Voting System

Software Design Document

Name (s): Nikhil Srikanth, Peter Linden, Alex Bohm, Andrew Petrescu

Date: 02/23/2021

Table of Contents

1. INTRODUCTION

1.1 Purpose

The purpose of this software design document is to illustrate the overall architecture of the Voting System, as well as the process flows of counting votes for the Instant Runoff and Open Party List voting methods. This document is intended for those who are interested in maintaining this software in the future or testers who wish to ensure features are working as intended.

1.2 Scope

The Voting System will be able to determine the results of an Instant Runoff or Open Party List election within 8 minutes. It will use the information from a final ballot file (which includes important information and votes from the election) in order to determine which process flow it must follow in order to achieve the desired outcome. For media representatives and auditors, separate files will be generated in order to meet their needs. This system benefits election officials and those who are interested in the outcome of an election as it will accurately determine the winner as well as provide some useful metrics.

1.3 Overview

This document provides the necessary information regarding the Voting System's overall architecture as well as the design and flows of the important software components. An overview of the system is provided first followed by the system architecture and the design of the important components. The document finishes with the design of the interface used for the system and a requirements matrix which maps the design choices to the requirements. Section titles and subtitles are bolded and larger in font with the content that follows in a regular format.

1.4 Reference Material

This section is optional.

List any documents, if any, which were used as sources of information for the test plan.

1.5 Definitions and Acronyms

- IR - Instant Runoff
- OPL - Open Party List
- CSV - Comma separated values
- SDD - Software Design Document

2. System Overview

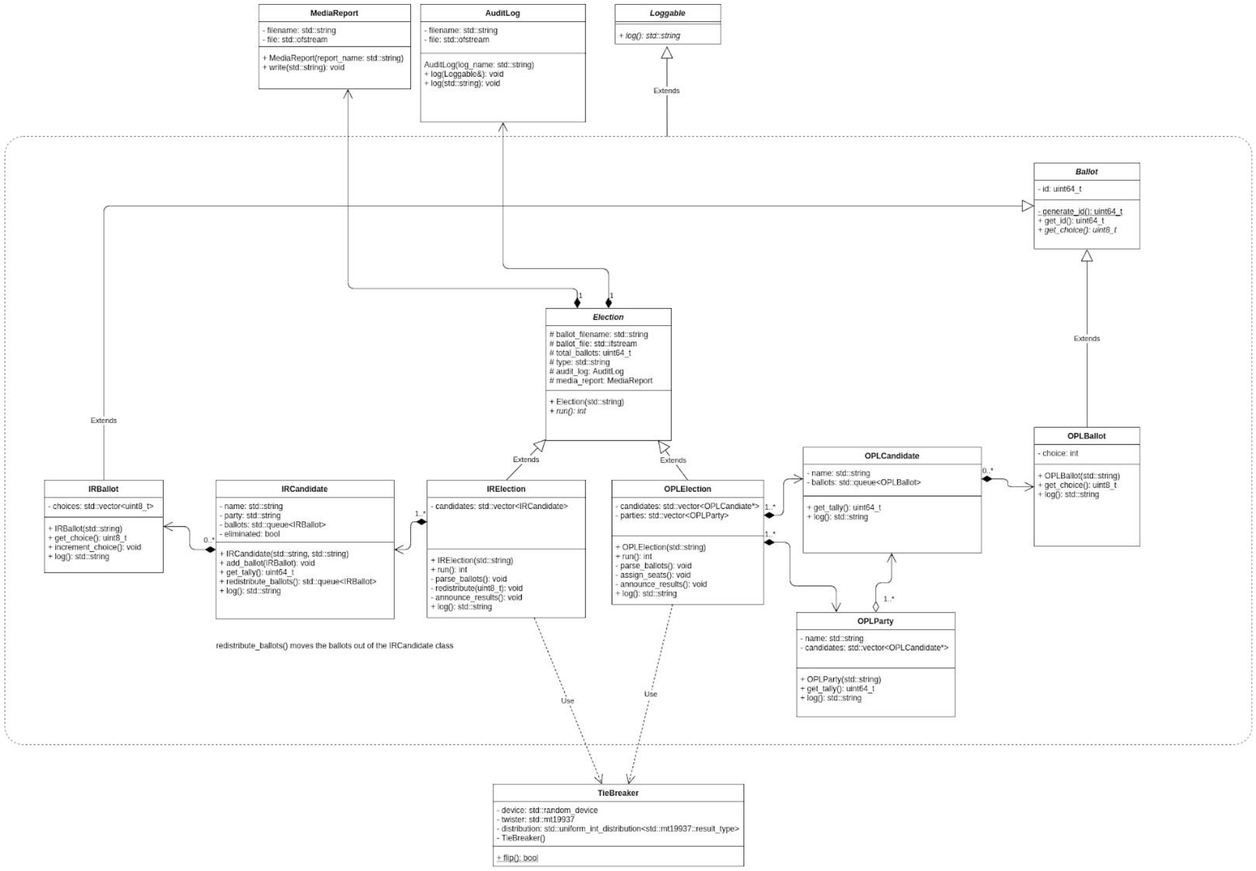
Give a general description of the functionality, context and design of your project. Provide any background information if necessary.

The Voting System is used by election officials to receive results from open party list (OPL) and instant runoff (IR) elections. The Voting System we create receives comma separated value (CSV) ballots which contain information regarding the election. When the type of election is selected, a specific algorithm will be run by the system that will analyse the ballots and retrieve information from them to produce results for the election. When the algorithm is calculating the winner of the election, the system will periodically output results to see the step by step process of how the winner was selected. The system will finally output an audit file which will contain information of the winner as well as more in depth information containing step by step processes of how the system selected the winner.

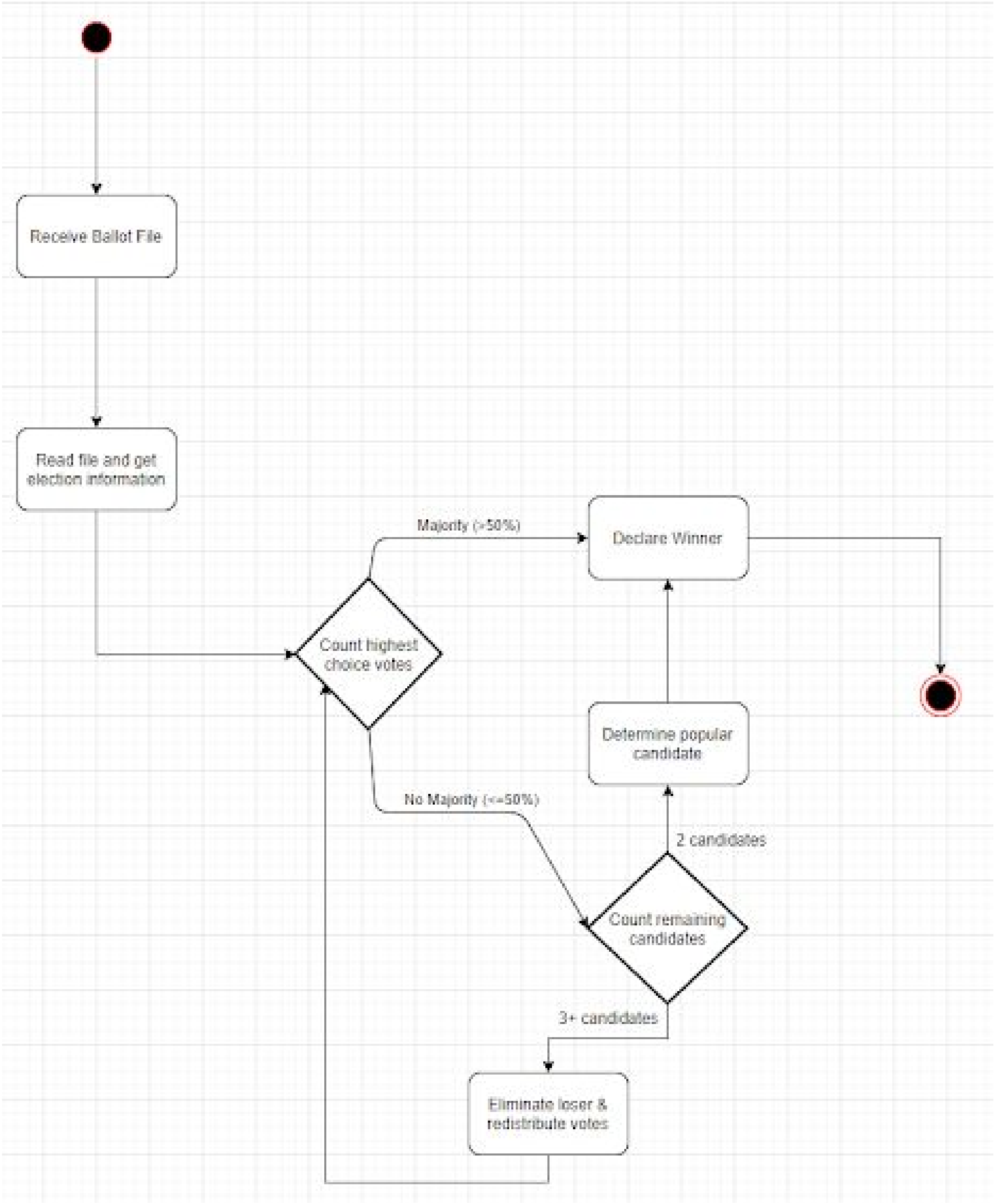
3. System Architecture

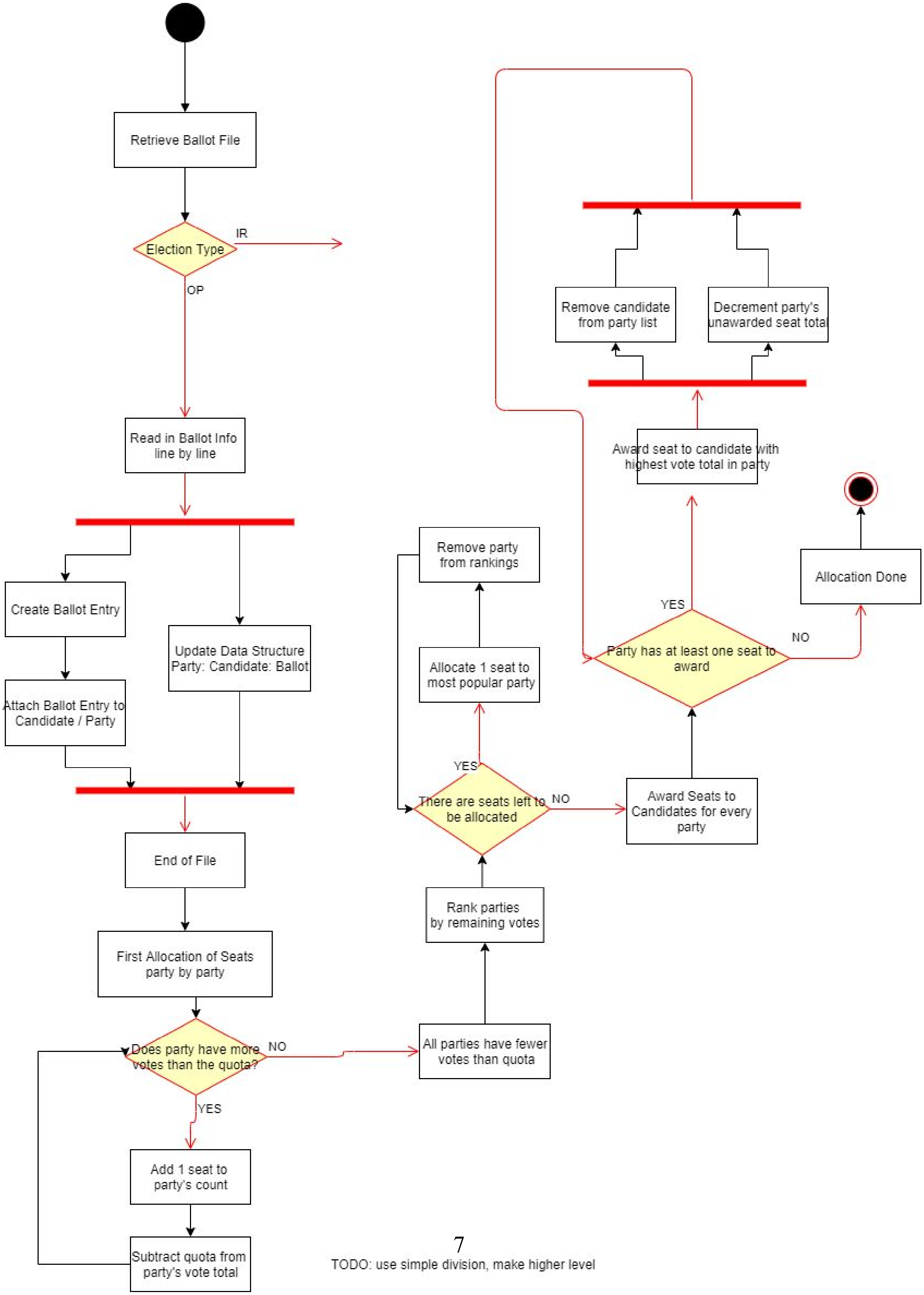
3.1 Architectural Design

Develop a modular program structure and explain the relationships between the modules to achieve the complete functionality of the system. This is a high level overview of how responsibilities of the system were partitioned and then assigned to subsystems. Identify each high level subsystem and the roles or responsibilities assigned to it. Describe how these subsystems collaborate with each other in order to achieve the desired functionality. Don't go into too much detail about the individual subsystems. The main purpose is to gain a general understanding of how and why the system was decomposed, and how the individual parts work together. Provide a diagram showing the major subsystems and data repositories and their interconnections. Describe the diagram if required.


MediaReport
- filename: std::string
- file: std::ofstream
+ MediaReport(report_name: std::string)
+ write(std::string): void
AuditLog
- filename: std::string
- file: std::ofstream
AuditLog(log_name: std::string)
+ log(Loggable&): void
+ log(std::string): void
Loggable
+ log(): std::string
Extends
Ballot
- id: uint64_t
- generate_id(): uint64_t
+ get_id(): uint64_t
+ get_choice(): uint8_t
1
1
Election
# ballot_filename: std::string
# ballot_file: std::ifstream
# total_ballots: uint64_t
# type: std::string
# audit_log: AuditLog
# media_report: MediaReport
+ Election(std::string)
+ run(): int
Extends
Extends
Extends
Extends
OPLBallot
- choice: int
+ OPLBallot(std::string)
+ get_choice(): uint8_t
+ log(): std::string
OPLCandidate
- name: std::string
- ballots: std::queue<OPLBallot>
+ get_tally(): uint64_t
+ log(): std::string
0..*
IRBallot
- choices: std::vector<uint8_t>
+ IRBallot(std::string)
+ get_choice(): uint8_t
+ increment_choice(): void
+ log(): std::string
0..*
IRCandidate
- name: std::string
- party: std::string
- ballots: std::queue<IRBallot>
- eliminated: bool
+ IRCandidate(std::string, std::string)
+ add_ballot(IRBallot): void
+ get_tally(): uint64_t
+ redistribute_ballots(): std::queue<IRBallot>
+ log(): std::string
1..*
IRElection
- candidates: std::vector<IRCandidate>
+ IRElection(std::string)
+ run(): int
- parse_ballots(): void
- redistribute(uint8_t): void
- announce_results(): void
+ log(): std::string
OPLElection
- candidates: std::vector<OPLCandiate*>
- parties: std::vector<OPLParty>
+ OPLElection(std::string)
+ run(): int
- parse_ballots(): void
- assign_seats(): void
- announce_results(): void
+ log(): std::string
1..*
1..*
1..*
OPLParty
- name: std::string
- candidates: std::vector<OPLCandidate*>
+ OPLParty(std::string)
+ get_tally(): uint64_t
+ log(): std::string
redistribute_ballots() moves the ballots out of the IRCandidate class
Use
Use
TieBreaker
- device: std::random_device
- twister: std::mt19937
- distribution: std::uniform_int_distribution<std::mt19937::result_type>
- TieBreaker()
+ flip(): bool

3.1.1 IR Activity Diagram

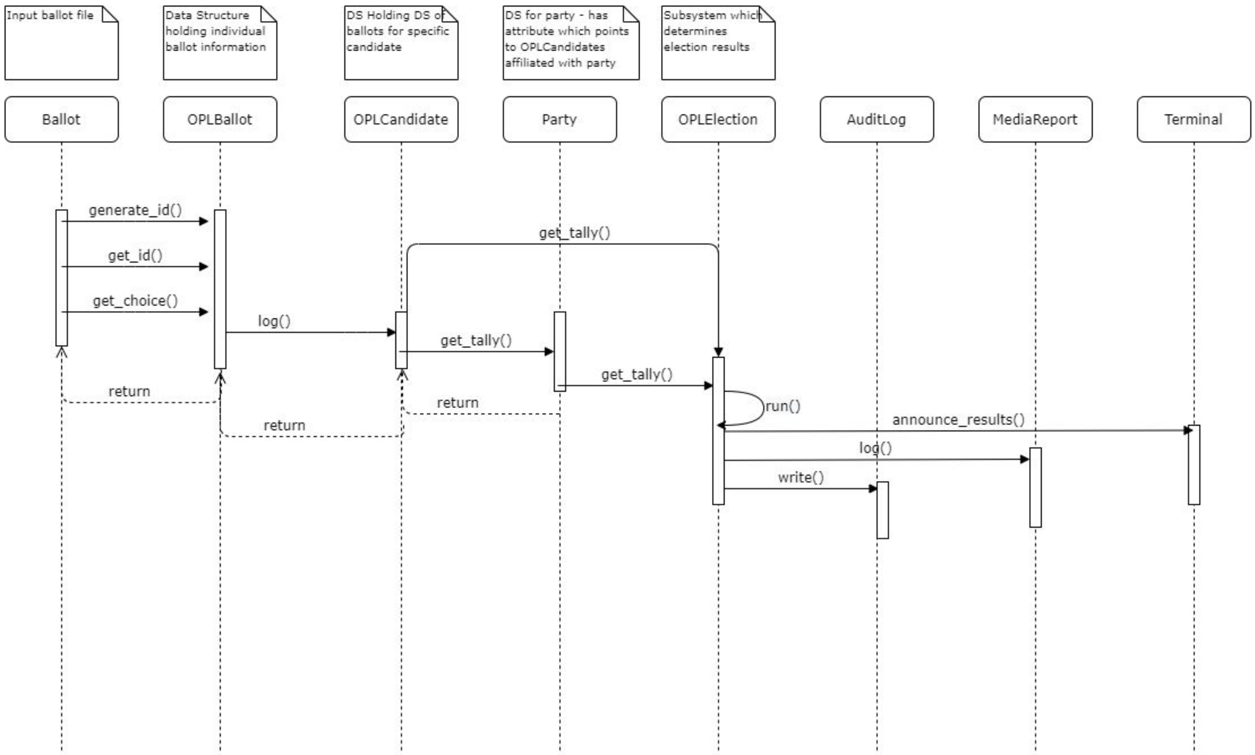

Receive Ballot File
Read file and get election information
Count highest choice votes
Majority (>50%)
Declare Winner
No Majority (<=50%)
Count remaining candidates
2 candidates
Determine popular candidate
3+ candidates
Eliminate loser & redistribute votes


Retrieve Ballot File
Election Type
IR
OP
Read in Ballot Info line by line
Create Ballot Entry
Attach Ballot Entry to Candidate / Party
Update Data Structure Party: Candidate: Ballot
End of File
First Allocation of Seats party by party
Does party have more votes than the quota?
YES
NO
Add 1 seat to party's count
Subtract quota from party's vote total
All parties have fewer votes than quota
Rank parties by remaining votes
There are seats left to be allocated
YES
NO
Allocate 1 seat to most popular party
Remove party from rankings
Award Seats to Candidates for every party
Party has at least one seat to award
YES
NO
Award seat to candidate with highest vote total in party
Remove candidate from party list
Decrement party's unawarded seat total
Allocation Done

TODO: use simple division, make higher level

3.2 Decomposition Description

Provide a decomposition of the subsystems in the architectural design. Supplement with text as needed. You may choose to give a functional description or an object oriented description. For a functional description, put a top level data flow diagram (DFD) and structural decomposition diagrams. For an OO description, put subsystem model, object diagrams, generalization hierarchy diagram(s) (if any), aggregation hierarchy diagram(s) (if any), interface specifications, and sequence diagrams here.


Input ballot file
Data Structure holding individual ballot information
DS Holding DS of ballots for specific candidate
DS for party - has attribute which points to OPLCandidates affiliated with party
Subsystem which determines election results
Ballot
OPLBallot
OPLCandidate
Party
OPLElection
AuditLog
MediaReport
Terminal
generate_id()
get_tally()
get_id()
get_choice()
log()
get_tally()
get_tally()
return
return
return
run()
announce_results()
log()
write()

3.3 Design Rationale

Discuss the rationale for selecting the architecture described in 3.1 including critical issues and trade/offs that were considered. You may discuss other architectures that were considered, provided that you explain why you didn't choose them.

4. Data Design

4.1 Data Description

Explain how the information domain of your system is transformed into data structures. Describe how the major data or system entities are stored, processed and organized. List any databases or data storage items.

4.2 Data Dictionary

Alphabetically list the system entities or major data along with their types and descriptions. If you provided a functional description in Section 3.2, list all the functions and function parameters. If you provided an OO description, list the objects and its attributes, methods and method parameters.

<<System>>

# ballots, # candidates

OPL:

ballots, candidates, parties

IR:

ballots, candidates

Auditor:

result

<<Ballot>>

id

OPL Ballot:

IR Ballot:

Party:

name, candidates

<<Candidate>>

name, votes

OPL Candidate:

IR Candidate:

5. Component Design (Sequence diagrams?)

In this section, we take a closer look at what each component does in a more systematic way. If you gave a functional description in section 3.2, provide a summary of your algorithm for each function listed in 3.2 in procedural description language (PDL) or pseudocode. If you gave an

OO description, summarize each object member function for all the objects listed in 3.2 in PDL or pseudocode. Describe any local data when necessary.

6. Human Interface Design

6.1 Overview of User Interface

Describe the functionality of the system from the user's perspective. Explain how the user will be able to use your system to complete all the expected features and the feedback information that will be displayed for the user.

6.2 Screen Images

Display screenshots showing the interface from the user's perspective. These can be hand drawn or you can use an automated drawing tool. Just make them as accurate as possible. (Graph paper works well.)

6.3 Screen Objects and Actions

A discussion of screen objects and actions associated with those objects.

7. Requirements Matrix

Provide a crossreference that traces components and data structures to the requirements in your SRS document.

Use a tabular format to show which system components satisfy each of the functional requirements from the SRS. Refer to the functional requirements by the numbers/codes that you gave them in the SRS.

8. Appendices

This section is optional.
Appendices may be included, either directly or by reference, to provide supporting details that could aid in the understanding of the Software Design Document.