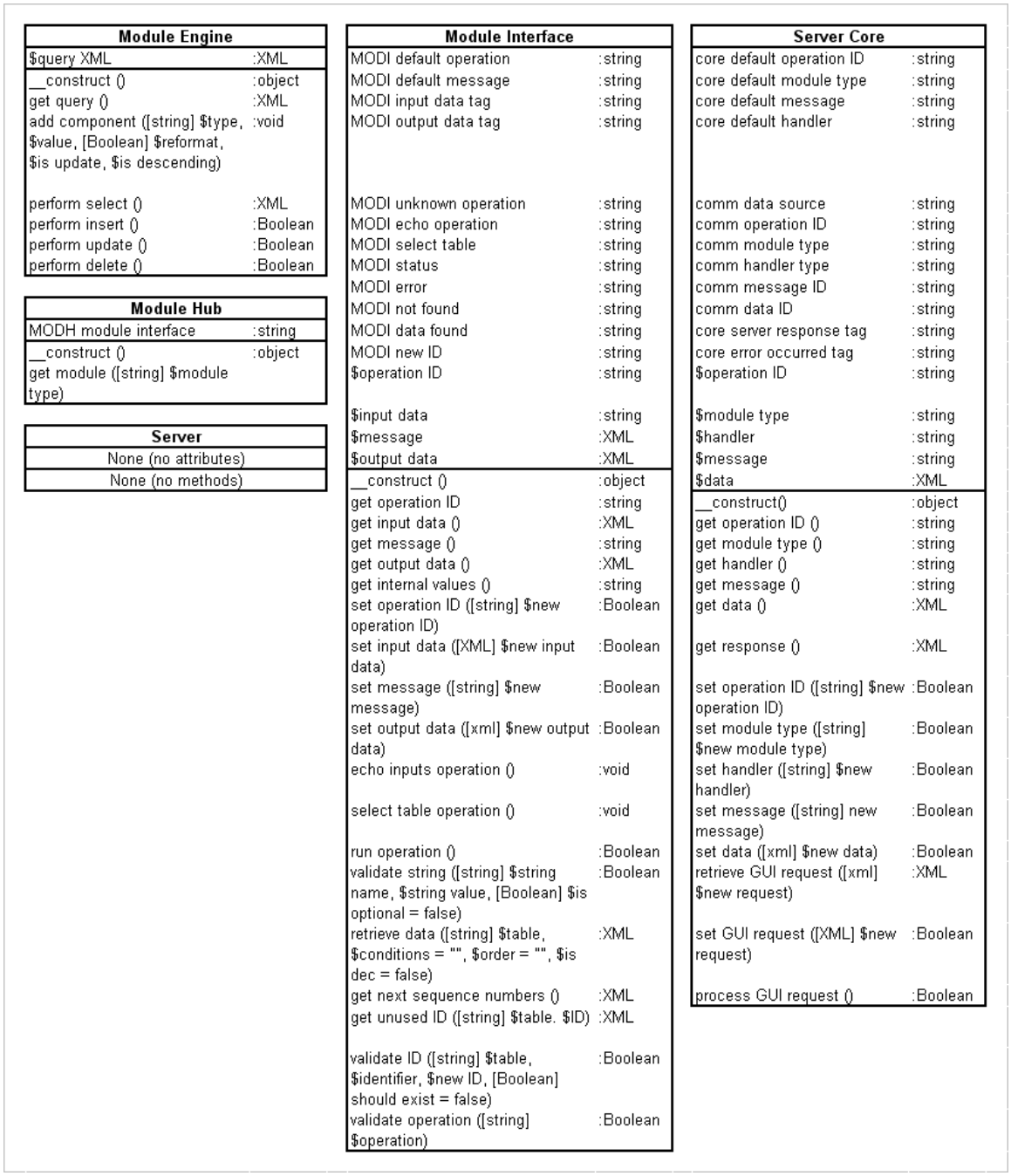

Figure 5.C: Server Core Classes

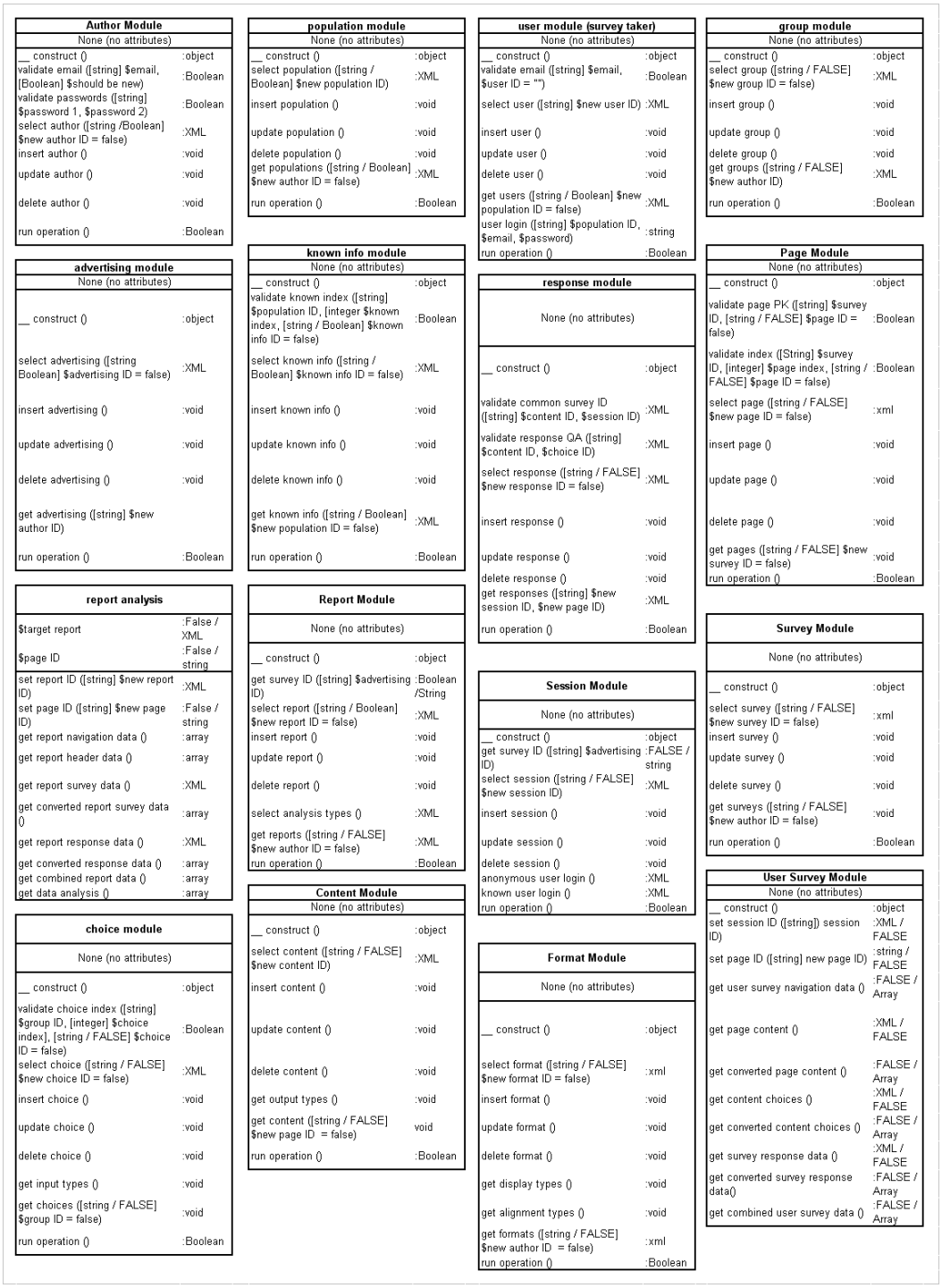

Figure 5.D: Server Modules

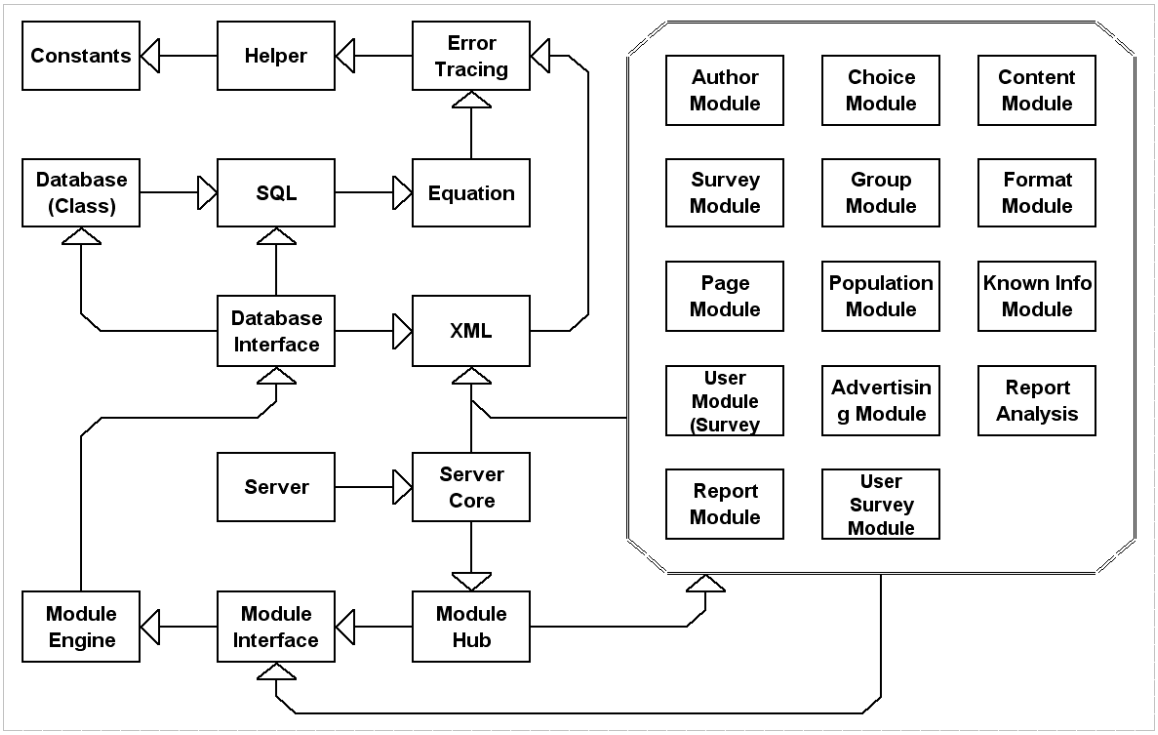

Figure 5.F: Class Diagram

8. TESTING

It is counter intuitive to say “a good test is one which finds a bug.” No developer ever wants to find bugs in their code however it is better to find a bug and be able to remove it then have code that crashes when a client uses it. Testing is a way of indicating whether or not a bug exists in a given peace of code. A set of tests will show one of two things either: that a bug exists (these inputs cause that output which is not expected) or that the code appears to be working (these inputs produce those expected outputs). Once a bug has been indicated to exist by a test the developer must debug the code, track down what is causing the bug and find a solution that will fix the issue. Testing is never foolproof; no matter how many tests are performed there could still be unfound bugs. The developer did extensive testing but at times had to go back to previous code in order to fix an issue which only became apparent when a subsequent part of the application used some of the previous code in a new or unique way.

Before describing the types of testing the developer did, one first needs to describe internal validation and documentation. Internal validation is the process of checking the form and content of input data; it is a battery of checks which are performed each time data is sent into or out of a method or class. This internal work is the developer’s way of trying to limit or define “good” and “bad” data. The format of a data value is the mechanical representation in which data is sent: string, Boolean, etc., and the content of data is the values or information that are being transferred. These checks help the developer to be confident that the information being worked with is valid and they directly affect the tests which are performed. Besides validating the data in each method, each method has to be described. It was important to note the purpose of each method as well as explain the expected inputs and outputs. This documentation is a necessity for large applications, for if the code is not described adequately then if one had to review old or previously created code it might not be understood. This documentation is used when debugging code as it describes how the code is expected to perform.

There were two parts of the application that needed to be tested, the server and the GUI; each had to be tested differently. This variation is due to the programming languages in which each was written. The server was written in strictly PHP and the GUI primarily in JavaScript and HTML. The server produces concrete responses; therefore test scripts were used to evaluate the server components. [See Figure 6 for a test script example] The GUI is more abstract. It has pre-loaded information but how it responds to inputs depends upon the order of inputs and interactions it has with the client and the server. In other words the GUI had to be tested live, no predefined testing.

In the server each class had an independent testing script made for it. As the output and interface of the application is web based it made sense to display the test results as HTML, a static description of the inputs, expected results, and outputs of each test. The developer found that in viewing such test pages, the tests themselves were rerun, meaning each time the developer refreshed the results page the developer was looking at the latest analysis of the underlying code. Some tests particularly the ones involving the GUI modules of the application affected the database, current data being displayed and or changed. To make these tests static the test changes to the database had to be undone by the test script. And yet because current information in the database affects what is displayed, expected results still went out of date.

The following [Figure 6] is a sample of a set of tests that were done on a given class method. What this example shows is that there were three tests performed, each test given an index value from op.01.01 to op.01.03. The first part [op.01] is the index of the method and the 01-03 are indexes for the given sub tests. In these tests the developer compared the expected values with the actual output values, if the values did not match this indicated that an error had occurred. The notes section simply describes the purpose of the test and includes any information of importance to the test that should be remembered.

Population.op.01				
Testing:	**[Ouput Data: STATUS/Entery: POPULATION_ID, AUTHOR_ID, NAME, (optional) PURPOSE]** **select_population()** **[Input Data: POPULAITON_ID]**			
#	**Input:**	**Expected:**	**Output:**	**Notes:**
op.01.01	`<Input_data/>`	`<Response_Data STATUS='ERROR' />` Invalid (POPULATION_ID) Retrieved	`<Response_Data STATUS='ERROR' />` Invalid (POPULATION_ID) Retrieved	Testing: invalid population_ID (none set) Note: error tracking shut off to save screen space
op.01.02	`<Input_data POPULATION_ID='Unknown' />`	`<Response_Data STATUS='ERROR' />` Required [POPULATION_ID] Not Found	`<Response_Data STATUS='ERROR' />` Required [POPULATION_ID] Not Found	Testing: invalid population_ID (not known by the database) Note: error tracking shut off to save screen space
op.01.03	`<Input_data POPULATION_ID='pop_1' />`	`<Response_Data STATUS='RETRIEVED' > <Entry POPULATION_ID='pop_1' AUTHOR_ID='0001' NAME='first population' PURPOSE='selectable pop' /> </Response_Data>` [POPULATION_ID] Found	`<Response_Data STATUS='RETRIEVED' > <Entry POPULATION_ID='pop_1' AUTHOR_ID='0001' NAME='first population' PURPOSE='selectable pop' /> </Response_Data>` [POPULATION_DATA] Found	Testing: valid population_ID Note: error tracking shut off to save screen space

Figure 6: Testing Example

Once the developer finished working on implementing and testing the server he moved to developing and testing the GUI. One bug the developer found when switching to the GUI was that the messages were being cached by the web browser. Caching web pages is a way of making the retrieval of static web data faster as the web browser has a copy of the data saved on the local machine. However when a web page is part of an Ajax application this caching of messages interrupts the flow of data. To solve this issue the developer had to add a header to all system messages and web pages which tells the current web browser to never cache this application's messages.

The only way the developer found to test a web base Ajax empowered GUI was to run the web page. In this case run means the developer had to use the available functions on the page and check whether or not the correct actions occurred. There are two types of actions, data must be sent to the server (and ultimately the database) and updating the displayed data when a server response or client input is retrieved. There is no record of the GUI testing; there is only the developer's assurance that each element of

the GUI has been tested. Once it was proven that a page's functionality worked, that page retrieved no more updates. As any update would require all of the functionalities on the page to be retested.

9. LIMITATIONS

The core of this application is fully developed, and yet there are additional features which should be added to the next version. There are two types of limitations in this application, components which were considered peripheral, and development issues which have not yet been addressed. The peripheral components were discussed in the prototyping meetings, were included in the initial design, but were not fully implemented. The development issues which would make the application more robust had to be delayed until the core functionality was working.

To streamline the production of this application there were several functionalities which were ignored or left incomplete. In terms of developing surveys the following were delayed: the use and importation of graphics and CSS and the ability to define paragraphs, or white space, and the ability to retrieve user text for open ended questions. Other content and improvements which was left out include the following:

- Being able to send a survey to a target population via email
- The Frequently Asked Questions system
- The applications Help System
- The ability to send cover letters to target population users
- Exporting report data to a text file
- Importing user data into a target population
- The cascade on delete system (the ability to delete or hide complex application components)
- Not proven to be handicap accessible

On the development side of this application there are several concerns which haven't been addressed. This application has the following security issues which need to be dealt with before it can be made public: the GUI should use the POST method of sending messages to the server (it currently uses the GET method); the input validation for the application needs to be able to filter out malicious inputs (such as foreign code or query statements); critical data should be encoded so that messages are not in clear text for any one to read; and database transaction consistency has to be tested. Other more basic issues which need to be dealt with include email address validation, and possibly

additional navigational options in the GUI. Beta testing, or allowing clients to use the application, would help the developer to find the quirks and inconsistencies which linger undetected. In other words, this application has not been out of its sanctuary and experienced the real world in any way, and it isn't prepared to deal with the types of issues that it is likely to face.

8. CONCLUSION

The point of this project, beyond creating a useful application for clients, was to find a puzzle in which the developer had to push him self in order to strategize and come up with solutions to the problems he encountered. For a significant portion of this project the developer felt as if he were working blind. He knew what the destination was but the developer did not have a clear path, and it was often difficult figuring out the next step. In developing this application he learned how important it is to gather the requirements, make oneself a roadmap of the steps involved in producing the application, and write down what he did, why he did it, and document every aspect of the application so that errors could be overcome. The developer learned that the life cycle models, diagrams, and documentation techniques which he had been shown are useful tools in organizing and planning how to form the structures, layers and interfaces which must be implemented in order to form a robust and adaptable application. Additionally the developer learned that despite ones best efforts there will be features to an application which may need additional work. Insight: An application, no matter how well constructed, ends up being a prototype for the next version of itself.

9. BIBLIOGRAPHY

[1] Damon Dean, Cascading Style Sheets for Dummies, 2001

[2] Ed Tittel and Natanya Pitts, HTML 4 for Dummies, 4th Edition, 2003

[3] Emily A.Vander Veer, JavaScript for Dummies, 3rd Edition, 2000

[4] Janet Valade, PHP 5 for Dummies, 2004

[5] Allen G. Taylor, SQL for Dummies, 5th Edition, 2003

[6] Ed Tittel, Chelsea Valentine, and Natanya Pitts, XHTML for Dummies, 2000

[7] Ed Tittel, Natana Pitts and Rank Boumphrey, XML for Dummies, 3rd Edition, 2002

[8] http://adaptivepath.com/ideas/essays/archives/000385.php

[9] http://www.w3.org/DOM/DOMTR#dom1

[10] http://developer.apple.com/internet/webcontent/xmlhttpreq.html

[11] http://msdn.microsoft.com/en-us/library/ms533050.aspx

[12] http://www.tizag.com/javascriptT/javascriptconfirm.php

[13]http://www.zvon.org/xxl/DOM2reference/Output/HTML/method_deleteRow_HTMLTableElement.html

[14] http://www.php.net/manual/en/

[15] http://www.oracle.com/technology/pub/articles/cioroianu-ajax-data.html?_template=/ocom/print

[16] http://developer.mimer.se/validator/parser200x/index.tml#parser

[17]http://www.interaktonline.com/support/articles/Details/AJAX%3A+Asynchronously+Moving+Forward-How+does+AJAX+work%3F.html?id_art=36&id_asc=308

[18] http://www.adaptivepath.com/ideas/essays/archives/000385.php

APPENDIX A: SUPPLEMENTARY DIAGRAMS

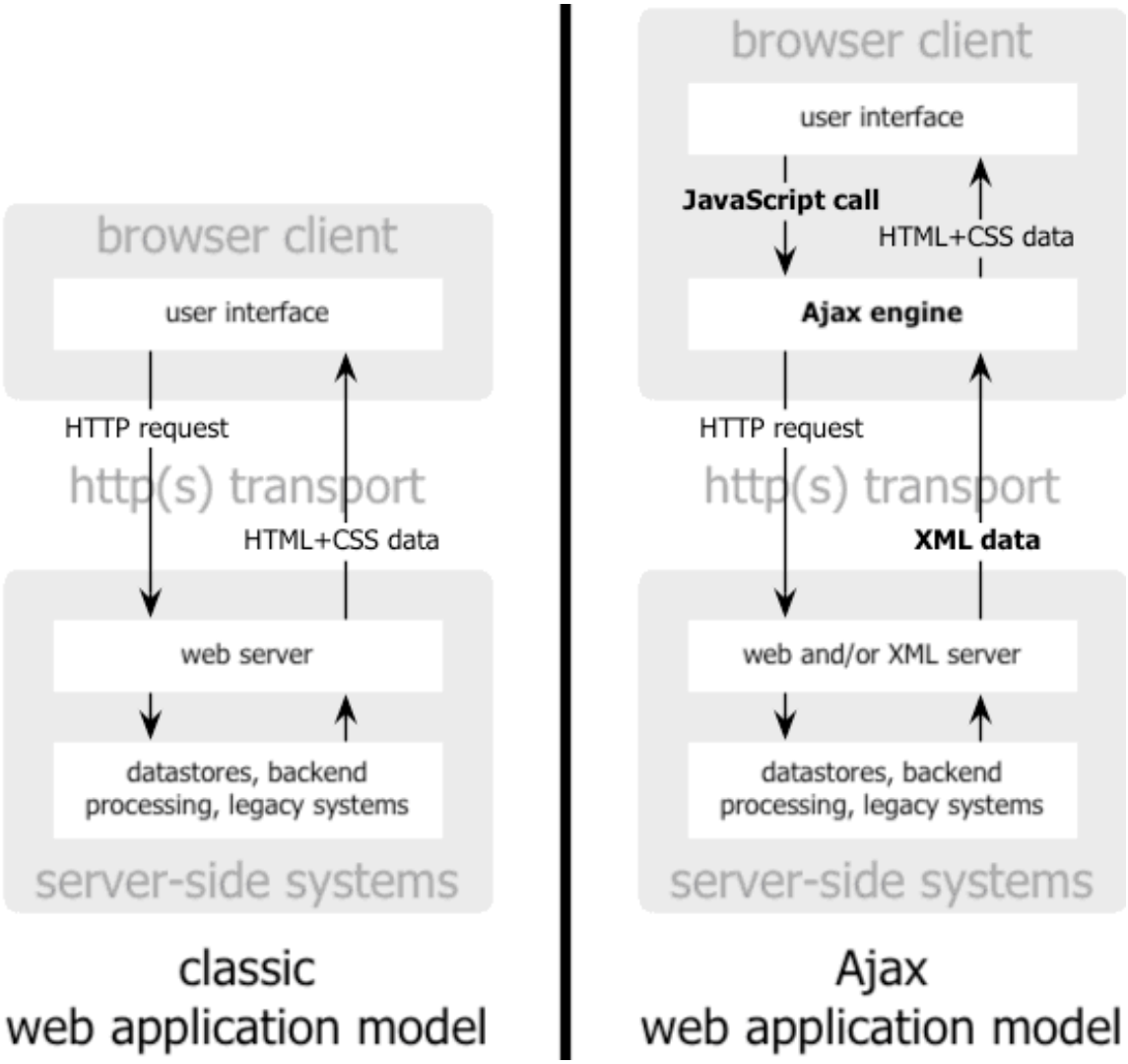

Figure A.1: Ajax vs. Classic Web Structure [18]

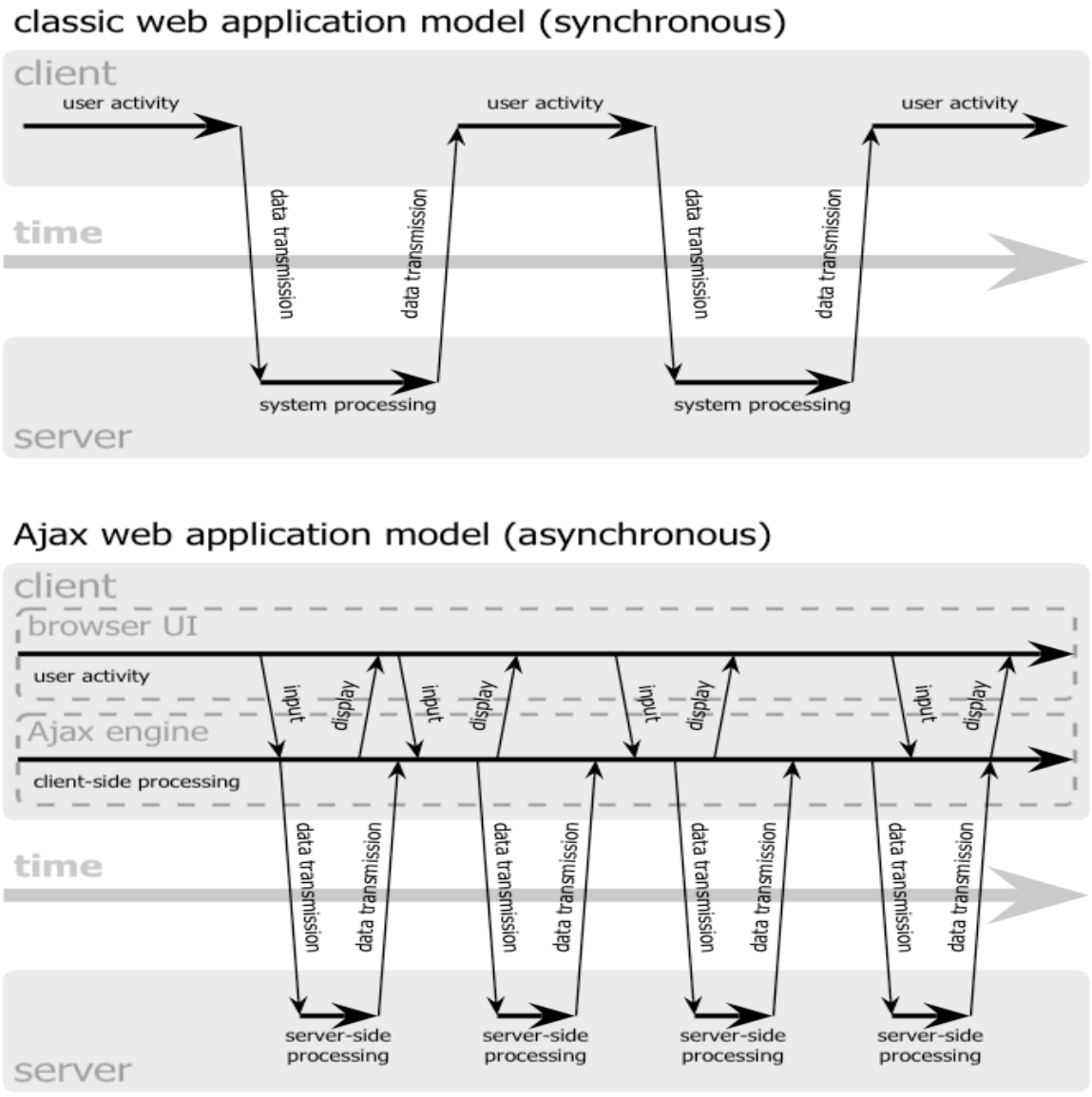

Figure A.2: Ajax vs. Classic Web Interactions [18]

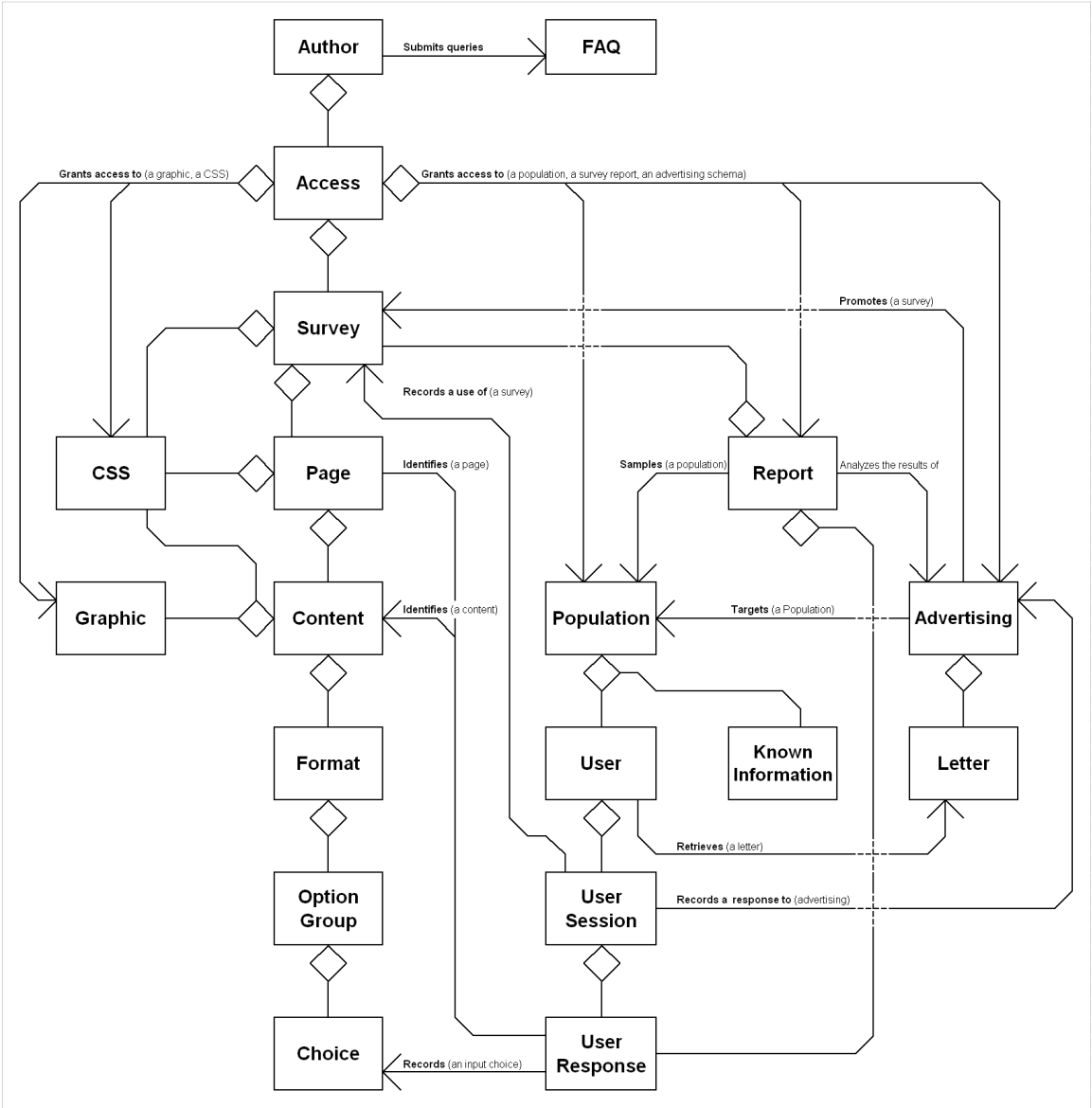

Figure A.3: Initial Module Class Diagram

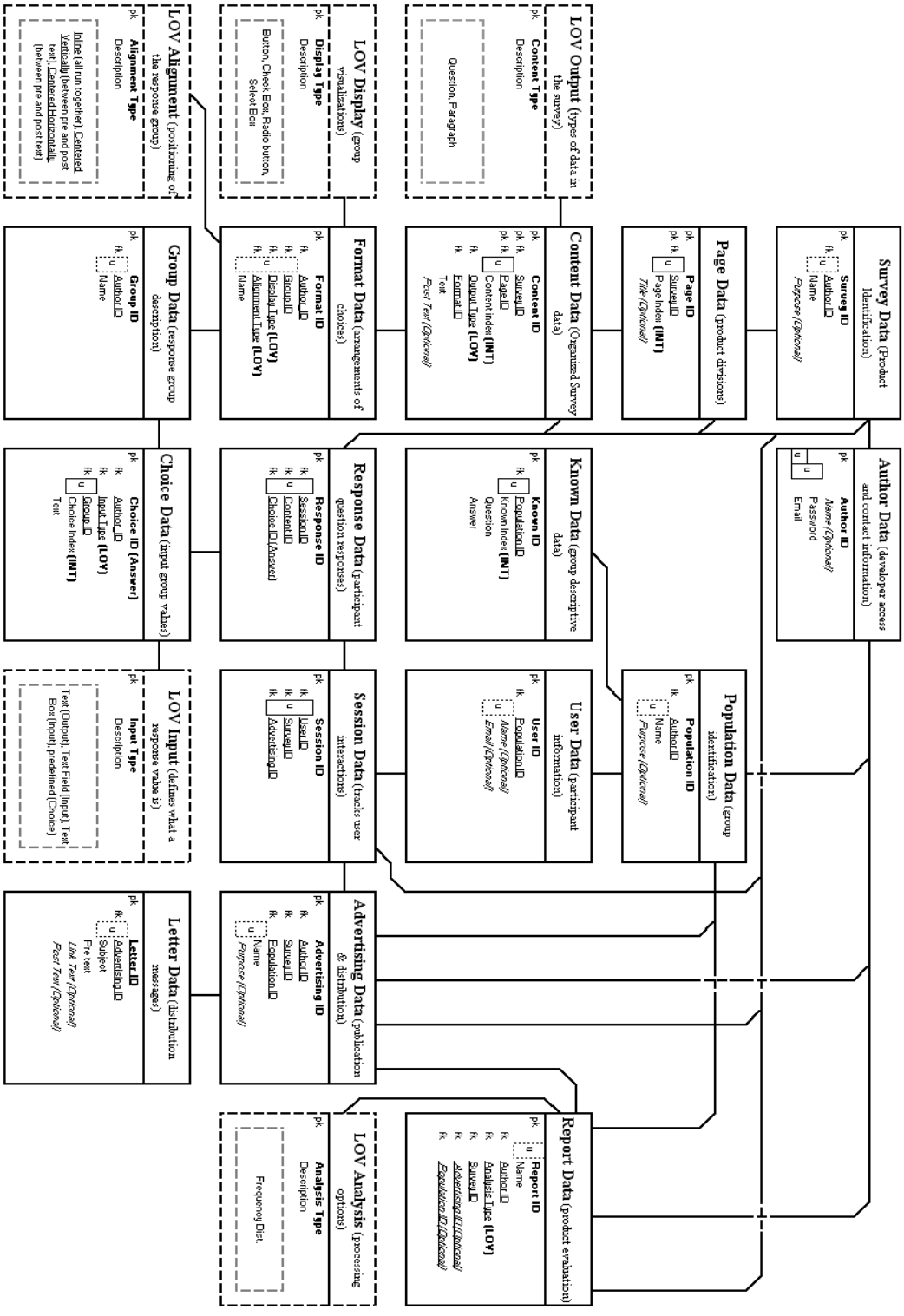

Figure A.4: Verbose Database ER Diagram

APPENDIX B: SCREEN SHOTS

Figure B.1: Author Login

Figure B.2: Author Registration

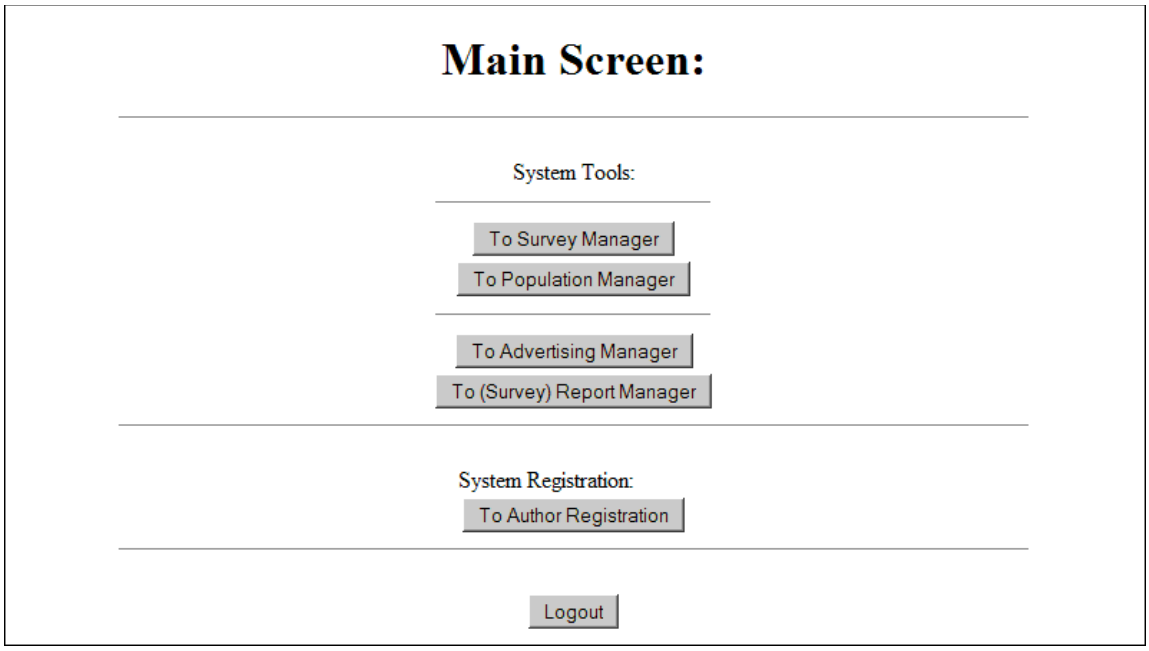

Figure B.3: Main Screen

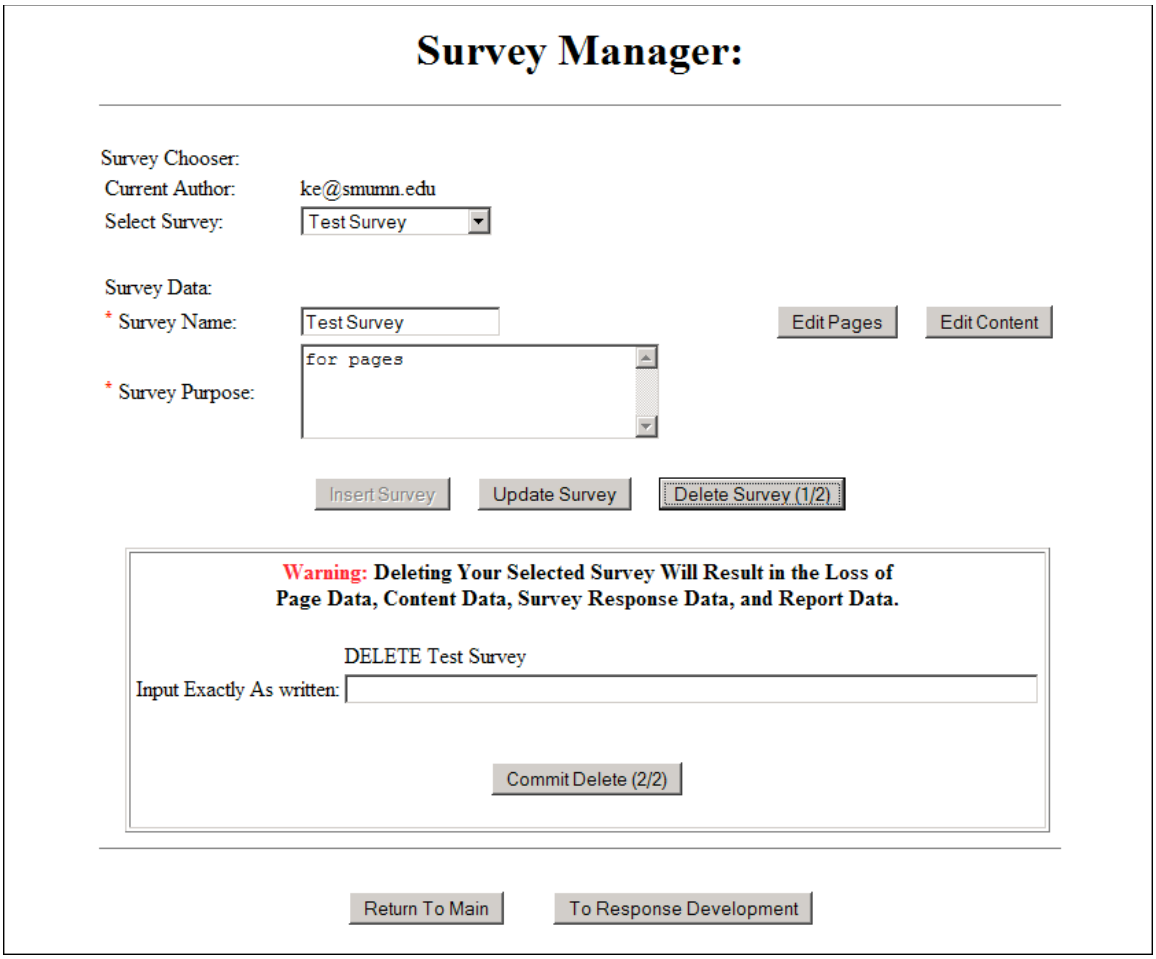

Figure B.4: Survey Manager

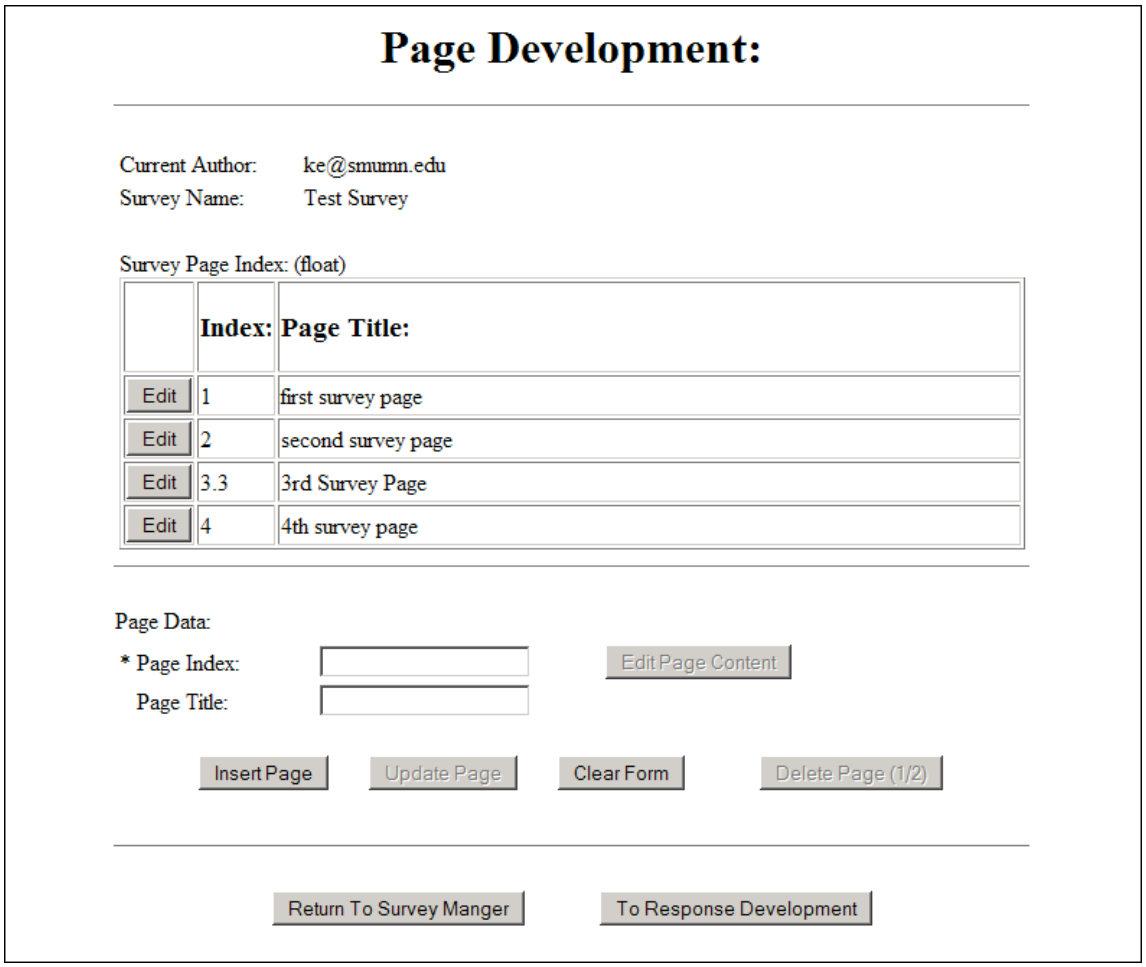

Figure B.5: Page Development

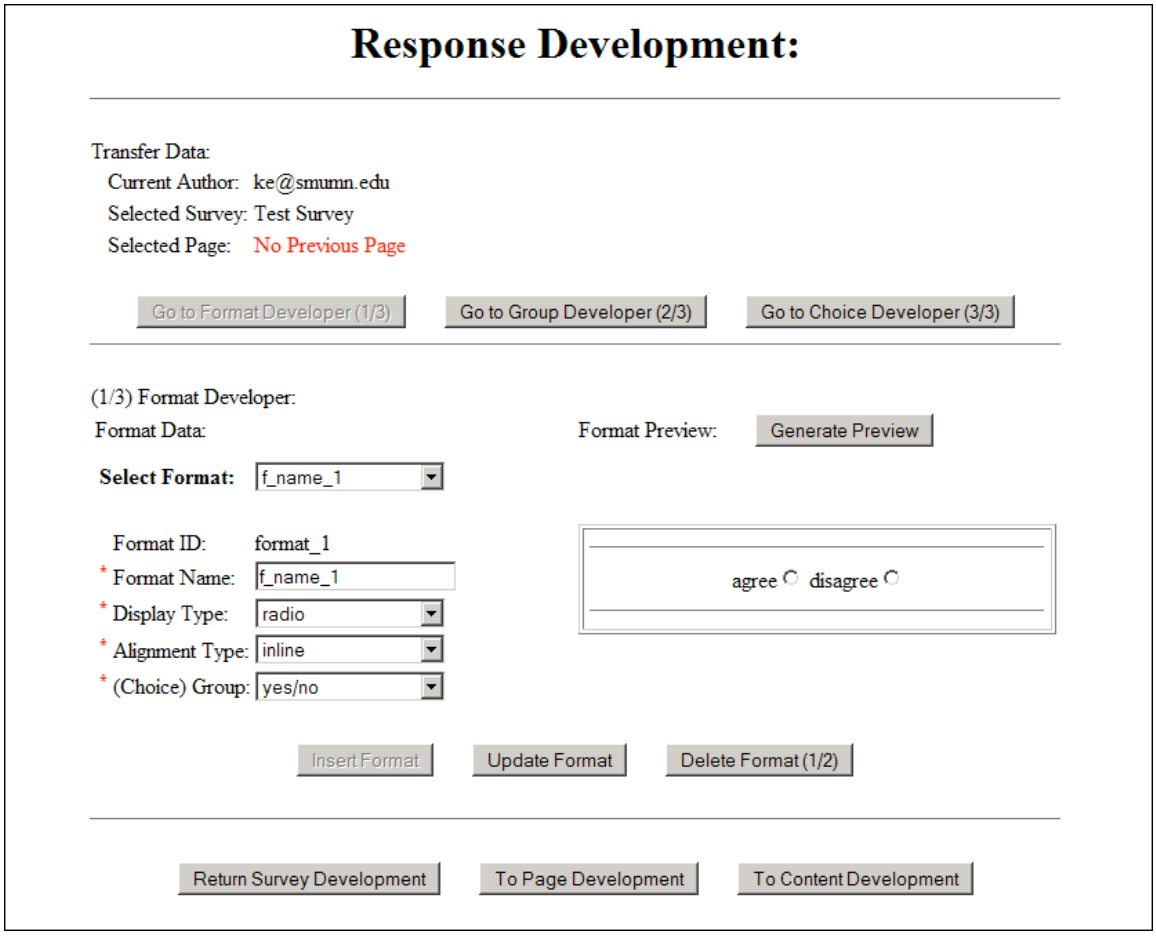

Figure B.6: Response Development (part 1)

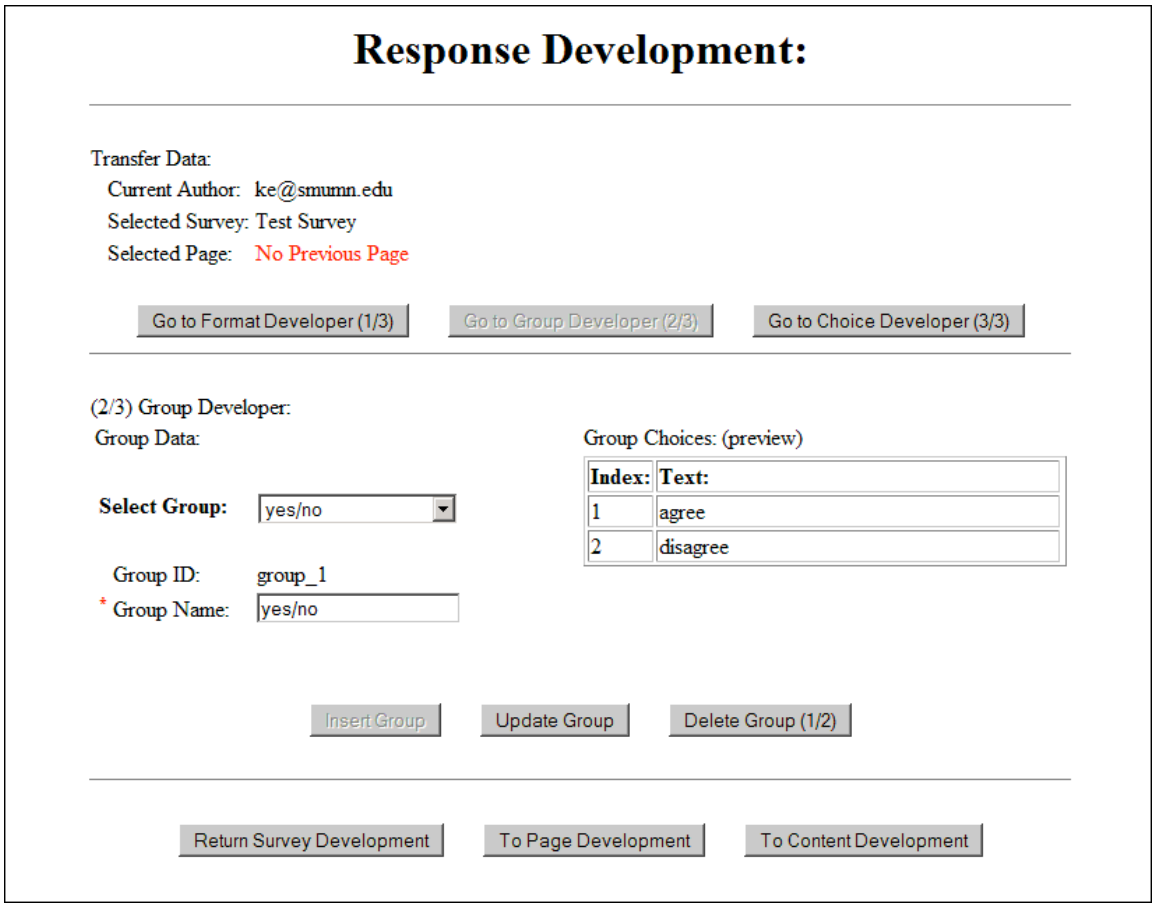

Figure B.7: Response Development (part 2)

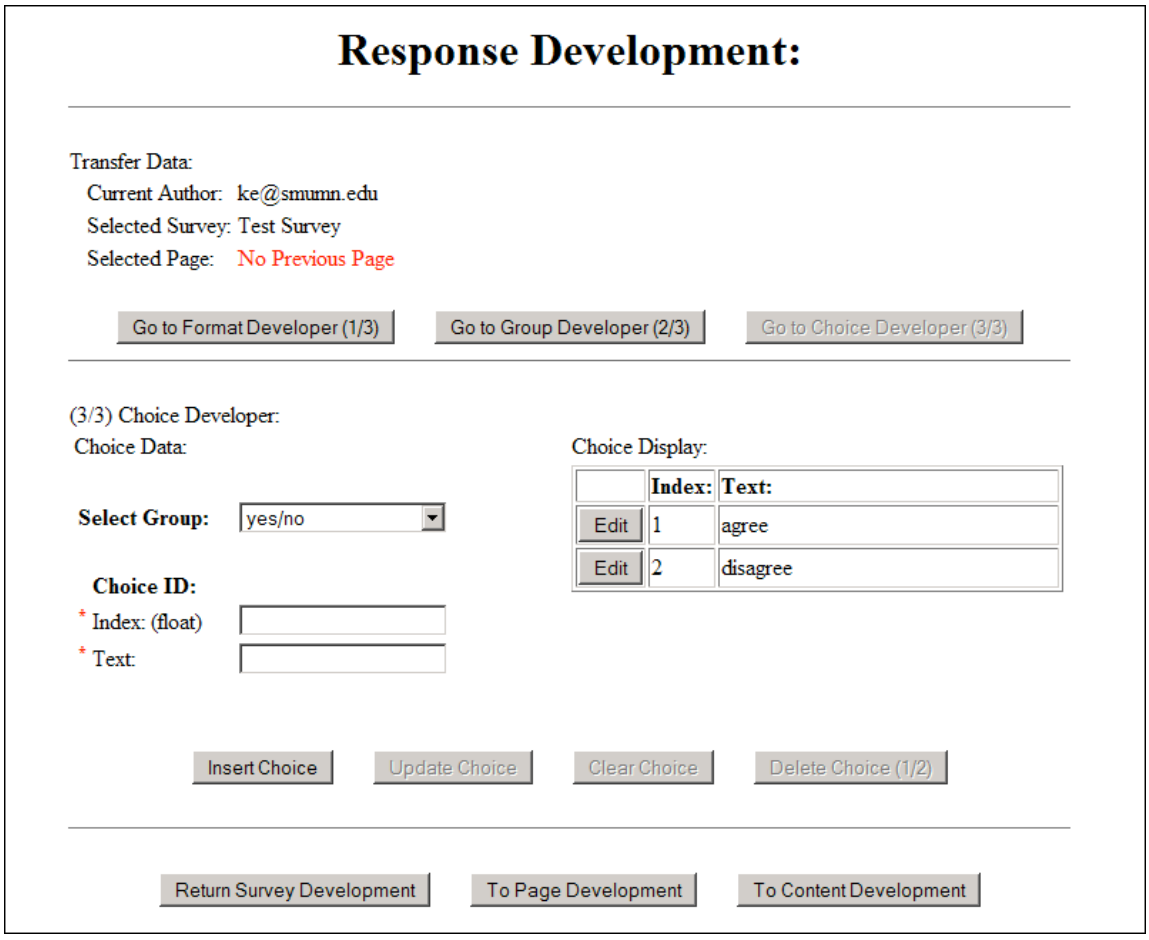

Figure B.8: Response Development (part 3)

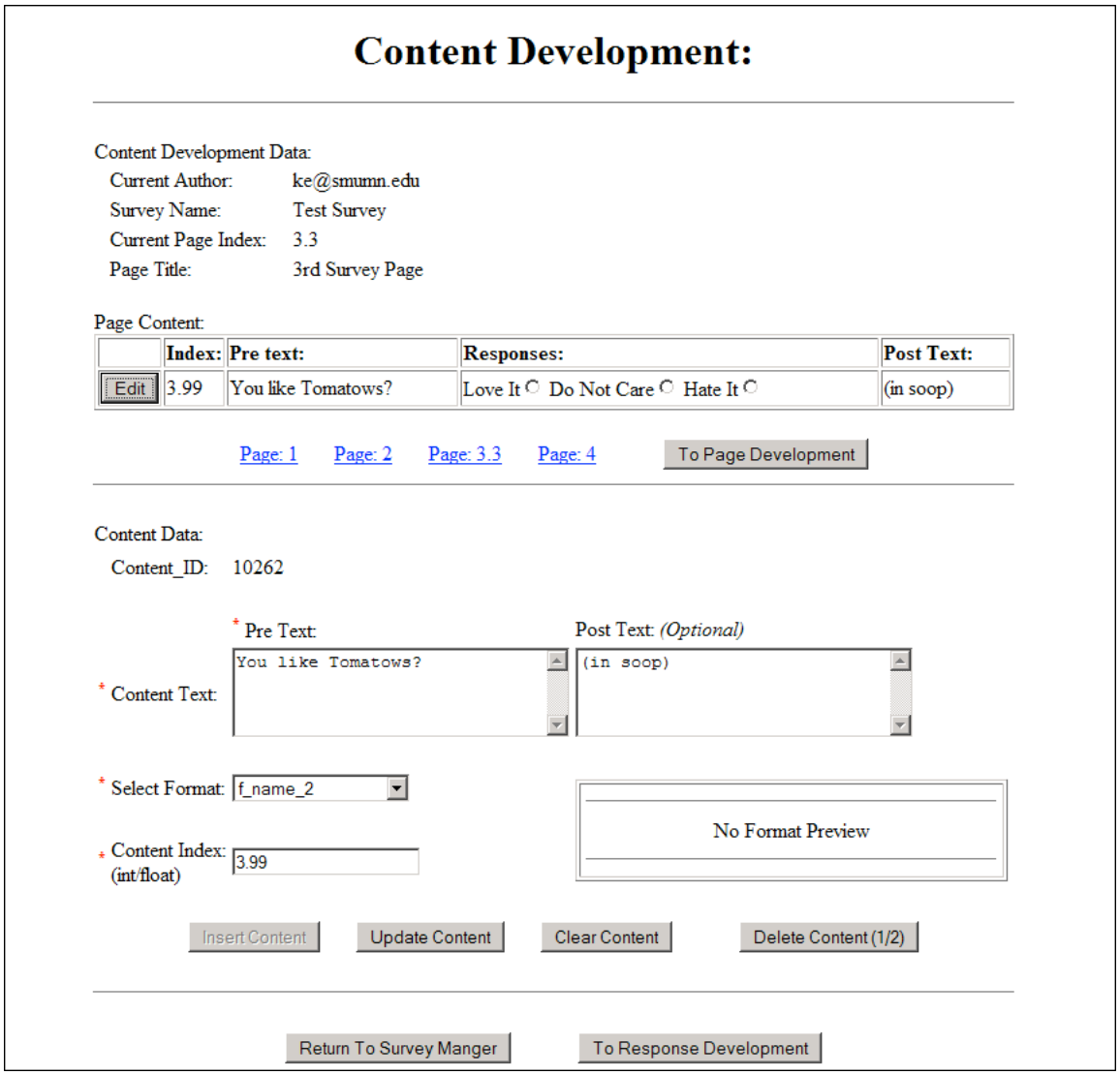

Figure B.9: Content Development

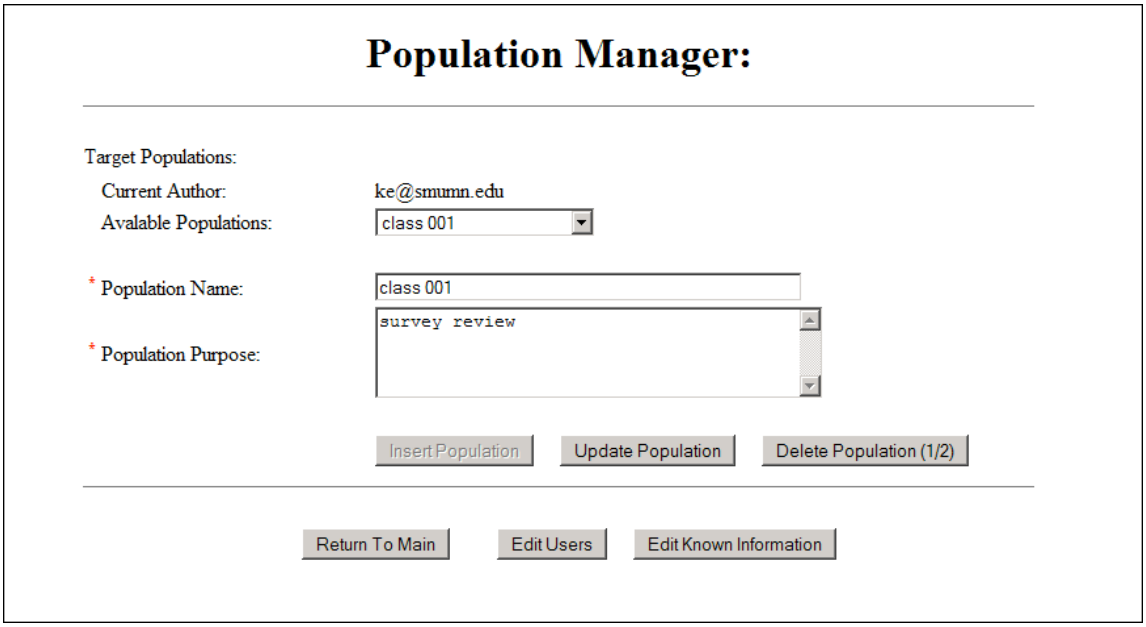

Figure B.10: Population Manager

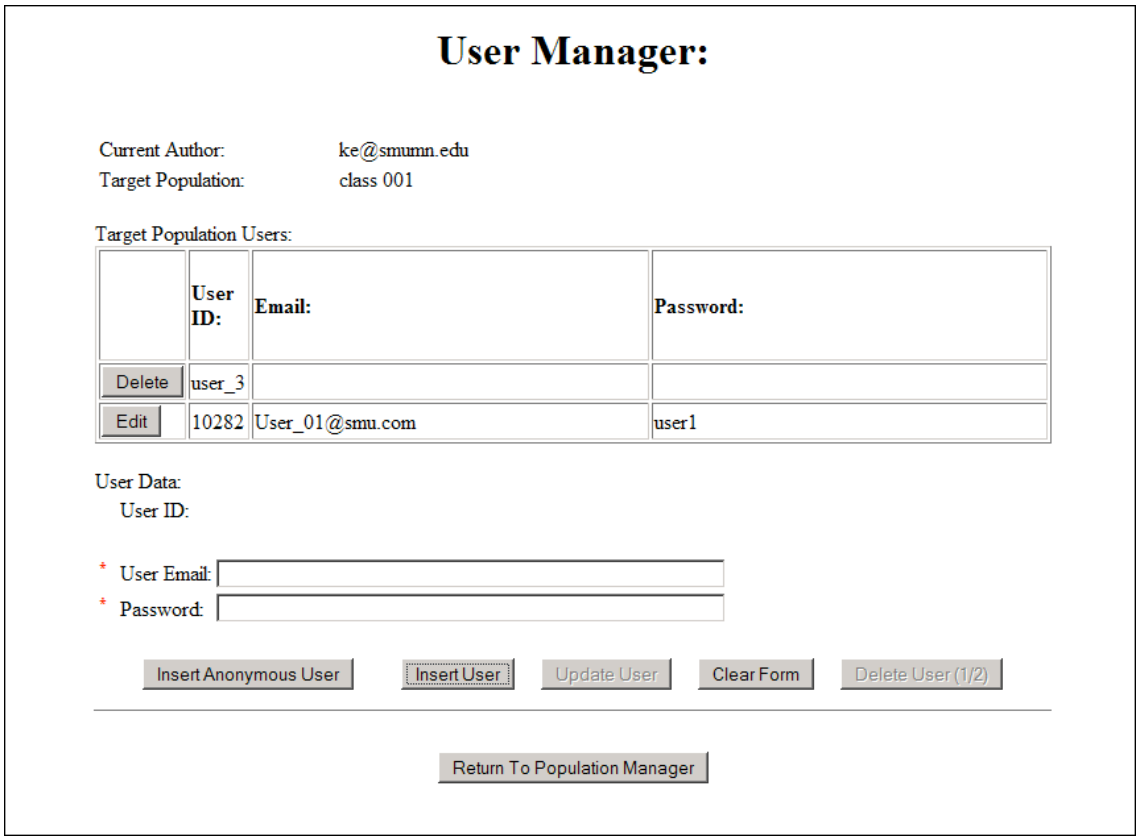

Figure B.11: User Account Development

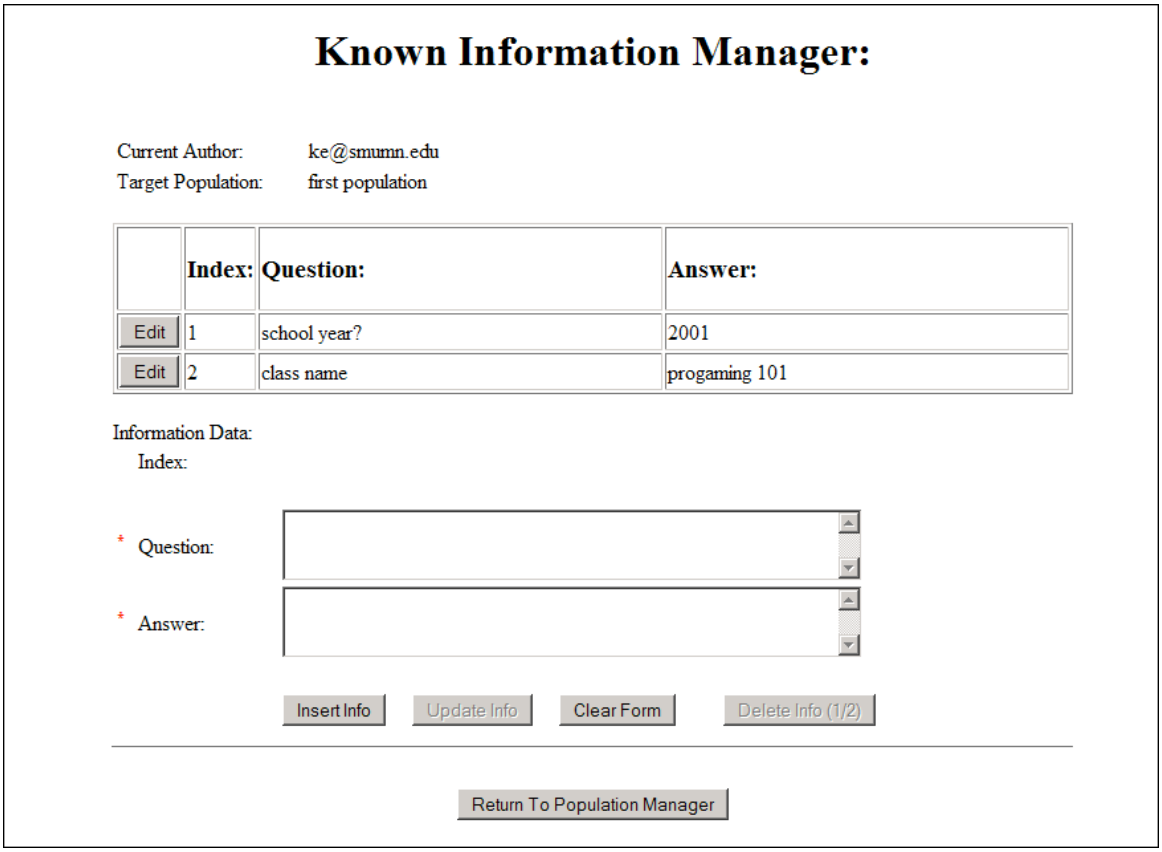

Figure B.12: Known Information Development

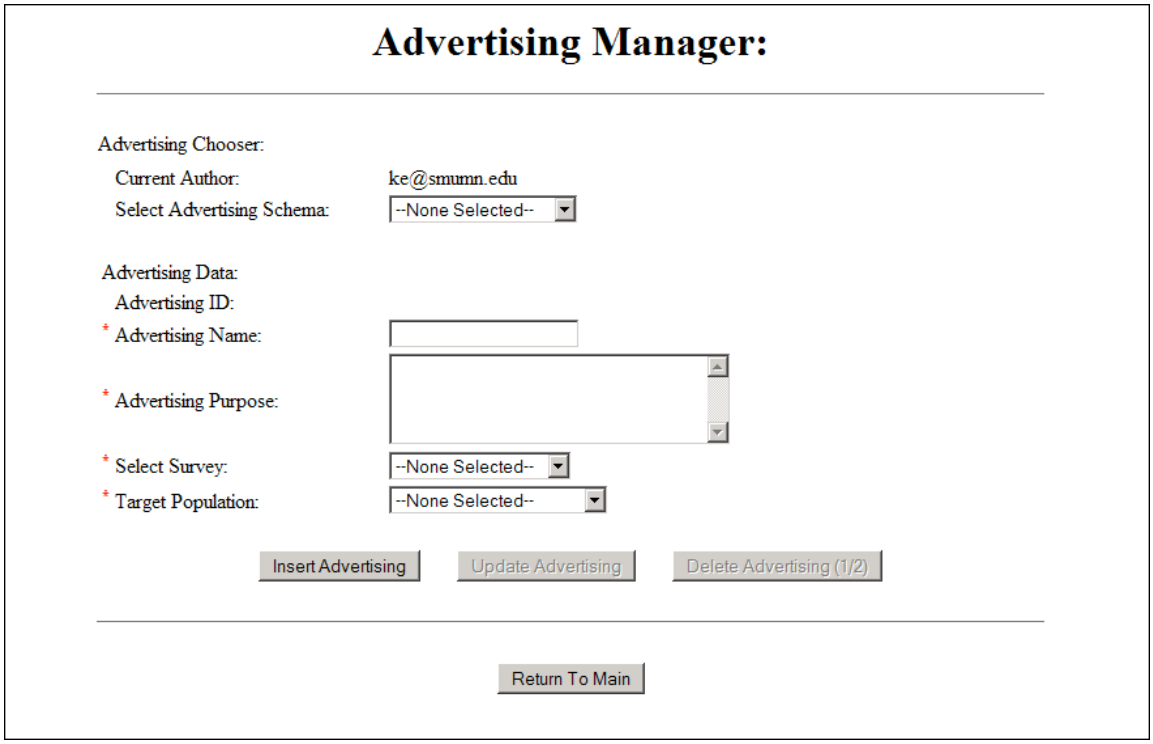

Figure B.13: Advertising Manager (Survey Publication)

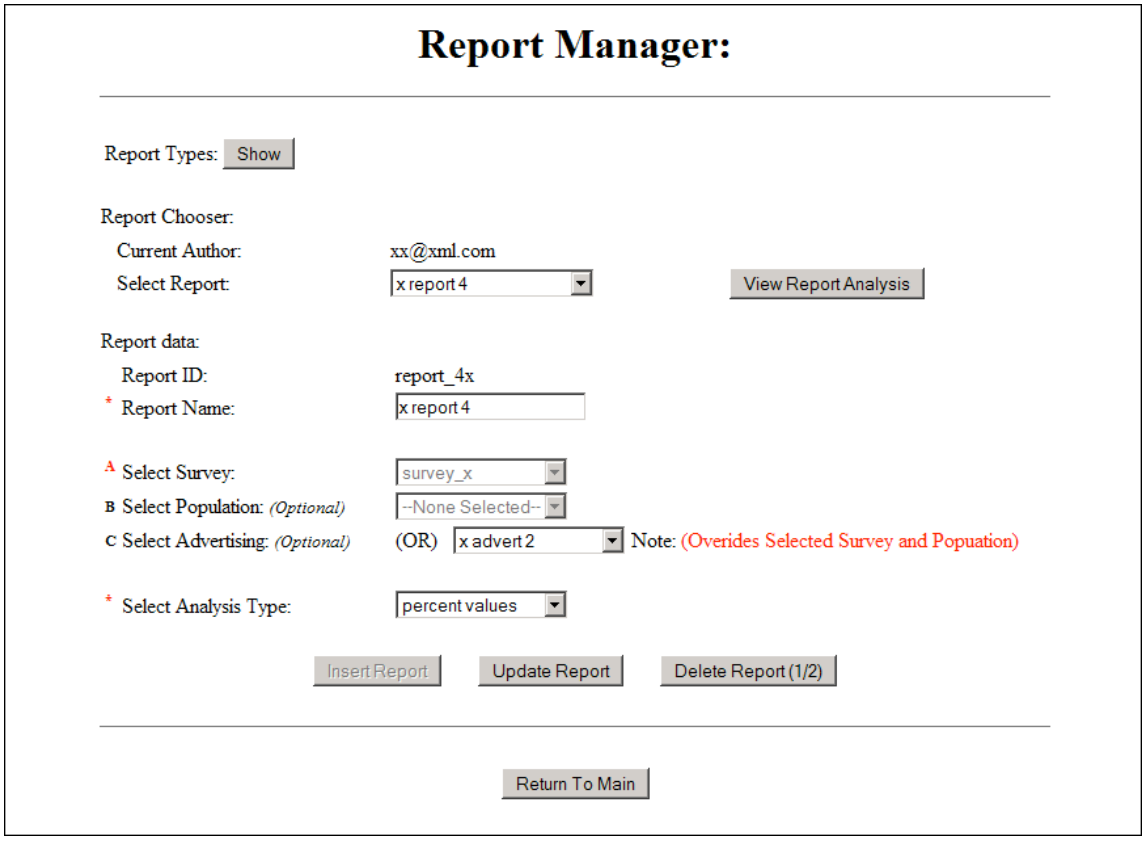

Figure B.14: Report Manager

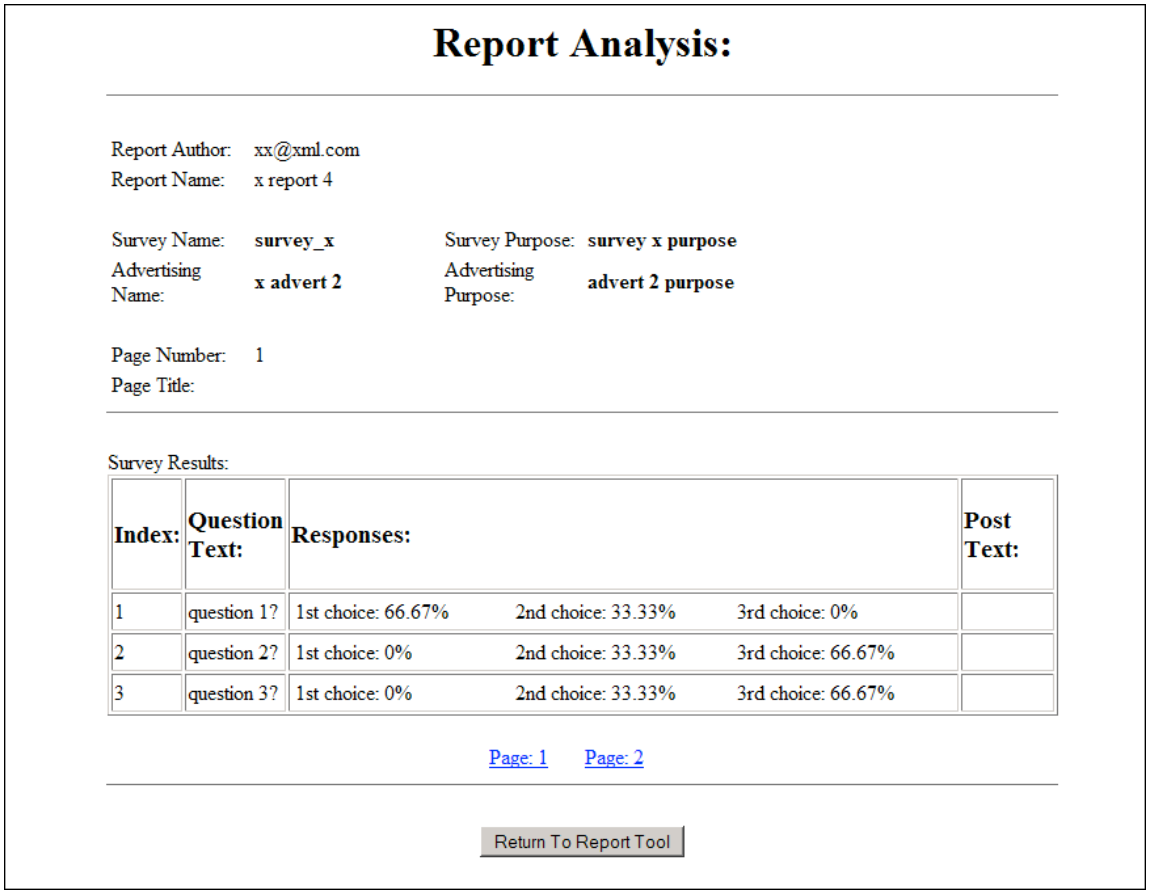

Figure B.15: Report Analysis

Figure B.16: Temp Survey Taker Login

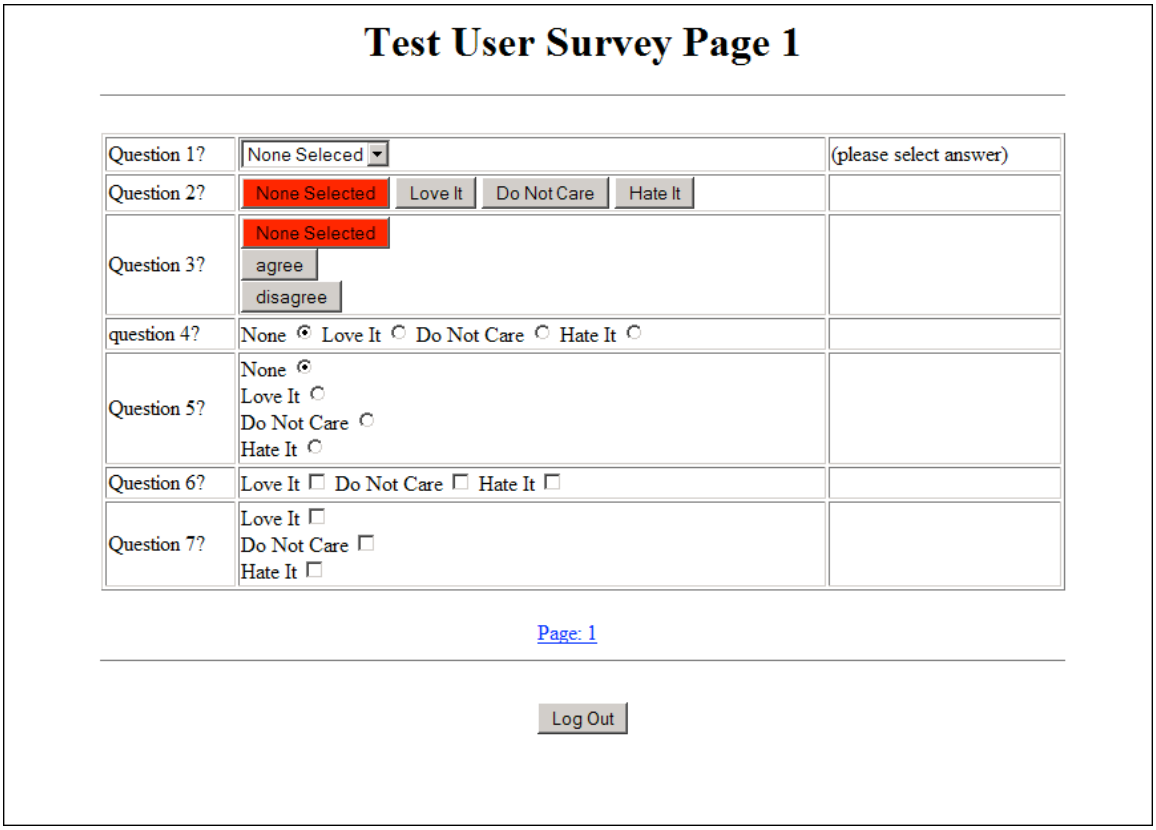

Figure B.17: Live Survey Example