ElectConnect

Final Interface Proposal
March 24, 2005

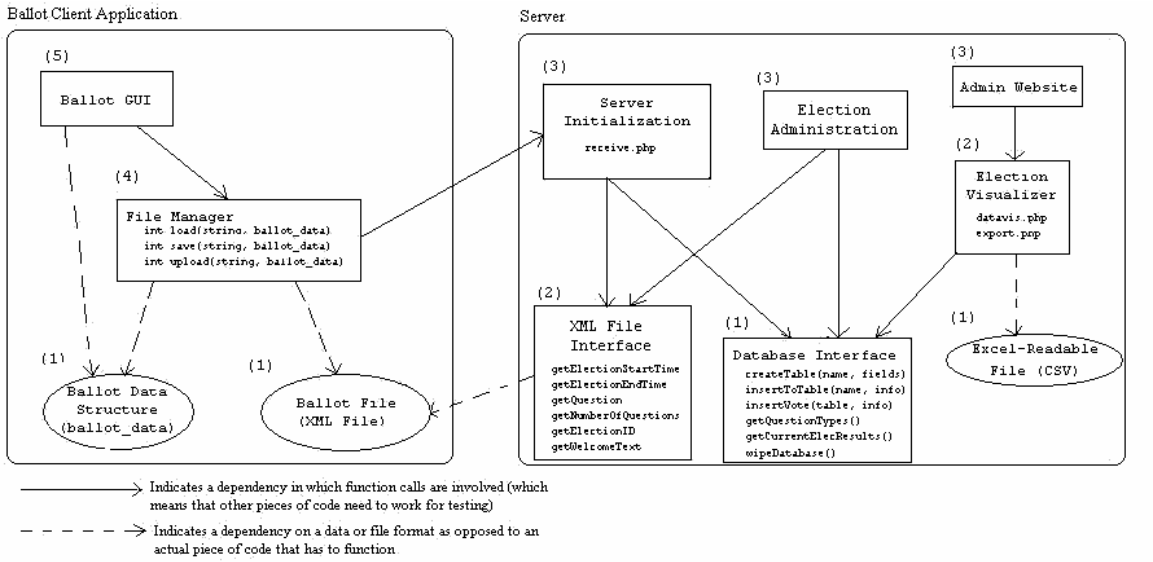
Mike Black
Xander Boutelle
Catherine Hill
Lars Johansson
Alex Kossey
Dan Silverman
Pawel Wrotek

1.0 Overview

This document provides an overview of the public interfaces for each of ElectConnect's modules. Also outlined are schedules for each component as well as a component level testing plan.

1.1 Revised Design Diagram

The diagram below represents the final high-level design of our project. We do not anticipate that this design will change.



1.2 Unchanged Interfaces

The following .h and .php files are being submitted unchanged from the original proposal:

- `Ballotdata.h`
- `FileManager.h`
- `Question.php`
- `Candidate.php`
- `Date.php`

1.3 Revised Interfaces

The following interfaces have materially changed or are being added:

`XMLFileInterface.php`

This interface has been revised to act as a XML independent (ie file-format independent) wrapper around the XML election data. The XML web component / file interface is simply an interface for extracting information out of the election xml data file. Its core is using PHP to parse the XML file and extract the necessary information about a given election to return to the user.

The user (the election-admin and some server-init) just inquires about information such as the number of questions, the start date of the election, the end date of the election, the questions that are being used etc... The XML web component parses the file and returns the answers to the user and that's that.

`DatabaseInterface.php`

Specifically, DatabaseInterface has been refined to act as a more high-level wrapper around the mySQL database.

The "database interface" is the external face of the database component. It provides functionality for wiping any data in the database and setting it up for the next election (createTable, insertIntoTable, wipeDatabase); it also gives higher-level abstractions for putting vote data into the database or reading it out (insertVote, getCurrentElecResults, and getQuestionTypes).

Server Initialization Interface

We did not discuss the server initialization interface in our original document, but think it is in fact worthy to note. Although the server initialization component has no formal interface (function calls, etc.) as other interfaces do, there is an arrow of dependency on the overall system diagram from the File Manager to the Server Initialization that needs to be explained. Over this arrow, two things need to happen: a file needs to get from the UCS computer to the server, and the Server Initialization needs to be informed that it has a new data file.

We feel that the most effective way to perform these two steps is to have an FTP server running on the CIS machine hosting the web site; the File Manager would connect to this FTP server and upload the .zip file. Then, it would contact the "receive.php" URL on the server. Once the File Manager contacts this script, its job is done. By doing so, it notifies the server that an election file has been uploaded; receive.php will take over, grab the .zip file, unpack it, and set up a database for the election.

`XML.txt`

Finally, the file `XML.txt` is submitted as well defined example of the file format we are using.

1.4 Revised Schedules

Web Election and Administration

The following is a slightly revised version of the schedule and testing plan for the web election and administration component. The original schedule and testing plan can be sound in sections 4.2 and 4.3 respectively.

Web-pages

- Vote information into database - submit.php
  - Done by: 4/13
  - Tested by: 4/16
  - Testing Plan:
    - The plan will consist of giving the script valid vote data, and allowing it to enter the data into the database. We will then retrieve the results and make sure everything is in order (using our Database class). Only one vote will happen initially, and then we will add more and more votes to ensure that nothing happens when a large number of votes are submitted. To test that script is fine with multiple people voting at once, bots will be used to submit several votes simultaneously.

- Welcome page - welcome.php
  - Done By: 4/6
  - Tested By: 4/7
  - Testing Plan:
    - Make sure that the appropriate text from the XML file is being displayed, and that the voters are linked to an appropriate page if there is no election in progress.
  - links to: Pages
    - No election
      - Done by and tested by: 4/3
      - Just plain HTML telling someone that there is no election.

- Verifying info (security) - verify.php
  - Done By: 4/8
  - Tested By: 4/13
  - Testing Plan:
    - Having group members log in and simply hit a submit button on a phony vote page to make sure that if they have not voted they can, and that after they have voted once they are not allowed to again. Make sure that this works for extended periods of time between voting and that it also works when they log out and log back in and when they simply stay logged in and keep hitting back to get to the submit page.

- Generating questions - election.php
  - Done by: 4/11
  - Tested by: 4/15
  - Testing Plan:
    - Create various sets of Questions (different types, multiple numbers etc...) and run the script on each set. View the resulting page and make sure everything appears properly. Also, have a testing script which, when the user clicks on submit, displays a page consisting of all of the answers selected and make sure that everything is there. Initially use only one question of each type to ensure that the Question's generateHTML method is working properly.

- Question verification - questionverify.php
  - Done by: 4/12
  - Tested by: 4/15
  - Testing Plan:
    - Testing consists of throwing the script all kinds of possible submissions (all valid, all invalid, a mix) with all the different question types. Initially, one question of a given type (valid or invalid) will be used to make sure that each subclass of Question has its verifyAnswers method working properly.

XML File

- xml.php
  - class that implements the XMLFile interface (in XMLFileInterface.php)
  - Done by: 4/5
  - Tested by: 4/10
  - Testing plan:
    - Make sure all methods correctly get information from various XML files
    - Start with simple files without a lot of candidates, questions etc... and make sure simple parsing (date etc...) things work
    - Build up to more complicated files that are more like actual ballot files that would be used
    - To test, output the results of the various get methods in the class to make sure the right data is returned.
    - Check the information in the Question classes that are created as well to make sure the correct instances are created for the appropriate questions in the ballot.

Ballot Creation

The schedule for the ballot creation application has been revised. As work began it became clear that it made sense to implement visual elements and logical features in

parallel rather than completing the visual aspect of the GUI and then moving on to the logical backend. The new schedule reflects this change. For reference, the original schedule can be found in section 2.2.

3/15: Empty GUI set up.
3/25: GUI and logic for adding and removing questions is complete
4/8: GUI and logic for inputting and editing general election information as well as adding candidates to questions is complete
4/16: Functionality that interacts with the parser complete (loading, saving, and uploading ballot to a server); initial integration
4/25: All required functionality complete and well tested
5/1: Full system integration

2.0 Ballot Creation

2.1 Interfaces

No other component depends on the GUI, so it has no associated interface file. The GUI depends on the ballot data structure (ballotdata.h), and the FileManager.

2.2 Schedule

3/15: Empty GUI set up.
3/25: GUI visually complete (not fully functional)
4/8: Core functionality complete (crating new election, inputting general election information, inserting new candidates and questions)
4/16: Functionality that interacts with the parser complete (loading, saving, and uploading ballot to a server); initial integration
4/25: Secondary functionality complete (moving, deleting questions)
5/1: Full system integration

2.3 Testing Plan

As each piece of functionality is added, the GUI will be tested by hand to make sure that it still looks/behaves as expected. When the functionality to interact with the parser is written (loading, saving, uploading), a dummy parser will be written with the following functionality for those three methods:

- load: Fill the ballot data structure with some known data and pass it to the GUI. Make sure that the GUI behaves correctly.
- save: Print the contents of the data structure passed by the GUI to the screen, and make sure that this corresponds to the actual data that the user tried to save.
- upload: Same as save, also print the name of the server to make sure that it is the one that the user wanted to connect to.

The GUI will call the methods of this dummy parser, and the output will be used to judge whether the GUI is handling its side of that functionality correctly

3.0 File Manager

3.1 Interfaces

The interface for the FileManager is provided by `FileManager.h`

3.2 Schedule

Note that here "testing" is used to mean finish testing.

Save/Load:

- Write save (4/8)
- Write load (4/8)
- Test save alone (4/10)
- Test load alone (4/12)
- Test load and save (4/16)

XML file:

- Do specifications for file format (3/16)
- Find a library to parse XML with: (3/16) (Behind schedule, looking for 3/18)

Uploading the file:

- Research how to create a connection to a server and run a PHP script (3/18)
- Write private zip method (4/2)
- Test private zip method (4/4)
- Write upload function (4/12)
- Test upload function (4/14)

3.3 Testing Plan

Component level testing for the file manager will be fairly straightforward. There are three methods in the interface: save, load, and upload.

Save and load depend on each other quite heavily, that is, testing either would be much easier with the other one already implemented. However, they can be tested separately.

Save: Take a simple ballot data structure and some filename to save it in, and save it, then look at it manually.

Individually: check times and other basic data

- check that saving a single question with a single candidate works
- check that adding text and a link to the question works
- check that adding text and a more info link to a candidate works

- check that adding more candidates works
- check that adding more questions works
- check that using the other types of questions works

Load will be tested very similarly to save, writing an XML file from scratch with just basic data, then a question, etc.

Upload is very, very tricky to test, because it depends on another component. Until the script it will be calling is up, I won't be able to fully test it.

4.0 Web Election Administration

4.1 Interfaces

Public interfaces for the web portion of ElectConnect are found in the following .php files:

- `DatabaseInterface.php`
- `XMLfileInterface.php`
- `Question.php`
- `Candidate.php`
- `Date.php`

The php v. c++ issue has not been completely resolved at this point, however we believe that these files provide interfaces which are closely analogous to those we would write in C++. Because these php files have been designed in a object oriented fashion, converting them to C++ would be largely a question on converting syntax.

4.2 Schedule

4/3: database.php - class that implements the Database interface (in DatabaseInterface.php)
4/4: Welcome page - welcome.php
4/6: xml.php - class that implements the XMLFile interface (in XMLFileInterface.php)
4/8: Verifying info (security) - verify.php
script to run when ballot creation has uploaded to the FTP server - reception.php
4/11: Generating questions - election.php
4/12: Question verification - questionverify.php
4/13: Vote information into database - submit.php
script to initialize the database - dbinitialize.php

4.3 Testing Plan

Web-pages

Vote information into database - submit.php

- Done by: 4/13
- Tested by: 4/16
- Testing Plan: The plan will consist of giving the script valid vote data, and allowing it to enter the data into the database. We will then retrieve the results and make sure everything is in order (using our Database class). Only one vote will happen initially, and then we will add more and more votes to ensure that nothing happens when a large number of votes are submitted. To test that script is fine with multiple people voting at once, bots will be used to submit several votes simultaneously.

Welcome page - welcome.php

- Done By: 4/4
- Tested By: 4/5
- Testing Plan: Make sure that the appropriate text from the XML file is being displayed, and that the voters are linked to an appropriate page if there is no election in progress.links to: Pages
    - No election
        - Done by and tested by: 4/3
        - Just plain HTML telling someone that there is no election.

Verifying info (security) - verify.php

- Done By: 4/8
- Tested By: 4/13
- Testing Plan: Having group members log in and simply hit a submit button on a phony vote page to make sure that if they have not voted they can, and that after they have voted once they are not allowed to again. Make sure that this works for extended periods of time between voting and that it also works when they log out and log back in and when they simply stay logged in and keep hitting back to get to the submit page.

Generating questions - election.php

- Done by: 4/11
- Tested by: 4/15
- Testing Plan: Create various sets of Questions (different types, multiple numbers etc...) and run the script on each set. View the resulting page and make sure everything appears properly. Also, have a testing script which, when the user clicks on submit, displays a page consisting of all of the answers selected and make sure that everything is there. Initially use only one question of each type to ensure that the Question's generateHTML method is working properly.

Question verification - questionverify.php

- Done by: 4/12
- Tested by: 4/15

- Testing Plan: Testing consists of throwing the script all kinds of possible submissions (all valid, all invalid, a mix) with all the different question types. Initially, one question of a given type (valid or invalid) will be used to make sure that each subclass of Question has its verifyAnswers method working properly.

Database

database.php - class that implements the Database interface (in DatabaseInterface.php)

- Done by: 4/3
- Tested by: 4/6
- Testing plan: Make sure all of the methods correctly modify the database starting with the creation/adding to database methods. Make sure mysql_query results are correct
    - these are library methods
    - Then check the getting information from the database

XML File

xml.php - class that implements the XMLFile interface (in XMLFileInterface.php)

- Done by: 4/6
- Tested by: 4/10
- Testing plan: Make sure all methods correctly get information from various XML files Start with simple files without a lot of candidates, questions etc and make sure simple parsing (date etc...) things work Build up to more complicated files that are more like actual ballot files that would be used. To test, output the results of the various get methods in the class to make sure the right data is returned. Check the information in the Question classes that are created as well to make sure the correct instances are created for the appropriate questions in the ballot.

Class Question and class Candidate (done), Question is abstract and needs classes implementing it for various question types. Both are used by xml.php

- Done by: 4/5
- Tested by: 4/8
- Testing plan:
    - First test Candidate to make sure that it doesn't mangle any data (it is just a storage class really, so nothing too complicated).
    - Second test Question (this is weird as it only is part of Question that really needs to be thoroughly tested right now)
        - a lot of methods are merely gets in the abstract class, so easily testable. The only part that needs to be tested for the XMLFile are the non-abstract methods in Question and the constructors for the subclass questions as well as the method getNumberChoices
        - The other two abstract methods are needed for the pre-submit (verification) and generation of the HTML ballot. However, the testing for these two methods should be underway at this point.

Fine tuning of the HTML generated and other small things will be more extensively later.

Initialization

script to run when ballot creation has uploaded to the FTP server - reception.php. Unzips the election file and puts it somewhere meaningful

- finished 4/8
- tested by 4/11: use any zip file containing a directory structure and check to see if the script can take the file and unpack it to the appropriate place with the appropriate structure intact.

script to initialize the database - dbinitialize.php

- finished 4/13
- tested by 4/15: use DatabaseInterface.php to test (it's supposed to be tested by 4/6). Use a dummy XML file to see if it can extract the question information (using question.php and XMLFileInterface.php, both of which are scheduled for completion before this point) from the file to create the appropriate tables in the database. Then use database.php to see if data can be inserted into and queried from these tables.

5.0 Web Data Visualization

5.1 Interfaces

The admin and data sections rely heavily on the mysql database and php (c++) server being up and running. I can implement my sections without them, but it will be very difficult to test and even further difficult to have a final integration without them working.

The other major dependency is determining the best way to secure the admin portion of the website, which I am currently doing research into.

No other component of the software depends on the data visualizer, so it does not have a public interface.

5.2 Schedule

3/25: Admin login page up and running
Admin users can log in and choose from elections already in database
Admin users can change password
Initial progress on exporting election data to Excel spreadsheet

4/11: Data exportation to Excel is done
4/17: Data visualization is done

4/25: Admin website is complete from top to bottom

5.3 Testing Plan

Admin Security:

A variety of tests will be performed to ensure Admin security. Mock users and passwords will be entered to ensure its stability.

Data Visualization

For data vis testing, a hardcoded election can be entered in the database. The data visualizer will then try displaying that data. A variety of hardcoded defective data will also be entered and the data visualizer will display the appropriate error.

Excel Exporting

To test Excel exportation, again, hardcoded election data can be entered in the database. User feedback will guide how the Excel exportation works (i.e. how data is organized and displayed in excel).