Verifying the Danish Voting System

By Ólavur Kjølbro

IT-University of Copenhagen, April 2011

Supervisors:
Dermot Cochran, Ph.D. student
Joseph R. Kiniry, Associate Professor

Student:
Ólavur Kjølbro
Birthday-code: 270377-olkj

Abstract

English

The Danish electoral law has been verified in this project. The electoral law is translated from the Danish law text via informal and formal Business Object Notation (BON) to a fully functional Java implementation with refinement Java Modeling Language (JML) annotations. The name of this system is DiVS. DiVS covers all the procedures from the Election Day and through to the final counting and computations – except for a few aspects. The soundness of DiVS is made on basis of an outlined proof of correctness, since a mechanical proof done by static checker tool is not possible today, when considering Java 1.6 implementations.

Danish

Den danske valglov er blevet verificeret i dette projekt. Lovteksten er oversat fra den danske version via Business Object Notation (BON) til en fuldt ud fungerende Java implementering forfinet med JML annotationer. Systemet er døbt DiVS. DiVS dækker alle aspekter vedrørende valget fra valgdagen og indtil fintællingen og den dertilhørende resultatudregning er lavet – bortset fra nogle få aspekter. Integriteten af DiVS er lavet på basis af et manuelt bevis for korrekthed, eftersom et mekanisk bevis assisteret af værktøj ikke er muligt i dag for Java 1.6 implementeringer.

Acknowledgements

I would like first of all to thank my supervisors Dermot Cochran and Joseph R. Kiniry. Thank you, Dermot, for always being available to help me out with my questions. You made impossible issues seem feasible and sometimes easy to solve. Thanks for sending me very relevant links on various occasions that I hadn't asked for. I could not have done it without you. Thank you, Joe, for being able to help me out with issues very hard to solve. Your inspiring lecture videos from spring semester 2010 have been of invaluable use, which I could not have done without. I would also like to thank co-students Siemen Baader and Martin Hansen for our good discussions that we had. Your useful feedback comments have contributed a great deal to my project. Last but not least, I would like to thank Joel Cole for correcting the English. Your work has improved my English skills a lot.

Copenhagen April 29th 2011

Ólavur Kjølbro

Table of Contents

Introduction

Denmark currently uses paper ballots in parliamentary and other elections. After the polls close, the votes are counted by hand. Administration for the election to parliament involves a 5-level system that assures accurate vote counting and that the right people are elected. A non open source program residing in a computer in the Ministry of the Interior assists with these complex computations[1]. This implies that Denmark, at least to some degree, is already using electronic voting (e-voting).

Before any form of e-voting is introduced anywhere, the voting system itself must be verified. This has not yet been done with the Danish voting system; i.e. no mathematical model of the Danish voting system has been checked for soundness against potential inconsistencies in the law or matters that need to be clarified. There are techniques for verifying any system. The Irish voting system (Vótail) and the tally component of the Dutch voting system (KOA) have been verified using the aforementioned techniques. After checking for soundness a higher confidence in the implementation can be achieved by thoroughly testing the system for all possible input.

Research question

Considering the things mentioned above here is the research question:

"What does a verified implementation of the Danish voting system look like?"

Method

In order to answer the question 5 tasks that must be completed:

1. Write a formal requirement analysis of the Danish voting system
2. Design and formally specify the Danish voting system according to the aforementioned requirement analysis
3. Implement the proposed design
4. Check the soundness of the specification
5. Describe the level of confidence that can be had in the implemented system

The 1st task will be handled by carefully analyzing the Danish Electoral Law text and translating it into informal Business Object Notation (BON). The 2nd will be fulfilled by translating informal BON into formal BON. The 3rd task will be implemented in Java refined with Java Modeling Language (JML) annotations. The soundness in task 4 will be checked using a static checker tool. This should be done, ideally, on both the Java skeleton with empty methods bodies and the finished implementation. Alternatively an outlined proof of correctness must be done. The 5th will involve analyzing the transitions between every step from electoral law text to the implementation, as well as analyzing the soundness and the test results combined.

[1] [Computerworld]

Outcome

The outcome of this project will be an API (divs.jar) that everyone can download for free from the projects repository on the web (cf. subsection 3.1.3). The code is released with a MIT license.

Report Structure

The main report has been structured after the tasks defined above. Chapter 1 contains the analysis as well as the informal BON. Chapter 2 deals with the design which includes the formal BON. Chapter 3 documents the implementation phase. Chapter 4 contains an outlined proof of program correctness. The documentation of the tests can be found in chapter 5. The conclusion can be found at the end of the main report.

1 Analysis

This chapter contains a thorough analysis of current Danish electoral law. Material written in English exists that describes and illustrates the election procedures in Denmark in details. However, the Danish text has been reviewed to ensure that nothing is lost in translation. From the analysis informal BON will be made as requirement specifications. The result of this chapter will be a transition of the law text into informal BON.

This analysis will cover the election process in Denmark, but not those in Greenland and the Faroe Islands, since the latter two follow different rules. What will mainly be considered are the procedures from the Election Day and till the final results are computed, since that is where this project has as its focus.

Chapter Outlines

This section contains a glossary (1.0.1) that the reader can use to become familiar with terms used throughout this project. The analysis can be found in the next section (1.1). From the analysis state charts can be derived (1.2). In addition to the analysis this section also includes a description of the architecture (1.3). It contains a description of the infrastructure of the system. The last section in this chapter is 'System Requirements' (1.4) which contains the actual voting system implementation requirements.

1.0.1 Glossary

The glossary contains terms and concepts used throughout the project. In order to minimize the risk of misunderstandings, many terms are taken from the 'Parliamentary Electoral System in Denmark'[2]. To further avoid any possible misunderstandings, the Danish term has been added in parenthesis.

- The Kingdom of Denmark – consists of Denmark, Faroe Islands, and Greenland
- Folketinget – the unicameral Danish parliament
- The Ministry of the Interior – ministry in charge of the election
- The Interior Minister – minister in charge of the Ministry of the Interior appointed by the Prime Minister
- Voter – a citizen of Denmark that is 18 years or older
- National Level – body in charge of holding the election. Currently (2011) it is the Ministry of the Interior
- Election Day – one particular date on which the election is held
- Regions – Denmark is divided into 5 regions. The regions do not have anything to do with the election procedures
- Province – Denmark is divided into three electoral provinces (short: province; Danish: landsdel)
- Constituency – each province is subdivided into many multi-member constituencies (Danish: storkreds)
- District – each constituency is further subdivided into nomination districts (Danish: opstillingskreds)
- Municipal Council – Denmark consists of 98 municipalities, all of which have a council (Danish: kommunalbestyrelse)
- Polling District – every municipal council is as a whole or partially divided into polling districts (Danish: afstemningsområde)

[2] [PESD]

- Polling Station – the physical location where voters from a certain polling district cast their votes
- The Electoral Map – consists of the provinces, the constituencies, the districts, and the polling stations
- Party – a political organization whose members all have the same aims or beliefs
- Party Letter – each party has a unique letter attached to it
- Electoral Register of Voters – a list that contains information about voters in a certain area (Danish: valgliste)
- Poll Card – a piece of paper with information sent to voters on the Electoral Register prior to the election (Danish: valgkort)
- Election Board – administration body on the national level whose responsibilities include to approve new parties (Danish: valgnævn)
- Electoral Management Board – a board that manages all districts within a Municipal Council (Danish: valgbestyrelse)
- Polling Supervisors – individuals who are responsible for controlling the election and the counting of votes (Danish: valgstyrere)
- Poll Book – a register in which all essential information about the election in a certain area is recorded
- Appointed Electors – voters appointed to assist at the Polling Station (Danish: tilforordnede vælgere)
- Candidate – a Danish citizen eligible for public office. A candidate is either independent or belong to a party
- Quota – the pure Hare quota used in computation step 3
- List Organizations – a party can run either standing-by-district or standing-in-parallel
- Standing-by-district – each district has its own candidate
- Party List – standing-by-district can be combined with a party list where party in a constituency registers a fixed order for its candidates
- Distributional Number – a value used when a party in a constituency uses a party list
- Standing-in-parallel – all the party's candidate are presented in all districts
- Ballot – a paper ballot on which the voter can cast his/her vote
- Polling Booth – place where voter with sufficient privacy can cast his/her vote on a ballot by marking his/her preferred candidate or party. There are 1 or more booths per polling station
- Advance Voting – voters unable to show up personally at their Polling Station on election day can cast their votes according to this procedure (Danish: brevstemme)
- Vote Casting – a voter can cast his/her vote by showing up personally at his/her Polling Station or by using Advance Voting
- Invalid Vote – a vote declared invalid
- Valid Vote – a vote not declared invalid
- Party Vote – a valid vote casted for a party
- Personal Vote – a valid vote casted for a candidate
- Preliminary counting – counting of votes on Election Night after all Polling Stations have closed
- Final counting – final meticulous (re)counting of votes
- Constituency Seat – a certain number of seats belong to every constituency (Danish: kredsmandat)

- Additional Seat – additional seats are used to level unevenness (Danish: tillægsmandat)[3]
- Threshold – a party must pass the electoral Threshold in order to gain a share of the Additional Seats (Danish: spærregrænse)
- Election Certificate – all elected candidates get a certificate from the Ministry of the Interior
- Substitute List – a list of non-elected candidates used if an elected individual cannot fulfill his/her responsibilities, e.g. a leave of absence, death etc.
- 5-level Administration System – consists of the national level, constituency level, district level, municipality level, and polling district level
- d'Hondt Method – is a divisor method. The divisors are 1,2,3,4, …
- Sainte-Laguë Method – is a divisor method. The divisors are 1,3,5,7, …
- The Danish Method – is a divisor method. The divisors are 1,4,7,10, …
- Quotient – a value resulting from a division of a dividend and a divisor
- Fraction – a real number value between 0 and 1
- Div – div is short for division. It has a quotient resulting from the number of votes divided by a divisor according to a certain divisor method
- Infrastructure – network structure consisting of e.g. servers, computers, and links between them
- Web Server – a computer program that serves content such as web pages
- Java – object oriented programming language released in 1995
- BON – acronym for Business Object Notation. BON is a method in the software engineering discipline developed in 1989-1993.
- JML – short for Java Modeling Language.
- E-voting – a term encompassing the electronic means of casting or counting of votes
- DRE Machine – acronym for Direct Recording Electronics. A machine used for vote casting
- VCE Machine – acronym for Vote Counting Electronics. A vote counting machine
- Remote Internet Voting – a concept where voters can cast their votes through the Internet by using a Web Browser
- Electronic Submission of Results – the electronically submitted results from a polling station to the Ministry of the Interior
- Voting System – is a method by which voters make a choice between options. The most common voting systems are: majority rule, proportional representation (PR), and plurality voting
- Eclipse IDE – free and open source development environment
- DIA – tool for drawing UML diagrams, ER diagrams, flow charts, etc.
- LAN – local area network
- GUI – graphical user interface
- https – http combined with SSL/TLS (added security)
- JAR – short for Java ARchive. File extension for e.g. Java libraries

[3] 'compensatory' has been left out deliberately, since it resembles 'constituency' too much. In coding phase it would cause unnecessary confusions

1.1 The Danish Electoral Law

Danish Electoral Law covers elections to the Folketing and referendum voting. It uses principles in the Danish Constitution chapter IV which states, among other things, that the election shall be by proportional representation and, when allocating seats to the electoral map, must pay attention to the number of inhabitants, the number of voters, and the population density[4].

1.1.1 Composition of the Law

Danish electoral law consists of three sections. Of these, only the first section is of particular interest for this project. Section II and III deal with fees and entry into force respectively.

1.1.2 Election to the Folketing

This project will consider election to the Folketing and not referendum voting, and although the focus of this project is not on the administration procedures of the election, a brief mention of them will be made.

Right of Voting and Eligibility

The right to vote is reserved for all people that are 18 years of age or older with a Danish citizenship. All people that have the right to vote are eligible to run for office[5].

Election Period

Elected seats are occupied for 4 years unless a new election is called before the term is up[6].

Number of Seats

The Folketing has a total of 179 seats of which the Faroe Islands and Greenland have 2 seats each[7]. 175 seats belong to Denmark.

The Electoral Map of Denmark

The electoral map of Denmark consists of three provinces. These are: Metropolitan Copenhagen, Sealand and Southern Denmark, and Northern and Central Jutland. Denmark is also divided into 10 constituencies; four constituencies belong to Metropolitan Copenhagen, three belong to Sealand and Southern Denmark, and three belonging to Northern and Central Jutland. The constituencies are in turn subdivided into a total of 92 districts. See the complete list in Appendix C.3.

Every municipality is, as a whole or partially, in a district divided into a polling district[8]. Every polling district has one single polling station[9]. Hence there might be several polling stations per district.

Of the 175 seats in Denmark, 135 of them are constituency seats and 40 are additional seats. The distribution of seats to the provinces and the constituencies are set every 5 years starting from the 1st of January 2010.

---

[4] [DC] § 31 stk. 3
[5] [DC] § 29, § 30
[6] [DEL] §6 stk. 1
[7] [DEL] §7 stk. 1
[8] [DEL] §9 stk. 1
[9] [DEL] §45 stk. 1

Before the election, it is determined exactly how many constituency and additional seats each province will get as well as how many constituency seats each constituency will get. The distribution is based on 1) population 2) number of voters in the last election and 3) area in square kilometers times 20. If the constituency 'Bornholms Storkreds' does not get at least 2 constituency seats it is still guaranteed 2 constituency seats[10].

Political Parties

The political parties that run for the election are the parties that won seats in the last election. New parties can run for election by following certain rules[11]. The Ministry of the Interior assigns the parties a unique, so-called party letter[12].

Electoral Register of Voters

All voters are registered in the electoral register of voters in the municipality where they live. Furthermore, one list is maintained per each polling district (i.e. on list per polling station[13].

Poll Card

All voters on the electoral register of voters get a poll card sent to their registered address of residence.

Municipal Council

There are 98 municipalities in Denmark which are elected every 4 years. The municipal council's tasks are to edit the electoral register of voters and send poll cards to the voters.

Election Board

The election board is an administration board appointed by the Interior Minister. Its task is to decide if a voter can be on the electoral register of voters or not, approve voter's declarations for new parties, and edit the register of parties.

Electoral Management Board, Polling Supervisors, and Appointed Electors

One electoral management board is appointed in every district. In larger municipalities one electoral management board is appointed for all the districts within the municipality borders. Polling supervisors and appointed electors are appointed to administer the election at the polling station. There are 5-9 polling supervisors per polling station.

Candidates

A person desiring a candidacy in a district can either run as a candidate for a party or as an independent candidate. Candidates running for a party must be approved by the party. No candidate can run in more than one constituency, and no candidate can run for more than one party or both for a party and as an independent candidate[14]. Registration of candidates must contain name, CPR no, occupation, and address[15].

---

[10] [DEL] §10
[11] [DEL] §11 stk. 1 and 2
[12] [DEL] §14
[13] [DEL] §15, §18 stk. 1
[14] [DEL] §32 stk. 1 and 2

List Organizations

A party can use either the standing-by-district or standing-in-parallel list organizations[16]. This has implications on the how the party votes are distributed to the candidates (cf. 'Ballots' below).

Standing-by-district means that one candidate is nominated in each district. This implies that the nominated candidate appears as the first candidate in the district's ballot under the party for which he/she is running. All other candidates are listed alphabetically. All party votes in the district are appointed to the nominated candidate in that district. Candidates elected status is determined by their vote count (most votes first, least votes last), except when a party list has been registered[17] (cf. candidate selection on p. 20).

Standing-in-parallel means that any given party nominates several candidates per district. Candidates are listed alphabetically on the ballot. The party can register a certain candidate to appear on top of the party's list of candidates on the ballot. The party votes are distributed between the candidates in the same ratio the candidate has personal votes. For instance, if a candidate in a district gets 50% of all personal votes, then the candidate must get a 50% share of the party votes in that district on top of his/her personal votes. Candidates elected status is determined by their vote count (most votes first, least votes last)[18].

Ballots

The electoral management board makes sure that enough paper ballots are provided for each polling station[19]. Parties on the ballot are ordered alphabetically by their party letter. Party candidates are listed according to party's list organization. Independent candidates are placed last on the ballot. A voter must be able to vote for either a party or an independent candidate. If a voter casts his/her vote for a party, the voter can choose either to cast a personal vote or a party vote[20].

Election Day

The Municipality Councils must provide polling stations with ample voting booths and ballot boxes. Voting booths must enable the voter to cast a vote anonymously[21]. A ballot box must be designed so that no ballots can be taken out without opening the box.

Voting starts at 9:00 and continues until 20:00. Before the voting starts, appointed electors must inspect the ballot boxes and confirm that they are empty. After this has been done the ballot boxes are locked or sealed. The polling stations close after 20:00 when no voter indicates that he/she wants to vote[22].

A voter can cast his/her vote by showing up at a polling station. Upon demand, a voter must provide his/her name, address, and date of birth. If the identity of the voter is questioned, the voter must provide

---

[15] [DEL] §33a stk. 2
[16] [DEL] §38
[17] [DEL] §39
[18] [DEL] §40
[19] [DEL] §43 stk. 1, §44
[20] [DEL] §43, §68
[21] [CAD] § 31 stk. 1
[22] [DEL] §46

documentation if necessary. When the voter has been marked off the electoral register of voters, he/she gets a paper ballot[23].

A voter can vote for either a party or a candidate. After the vote has been cast the voter puts the ballot in the sealed ballot box[24].

Advance Voting

Any voter unable to show up at the polling station on the Election Day can use advance voting at any Population Registrar office in the country[25]. Other people e.g. patients at a hospital may also use advance voting[26]. Advance voting may normally occur beginning three weeks before the Election Day and up to the second to last weekday before the Election Day[27].

Preliminary Vote Counting

When the polling stations close, the votes are counted by polling supervisors and appointed electors. Both normal ballots and advance votes are considered. The counting is public. It is determined how many votes each party received and how many votes the independent candidates received. A vote cast for a candidate belonging to a party is regarded as a vote for the party[28]. A vote can be considered invalid according to specific rules. Advance voting can be considered invalid according to specific rules[29]. Personal votes are not necessarily counted immediately after the election. This step, for the purposes of this project, is referred to as the preliminary counting (cf. state charts).

When the counting is completed and the results are entered into the poll book, the book is signed by the election controllers. The results are announced to the people present at the polling station. The head of the election controllers must immediately forward the results to head of the electoral management board[30]. When the head of the electoral management board has received the results from all polling stations in a district, the votes are added together. Those results must immediate be submitted to the Ministry of the Interior, including the breakdown of how many votes each party has and how many votes each independent candidate receives[31]. This allows that the Ministry of the Interior to calculate the preliminary results of the election late on election night. These results are only partial, since there is, at this point, no basis to select the candidates for the parties.

Final Vote Counting

No later than one day after the Election Day the electoral management board administrates a final computation of the results from each district. The municipal council makes sure beforehand that the poll

[23] [DEL] §47
[24] [DEL] §48
[25] [DEL] §53
[26] [DEL] §54
[27] [DEL] §56
[28] [DEL] §68
[29] [DEL] §69
[30] [DEL] §70
[31] [DEL] §71

books, the electoral registers of voters, the ballots and the advance voting material are present[32]. The final computation of the results includes a final meticulous (re)count and assessment of the votes in the area that the electoral management board is in charge of. The results are entered in the poll book. Votes are counted for each party and independent candidates as well as the votes for parties are divided into party votes and personal votes for candidates. Thereafter the party votes are distributed among the candidates according to the list organization of the party. If two or more candidates have just claim on a party vote, then lots are drawn. This means that lots are drawn on the district level before the results are submitted to the Ministry of the Interior. The results of the whole district are entered into the poll book and announced to the people present[33].

The head of the electoral management board submits a copy of the poll book to the Ministry of the Interior. The copy must be certified by the head of the electoral management board. The Interior Minister decides how the submission procedure is done and can decide if and how the poll book is submitted electronically to the Ministry of the Interior[34].

Computation of the Results of the Election

When the Ministry of the Interior has received all certified copies of the poll books, the final results of the election are computed[35]. In short this is done in a six-step manner:

1. Allocation of constituency seats
2. Determination of the electoral threshold
3. Allocation of additional seats on the national level
4. Allocation of additional seats on the provincial level
5. Allocation of additional seats on the constituency level
6. Selection of candidates

1. Allocation of Constituency Seats

For each party and for each independent candidate within a constituency the votes are added up. The number of votes for each party and every independent candidate are thereafter divided by 1, 2, 3 etc. (the d'Hondt method) until there are, for each, a number of quotients that corresponds to constituency seats allocated to the constituency before-hand. The party or candidate with the highest quotient gets the first seat, the second highest gets the second seats, and so forth. If two or more quotients are equal then lots are drawn[36].

2. Determine the Electoral Threshold

The electoral threshold is determined on the national level for each party. Only the parties passing this threshold will get additional seats. The threshold is passed if at least one of the following is fulfilled 1) the party receives at least one constituency seat 2) the party obtains in two out of three provinces, a number of votes at

[32] [DEL] §72
[33] [DEL] §73
[34] [DEL] §74
[35] [DEL] §75
[36] [DEL] §76

least corresponding to the average valid votes per constituency seat within province 3) the party receives at least 2.0 percent of all valid votes nationally[37].

3. Allocation of Additional Seats on the National Level

On the national level, all valid party votes passing the threshold are added up. The sum is divided by 175 less the country total of independent candidates that have received a constituency seat. This produces the quota (cf. glossary). On the national level the total votes for the passing parties is now divided by the quota. The produced quotients (without fractions) show how many seats the parties should get as a minimum; there might be 1 more. If all 175 seats are not taken, then the parties get seats allocated using the largest fractions rule. If two or more fractions are of equal size, then lots are drawn. If a party receives more constituency seats in total than the aforementioned allocation justifies, then the calculation is redone, leaving out the parties that have exactly enough seats or too many seats. The number of additional seats is found by subtracting the total number of seats by the total number of constituency seats[38]. The calculations are redone until the parties considered don't receive too many constituency seats compared to what these calculations allow.

4. Allocation of Additional Seats on the Provincial Level

The total number of votes in each province is added up for each party passing the threshold. These sums are divided by 1, 3, 5 etc. (the Sainte-Laguë method) creating numerous quotients that must correspond to the number of constituency seats that the party has plus 1. The first few quotients are crossed out so that there is only one left per party per province. The province and party that has the largest quotient gets the first additional seat, the province and party with the second highest quotient gets the second seat, etc. When a province or a party gets the number of additional seats that it should, then the province or party are put to rest, meaning that the province or party are left out of future considerations in the computation. The allocation continues until all provinces and all parties get their additional seats. If a party has not received votes in all provinces, then it must beforehand get its seats in the provinces it received votes[39]. If there are ties between provinces and/or parties then lots are drawn[40].

5. Allocation of Additional Seats to Constituencies

The total number of votes in each constituency are divided by 1, 4, 7 etc. (the Danish method) for each party creating numerous quotients. These quotients that must correspond to the number of constituency seats that the party has plus 1. The first few quotients are crossed out so that there is only one left per party per constituency. The constituency with the highest quotient gets the first additional seat; the second highest quotient gets the second additional seat, etc., until all additional seats are allocated to constituencies. If there are ties between constituencies, then lots are drawn[41].

[37] [DEL] §77 stk. 1
[38] [DEL] §77 stk. 3
[39] [DEL] §78
[40] [DEL] §79 stk. 3
[41] [DEL] §79

6a. Selection of Candidates

The candidates of a party are selected on basis of the vote count. For every constituency, the votes for the candidate are added up (including possible party votes). Candidates are elected in the order of votes received (most votes first, least votes last). If two or more candidates get the same number of votes, then lots are drawn. If a party does not have sufficient candidates nominated in a constituency, then unelected candidates of a constituency in the same province get elected instead. If there are no un-elected candidates left in the same province for the party, then the seats are given to the other two other provinces; the first unelected candidate in the constituency in these two provinces gets the first seat, etc. If nationally there no substitutes, then the Folketing decides what should be done[42]. If a party has registered a party list in the constituency, then the candidates are selected in the following way: 1) Party votes in the constituency are divided by the number of seats obtained in the constituency plus 1. The number thereby obtained is rounded up to the next whole number and becomes the party's distributional number[43]. 2) If a candidate after the summation of votes has achieved the distributional number or higher, then the candidate is elected. If there are many candidates that have achieved the distributional number or higher, then they are elected according to the order in the party list. 3) If, after the previous point, there still are not enough candidates elected, then the other candidates on the party list are elected in the order they appear on the party list. 4) If no candidate has obtained the distributional number, then the candidates are elected in the order they appear on the party list.

6b. List of Substitutes

The Ministry of the Interior composes a list of substitutes which consists of the candidates that didn't get elected. The list is used when e.g. a candidate becomes a part of the government. The list is composed distinctly for every party in every province. The calculations are carried out in the same way as when additional seats are allocated to constituencies (cf. '5. Allocation of Additional Seats to Provinces' above). This means that the computation starts where the aforementioned computations left off, and that the constituencies are put to rest, when there are no un-elected candidates left. The constituency with the largest quotient gets the first seat, the second largest the second seat etc. until all candidates in the province are ranked inside the province on the substitute list. If a party list has been registered then the order is determined by the order in which the candidates are on the party list[44].

Election Certificate

All the elected candidates get an election certificate from the Ministry of the Interior.

Approval of the election

The Folketing decides if the calculations that the Ministry of the Interior can be approved or it should be done again. Any voters can submit complaints addressed to the Folketing and send them to the Ministry of the Interior. Complaints must reach the Ministry of the Interior no later than the very next weekday after the Election Day. If the Folketing decides that the election in a district is invalid then the approval of elected

[42] [DEL] §81, §92 stk. 2-4
[43] [PESD] p. 12, 42
[44] [DEL] §84, §85

candidates from that constituency is postponed. The candidates are, however, temporarily considered legitimate members of the Folketing.

1.1.3 Overview of different actors in the election

From the analysis above a list of all actors is derived.

- Voter
- The Ministry of the Interior
- The Interior Minister
- Municipal Council
- Election Board
- Election Management Board
- Head of Election Management Board
- Polling Supervisor
- Head of the Polling Supervisors
- Appointed Elector
- Population Registrar
- Party
- Candidate

1.2 State Charts

Considering the information above about the election procedures on and after the Election Day state charts can be derived showing which states the election can be in. The states have been grouped together on two levels so that the result is a two-tier state machine. These are referred to as the outer and the inner state machines respectively. The former covers the election procedures, and the latter covers the computation of the results. This section could have included a state chart derived from the advance voting. This has been left out, since the votes from advance voting are counted together with the rest after the polling stations close.

The states in the state charts are capitalized in case it is a normal state representing the state that the system (the election) is in currently, e.g. 'Election Closed'. The action states, which by the way represent the transition between the states, consist of lower case letters in order to distinguish them from normal states. The actions are denoted on the charts with more rounded edges that the normal states.

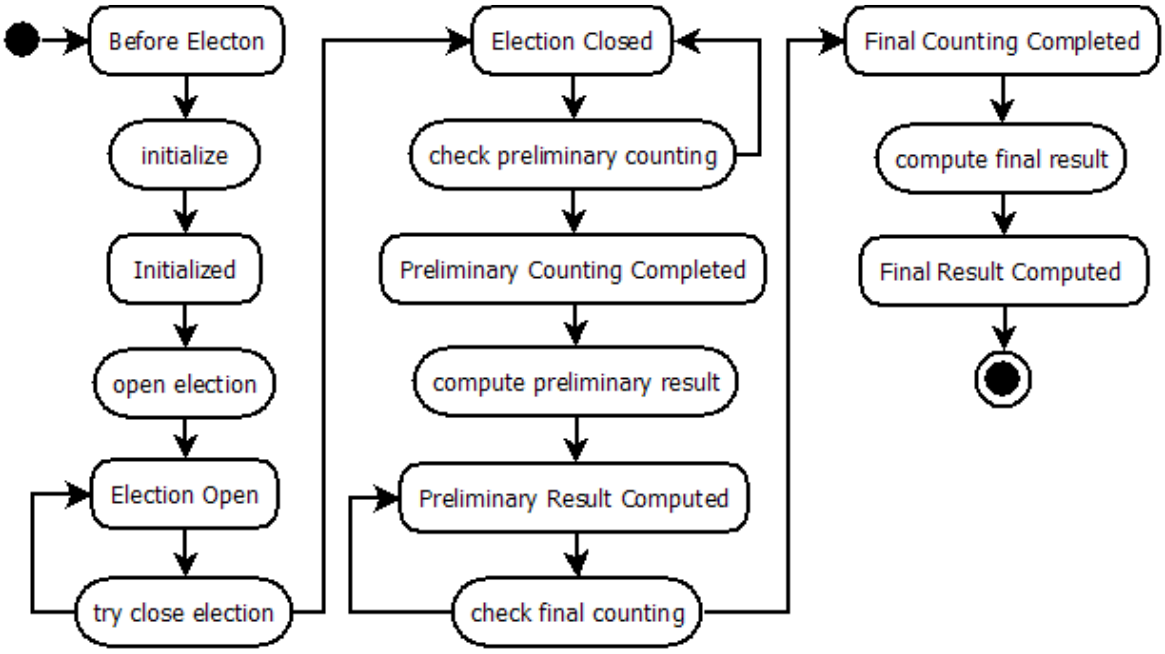


Figure 1 State chart for the election at national level for the main course of events

The states in the outer state machine are all sequential. The state 'Before Election' is a state before the election where everything is prepared. The states 'Election Open' and 'Election Closed' are on the Election Day. 'Preliminary Results Computed' is usually late on election night. The final counting starts the very next day after Election Day and the 'Final Results Computed' is usually about two days after the election.

Inside the 'compute preliminary results' and 'compute final results' action state there are more states that are shown in the chart below. The former does not reach the 'candidate selection' action, whereas the latter does.

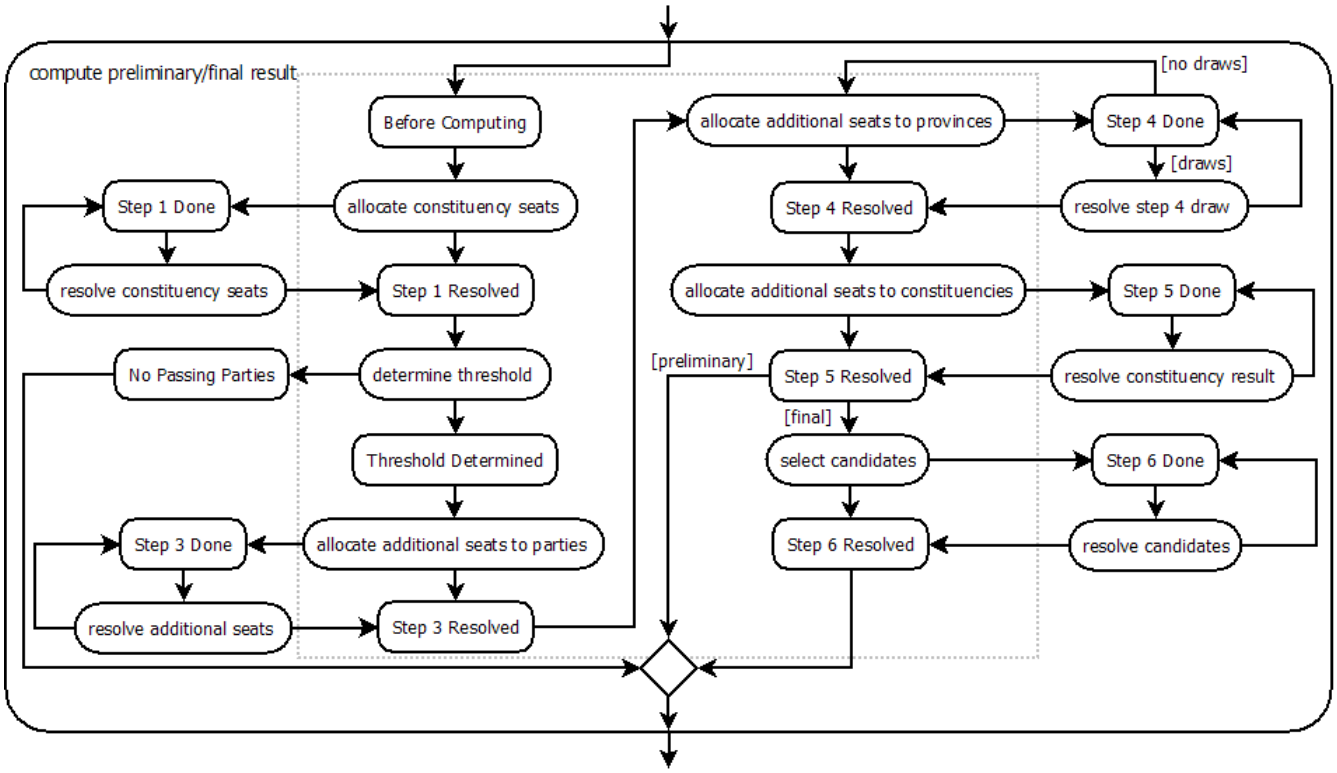


Figure 2 State chart of the inner state machine – the computations of the results

These states follow directly from the procedure described in the Danish electoral law[45]. If there are no ties that must be broken, then only the action states in the middle are used (i.e. all states/transitions inside the dotted gray square). The action states starting with the word 'resolve' are meant for resolving draw issues. According to the law, ties must be broken by drawing lots. The state 'No Passing Parties' is added, since the law does not guarantee that any party will pass the threshold. The next transition from 'Step 4 Done' is either 'resolve step 4 draw' or 'allocate additional seats to provinces' dependent on the presence of draws (cf. subsection 4.2.1 p. 64).

1.3 Architecture

The law explicitly mentions the opportunity for having the electoral register of voters maintained electronically[46], as well as submitting the results to the Ministry of the Interior electronically[47]. The law says that the Interior Minister should decide if and how this should be done. Options would be sending a spreadsheet attached to an email or logging on to a web interface with a username and password. The software system at the national level residing in the Ministry of the Interior must therefore support either

[45] §76 to §85 (except for §83)
[46] [DEL]§19, §64
[47] [DEL] §74

typing in the results through a GUI or submission of the results from a web server. Assuming the latter option the architecture or infrastructure would roughly look like this:

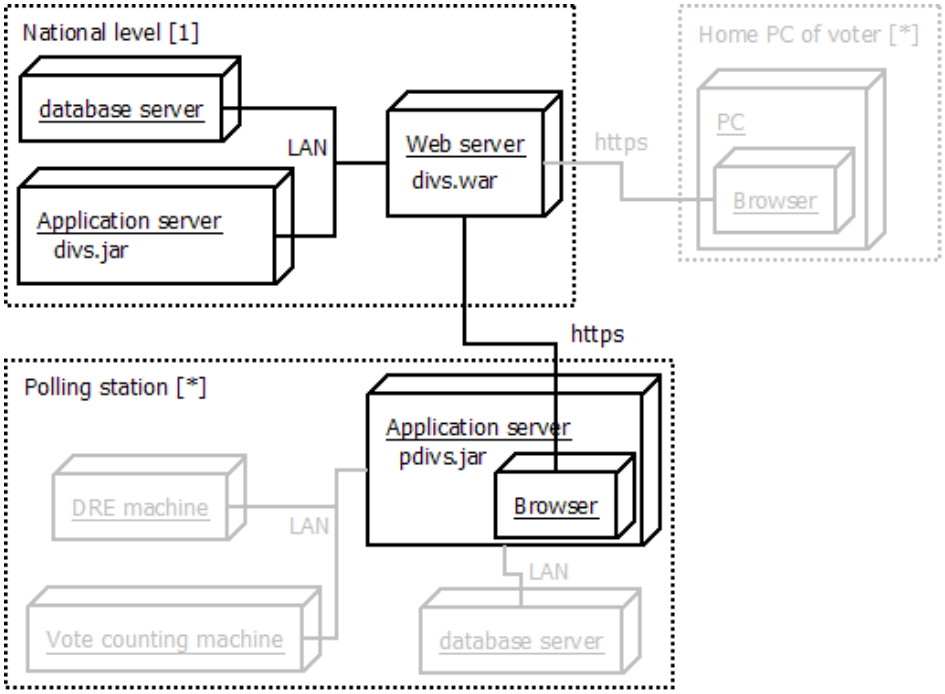


Figure 3 Deployment diagram of the architecture

The architecture is up to the Interior Minister to decide[48], and therefore it will not be discussed in much detail here. The point is to give the reader a hint of how the architecture could be like.

On the national level there are three servers connected on a LAN. The web server is the gateway to the outside world. It uses the backend on the application server, as well as the database server. The polling stations use a web browser to communicate with the national level using the https protocol. The gray areas on the diagram are possible future extensions. They are not discussed here, because they are pure speculations, which may require changes to the current electoral law.

1.4 System Requirements

This section contains the system requirements as informal BON made on basis of the analysis above. A brief introduction to BON can be seen in Appendix A. Readers unfamiliar with BON may check appendix B which contains classic requirement specifications in order to understand the BON method better.

1.4.1 Naming

The name of this project is DiVS, which is short for Danish Voting System. The name could have been DVS, which according to the website acronymfinder.com does not conflict with any similar systems or concepts, but

[48] [DEL] § 74 stk.3

DiVS with lowercase “i” coming from the “i” in Danish makes the name unique. In the computation state or phase many divisions are carried out; ‘divs’ is also short for divisions.

1.4.2 Informal BON

This section contains the informal BON. The first part will show the architecture of DiVS; i.e. the arrangement of different clusters and classes. Then all classes are described and outlined (except for the indexing clauses).

When different clusters and classes are mentioned for the first time they will be underlined. Throughout the rest of this section clusters and classes are written in capital letters, since it is a BON convention.

System chart

The system chart is called DiVS. It has one top-level cluster referring to the software system functioning only on the national level of the election.

```
system_chart DiVS --DANISH_VOTING_SYSTEM
indexing
      author: "Ólavur Kjølbro"
explanation "Procedures of the election to the Folketing"
cluster MODEL description "Model cluster on the national level"
end --DiVS
```

Clusters

A cluster is simply a collection or group of related concepts. The MODEL cluster contains a few classes and more clusters (which are underlined in this paragraph). The central classes reside inside the PROCESS cluster. These classes handle the election and computation of the election results. When the application is initialized, large amounts of data is eventually loaded into the memory of the machine running DiVS. These are mostly classes that reside inside the DATA cluster or nested clusters inside DATA. Classes used specifically in the COMPUTATION phase reside inside the COMPUTATION cluster and classes that represent the electoral map of Denmark reside in MAP cluster. It is at this point that DiVS requires a gateway to a database storing information about parties, candidates, the electoral map, and results coming in from polling stations. A cluster GATEWAYS is created, and it contains the cluster DATABASE where the database gateway resides.

```
cluster_chart MODEL
explanation "Main cluster on the national level"
class ELECTION_CONSTANTS description "Contains information about the composition /
      of the parliament, the electoral map of Denmark, different divisor methods, /
      and different list organizations"
class ELECTION_STATUS description "Contains all possible states of the election - /
      in both the outer and the inner state machines"
cluster DATA description "Contains data classes that are used in the election. /
      E.g. constituency, party, ballot etc."
cluster GATEWAYS description "Contains gateways to e.g. the database"
cluster PROCESS description "Contains the classes that handle the election and /
      calculate the election results"
end --MODEL

cluster_chart DATA
explanation "Contains data classes that are used in the election. E.g. /
```

```
      constituency, party, ballot etc."
class PARTY description "Political party that runs at the election"
class CANDIDATE description "Candidate may belong to a party or be independent"
class BALLOT description "Each polling station has its own ballot"
class BALLOT_JOURNAL description "Each ballot has a journal with both parties and /
      candidates"
cluster MAP description "Containing classes that comprises the complete electoral /
      map of Denmark"
cluster COMPUTATION description "Contains data classes that are needed in the /
      process of computing the results of the election"
end --DATA

cluster_chart MAP
explanation "Containing classes that comprises the complete electoral map of /
      Denmark"
class PROVINCE description "The Danish electoral map is divided into 3 electoral /
      provinces (short notion: province)"
class CONSTITUENCY description "Each province has three or four multi-member /
      constituencies (short notion: constituency)"
class DISTRICT description "Each multi-member constituency has many nomination /
      districts (short notion: district)"
class POLLING_STATION description "Each district has one or more polling stations /
      where people turn up and vote"
end --MAP

cluster_chart COMPUTATION
explanation "Contains data classes that are needed in the process of computing /
      the results of the election"
class DIV description "Class can be considered as holding a quotient value. It is /
      used to assist in the computation of constituency seats and when additional /
      seats are allocated to provinces and constituencies"
class CONSTITUENCY_RESULT description "Contains the final results of a /
      constituency with regards to number of total party votes and list of /
      candidates and their votes"
class PROVINCE_RESULT description "Contains the final results of a province with /
      regards to number of total party votes"
class PARTY_RESULT description "Contains information about the final results of a /
      party on different levels with regards to number of total votes and seats /
      taken"
class CANDIDATE_RESULT description "Contains votes for a certain candidate plus /
      information on ranking and elected"
end --COMPUTATION

cluster_chart GATEWAYS
cluster DATABASE description "Contains gateway to database"
end --GATEWAYS

cluster_chart DATABASE
explanation "Contains gateway to the database on the national level (at the /
      Ministry of Interior)"
class DATABASE_GATEWAY description "Gateway to the database"
end --DATABASE

cluster_chart PROCESS
```

```
explanation "Contains the classes that handle the election and calculate the /
      election results"
class GUI_INTERFACE description "Interface that informs the GUI about which /
      methods to use"
class WS_INTERFACE description "Interface that informs the web server about which /
      methods to use when polling station submits the results through a /
      web interface"
class ELECTION description "Contains methods that change the state of the /
      election on a high level"
class ELECTION_RESULT description "Contains all methods that compute the /
      preliminary and final results of the election"
end -PROCESS
```

Classes

A class or classifier represents an individual concept in the analysis. This subsection deals with the classes defined in DiVS.

ELECTION_CONSTANTS is a class that you can query only for numerous constants in the election.

```
class_chart ELECTION_CONSTANTS
explanation "Contains information about the composition of the parliament, the /
      electoral map of Denmark, different divisor methods, and different list /
      organizations"
query
      "How many seats are there in total?",
      "How many seats are allocated to Denmark?",
      "How many seats are allocated to Faroe Islands?",
      "How many seats are allocated to Greenland?",
      "How many seats are constituency seats?",
      "How many seats are additional seats?",
      "How many provinces are there in Denmark?",
      "How many constituencies are there in Denmark?",
      "How many districts are there in Denmark?",
      "How much larger is the next divisor using different divisor methods?",
      "What are the different list organization that a party can use?",
constraint
      "The number of seats in total is 179",
      "The number of seats allocated to the Faroe Islands is 2",
      "The number of seats allocated to the Greenland is 2",
      "The number of seats allocated to the Denmark is 175",
      "The number of constituency seats is 135",
      "The number of additional seats is 40",
      "There are 3 provinces in Denmark",
      "There are 10 constituencies in Denmark",
      "There are 92 districts in Denmark",
      "The d'Hondt method makes the next divisor plus 1",
      "The Sainte Laguë method makes the next divisor plus 2",
      "The Danish method makes the next divisor plus 3",
end --ELECTION_CONSTANTS
```

ELECTION_STATUS has all the different states that the election can be in the outer and the inner state machine. The class also contains the states the polling stations can be in. The reason this is important is because the

counting of votes must not start at any polling station before all the polling stations have closed. The different states are not listed explicitly here (cf. subsection 2.2).

```
class_chart ELECTION_STATUS
explanation "Contains all possible status of the inner and outer state machine"
query
      "What are the different states that the election can be in the outer state /
      machine (handling the election)?",
      "What are the different states that the election can be in the inner state /
      machine (during computation of the results)?",
      "What are the different states that polling station can have during the /
      election?"
end --ELECTION_STATUS
```

Each PROVINCE has a name and a certain number of constituency and additional seats. The number of constituency and additional seats is known before every election. The law offers no guarantee that all provinces will have a certain amount of seats. The constraint on how many seats of the two types ranges therefore from 0 to the country total.

```
class_chart PROVINCE
explanation "The Danish electoral map is divided into 3 provinces"
query
      "What is the name of the provinces?",
      "How many constituency seats does the provinces have in the parliament?",
      "How many additional seats does the provinces have in the parliament?",
constraint
      "Number of constituency seats must be between 0 and the total number of /
      constituency seats",
      "Number of additional seats must be between 0 and the total number of /
      additional seats",
end --PROVINCE
```

A CONSTITUENCY has a name, it must also belong to a province, and its number of constituency seats must be between 0 and the number that its PROVINCE has. If the CONSTITUENCY is 'Bornholms Storkreds' then it must have at least 2 constituency seats.

```
class_chart CONSTITUENCY
explanation "Each province has one or more multi-member constituencies"
query
      "What is the name of the constituency?",
      "What province does this constituency belong to?",
      "How many constituency seats does the constituency have in the parliament?",
constraint
      "Constituency must belong to a province",
      "Number of constituency seats must be between 0 and the number of /
      constituency seats belonging to province",
      "If the constituency is 'Bornholms Storkreds' then the minimum number of /
      constituency seats is 2"
end --CONSTITUENCY
```

A DISTRICT has a name and must belong to a CONSTITUENCY.

```
class_chart DISTRICT
explanation "Each multi-member constituency has one or more nomination districts"
query
      "What is the name of the nomination district?",
      "To what constituency does it belong?",
constraint
      "District must belong to a constituency",
end --DISTRICT
```

Each POLLING_STATION has a name, and it must belong to a DISTRICT. Each POLLING_STATION has a certain amount of voters assigned to it. The reason for including this number is that the total number of counted votes coming from the POLLING_STATION must not exceed this total. POLLING_STATION is classified an open state, that must be found in ELECTION_STATES.

```
class_chart POLLING_STATION
explanation "Each district has one or more polling stations, where voters /
      physically can cast their vote"
query
      "What's the name of the polling station?",
      "To what district does it belong?",
      "What is the 'open state' of the polling station?",
      "How many voters belong to the polling station?",
command
      "Open polling station!",
      "Close polling station!"
constraint
      "Polling station must belong to a district",
      "Number of voters must be non-negative",
      "The 'open state' is always between 'Not Open' and 'Closed' (both included)",
end --POLLING_STATION
```

Each political PARTY has a name, a party letter, a list organization and a (possibly empty) party list.

```
class_chart PARTY
explanation "Political party that runs at the election"
query
      "What is the name of the party?",
      "What is the party letter of this party?",
      "What principal form of list organization does it use?",
      "What does the party list look like?"
constraint
      "The list organization must be either standing-by-district or /
      standing-in-parallel",
      "If a party list is used, then list organization must be /
      standing-by-district"
end --PARTY
```

Each CANDIDATE has a name and a personal identification number in order to distinguish them from another CANDIDATE with the same name. CANDIDATE must also have a position and an address. CANDIDATE must be either belonging to a party or independent.

```
class_chart CANDIDATE
explanation "A candidate belongs to a party or is independent"
```

```
query
      "What is the name of the candidate?",
      "What is the personal identification number of the candidate?",
      "What is the position of the candidate?",
      "What is the address of the candidate?",
      "What party does the candidate belong to?",
      "Is the candidate independent?",
constraint
      "Candidate must have a name",
      "Candidate must belong to a party or be independent"
end --CANDIDATE
```

A BALLOT has a list of ballot journals (cf. next classifier). This ballot journal consists of parties, candidates of the parties, and the independent candidates. Eventually a BALLOT will contain the results of a POLLING_STATION. BALLOT must belong to a POLLING_STATION. Part of the results is the invalid vote count.

```
class_chart BALLOT
explanation "Each polling station has its own ballot"
query
      "Which polling station does this ballot belong to?",
      "What does the ballot journal look like?",
      "How many invalid votes are there?"
constraint
      "Ballot must belong to a polling station",
      "Ballot must have a non-empty ballot journal",
      "Ballot must have a non-negative number of invalid votes"
end --BALLOT
```

A BALLOT_JOURNAL represents either a party exclusive-or a candidate. BALLOT_JOURNAL contains a command enabling the number of votes to be registered after the creation of the BALLOT_JOURNAL. This vote count must be non-negative, since no one can get less than zero votes.

```
class_chart BALLOT_JOURNAL
explanation "A ballot has a list of ballot journals. Each contains either a party /
      or a candidate"
query
      "What is the party of this ballot journal?",
      "What is the candidate of this ballot journal?",
      "How many votes does this ballot journal have?"
command
      "Set the number of votes!"
constraint
      "Ballot journal has either a party on it exclusive or a candidate",
      "Number of votes must be non-negative",
end --BALLOT_JOURNAL
```

After quotients are calculated they are stored in DIVs; the term ‘div’ is short for division.

```
class_chart DIV
explanation "Class can be considered as keeping a quotient. It is used to assist /
      in the computation of constituency seats and when additional seats are /
      allocated to provinces and constituencies"
query
```

```
      "What is the quotient?",
      "What party or candidate does it represent?",
constraint
      "Value must be non-negative",
      "Party must be set xor the candidate must be set",
end --DIV
```

A PARTY_RESULT represents the results of a PARTY. The results are either on the national, CONSTITUENCY, or PROVINCE level. First of all the results refer to the number of votes the party has. PARTY_RESULT must belong to a party and the divisor method must be set to either d'Hondt, Sainte Laguë, or the Danish method. After computation step 1 the party might be involved in a draw for a constituency seat. The opponent is either another party or an independent candidate. During election results computations in step 4 and 5, the div list must be initialized using the right divisor method; the size of the div list must correspond to the number of constituency seats + 1. The analysis mentions that quotients must be crossed out. This can be achieved by always using the last element in the div list; i.e. by always using the last quotient in the div list the others *are* crossed out. PARTY_RESULT also contains information about whether a party passed the threshold and how many seats party receives.

```
class_chart PARTY_RESULT
indexing
      author: "Ólavur Kjølbro";
      created: "2010-12-09";
explanation "Class represents the party results on the national, province, and /
      constituency level"
query
      "What party is it?",
      "How many votes in total did is get?",
      "What divisor method is used?",
      "What does the div list look like?",
      "Is the party involved in a draw on the constituency level?",
      "Did it pass the threshold?",
      "How many constituency seats did it get?",
      "How many additional seats did it get?",
      "How many seats in total did it get?",
      "What is the list of candidates? (used on the constituency level)",
      "What is the party's distributional number?",
      "What does the list of substitutes look like?",
command
      "Set the number of votes!",
      "Set threshold passed status for party!",
      "Set number of constituency seats!",
      "Set number of additional seats!",
      "Add one div to div-list where the value is calculated according to the /
      right divisor method!",
      "Initialize div-list for step 4 or step 5!"
constraint
      "It must belong to a party",
      "Number of constituency seats must be between 0 and the total number of /
      constituency seats in Denmark",
      "Total number of seats must be between 0 and the total number of seats in /
      Denmark",
```

```
      "Total number of seats must equal constituency seats plus additional seats",
      "Total number of votes must be non-negative",
      "Divisor method must be d'Hondt, Sainte Laguë, or the Danish method",
      "Number of votes must be non-negative",
      "Number of constituency seats must be non-negative",
      "Number of additional seats must be greater than the number of constituency /
      seats times -1",
end --PARTY_RESULT
```

For each CONSTITUENCY there is one CONSTITUENCY_RESULT. It represents the results on the constituency level for all parties; hence it contains a list of PARTY_RESULT and a list of the independent candidates. The results for a constituency are computed in the first step when the constituency seats are allocated. The results are then used throughout the rest of the computation process, including the final step when the computation of the elected candidates is done.

```
class_chart CONSTITUENCY_RESULT
query
      "What constituency is it?",
      "How many registered voters are there in the constituency?",
      "What does the party results look like?",
      "What does the list of independent candidates look like?",
      "How many seats are taken so far?",
command
      "Create initial divisor list!"
constraint
      "Constituency must be set",
end --CONSTITUENCY_RESULT
```

For each PROVINCE there is one PROVINCE_RESULT. It represents the results on PROVINCE level for all parties. A PROVINCE_RESULT contains therefore a list of PARTY_RESULT. The number of votes can be added together as the results are received from all districts. After the threshold is computed, the number of constituency seats per province is set. In step 4 the additional seats per province are set. It does not contain a list of independent candidates.

```
class_chart PROVINCE_RESULT
indexing
      author: "Ólavur Kjølbro";
      created: "2010-12-09";
query
      "What province is it?",
      "What does the party results look like?",
constraint
      "Province must be set",
end --PROVINCE_RESULT
```

In step 6 the candidates are selected; i.e. either elected or added to the list of substitutes. This is done on basis of the personal and party votes dependent on the list organization that the party has chosen. The purpose of CANDIDATE_RESULT is to assist in these calculations. It contains a CANDIDATE (must be set), number of votes, and the elected status. After computation step 1 the candidate (when independent) can be involved in a draw for a constituency seat. The opponent is either another independent candidate or a party.

```
class_chart CANDIDATE_RESULT
query
      "To which candidate does the votes belong to?",
      "How many personal votes did the candidate get?",
      "How many party votes did the candidate get?",
      "How many votes in total did the candidate get?",
      "Is the candidate elected?",
      "Is the independent candidate involved in a draw on the constituency level?",
      "Does the candidate obtain the distributional number?",
command
      "Set the number of personal votes!",
      "Set the number of party votes!",
      "Set the elected status!",
constraint
      "The candidate must be set",
      "The number of party votes must be non-negative",
      "The number of personal votes must be non-negative",
      "Total votes must be the sum of party votes and personal votes"
end --CANDIDATE_RESULT
```

There is a need for an interface to a database or a persistence solution of some kind. The obvious reason is that it makes the system less vulnerable to power outages (cf. section 2.3). Before the election the database collects all information about electoral map of Denmark, political parties, and candidates – either party related or independent. When all the polling stations have closed the counting can commence and the results can be sent. Results are stored in the database and fetched to DiVS through the interface, DB_INTERFACE. When the results from all POLLING_STATIONs have been sent and stored in the database the computation of the results begins. I.e. the results grouped by CONSTITUENCY, PROVINCE, or PARTY are fetched from the database and made available to the class that computes the results of the election. The status of the election is not stored in the database, hence in case of a power spike the Ministry of the Interior must in DiVS go through the outer states again in order to come the same point that it was before; e.g. if DiVS was in the middle of the preliminary counting transition, then the Ministry of the Interior must open and try close the election, check counting and start computing again. The same state is then always reachable, since the data rests safely in the database.

```
class_chart DB_INTERFACE
explanation "Interface to the local database"
query
      "What does the ballot of polling station X look like?",
      "What is the elections' results on the constituency level with party votes /
      grouped together?",
      "What is the elections' results on the national level with party votes /
      grouped together?",
      "What is the elections' results on the provincial level when party votes /
      grouped together?",
      "What is the elections' results on the constituency level when personal /
      votes are grouped together?",
      "Is the preliminary results from all polling stations submitted?",
      "Is the final results from all polling stations submitted?",
      "How many valid votes were casted in total in the election?",
command
```

```
      "Register the electoral map of Denmark!",
      "Register parties, candidates and the relationship between them!",
      "Register preliminary results from polling station X!",
      "Register final results from polling station X!",
end --DB_INTERFACE
```

The GUI uses the commands in the GUI_INTERFACE to control the procedures of the election. The computation of the preliminary and the final results are done in several steps, but are not outlined here. These two computation commands correspond to 'compute preliminary results' and 'compute final results' referred to in the outer state machines (cf. figure 1 and 2).

```
class_chart GUI_INTERFACE
query
      "What was the preliminary results of the election?",
      "What was the final results of the election?",
command
      "Register electoral map of Denmark!",
      "Register parties and candidates!",
      "Initialize the election!",
      "Open the election!",
      "Try to close the election!",
      "If preliminary results is received from all districts, then preliminary /
      counting is completed!",
      "Compute preliminary results in five steps!",
      "Resolve preliminary results in all steps except for step two!",
      "If final results is received from all districts, then final counting is /
      completed!",
      "Compute final results in six steps!",
      "Resolve final results in all steps except for step two!",
end --GUI_INTERFACE
```

Assuming that the Head of the Election Management Board of a certain district submits results to the national level through a web interface, the WS_INTERFACE provides the commands that the web server needs. When the preliminary or final results are submitted, the web server uses a BALLOT and adds the invalid votes first. Then the web server adds the number of votes for each BALLOT_JOURNAL.

```
class_chart WS_INTERFACE
query
      "What is polling station no X?",
      "What is the open status of polling station no X?",
      "What ballot belongs to polling station no X?",
      "Are all polling stations closed?"
command
      "Open polling station no X!",
      "Close polling station no X!",
      "Submit the preliminary results of district no Y!",
      "Submit the final results of district no Y!",
end --WS_INTERFACE
```

The ELECTION class is the central class of the election to the Folketing. It is the outer state machine in the election. It inherits all queries and commands from GUI_INTERFACE and WS_INTERFACE, and therefore the

queries and commands are not repeated below. The state of the election must be a value defined in ELECTION_STATUS.

The real reason to include the GUI_INTERFACE and WS_INTERFACE is that although this project will not deal with implementation of a GUI or a web application, there is a need to educate a future GUI and web developer on how DiVS should be used.

```
class_chart ELECTION
explanation "Outer state machine in the election. It inherits both GUI_INTERFACE /
      and WS_INTERFACE"
query
      "What is the current state of the election?",
constraint
      "Status of the election is always between 'Before election' and /
      'Final results computed' (both included)"
end --ELECTION
```

When all necessary information becomes available and votes are counted (preliminary or final) the ELECTION_RESULT is computed. ELECTION_RESULT represents the inner state machine. The computation is done as described in the 6 steps (cf. subsection 1.1.2 p. 18-20).

```
class_chart ELECTION_RESULT
explanation "Computation of the results"
query
      "What is the inner status of the election?",
      "What does the constituency results look like with regards to parties and /
      independent candidates?",
      "What does the national results look like with regards to parties?",
      "What does the province results look like with regards to parties?",
      "What does the constituency results look like with regards to candidates?",
      "Which parties passed the threshold and which parties did not?",
      "How many valid votes are there in total?",
      "How many invalid votes are there in total?",
      "How many valid votes are there in total for parties passing the threshold?",
      "How many seats are taken so far?",
      "How many additional seats are taken so far?",
      "Has any party got more constituency seats in total than allocated seats in /
      total?",
      "How many independent candidates are elected?",
      "What do the draw issues look like after step 1, 3, 4, 5, and 6",
command
      "Inform about constituency results with regards to party votes and votes /
      for independent candidates!",
      "Inform about party votes on the national level!",
      "Inform about party votes on the provincial level!",
      "Inform about constituency results and personal votes!",
      "Set total votes casted in the election!",
      "Set total votes casted for passing parties in the election!",
      "Determine the quota from total passing votes divided by total number of /
      seats in Denmark!", --step 3
      "Step 1: allocate all constituency seats to the parties and independent /
      candidates within the constituencies!", --step 1
```

```
      "Step 2: compute the electoral threshold for each party running!",
      "Step 3: allocate additional seats to parties on the national level!",
      "Step 4: allocate additional seats on the provincial level to parties!",
      "Step 5: allocate additional seats on the constituency level!",
      "Step 6: determine which candidates have got a seat in parliament!",
      "Resolve ties after step 1, 3, 4, 5, and 6!",
constraint
      "Inner status of the election must be between before computing and results /
      computed",
end --ELECTION_RESULT
```

Scenarios

There are two main actors in the system: the Ministry of the Interior and the head of the electoral management board. These control the election procedures through the GUI_INTERFACE and WS_INTERFACE respectively.

```
scenario_chart DiVS
scenario "Edit electoral map" description "Edit the electoral map in the database"
scenario "Edit parties and candidates" description "Edit the parties and /
      candidates in the database"
scenario "Initialize the election" description "Change state to 'Initialized'"
scenario "Open the election" description "Enable polling stations to open by /
      changing state to 'Election Open'"
scenario "Try to close the election" description "If all individual polling /
      stations are closed then election is closed. Otherwise it is still open"
scenario "If preliminary results is received from all districts, then preliminary /
      counting is completed!" description "If all districts have registered their /
      preliminary results the election changes state to 'preliminary counting /
      completed'"
scenario "Compute step 1 (preliminary or final)" description "Allocate /
      constituency seats to the parties in the constituencies"
scenario "Resolve step 1 (preliminary or final)" description "Resolve draws /
      between parties for constituency seats"
scenario "Compute step 2 (preliminary or final)" description "Determine the /
      electoral threshold for the parties"
scenario "Compute step 3 (preliminary or final)" description "Allocate additional /
      seats to passing parties on the national level"
scenario "Resolve step 3 (preliminary or final)" description "Resolve draws /
      between parties for additional seats"
scenario "Compute step 4 (preliminary or final)" description "Allocate additional /
      seats to provinces"
scenario "Compute step 5 (preliminary or final)" description "Allocate additional /
      seats to constituencies"
scenario "Resolve step 5 (preliminary or final)" description "Resolve draws /
      between constituencies for additional seats"
scenario "Compute step 6 (final)" description "Select candidates for parliament /
      or substitute list"
scenario "Resolve step 6 (final)" description "Resolve draws between candidates /
      in a constituency"
scenario "If final results is received from all districts, then final counting is /
      completed!" description "If all districts have registered their final /
      results the election changes state to 'final counting completed'"
scenario "Compute the final results" description "The final results of the /
```

```
      election is computed according to the six steps of the law"
scenario "Open polling station no X" description "Let national level know that /
      polling station has opened"
scenario "Close polling station no X" description "Let national level know that /
      polling station has closed"
scenario "Submit the preliminary results of district no Y" description "Notify /
      the national level about the preliminary results of all polling stations in /
      district Y"
scenario "Submit the final results of district no Y" description "Notify the /
      national level about the final results of all polling stations in district Y"
end --scenario_chart
```

Events

The list of events is compiled by carefully looking at the scenarios and breaking the scenarios into sub-scenarios: events. The incoming events are resulting from the scenarios and involve commands only. The comment at the end of each line denotes from where the initial call came. The events are roughly ordered according to the scenarios that use them.

```
event_chart DiVS
incoming
event "Edit provinces" involves ELECTION, DB_GATEWAY
event "Edit constituencies" involves ELECTION, DB_GATEWAY
event "Edit districts" involves ELECTION, DB_GATEWAY
event "Edit polling stations" involves ELECTION, DB_GATEWAY
event "Edit parties" involves ELECTION, DB_GATEWAY
event "Edit candidates" involves ELECTION, DB_GATEWAY
event "Initialize state to 'initialized'" involves ELECTION
event "Change state to 'election open'" involves ELECTION
event "Open polling station no X" involves ELECTION, DB_GATEWAY
event "Close polling station no X" involves ELECTION, DB_GATEWAY
event "Change state to 'election closed'" involves ELECTION
event "Submit preliminary results of polling station no X"
      involves ELECTION, DB_GATEWAY
event "Change state to 'preliminary counting completed'" involves ELECTION
event "Submit final results of polling station no X" involves ELECTION, DB_GATEWAY
event "Change state to 'final counting completed'" involves ELECTION
event "Change state to 'final results computed'" involves ELECTION, ELECTION_RESULT
event "Load constituency results from database"
      involves ELECTION, DB_GATEWAY, ELECTION_RESULT
event "Load national results from database"
      involves ELECTION, DB_GATEWAY, ELECTION_RESULT
event "Load province results from database"
      involves ELECTION, DB_GATEWAY, ELECTION_RESULT
event "Load candidate results from database"
      involves ELECTION, DB_GATEWAY, ELECTION_RESULT
event "Allocate constituency seats"  involves ELECTION, ELECTION_RESULT,
      CONSTITUENCY_RESULT, PARTY_RESULT, PROVINCE_RESULT, DIV
event "Fetch draw issues for constituencies and parties (or independent /
      candidates)" involves ELECTION, ELECTION_RESULT, CONSTITUENCY_RESULT,
      PARTY_RESULT, CANDIDATE_RESULT
event "Resolve constituency results" involves ELECTION, ELECTION_RESULT,
      CONSTITUENCY_RESULT, PARTY_RESULT, PROVINCE_RESULT, DIV
event "Update national results" involves ELECTION, ELECTION_RESULT,
```

```
      CONSTITUENCY_RESULT, PARTY_RESULT, PROVINCE_RESULT, DIV
event "Determine threshold" involves ELECTION, ELECTION_RESULT,
      CONSTITUENCY_RESULT, PARTY_RESULT, PROVINCE_RESULT, DIV
event "Calculate passing votes" involves ELECTION, ELECTION_RESULT,
      CONSTITUENCY_RESULT, PARTY_RESULT, PROVINCE_RESULT, DIV
event "Determine quota" involves ELECTION, ELECTION_RESULT, CONSTITUENCY_RESULT,
      PARTY_RESULT, PROVINCE_RESULT, DIV
event "Allocate additional seats to parties" involves ELECTION, ELECTION_RESULT,
      CONSTITUENCY_RESULT, PARTY_RESULT, PROVINCE_RESULT, DIV
event "Fetch draw issue for parties" involves ELECTION, ELECTION_RESULT,
      PARTY_RESULT
event "Resolve draws between parties for additional seats" involves ELECTION,
      ELECTION_RESULT, CONSTITUENCY_RESULT, PARTY_RESULT, PROVINCE_RESULT, DIV
event "Update province results" involves ELECTION, ELECTION_RESULT,
      CONSTITUENCY_RESULT, PARTY_RESULT, PROVINCE_RESULT, DIV
event "Initialize div lists in provinces" involves ELECTION, ELECTION_RESULT,
      CONSTITUENCY_RESULT, PARTY_RESULT, PROVINCE_RESULT, DIV
event "Allocate additional seats to provinces" involves ELECTION, ELECTION_RESULT,
      CONSTITUENCY_RESULT, PARTY_RESULT, PROVINCE_RESULT, DIV
event "Fetch draw issue for parties and provinces" involves ELECTION,
      ELECTION_RESULT, PROVINCE_RESULT, PARTY_RESULT
event "Resolve draws between parties and provinces" involves ELECTION,
      ELECTION_RESULT, PROVINCE_RESULT, PARTY_RESULT, DIV
event "Initialize div lists in constituencies" involves ELECTION, ELECTION_RESULT,
      CONSTITUENCY_RESULT, PARTY_RESULT, PROVINCE_RESULT, DIV
event "Allocate additional seats to constituencies" involves ELECTION,
      ELECTION_RESULT, CONSTITUENCY_RESULT, PARTY_RESULT, PROVINCE_RESULT, DIV
event "Fetch draw issues for parties and constituencies" involves ELECTION,
      ELECTION_RESULT, CONSTITUENCY_RESULT, PARTY_RESULT, PROVINCE_RESULT
event "Resolve draws between constituencies for additional seats" involves
      ELECTION, ELECTION_RESULT, CONSTITUENCY_RESULT, PARTY_RESULT,
      PROVINCE_RESULT, DIV
event "Sort candidates descending by vote count" involves ELECTION,
      ELECTION_RESULT, CONSTITUENCY_RESULT, PARTY_RESULT, PROVINCE_RESULT, DIV
event "Select candidates" involves ELECTION, ELECTION_RESULT, CONSTITUENCY_RESULT,
      PARTY_RESULT, CANDIDATE_RESULT
event "Fetch draw issues for constituencies, parties, and candidates" involves
      ELECTION, ELECTION_RESULT, CONSTITUENCY_RESULT, PARTY_RESULT,
      CANDIDATE_RESULT
event "Resolve draws between candidates" involves ELECTION, ELECTION_RESULT,
      CONSTITUENCY_RESULT, PARTY_RESULT, CANDIDATE_RESULT
event "Select un-elected candidates for the substitute list" involves ELECTION,
      ELECTION_RESULT, CONSTITUENCY_RESULT, PARTY_RESULT, CANDIDATE_RESULT
event "Change state to 'preliminary results computed'"
      involves ELECTION, ELECTION_RESULT
event "Check if all polling stations are closed" involves ELECTION, DB_GATEWAY
event "Check if the preliminary results of all polling stations have been /
      submitted" involves ELECTION, DB_GATEWAY
event "Check if the final results of all polling stations have been submitted"
      involves ELECTION, DB_GATEWAY
event "Return the preliminary results of the election"
      involves ELECTION, ELECTION_RESULT
event "Return the final results of the election" involves ELECTION, ELECTION_RESULT
event "Return the open status of polling station no X" involves ELECTION
```

```
event "Return polling station no X"
      involves ELECTION, DB_INTERFACE, POLLING_STATION
event "Return ballot belonging to polling station no X" involves ELECTION,
      DB_INTERFACE, BALLOT, POLLING_STATION, BALLOT_JOURNAL, PARTY, CANDIDATE
end --event_chart (incoming)
```

The outgoing events are partially resulting from scenarios when technical problems arise.

```
event_chart DiVS
outgoing
indexing
      author: "Ólavur Kjølbro";
event "Database connection goes down" involves ELECTION, DB_GATEWAY
event "System is out of memory" involves ELECTION, DB_GATEWAY, ELECTION_RESULT
end --event_chart (outgoing)
```

Creation Chart

Here is the Creation Chart of the system. The DB_GATEWAY creates most instances. DIVs are only created by PARTY_RESULT. ELECTION does not get created from within the analyzed system.

```
creation_chart DiVS
explanation
      "List of classes creating objects in the system"
creator ELECTION creates DB_GATEWAY, ELECTION_RESULT
creator DB_GATEWAY creates PROVINCE, CONSTITUENCY, DISTRICT, POLLING_STATION,
      PARTY, CANDIDATE, BALLOT, BALLOT_JOURNAL, CONSTITUENCY_RESULT,
      PROVINCE_RESULT, PARTY_RESULT
creator PARTY_RESULT creates DIV
end --creation_chart
```

1.4.3 Election Coverage

During the election night, the mass media covers the election to the Folketing. The current counting on the national level and projections of the final preliminary results are shown on a regular basis on TV, websites, etc. The electoral law itself does not mention these features and therefore they are irrelevant to this project.

2 Design

On basis of the thorough analysis this chapter contains the design of DiVS. Among other things, this consists of the translation of informal BON into formal BON. Before that, there is a need to list the limitations that the formal BON will not include.

2.0.1 Limitations

DiVS is designed using the software architectural design pattern Model-View-Controller. This project will, however, only consider the model part, because the GUI does not have any scientific relevance for this project.

The complete requirements will not be designed in its entirety; from what is considered relevant in the analysis chapter. The design should, however, maintain extensibility properties so that the features left out can easily be added later (cf. section 3.5). Here are the requirements that will be left out:

- Register electoral map
- Register parties and candidates
- Party list
- Party gets more seats in a constituency than there are candidates
- Substitution list
- Lots drawn on district level when distributing party votes
- Handling rounding errors

The goal of this thesis is to verify the Danish voting system. Leaving these features out of the system does not jeopardize verification integrity. The first two are about making the design support the insertions/updates/deletions of the electoral map, parties and candidates. It does not influence the results of the verification that the registrations are supported or not. It would be nice to verify the third, fourth, and fifth features mentioned; these things are just considered the least important in this project during the computation phase on which this project focuses. The sixth limitation is not very important, but these should be registered properly. The handling of rounding errors is important, but has not been prioritized.

Chapter outlines

The first thing discussed is the model (2.1) of the project. Formal BON (2.2) designed from the informal BON is the second section. The database design (2.3) supporting the formal BON is designed last.

2.1 The Model

As mentioned previously only the model is of interest in this project. The illustration below shows the model of DiVS application on the national level. DiVS has an interface to the database, an interface to the web server, and an interface to the GUI. The web application and the GUI are shown in gray color, since they are irrelevant to this project. Just beneath the web interface and the GUI is a controller, also shown in gray color. It serves as a layer on top of the model to which the GUI and the web application connect. This must not be confused with the controller of the GUI or web application.

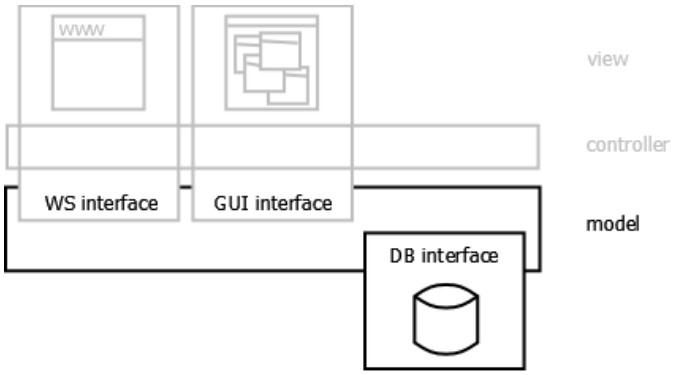


Figure 4 Model of DiVS

2.2 Formal BON

This section discusses the formal BON, which has been made in a transition from informal BON. The transition is complete, excluding those points mentioned in the introduction of this chapter. The reader can get a brief introduction to the BON method in appendix A.

There exists a tool called BONc that can translate informal BON into formal BON. BONc is an Eclipse plug-in (cf. subsection 3.1.1). Since the feature mentioned has not been released yet, it has not been used for this project.

2.2.1 Clusters

The clusters have already been introduced in the beginning of section 1.4.2. The illustration below shows the clusters and class organization:

```
NATIONAL
   C: ELECTION_CONSTANTS, ELECTION_STATES
   DATA
      C: PARTY, CANDIDATE, BALLOT, BALLOT_JOURNAL
      MAP
         C: PROVINCE, CONSTITUENCY, DISTRICT, POLLING_STATION,
      COMPUTATION
         C: PARTY_RESULT, CANDIDATE_RESULT, DIV, CONSTITUENCY_RESULT, PROVINCE_RESULT
   GATEWAYS
      DATABASE
         C: DB_INTERFACE, DATABASE_GATEWAY
   PROCESS
      C: GUI_INTERFACE, WS_INTERFACE, ELECTION, ELECTION_RESULT
```

Figure 5 Clusters and class organization

The lines prefixed by `C:` are classes separated by commas. The others are clusters. The indentation shows in which cluster the classes or clusters reside.

2.2.2 Classes and Interfaces

In this subsection the interesting concepts are shown for each class. For most classes a portion of the BON is shown in order to prove some points (see below). The order of the class will follow the order in figure 5. The

complete formal BON can be seen by checking out and examining the project from the repository (cf. subsection 3.1.3).

Most non primitive data type attributes are required and must be non void (null). Most integer values must at least be non-negative. These cases are therefore not shown below among the invariants unless needed to prove some point. The integers that end with `_id` are IDs that correspond to the unique ID or primary key in the database. The invariant on these features is always a number greater than 0. These are not shown either.

class ELECTION_CONSTANTS

All features in ELECTION_CONSTANTS are declared as integers, since the constraints on the queries all include integer value.

```
class ELECTION_CONSTANTS
feature
	…
	no_of_provinces : INTEGER
	no_of_constituencies : INTEGER
	no_of_districts : INTEGER
	…
invariant
	…
	no_of_provinces = 3;
	no_of_constituencies = 10;
	no_of_districts = 92;
	…
end --ELECTION_CONSTANTS
```

class ELECTION_STATUS

The ELECTION_STATUS only contains integers (like ELECTION_CONSTANTS does). Each feature represents a certain state for the outer and inner state machines as well as for a polling station. The values of the first states in the outer state machine are shown below. All values of states in the same state machine must be unique; otherwise the state machine would be confused about what state it was in.

```
class ELECTION_STATUS
feature
	before_election : INTEGER
	initialized : INTEGER
	election_open : INTEGER
	election_closed : INTEGER
	preliminary_counting_completed : INTEGER
	…
invariant
	before_election = 0;
	initialized = 1;
	election_open = 2;
	election_closed = 3;
```

```
        preliminary_counting_completed = 4;
        …
end --ELECTION_STATUS
```

class PARTY

The PARTY class must have its `list_organization` correspond to what is defined in the ELECTION_CONSTANT class.

```
class PARTY persistent
        …
invariant
        …
        list_organization = ELECTION_CONSTANTS.standing_by_district or
                list_organization = ELECTION_CONSTANTS.standing_in_parallel;
end --PARTY
```

class CANDIDATE

The CANDIDATE class has a `party`. When this value is `void` it means that the candidate is independent and doesn't belong to a political party.

```
class CANDIDATE persistent
feature
        …
        party : PARTY
        …
end --CANDIDATE
```

class BALLOT

The BALLOT belongs to POLLING_STATION `ps`. This might come as a surprise, since in the ER diagram it is shown that a BALLOT has a 1-to-1 relationship with a district and not a polling station (cf. section 2.3). The reason is, however, that when the results are registered, they are registered per polling stations, allowing the national level to know exactly what the results look like at each polling station, and not just on district level. The `journal` is a list of BALLOT_JOURNALs. Its parties must be ordered by `party_letter` and its candidates according to the list organization of the party. `journal` has the independent candidates located at the end.

```
class BALLOT persistent
feature
        …
        ps : POLLING_STATION
        journal : LIST[BALLOT_JOURNAL]
        …
end --BALLOT
```

class BALLOT_JOURNAL

The BALLOT_JOURNAL contains both party and candidate. However, exactly one of them has to be `void`, because each BALLOT_JOURNAL must represent either a `party` exclusive-or a `candidate`, not both. The total

votes must always correspond to the total `party_votes` plus the total `personal_votes`. A `party` can never get `personal_votes`, whereas all candidates get `party_votes` distributed between them.

```
class BALLOT_JOURNAL
feature
	…
	party : PARTY
	candidate : CANDIDATE
	total_votes : INTEGER
	party_votes : INTEGER
	personal_votes : INTEGER
invariant
	…
	party /= Void xor candidate /= Void;
	total_votes = party_votes + personal_votes;
	party /= Void -> personal_votes = 0;
end --BALLOT_JOURNAL
```

class PROVINCE

Each PROVINCE class gets a certain number of `constituency_seats` and `additional_seats` allocated before the election. Since there is no guarantee of a minimum or maximum, the invariant only states that the seat count is between 0 and the country total of constituency and additional seats respectively.

```
class PROVINCE
feature
	…
invariant
	…
	constituency_seats >= 0;
	constituency_seats <= ELECTION_CONSTANTS.constituency_seats;
	additional_seats >= 0;
	additional_seats <= ELECTION_CONSTANTS.additional_seats;
end --PROVINCE
```

class CONSTITUENCY

There is no guarantee that a CONSTITUENCY will get a certain number of seats, unless `bornholm` is true, in which case, at least 2 constituency seats are guaranteed. With this said, `constituency_seats` must not be greater than the constituency seat count of `province` to which it belongs.

```
class CONSTITUENCY
feature
	…
	province : PROVINCE
	constituency_seats : INTEGER
	bornholm : BOOLEAN
invariant
```

```
	…
	constituency_seats >= 0;
	constituency_seats <= province.get_constituency_seats;
	bornholm = true -> constituency_seats >= 2;
end --CONSTITUENCY
```

class POLLING_STATION

Each POLLING_STATION belongs to a `district`. POLLING_STATION must have an `open_state` that must be found in ELECTION_STATUS. The change of state can only be made by the features `open` and `close`.

```
class POLLING_STATION
feature
	…
	district : DISTRICT
	open_state : INTEGER
feature
	…
	open
		require
			open_state = ELECTION_STATUS.PS_NOT_OPEN;
		ensure
			open_state = ELECTION_STATUS.PS_OPEN;
			delta open_state;
		end
	close
		require
			open_state = ELECTION_STATUS.PS_OPEN;
		ensure
			open_state = ELECTION_STATUS.PS_CLOSED;
			delta open_state;
		end
invariant
	…
	district /= Void;
	open_state = ELECTION_STATUS.PS_NOT_OPEN or
		open_state = ELECTION_STATUS.PS_OPEN or
		open_state = ELECTION_STATUS.PS_CLOSED;
end --POLLING_STATION
```

class PARTY_RESULT

The PARTY_RESULT contains the feature `method` which is used during the computation process. The visible features for the computation phase are also shown below. `add_one_to_div_list` is used when initializing when constituency seats are allocated and when a seat has been taken. `create_init_divs` is used for initializing purposes when additional seats are distributed to provinces and constituencies. The former ensures that the size of `divs` is always one larger than before, and the latter ensures that the size of `divs` equals `constituency_seats` + 1. The `method` must be set to a value found in the ELECTION_CONSTANTS class.

When an instance of PARTY_RESULT is used on the constituency level, the instance of candidate_results contains CANDIDATE_RESULT elements. This is only the case in the preliminary computations. `no_of_elected` represents the number of elected candidates; it's declared for convenience reasons so that the contracts are easier to write. The `total_seats` must always equal `constituency_seats` plus `additional_seats`, but `additional_seats` might be a negative number. This is useful during computation step 3 to see if a party already has too many seats allocated.

```
class PARTY_RESULT
feature {NONE}
	…
	total_seats : INTEGER
	constituency_seats : INTEGER
	additional_seats : INTEGER
	no_of_elected : INTEGER
	candidate_results : LIST[CANDIDATE_RESULT]
	method : INTEGER
	divs : LIST[DIV]
feature
	…
	add_one_to_div_list
		ensure
			divs.count = (old divs.count) + 1;
			delta divs;
		end
	create_init_divs
		ensure
			divs.count = constituency_seats + 1;
			delta divs;
		end
invariant
	…
	total_seats = constituency_seats + additional_seats;
	constituency_seats >= 0;
	no_of_elected >= 0;
	candidate_results /= Void;
	method = ELECTION_CONSTANTS.dhondt or
		method = ELECTION_CONSTANTS.sainte_lague or
		method = ELECTION_CONSTANTS.danish_method;
	divs /= Void;
end --PARTY_RESULT
```

class CANDIDATE_RESULT

The CANDIDATE_RESULT has the feature `total_votes` which must equal `party_votes` plus `personal_votes`. `set_elected` is used for either electing `candidate` or un-electing `candidate`; initially `elected` is false for all CANDIDATE_RESULTs.

```
class CANDIDATE_RESULT
```

```
feature {NONE}
	…
	total_votes : INTEGER
	party_votes : INTEGER
	personal_votes : INTEGER
	elected : BOOLEAN
feature
	…
	set_elected
		-> e : BOOLEAN
		require
			e = false -> elected = true;
		ensure
			elected = e;
			delta elected;
		end
invariant
	…
	total_votes = personal_votes + party_votes;
end --CANDIDATE_RESULT
```

class DIV

The DIV class must have `pres` exclusive-or `cres` set. Its `ranking` can be set to a positive integer value; the `ranking` itself must be non-negative.

```
class DIV
feature {NONE}
	pres : PARTY_RESULT
	cres : CANDIDATE_RESULT
	quotient : NUMERIC
	ranking : INTEGER
feature
	…
	set_ranking
		-> rank : INTEGER
		require
			rank > 0;
		ensure
			ranking = rank;
			delta {ranking};
		end
invariant
	pres /= Void xor cres /= Void;
	quotient >= 0;
	ranking >= 0;
end --DIV
```

class CONSTITUENCY_RESULT

Each CONSTITUENCY_RESULT must have the `party results` and `independent_candidates` set to non `void`. The `seats_taken` is used to keep track during the computation phase or state how many seats are taken on the constituency level.

```
class CONSTITUENCY_RESULT
feature {NONE}
      …
      party_results : LIST[PARTY_RESULT]
      independent_candidates : LIST[CANDIDATE_RESULT]
      seats_taken : INTEGER
invariant
      …
      party_results /= Void;
      independent_candidates /= Void;
      seats_taken >= 0;
end --CONSTITUENCY_RESULT
```

class PROVINCE_RESULT

Each PROVINCE_RESULT must have the `party_results` set. The `seats_taken` is used to keep track during the computation phase on how many seats are taken on the provincial level.

```
class PROVINCE_RESULT
feature {NONE}
      …
      party_results : LIST[PARTY_RESULT]
      seats_taken : INTEGER
invariant
      …
      party_results /= Void;
      seats_taken >= 0;
end --PROVINCE_RESULT
```

class DATABASE_GATEWAY

The DATABASE_GATEWAY inherits from the DB_INTERFACE. It stores ballot results and can retrieve that data in several ways depending on what level it is on: constituency, province, or national level.

class ELECTION

The ELECTION class has a non-visible feature, `status`, declared as an integer, which represents the state that the outer state machine is in. The invariant tells what restrictions there are on `status`. ELECTION has both `preliminary_result` and `final_result` declared as features. The features with the suffix `_preliminary` deal with the preliminary results computations and the features with the suffix `_final` deal with the final results. Features starting with `computing_step_` compute in a certain step from 1 to 6. Features starting with `resolve_step_` resolve draws that the computing might have produced. Section 4.1 explains the dynamics of the election class; i.e. outer state machine.

```
effective class ELECTION
inherit GUI_INTERFACE; WS_INTERFACE
feature {NONE}
	status : INTEGER --status of outer state machine
	preliminary_result : ELECTION_RESULT
	final_result : ELECTION_RESULT
	…
feature
	…
	redefined compute_step_one_preliminary
		require
			status = ELECTION_STATUS.preliminary_counting_completed;
			preliminary_result.get_inner_status =
				ELECTION_STATUS.before_computing;
		ensure
			preliminary_result.get_inner_status =
				ELECTION_STATUS.step_1_done or
				preliminary_result.get_inner_status =
					ELECTION_STATUS.step_1_resolved;
			delta preliminary_result;
		end
	…
	redefined resolve_step_three_final
		-> pr : PARTY_RESULT
		require
			final_result.get_inner_status = ELECTION_STATUS.step_3_done;
			pr /= Void;
		ensure
			final_result.get_inner_status = ELECTION_STATUS.step_4_resolved;
			delta final_result;
		end
	…
invariant
	…
	status = ELECTION_STATUS.before_election or
		status = ELECTION_STATUS.initialized or
		status = ELECTION_STATUS.election_open or
		status = ELECTION_STATUS.election_closed or
		status = ELECTION_STATUS.preliminary_counting_completed or
		status = ELECTION_STATUS.preliminary_result_computed or
		status = ElectionStatus.final_counting_completed or
		status = ElectionStatus.final_result_computed;
end --ELECTION
```

class ELECTION_RESULT

As shown above, the ELECTION class has two instances of ELECTION_RESULT. The features that are called from ELECTION class above are shown below. Section 4.2 shows and discusses the dynamics of the inner state machines.

```
class ELECTION_RESULT
feature {NONE}
	inner_status : INTEGER --status of inner state machine
	const_result : LIST[CONSTITUENCY_RESULT]
	national_result : LIST[PARTY_RESULT]
	province_result : LIST[PROVINCE_RESULT]
	quota : NUMERIC
	total_votes : INTEGER
	…
feature
	…
	allocate_constituency_seats
		require
			inner_status = ElectionStatus.BEFORE_COMPUTING;
			const_result.count = ElectionConstants.NO_OF_CONSTITUENCIES;
		ensure
		  inner_status = inner_status = ELECTION_STATUS.step_1_done or
			inner_status = ELECTION_STATUS.step_1_resolved;
		  not isStep1Resolved -> inner_status = ELECTION_STATUS.step_1_done;
		  isStep1Resolved -> inner_status = ELECTION_STATUS.step_1_resolved;
		  delta {inner_status, const_result};
		end
	…
	resolve_additional_seats
		-> param_pr : PARTY_RESULT
		require
			inner_status = ELECTION_STATUS.step_3_done;
			param_pr /= Void;
			param_pr.is_draw;
			param_pr.get_additional_seats > 0;
			seats_total > ELECTION_CONSTANTS.seats_denmark;
		ensure
			(old seats_total) = seats_total + 1;
			(seats_total = ELECTION_CONSTANTS.seats_denmark) ->
				inner_status = ELECTION_STATUS.step_3_resolved;
			delta {inner_status, national_result, seats_total};
		end
	…
invariant
	inner_status = ELECTION_STATUS.before_computing or
		inner_status = ELECTION_STATUS.step_1_done or
		inner_status = ELECTION_STATUS.step_1_resolved or
```

```
            inner_status = ELECTION_STATUS.threshold_determined or
            inner_status = ELECTION_STATUS.step_3_done or
            inner_status = ELECTION_STATUS.step_3_resolved or
            inner_status = ELECTION_STATUS.step_4_done or
            inner_status = ELECTION_STATUS.step_4_resolved or
            inner_status = ELECTION_STATUS.step_5_done or
            inner_status = ELECTION_STATUS.step_5_resolved or
            inner_status = ELECTION_STATUS.step_6_done or
            inner_status = ELECTION_STATUS.step_6_resolved;
end --ELECTION_RESULT
```

2.3 Database design

The database that supports the system will be presented here as an ER diagram. It will clearly and unambiguously show the relationships between all entities in the database. From the ER diagram a DDL has been written.

2.3.1 ER diagram

The ER diagram is shown and discussed below:

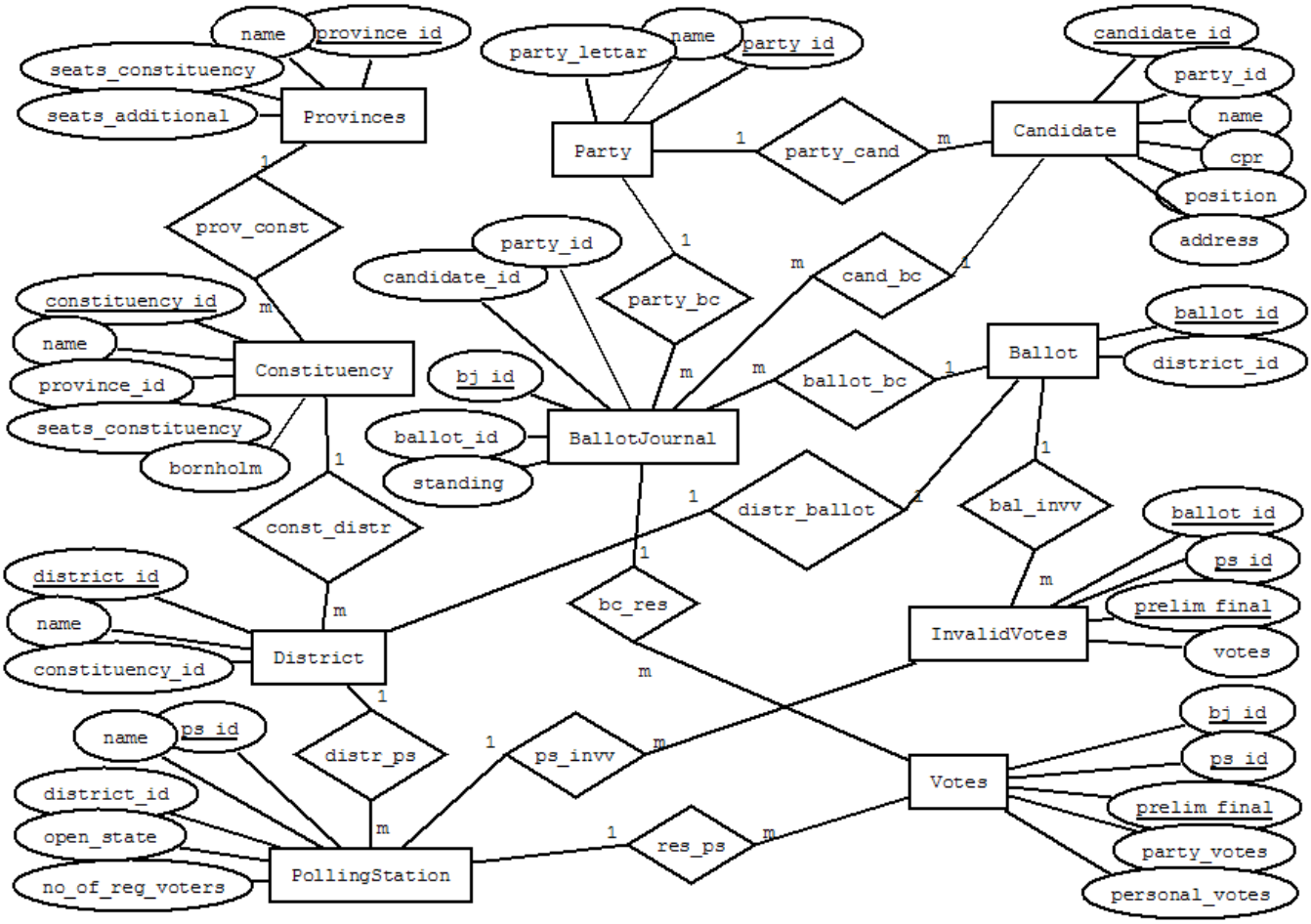


Figure 6 ER diagram

Starting in the upper left corner a `Province` has many `Constituencies`, a `Constituency` has many `Districts`, and a `District` has many `PollingStations`. A `Party` has many `Candidates`, and both `Party` and `Candidate` may be referenced from a `BallotJournal`. A `Ballot` has a 1-to-1 relationship with `District` and not with `PollingStation` (as the previous section might suggest). The reason is that in every `District` all the `Ballots` are identical. This makes the design more normalized[49]. The results of the election (i.e. the number of counted votes) belong to `Votes` and `InvalidVotes` entities. `InvalidVotes` has a many-to-1 relationship to both `Ballot` and `PollingStation`, whereas `Votes` has `PollingStation` and `BallotJournal` as its parents.

All entities have a one-attribute primary key except for `Votes` and `InvalidVotes`. This makes it easy to identify the records. The results from `PollingStation` can be either preliminary or final. This is the reason for the `prelim_final` attribute on `Votes` and `InvalidVotes` entities, which is also included in the primary key of entities mentioned.

[49] [Elmasri&Navathe07] Ch. 10

2.3.2 Data Definition Language (DDL)

The DDL can be seen in the appendix D.1. It contains creations of tables that include foreign keys and more.

3 Implementation

On basis of the formal BON this chapter will contain documentation of the implementation; i.e. it will document the transition from formal BON to Java with JML annotations. There exists a tool that ensures that the formal BON and the Java code include JML annotations that will match each other, except for a certain type of annotations. What the tool ca not do, has been done by hand.

Chapter Outlines

The chapter will start with information of practical nature regarding the tools that have been used in this project (3.1). The second section is about the transition from formal BON into Java with refinement JML annotations (3.2). The third section will discuss the soundness checking (3.3) mentioned in the introduction. The fourth section will discuss various choices made during the implementation phase (3.4). The fifth section will discuss if the design/implementation is extensible (3.5).

3.1 Language and Tools

This section contains information and discussions about which tools have been used in this project.

3.1.1 Programming Language

The programming language used for this project is the Java programming language.

3.1.2 IDE

The Eclipse IDE has been chosen, since there exist several very useful plug-ins that are very relevant to this project. These are Beetlz, BONc, ESC/Java2, OpenJML, and more. The plug-ins save much development time compared to simply using command-line tools, since the plug-ins are more usable.

3.1.3 Database

There are several different database solutions that could have been used in this project. For a project like this it is important that the database of choice is free of charge and be able to run on the most commonly used operating systems Windows, Mac OS, and Linux. Furthermore a database solution that does not require installations or additional software except for the database itself would be preferable, since it becomes more portable. MySQL is free, runs on all mentioned operating systems, but usually requires installation. H2 and Apache Derby (also known as Java DB) are all free of charge, they all support the platforms mentioned, and their disk footprint is quite low. There were some issues with H2 regarding Unicode letters although it claims to support Unicode. H2 in practice also seems to be rather slow. Apache Derby on the other hand supports Unicode and is much faster than H2. Therefore Apache Derby has been chosen.

3.1.4 Versioning System

SVN on Google Code has been chosen as the versioning system. It is free, and doesn't have any restrictions on disk space, number of developers, and more. There is however a restriction that it should be open-source software (OSS), which suites this project perfectly, since DiVS is free and open source. Google Code doesn't feature Trac, which would be nice to have for bug tracking purposes, but since there has been just one

developer on the project, the bug tracking need can easily be covered by a simple todo-list in e.g. a Word document.

All artifacts that have any relevance have been committed to the versioning system.

The URL is: http://code.google.com/p/danishvotingsystem/.

3.1.5 Organization and Management Tools

The organization of this project has been done with a Word document. The document contains a todo-list, a list of milestones, a Gant chart, and list of all questions for the supervisors. An alternative could be Google Docs. There hasn't been any need for synchronous document writing, because there has been only one developer. The document has, like all other artifacts, been submitted to the SVN.

The diagrams in this report have been drawn using the tool Dia v.0.97.1. It can be downloaded for free at: http://live.gnome.org/Dia.

3.1.6 Static Checkers

In order to check for soundness both ESC/Java2 and the ESC feature of the OpenJML plug-in package have been used; static checkers are often referred to as ESC, which stands for Extended Static Checking. Unfortunately the former does not support Java 1.6, which DiVS has been coded in. This was the direct reason for using the latter, or at least trying to use the latter. OpenJML is still under development. Although its parser and type checker are fully functional the JML compiler, JML4c, does not yet support the quantified JML expressions \forall, \exists, and \sum that are or have been used in the JML contracts. The result is that ESC is not of much use to this project.

Neither static checkers tools are complete. This means that the outcome of the static checkers might be false negatives or false positives.

The former is when the static checker reports a warning that shouldn't have been reported. These can be dealt with by reviewing the code in order to do some manual static checking. If static checker is proved wrong, then the `//@ nowarn` annotation can be used to get rid of the warning. If the static checker is proved correct, then the code needs to be revised.

The latter is when the static checker doesn't report a warning although it should. This possibility implies that one should not trust the tools blindly. The likelihood for false positives is reduced, when more JML specifications are added[50]. This is, however, a challenge when using static checkers. When OpenJML is finished implemented the idea is, that it should be complete[51].

[50] [Kiniry et.al06]
[51] [Dermot Cochran] meeting 12th of April 2011

3.1.7 Other Plug-ins Used in Eclipse

BONc can be used for type checking with regards to both informal and formal BON; i.e. checking the syntax. The plug-in can also generate BON documentation. Future releases of BONc will also be able to translate informal BON into formal BON[52].

Beetlz can be used to translate the formal BON into Java (cf. next section). The tool also supports runtime comparison of the formal BON and the Java code. When the BON and Java don't match, the user is notified by a red error marker. The tool is not finished yet; i.e. there are still many bugs, that need to be resolved, and the translation of quantifiers hasn't been implemented yet.

3.1.8 Release Number

The current release number is 0.9.1. The source folder, `src`, contains a text file with this information.

3.2 Generating the Java Skeleton

The Java skeleton containing contracts has been generated using the Beetlz plug-in. Beetlz has, as mentioned above, its limitations regarding quantified expressions. Therefore the contracts are quite spartan from what BON source contains in this process. Throughout the project's lifespan, when, for instance new classes have been added, or when a class has undertaken some non-minor changes, Beetlz' translation feature has been used. For minor changes the Beetlz runtime checker has been adequate, and the required changes have been made by hand. The runtime checker has ensured that the BON and Java match to a large extent.

The Java architecture is the same as shown in figure 5, which shows clusters and classes. Clusters in BON correspond to packages in Java, whereas classes in BON are classes or interfaces in Java.

There are, however, a few classes implemented in Java that don't exist in formal BON. These reside in the utilities package name `utils` (cf. subsection 3.4.1).

3.3 Checking the Java Skeleton for Soundness

Cf. section 4.1.

3.4 The Implementation Phase

During the implementation phase there have been various choices made, which are documented here.

3.4.1 Coding Style

It has been strived in this project to use the coding standard of KindSoftware. There might, however, be some minor differences. The choices made in this context are the following:

- The methods are all written using the camel case

[52] [Joseph R. Kiniry]

- Attributes representing several words are separated by an underscore
- Javadoc comments always precede the JML pre and post conditions. The motivation is that the code reader should be introduced to the functionality as gently as possible. It is more usable to have the natural language text first and then the formal JML contracts.
- The local declarations should precede the constructors. The constructors should precede the methods.
- The order of the methods should be public first, protected next and in private at the end. The reason is that the code reader can look from top down and more quickly see what is visible from outside
- The "getter" methods should precede the "setter" methods. This usually implies that the pure methods are listed above the non-pure methods
- The most common behavior must be stated first for when there are more than one in the JML contracts
- The JML invariants should come in-between the declarations in the top, and the first constructor. The motivation is that it is more usable to see both the declarations and the invariants without scrolling too much up and down
- "Setters" are not provided to a class if the variable that the setter represents must not change during the lifespan of the object. This has been done in order to maintain data integrity. All constructors therefore receive parameters needed so that an instance never is in an illegal state. For instance the class Party receives party_id, name, letter, standing and party list. The JML invariants on class Party have restrictions on these fields that say, that party_id must be greater than 0, name and letter should be set, and the standing must either be standing-by-district or standing-in-parallel. If the constructor didn't receive any parameters, then the responsibility would lie outside the Party class to ensure the legal state, e.g. that name would be set right after the instantiation of a Party object.
- Integers have been chosen instead of the long data type although KindSoftware advocates using long. The reason is that the integers never store any number higher than the total number of votes in an election in Denmark. Denmark has 5.5 million inhabitants which makes the probability of rollovers non-existent[53]. In the generator framework (cf. section 5.3.1), integer values can take higher values, but these are still much less that what an integer can take.
- It has been strived to make the code as DRY as possible. DRY is an acronym for 'Don't Repeat Yourself'[54].

3.4.2 Variable Naming

The naming of variables has been done consciously and consistently throughout DiVS. Most of them are self-explanatory. Below is a list the names used for the classes, together with an explanation.

Type	Declared names	Explanation
CandidateResult	cres	cres is short for candidate result
PartyResult	presn, presp, presc	pres is short for party result. Trailing n, p, or c denote that PartyResult is on the national, provincial, or constituency

[53] [KindSoftware]
[54] [Hunt&Thomas99] p.27-32

		level
ConstituencyResult	cr	cr is short for constituency result.
ProvinceResult	pr	pr is short for province result.
List<ConstituencyResult>	constituency_results	results is written in plural to denote presence of a list
List<ProvinceResult>	province_results	results is written in plural to denote presence of a list
List<PartyResult>	party_results	results is written in plural to denote presence of a list
parameters	p_	p_ is used to distinguish it from locals of same type

3.4.3 The Utility Package

The utils package contains a few classes that have a specific purpose in the Java implementation. These are discussed below.

Class Cmp

The class implements the Comparator interface parameterized to CandidateResult. Interface requires that the compare method is overridden. The Cmp class is used to sort the candidates descending by their total votes right before the candidate selection. This is done by the following line of code in ElectionClass:

```
Collections.sort(pres.getCandidateResults(), new Cmp());
```

Class DivsException

DiVS has its own exception class which is practical to have because it allows one to define a complete set of error codes with numbers that fit together in certain ranges. DivsException error codes are always two numbers. If the first digit is the same, then the exceptions belong together in the same category. For instance IO_ERROR has error code number 30 and FILE_NOT_FOUND 31. These two are in the same category, since they are both in the thirties.

The class is the only exception class thrown in the DiVS. For instance, when a SQLException is thrown, it is always caught by a try-catch and re-thrown as a DivsException as illustrated below:

```
try {
      // fetch data here
} catch (SQLException e) {
      throw new DivsException(DivsException.ERROR_SQL,
                  "Fetch data result from
                  database:\n"+e.getMessage());
}
```

Class LocalDB

The class is singleton and initiates the connection to the database. Database Tuning principle number 3 says: “Start-up costs are high; running costs are low”[55]. This principle is the direct reason for making the class

[55] [DBT] p.2

singleton. The alternative would be to (re)connect to the database for every transaction, and that would waste lots of time and memory.

3.4.4 Protecting the Database with a Password

The current implementation of DiVS probably would not have been used by the Ministry of the Interior for reasons like there is no GUI supporting the model and there is no supporting web application that the districts can use for e.g. registering the results. The database has, however, been protected with an encryption key. The reason is just that this is a good habit. This has, of course, an implication on how to connect to the database. The constructor of the Election class therefore receives two string parameters, `jdbc_driver` and `db_url`. These are passed on to the LocalDB.

It's a bad habit to hard code passwords. The JUnit tests that test DatabaseGateway, Election, and ElectionResult classes therefore fetch the password from a property file, './env/env.properties'. The env folder has been added to svn:ignore, which means that when checking out the project from the svn the './env/env.properties' must be created with the keys: jdbc_driver and db_url. The keys' values must be set to what's needed to connect to the database.

Below are instructions on how to create an encrypted Apache Derby database and load the DDL and DML:

1. Download and unzip Apache Derby (Java DB) from http://db.apache.org/derby
2. Through a command or terminal window start the 'ij' tool
3. Write the following commands where 'xxx' is the encryption key and `<p1>` is the absolute path to where the database will reside (e.g. /db/divs corresponds to c:\db\divs on Windows) and `<p2>` is the absolute path to this project (e.g. c:/workspace/divs). If no encryption is desired then leave out what if after `create=true;` An unencrypted database uses less than half the disk space and is faster.

```
driver 'org.apache.derby.jdbc.EmbeddedDriver';

connect
'jdbc:derby:/<p1>;user=divs;create=true;encryptionAlgorithm=Blowfish/CBC/NoPadding;
bootPassword=xxx';

run '<p2>/sql/divs.sql';

run '<p2>/sql/data.sql';
```

At this point the database should be created and ready. Note that Apache Derby by default does not allow more that one connection.

3.4.4 Method 'initialize' in Election Class

The `initialize` method in `Election` class doesn't do anything except for change the state to 'Initialized'. This is the same as the `openElection` method. Both methods are kept, because it is likely that 'initialize' in future releases will have something more to do.

3.4.5 Serializable

A number of classes implement the Serializable interface. The classes involved are Ballot and the classes contained in Ballot; i.e. BallotJournal, Party, and Candidate. The reason is that persisting Ballots makes testing of special test cases much easier.

3.4.6 Override the Equals Method or Not

Conscious choice has been made not to override the 'equals' method in any class. The reason is that comparisons of different classes is more usable to read using the '==' than using the '.equals'. What precedes the comparison is often a long chain of method calls, and therefore the '==' is simply easier to read.

3.4.7 Other Database Solutions

If the Ministry of the Interior uses DiVS and chose to use a different database solution than Apache Derby it would not cause much concern. The DatabaseGateway class contains no DDL and will only use the Data Manipulation Language (DML) statements[56] `insert`, `update`, `delete`, and `select`. Although a few aggregate functions are used like `count` and `sum` the syntax itself is simple and should not require any dramatic changes. And, of course, the right library (jar file) must be added to the project.

3.4.8 Semantics of Resolve Methods

The semantics of the resolve methods is not always the same. Some of them have a semantic that parameters passed to the methods lose the draw whereas others have the semantic that the parameters passed to the method win the draw. Hopefully the confusion that this might lead to is not too grave (cf. Appendix G.1.3).

3.4.9 Representation of Draw Issues

The computation phase in step 4 is complicated. Should draws occur, then they involve parties and provinces. This has been the direct reason to add the class `StepFourDraw` in the `computation.draw` package. The class has two list attributes, parties and provinces. The elements of these lists belong together in pairs; i.e. first element in the parties list and provinces list represent a party in a province involved in a draw. This implies that there should always be 2 or more elements in each list when there is a draw, and the size of the lists must always be equal. Similar classes have been created to represent draws after the other computation steps. The draw issue objects are retrieved by calling the getStepXDraw in the ElectionResult class; X takes the values One, Three, Four, Five, and Six. These classes have been added late in the process and therefore the BON is not yet reverse engineered.

3.4.10 Checking Correct Party Vote Distribution

The electoral management board has the responsibility to distribute the party votes to the candidates according to the party's list organization. DiVS doesn't check that this is done correctly; it assumes that this is the case.

[56] [Elmasri&Navathe07] p.37-38.

3.4.11 Optimizations

Not much focus has been on optimizations during the implementation phase. There are, however, some places where optimizations have played a role. The DatabaseGateway fetches the results from the database when the `getConstituencyResult` is invoked. The method has these lines in the beginning:

```
List<Province> provinces = this.getProvinces();
List<Constituency> constituencies = this.getConstituencies(provinces);
List<Party> parties = getParties();
boolean indep_cand = true;
List<Candidate> candidates = getCandidates(parties, indep_cand);
```

When the constituencies are selected through a local call in line 2 the provinces are shipped along as a parameter, so that `getConstituencies` doesn't have to fetch the provinces again. Note that all constituencies must belong to a province. The same point goes to the `getCandidate` method; it receives parties as a parameter.

3.5 Extensibility

The limitations mentioned in the introduction of the design chapter can be added later. This section discusses if it will be is easy or hard to add the limitations mentioned.

3.5.1 Register Electoral Map and Register Parties and Candidates

The registering of the electoral map would be easy to implement in DiVS. It is already stated in informal BON that register electoral map in addition to register parties and candidates should be an additional feature of the election class (since it inherits from the GUI_INTERFACE). The DB_INTERFACE also has these commands described. The same goes with the registering of the parties and the candidates. It might, however, require many lines of code in the concrete Java class representing DATABASE_GATEWAY. This task should, however, be straight forward to solve.

3.5.2 Party List

Adding the party list would require a few changes to the database design as well as the computation of the results. The ER diagram would need an entity `PartyList` with the attributes `party_id`, `candidate_id`, and `standing`; the first two are the primary key attributes. A candidate cannot run in more than one constituency, and that information is already available in `BallotJournal`, `Ballot`, and `District`. Therefore `PartyList` doesn't need a `constituency_id` attribute. `PartyList` should have a many-to-1 relationship to both `Party` and `Candidate`. The usage of `PartyList` should be that the entity contains no rows representing candidates, and that they are not on the party's party list.

The PartyResult class must accommodate calculating the distributional number. It is sufficient to calculate it runtime since all information necessary is present in the class: `total_seats` and `total_votes`.

The `select_candidates` method of the ElectionResult class would not require many changes. The arrangement of the candidates, which is dealt with in the `Cmp` class, must be edited. The `compareTo` method in `Cmp` class should take into account the distributional number in addition to the standing on the party list.

3.5.3 Party Receives More Seats, Than it Has Candidates

In computation step 6 the candidates are selected. In some constituencies there might be a situation where there are less candidates than there are seats. According to the law the extra seats should first be allocated to a constituency in the same province. If not applicable the seats must be distributed to the other two provinces when these are merged with regards to the party results. The computations resemble the computations in step 4 and 5, but here they take into account the number of running candidates.

The select_`candidates` method must, just after the current selection of candidates has finished, do the computations described above – for each party first on the provincial level and then on a level where the two other provinces are merged and the constituencies are treated as belonging to that "province". The computations terminate when all seats are allocated to a candidate. These calculations are feasible although they are fairly complex. What needs to be added to the design is a class that contains a candidate and a constituency. In the `PartyResult` class a list parameterized to this class should be empty when there are enough candidates. When the party gets more seats than it has candidates then the algorithm that finds the un-elected candidates must maintain the list mentioned.

3.5.4 Substitution List

The implementation of the substitution list resembles the computations above. The substitute list should carry on where the computations above left off. The difference is now, that instead of electing the candidates the candidates are ranked in the substitute list for the party in the province. This means that for each party in each province there is a need of a list of `CandidateResult`. The list contains references to the non-elected candidates in the province and when necessary, references to non-elected candidates in other provinces. The substitution list might have draws that need to be resolved, just like the elected candidates. Therefore an additional resolve method is needed to be written that resolves draws on the substitute list. This resolve method will look like the other method mentioned.

3.5.5 Handling Rounding Errors

Unhandled rounding errors might create bugs. For instance two quotients that are almost the same and should be exactly the same can be considered not equal. This could mean that draws that should have been draws are not discovered and one of the parts is favored.

This can be dealt with in Java by using the class `BigDecimal` instead of the primitive data type `double`. In comparisons the `compareTo` (and not `equals`) `BigDecimal` can be used to check for the return value -1, 0, and 1 meaning less than, equal to, and greater than respectively. This would, however, only reduce the number of bugs. In order to be absolute sure the Div class should store the dividend and divisor integer values. The comparisons between these integers should be made after the lowest common denominator has been found. This requires a `compareTo` method in the Div class that is straight forward to write.

3.5.6 Lots Drawn on District Level

According to the analysis, lots are drawn on the district level between candidates that have just claims on a party vote. DiVS does not yet support this on the national level; i.e. DiVS does not know in the current design

which candidates are involved and who won the draw. It should be easy to add an attribute or something similar to the `CandidateResult` class that represents this information.

3.5.7 Projections and Aggregate Counting

Should the Ministry of the Interior choose to use DiVS in the future, they would most likely require some form of data sharing with the mass media. Projections are based on the results from the previous election and the difference between the previous election and the current election. For instance if a party in the last election received 5 percent of the votes and in current election gets 2 percent more votes in the results, that have been registered so far, then the party is projected to get 7 percent of the votes.

One solution would be to store the results from the last election in the election before the Election Day, and then a method in the DatabaseGateway could just compare the current results with the old results. This would require the database to e.g. add one more database schema in the same database with exactly the same tables. In addition, a method must be added that calculates the projection. If the results from the previous election are ready and available then the method that makes the projection simply needs to project the difference in percentages. The computation in step 3 would then be repeated in order for the projection to include the number of seats this time and the last time. In short, this functionality requires only the data from the previous election to be available and a method that can compare the results that have been registered so far with the previous election.

3.5.8 Conclusion of Extensibility

All the features mentioned here can be implemented without altering the design. Therefore it must be concluded that DiVS is extensible.

4 Verification

This chapter contains the verification of the program correctness. As mentioned in the introduction this should be done using static checking tools. The tools currently available (April 2011) do not support static checking of Java 1.6 to a satisfactory level; OpenJML doesn't support usage of quantifiers yet. Therefore, the verification must be done by other means. As the introduction suggests, an outlining of the proof has been chosen as the alternative to static checking. This can be thought of as manual static checking.

The reader unfamiliar with the Danish electoral law can inspect appendix E to see simplified example results before reading through this chapter. This will allow the reader to know what the algorithms do.

A verification using a static checker tool rests on the JML contracts being correct. The JML contracts have been type checked by the OpenJML tool. The JML contracts might, however, need smaller revisions when the OpenJML tool is finished implemented.

4.1 Static Checking of the Java Skeleton

Generally, it's a good idea to use a static checker tool as soon as the Java skeleton with JML contracts is ready. The checking should focus on invariants and pre or post conditions that are in conflict. For instance, if an invariant states that an integer must be greater than or equal to 0 and a pre condition states that this exact same variable should be less than 0, then these two are clearly in conflict. These conflicts are not so easy to spot in a large system, making static checker tools very useful. They might, however, produce false negatives and false positives. The most important benefit is that the checking of the skeleton with contracts can, early in the process, reveal design flaws and not just errors. The sooner one finds flaws, the less work is wasted.

In this project, if all transitions from the electoral law through BON to the Java skeleton are sound, then the static checking is really a soundness checking of the electoral law. Inconsistencies in the Java skeleton would be a result of inconsistencies in the law, since the Java implementation can be considered a mathematical model of the law.

Due to issues with the static checker tools, this has not been done. This checking would be a subset of the proof outlining anyway, and is therefore not essential for verifying the program correctness.

4.2 Outlined Proof of Program Correctness

This section will deal with the program correctness involved in both the outer and inner state machine. In order to outline proof of correctness, both the outer and inner state machines combined must both be sound and complete. It must also be confirmed that all transitions will terminate. If this implies that the final state is reachable, then program correctness is assumed.

The reader is encouraged to view state charts in figure 1 and 2 before reading this section.

4.2.1 Outer state machine

The state chart of the outer state machine is linear – except for a few self loops. All the transitions have both pre and post conditions. The discussion starts in the start state, 'Before Election'. Thereafter, all transitions will be discussed in the same order as they appear on the state chart by following the transitions.

Before Election

This is the start state. There is only one outgoing transition from here: 'initialize'.

initialize

The only thing here is that the outer state is set to 'Initialized'. It is therefore trivial to see that the transition has been verified.

open election

Here, the outer state is set to 'Election Open'. This means that polling stations can inform the national level that they are now open.

try close election

The outcome of this state is either 'Election Open' or 'Election Closed'. There is a finite number of polling stations in Denmark, which can open and close. The transition will only go from the former to the latter if all the individual polling stations have closed. The transition checks the number of polling stations that have not yet closed. If this number is greater than zero, then outer state will not change. If zero, then the state changes to 'Election Closed'.

check preliminary counting

When outer state reaches 'Election Closed', it enables polling stations to initiate the counting and eventually the registration of the results. The outcome of this state is possibly 'Preliminary Counting Completed', but only if all the individual polling stations have registered their preliminary results. In other words, if there is any polling station that has not yet registered the preliminary results, the state will not change. When the outer state reaches 'Preliminary Counting Completed', the inner state machine is informed about all necessary data from the registered preliminary results.

preliminary computing steps

The computing steps are dealt with in the next section. There are, however, several interesting steps seen from the point of view of the outer state machine. These are listed below:

compute step two preliminary

After the electoral threshold has been determined, there can develop a situation where there are no parties that pass the threshold. If this happens, then the computation of the preliminary results is finished, and the outer state becomes 'Preliminary Results Computed'.

compute step five preliminary

If, after computing step five, the inner state is 'Step 5 Resolved', then the outer state changes to 'Preliminary Result Computed'. This means that there are no draws that need to be resolved.

resolve step five preliminary

If, after computing step five, the inner state is 'Step 5 Done', it means that there is at least one draw that needs to be resolved. Draws can be resolved in the inner transition 'resolve constituency results'. For every transition there is one less issue that needs to be resolved, and since the number of issues is finite, it means that at some point there will be no issues left. The outer state machine then changes state to 'Preliminary Result Computed'.

check final counting

When outer state reaches 'Preliminary results computed' it enables polling stations to initiate the final counting and the registration of the final results. The outcome of this state is either 'Preliminary Result Computed' or 'Final Counting Completed'. The transition will only go from the former to the latter, if and only if all the individual polling stations have registered their final results. In other words, if there is any polling station that has not yet registered the final results, the state will not change. When the outer state reaches 'Final Counting Completed', then the inner state machine is informed about all necessary data from the registered final results.

final computing steps

The final computing steps are dealt with in the next section. There are, however, several interesting steps seen from the outer state machine, and they are listed below:

compute step two final

This is the same situation as described above, and need not be repeated

compute step six final

If, after computing step six, the inner state is 'Step 6 Resolved', then the outer state changes to 'Preliminary Result Computed'. This means that there are no draws that need to be resolved.

resolve step six final

If, after computing step six, the inner state is 'Step 6 Done', then there is at least one draw that needs to be resolved. This can be resolved by the inner transition 'resolve candidates', with parameters of involved candidate, party, and constituency. For every time this is done, there is one less issue that needs to be resolved, and since the number of issues is finite, it means that at some point there are no issues left. The outer state machine then changes state to 'Final Result Computed'.

Final Results Computed

This is the final state. It can be reached assuming that inner state machine works as intended – which is exactly what the next section examines.

4.2.2 Inner state machine

The outer state machine has two instances of the inner state machine. These are called preliminary and final result. The preliminary result will not go past 'Step 5 Resolved', whereas the final result will – assuming that neither will reach the special state: 'No Passing Parties'. This subsection will discuss all transitions in the same order they appear on the inner state chart.

It is assumed that the data the inner state machine has received is consistent with regards to the number of seats each province and constituency has been allocated before the election. For instance if a province has X seats allocated, then all constituencies within that constituency have a number of seats added up that together equals X, and so forth. It is also assumed that a party has more candidates running in each constituency than it will get seats (cf. limitations 2.0.1).

Moreover, it is assumed, that in every constituency there will be enough votes cast in order to determine who receives the constituency seats. Although the law does not explicitly specify, that you need to get a vote in order to get elected it is assumed here, that a party or independent candidate must get at least 1 vote in order to be considered. It becomes the responsibility of the Folketing to decide what to do if not enough parts get votes. The point is, however, that if not enough people get votes then the computations in step 1 will be incomplete.

All transitions will include the pre conditions, the post conditions, and the pseudo-code of the algorithm inside the transition. The pseudo code is written as sparse as possible without taking away any key points. The last pre and post condition is always about the inner state. The pseudo code will not specify the resulting post conditions, since the inner state will always become 'Step X Done' or 'Step X Resolved' if there are no draws and e.g. all seats are allocated (X will hold the values 1, 3, 4, 5, and 6). The only exception is 'determine threshold', which will result in either 'Threshold Determined' or 'No Passing Parties'. Because of the property mentioned about 'Done' vs. 'Resolved', these post conditions are not discussed below.

The JML contracts are not included in this subsection. The pre and post conditions listed here represent what they express (cf. the code in the SVN repository).

allocate constituency seats

The preconditions for this transition are:

- For each constituency result there must be a list containing all parties and a list of all independent candidates and their votes. The list of constituency results must be of size 10, which is the number of constituencies in Denmark
- For each constituency result the number of seats taken so far must equal 0
- For all parties in all constituency results the number of seats taken so far must be 0
- For all candidates the elected status is set to non-elected
- All parties running in the constituencies must be found in the list of parties on the national level as well as on the provincial level for the province in which the constituency belongs
- For all constituency result elements at least one party will get at least 1 vote or there are at least the number of independent candidates getting at least 1 vote as there are constituency seats in the constituency
- The inner state must be 'Before Computing'

The first point says that the inner state machine must be informed about the result on the constituency level. The second through fourth say that everything must be initialized to zero or non-elected. The reason for the

fifth pre condition point is that when a party wins constituency seats, it must be possible to accumulate these seats on the national and provincial level after this transition. The fifth is very interesting: if it is true, then it is possible to allocate all constituency seats. If it is false, then it is not possible to allocate all constituency seats. The start state of the inner state machine is always 'Before Computing'.

The post conditions for this transition are:

- For all constituency results the number of seats taken is greater or equal to the number of constituency seats belonging to constituency
- If there exists a constituency result with more seats taken than constituency has seats allocated, then the parties and/or independent candidates involved in draws are marked. The number of involved parts is always greater than 1
- If there are no draws then all parties on the national level are updated with the number of constituency seats won on the constituency level
- The inner state changes to 'Step 1 Done' or 'Step 2 Resolved' (if there are no draws)

The first point says that all the constituency seats are allocated and perhaps more seats are allocated than there should be. The second point says that if there are more seats taken than allocated, then the involved parts are marked so that lots might be drawn outside this system. The outcome of the lots drawn is afterwards registered in the system.

In order to verify the post conditions, it is necessary to look into how the algorithm works:

1. for all constituency results
2. FindSeats(ConstituencyResult$_i$)
3. if for all constituency seats taken = constituency seats allocated to constituency then
4. update result on the national level

FindSeats(ConstituencyResult cr)

5. create one div and add it to the div list of every party in 'cr'
6. while seats taken < constituency seats allocated to constituency do
7. find the highest quotient among parties and vote count for unelected independent candidates
8. locate party and/or un-elected independent candidate with highest div value
9. if party then add 1 seat to party and add 1 more div to div list using the d'Hondt method
10. if independent candidate then set him/her to elected
11. if seats taken > constituency seats available in constituency then
12. mark all parties and/or independent candidates with quotient = highest

The sixth pre condition ensures that as long as the predicate in the while-loop is true, we will always find the highest value on line 7. If at least one party gets at least 1 vote, then the quotient of those parties will always be larger than 0. This is because the number of votes divided by 1, 2, 3, etc. is always larger than 0. At some point the seats taken will be equal or greater than the number of constituency seats belonging to constituency,

at which time the algorithm terminates. In the last while-loop iteration, however, there might be quotients that are equal among the involved parties and independent candidates. If they are, then multiple seats get allocated in this last loop, and therefore the number of seats taken in the province might be more than allocated to the province. This is exactly what the first post conditions claims and is hereby verified.

The last if-statement of the algorithm says that if there are more seats taken than there are seats available, then all involved parts are marked. If there is, during the last iteration of the while-loop, only 1 seat allocated, then 'seats taken' would equal constituency seats available. But if the if-statement in line 11 is true, then there must be more than 1 allocated during the last iteration and involved parts are marked. This is what the second post condition claims, and it is hereby verified.

At the end of the transition there is a check that investigates the presence of any issues/draws in the constituencies (line 3). If there are no issues, then the number of seats on the national level is updated. This satisfies the third post condition.

It has already been shown that the algorithm will terminate.

resolve constituency seats

This method is used in case there are draws after the previous transition. The method receives three parameters: a party result, a candidate result, and a constituency result. The semantic of the parameters is that in the constituency result, the party or candidate result has lost the draw.

The pre conditions for this transition are:

- Party result parameter is not null exclusive-or candidate result is not null
- Party result parameter (if not null) has at least 1 constituency seat and it has been marked for a draw
- Candidate result parameter (if not null) is elected and has been marked for a draw
- The party result parameter (if not null) can be found in the constituency result parameter
- The candidate result parameter (if not null) can be found among the independent candidates
- The constituency result parameter has more seats taken than it has seats to allocate, and there exists at least 1 opponent that is drawing for the same seat within the constituency result
- The inner state must be 'Step 1 Done'

The first point says that either the party result or the candidate result must be null, since if both were non null, then the algorithm would not know who lost the draw. The second and third points say that the involved part must be marked for draw beforehand. This would make it possible to take a seat (either by subtracting number of seats taken for the party or by switching from elected to non-elected for the independent candidate). The fourth and fifth points say that the party or the independent candidate must exist as running in the constituency. The sixth pre condition says that in the constituency result, there must be more seats taken than there are available. Otherwise there is not a draw issue present. It also says that there is an opponent in the draw that has been marked. This follows from the post conditions of the previous transition when there are draw issues to resolve.

The post conditions for this transition are:

- If party result was not null then party result is no longer involved in the draw, and it has lost one seat
- If independent candidate was not null then he/she is no longer involved in the draw, and he/she is no longer elected
- Number of seats taken in constituency is one less than before
- If all issues in all constituencies are resolved then the parties are updated on the national level with the number of constituency seats won in the constituencies
- The inner state changes from 'Step 1 Done' to 'Step 2 Resolved' if there are no draws

In order to verify the post conditions, we need to look at the algorithm:

1. locate party or independent candidate (involved party)
2. withdraw 1 seats from involved part
3. un-mark involved part
4. withdraw 1 seat from the constituency
5. if for all constituency seats taken = constituency seats allocated to constituency then
6. update result on the national level

The pre conditions tell us that involved part (party or independent candidate) exists in the constituency and they can lose 1 seat. This means that they can be found and one seat can be taken away from them. They are at the same time un-marked and the number of seats taken in the whole constituency is reduced by 1. This is exactly what the first three post conditions claim, and since there is no reason to suggest that the algorithm will not terminate, it is therefore verified.

The last post condition is the same as in the previous transition, and since the same arguments apply here it is also verified.

determine threshold

The pre conditions for this transition are:

- The list of provincial results must each contain a list of all parties and their votes. The list must be of size 3, which is the number of provinces in Denmark
- The total number of valid votes is known on the national and the provincial level
- All parties in all provinces must be found in the list of parties on the national level
- For all constituencies the number of seats taken must equal the number of seats available
- The number of constituency seats taken on the national level for the parties plus the number of independent candidates elected must equal the total number of constituency seats in Denmark (135)
- The threshold is set to non-passed for all parties
- The inner state must be 'Step 1 Resolved'

The first point says that the list of provincial results must be set and it must contain the votes for all parties. The second pre condition says that the number of valid votes on the provincial level must be known; it is used in the second option to pass the threshold. The second pre condition also says that the number of valid votes on the national level must be known; this number is used in the third option to pass the threshold. The third point says that all parties running in the provinces must exist among the parties on the national level. The fourth pre condition says that there must not be any draw issues left after the first transition. The fifth pre condition says that the number of constituency seats for the parties on the national level must be updated. The last pre condition specifies that threshold passed must be initialized to false.

The post conditions for this transition are:

- If a party has at least one constituency seat then it passes the threshold
- If a party in two provinces has more votes than the votes/seats ratio, then it passes the threshold
- If a party receives at least 2 percent of all valid votes then it passes the threshold
- If the outcome is that there is at least 1 party passing the threshold then the total number of passing votes will be set to match all the votes for all the parties passing the threshold. This implies that the pure Hare quota will be calculated and it will be of a value that equals the number of passing valid votes divided by the total number of seats to Denmark (175) minus the number of seats taken by independent candidates
- The inner state changes to 'Threshold Determined' or 'No Passing Parties' (the latter if and only if no party passes the threshold)

The first three post conditions list the three tests for passing the threshold. Any party passing one of them passes the electoral threshold. The fourth post condition specifies that if any party passes the threshold then the total number of passing votes will be calculated, and the quota will be calculated.

In order to verify the post conditions we need to look at the algorithm:

1. for all parties on the national level
2. IsThresholdPassed($Party_i$)
3. if at least one party passed then
4. calculate total passing votes and determine the quota
5. set inner state to 'Threshold determined'
6. else
7. set inner state to 'No Passing Parties'

IsThresholdPassed(Party)

8. if party has got at least 1 constituency seat then
9. set threshold passed
10. if party in two out of three provinces has votes >= than the valid votes / seats in province then
11. set threshold passed
12. if party has got at least 2.0 percent of the votes on the national level then

13. set threshold passed

The if-statements on line number 8, 10, and 12 correspond to the first three post conditions. One of the pre conditions says that the threshold for all parties is initialized to 'not passed'. The threshold will be set to 'passed' only when the party passes one of the three tests; otherwise it will keep the false value. For every party then the algorithm is correct and therefore the first three post conditions are satisfied.

For the second post condition the outcome is either that no parties are passing the threshold or the negation of that: at least one party is passing the threshold. If the former is true then nothing happens (except for the change of inner state). If the latter is true then the outcome is that the total number of passing votes and the quota is calculated (lines 4 and 5).

There is no reason to suggest that the algorithm will not terminate.

Inner State: "No Passing Parties"

It has been shown above that there is a chance that no parties will pass the threshold. For this to happen, the following pre conditions must be satisfied:

1. all constituency seats are allocated to independent candidates
2. no party gets the vote ratio on the provincial level for 2 out of 3 provinces that matches at least the votes divided by seats
3. all parties get less than 2.0 percent of all valid votes

If this becomes the case, then no party will be entitled to a share of the additional seats. Furthermore, it means that the parliament would consist of only 135 seats representing Denmark. This is in conflict with the Danish electoral law, which specifies that there must be exactly 175 seats representing Denmark.

The probability of this happening is low, but the possibility is present. Danish electoral law should therefore accommodate this contingency, and this is why this project is introducing the inner state 'No Passing Parties' (cf. figure 2). This has been added to the recommendations and comments to the electoral law (cf. Appendix F).

allocate additional seats to parties

The pre conditions for this transition are:

- For all party results on the national level the number of additional seats is 0
- There exists at least one party passing the threshold
- The quota has been calculated
- The inner state must be 'Threshold Determined'

The first point shows that the number of additional seats should be initialized properly to 0. The next two are merely consequences of the previous transition when at least one party has passed the threshold.

The post conditions for this transition are:

- The total number of seats taken on the national level is greater than or equal to the total number of seats in Denmark (175)
- The upper bound of the total number of seats taken is 175 + the number of passing parties – 1
- If the total number of seats is greater than 175 then the parties with equal fractions in the last iteration of the while-loop are marked for draw. This means that there are anywhere from two to all passing parties involved
- If the total number of seats is equal to 175 then all parties on the provincial level have the number of constituency seats updated, and the div list is initialized according to the Sainte Laguë method
- The number of additional seats of a party is the total number of seats minus the constituency seats
- The inner state changes to 'Step 3 Done' or 'Step 3 Resolved' (if there are no draws)

In order to verify the correctness we need to look at the algorithm that allocates the seats to the parties. The algorithm uses the sub-routines: AddIntegers and BreakFractions:

1. int seats_total := number of independe candidates elected
2. AddIntegers()
3. BreakFractions()
4. if any party has a negative number of additional seats then
5. seats_total := total number of independent candidates elected
6. arrest parties with negative additional seats count or additional seats equal to 0
7. AddIntegers()
8. BreakFractions()
9. if seats_total = 175 then
10. update number of constituency seats on the provincial level from the constituency result
11. initialize the div list for each party on the provincial level according to the Sainte Laguë method

AddIntegers()

12. for all parties on the national level
13. if party passed the threshold then
14. set total number of seats for party to (total votes for party / quota)
15. accumulate total_seats with the same number. Ensure that additional seats = total seats – constituency seats

BreakFractions()

16. while seats_total < 175 do
17. find highest fraction among all un-arrested parties passing the threshold
18. locate parties with highest fractions, add 1 seat, arrest party, and accumulate total_seats by 1. Ensure that additional seats = total seats – constituency seats
19. if seats_total > 175 then
20. locate parties with highest fractions again and mark them

First let's look at the AddInteger algorithm. Keeping in mind that we are currently distributing all seats not taken by independent candidates, and that the quota is calculated on basis of the total *passing* votes divided by 175 (less the number of constituency seats taken by independent candidates). Also we are only considering parties that have passed the threshold. The quota is quite central in that algorithm. Line 14 says that a party should get seats corresponding to its number of votes divided by the quota and rounded down to nearest integer. This means that the number of seats taken after this step will approach 175 minus the number of independent candidates. Note that the gap up to 175 cannot be higher than the number of passing parties minus 1, because the gap equals the fractions of all the passing parties. Now the fractions can be broken; i.e. BreakFractions is called. Since the gap is less than the number of passing parties, it means that the gap will always be closed by the BreakFractions algorithm. This also means that during the last iteration there might be a draw between parties if their fractions are equal. This satisfies the first post conditions. Also, since there is a possibility of a draw in the last iteration, it also means that there might be more than 175 seats taken. However, the reason for this may be that one or more parties have already received more constituency seats than the calculations above allow. When this happens another round must be made and AddInteger and BreakFraction must be called upon once again (line 4). The difference in the second round is that the parties that have received too many seats or exactly the number of seats allowed are put to rest. The AddInteger algorithm will then compute exactly the same number of seats as before for the un-arrested parties (i.e. the parties that are put to rest) while BreakFraction will allocate 1 more seat to at least one party, never allocating more than 1 more seats to any one party (cf. Appendix F). This implies that before the second round, it is impossible for any un-arrested party to be in a situation where they will get more seats than allowed. Therefore the first post condition is satisfied when the if-statement on line 4 is true.

If there are no fractions to break after AddInteger has finished, the number of seats taken is exactly 175; this follows with how the quota is calculated. If there is a quotient that does not equal an integer number, then the highest number of seats taken is 174, since the fractions of the parties with fractions will add up to 1. There is the possibility that all quotients have fractions and that these fractions are the same. This involves all parties in a draw. Therefore the upper bound for the total number of seats is 174 + number of passing parties – 1. This is exactly what the second post condition claims.

The third post condition is a consequence of when seats_total is larger than 175. The involved parties are marked by the code on line 19 and 20. In the last iteration there must be at least two parties involved, otherwise the condition of the while-loop will be false. This satisfies the second post condition.

The fourth post condition is a consequence for when seats_total equals 175. The province result is updated by the code on line 10 and the div list of each party is initialized by the code on line 11.

There is a while-loop present inside BreakFractions, which might suggest that it might not terminate. Since BreakFractions will always close the gap, the while loop will terminate.

resolve additional seats

The following transition is used when there are draws after the previous transition. The transition receives one parameter: a party result. The semantic of the parameter is that in the party result loses the draw.

The pre conditions for this transition are:

- The party result parameter must be non null, marked for a draw, and have at least 1 additional seat on the national level
- The party result parameter must be found among the parties on the national level
- On the national level there must exist another party result that is also marked for a draw
- The total number of seats taken must be larger than 175 (the total for Denmark)
- The inner state must be 'Step 3 Done'

The first pre condition is self-evident: there must be at least 1 additional seat taken, otherwise there is nothing to give away in this transition. The second point ensures that the party exists and can be found. The third and fourth pre conditions are a consequence of the previous transition – when too many seats are taken.

The post conditions for this transition are:

- The party result parameter is not marked for draw and the number of additional seats for party result parameter is 1 less
- The total number of seats is 1 less than before
- If the total number of seats equals 175 then all parties on the provincial level are updated with regards to the number of constituency seats. The div list is also initialized according to the Sainte Laguë method
- The inner state changes from 'Step 3 Done' or 'Step 3 Resolved' (if there are no draws)

In order to verify the correctness we need to look at the algorithm:

1. locate party result on the national level
2. un-mark party result and take 1 additional seat from party
3. decrement seats_total by 1
4. if seats_total = 175 then
5. update number of constituency seats on the provincial level from the constituency result
6. initialize the div list for each party on the provincial level according to the Sainte Laguë method

The pre condition says that the party result can be found. When it is found it can be un-marked for draw, and 1 seat can be withdrawn from it. This satisfies the first post condition. The second post condition is satisfied by line 3. Lines 4, 5, and 6 satisfy the third post condition.

There is no reason to suggest that the algorithm will not terminate.

allocate additional seats to provinces

The algorithm that allocates additional seats to the provincial level is complicated. The premise of this algorithm is already described in the analysis chapter (cf. subsection 1.1.2 p. 19). This subsection verifies both the allocation of the additional seats and the resolving of possible draws.

The pre conditions to this transition are:

- The number of additional seats taken must be 0
- For all provinces the number of additional seats taken must be 0
- For all parties in all provinces the number of additional seats taken must be 0
- The total number of additional seats allocated to parties on the national level must be 40
- The total number of additional seats allocated to the provinces is equal to 40
- For all parties in all provinces the size of the div list must equal the number of constituency seats + 1
- All passing parties get at least 1 vote in all provinces

The pre conditions to this computational step are that all initializations are done properly and no unresolved draws are left from previous transitions. Keep in mind pre condition number 7 - it states the most common case in the elections. If not stated, then the verification falls apart.

The post conditions are:

- The number of additional seats taken is 40 (total number of additional seats in Denmark)
- All provinces get exactly the number of additional seats that they are allocated before hand
- On the provincial level, all parties, in total, receive the same number of additional seats that they get on the national level
- The inner state is 'Step 4 Resolved'

All post conditions in this transition merely state that there are no draws and that all seats get allocated to the parties on the provincial level.

In order to verify the post conditions, we need to look at the algorithm. The algorithm that allocates the additional seats on the provincial level is shown in pseudo code below. An explanation follows.

```
1.  while additional_seats_taken < 40 and not TooManySeatsTaken() do
2.      highest := -1
3.      for all un-arrested passing parties in all un-arrested provinces
4.          find highest quotient
5.      if  no highest quotient found  then  break
6.      ranking := additional_seats_taken
7.      count := 0
8.      for all passing parties in all provinces
9.          for each quotients/div with value = highest
10.             count := count + 1;
11.             set quotients ranking = ranking
12.             add 1 to province.additional_seats_taken
13.             add 1 to party.additional_seats_taken (national level)
14.             add 1 to party.additional_seats (province level)
15.             add 1 to div list for party on province level
16.     accumulate additional_seats_taken by count
```

```
    TooManySeatsTaken()
17. for all province results
18.     if  additional_seats_taken by province result > additional_seats allocated to province  then
19.         return true
20. for all passing parties on the national level
21.     if  additional_seats_taken by party > additional_seats  then
22.         return true
23. return false
```

A party result is put to rest if the party result has received its additional seats allocated on the national level or more. A party that is un-arrested has not yet received its additional seats. The same reasoning applies for provinces. Line 4 states: “find highest quotient”. This means implicitly “find highest quotient that are not crossed out” (cf. p. 31).

The most common case involves no draws. Let’s first look into that.

The while-loop (line 1) will stop when at least 40 seats are taken. Because of the seventh pre condition it will always be possible to find a highest quotient (line 3 and 4), and therefore the if-statement on line 5 is always false. This causes every iteration of the while-loop to allocate at least one additional seat to a party and a province. The algorithm will therefore terminate at some point since all the additional seats will get allocated. It is possible, however, that while-loop iterations allocate more than just one additional seat. This doesn’t imply, however, that a draw is created; which is what we assume for the moment. Therefore, when there are no draws, all post conditions get verified.

The while-loop will terminate as soon as a draw is present. The definition of a draw is: a party result takes, on the national level, more additional seats than it has allocated, or a province takes more additional seats than it has allocated. The sub-routine TooManySeatsTaken illustrates how the presence of draws is found.

Let’s take a look at the algorithm again. The only possible way for a draw to occur is that two or more quotients are equal. Furthermore, these quotients must belong to either the same party in several provinces (e.g. the last additional seat of a party getting allocated to two provinces) or to the same province for several parties (e.g. the last additional seat of the province getting allocated to two parties). The worst case scenario is when several parties in several provinces draw for the same seat; i.e. all parties in all provinces have a just claim on a seat, because their quotients are the same. This would be the result of all possible combinations of all passing parties in all provinces; e.g. if there are 10 passing parties then there are 30 involved parts in the draw, taking into account that there are 3 provinces in Denmark.

When the allocation of additional seats to the provincial level stops and the inner state is not ‘Step 4 Resolved’, it means that there is a draw present. The algorithm below works out the involved parts:

```
    GetStepFourDraw
24. ranking := -1
```

25. for all passing parties in all provinces
26. find the highest quotients ranking among the quotients
27. for all passing parties in all provinces
28. if quotients.ranking = ranking
29. StepFourDraw.addParty
30. StepFourDraw.addProvince

The algorithm first locates the highest ranking value among the parties on the provincial level. The reason for this is that since the while-loop of the main algorithm breaks as soon as a draw presents itself, the involved parts must have the highest ranking number; cf. line 9. In order to know the involved parts, one simply has to invoke GetStepFourDraw. Lines 29 and 30 add a party-province pair to a StepFourDraw instance (cf. 3.4.9).

The law says that lots must be drawn to resolve draw issues. DiVS must be informed about the result of the lots drawn. DiVS resolves issues through the algorithm ResolveStep4Draw. The algorithm receives three parameters: rank, party result, and province result. The semantics of the parameters is that the party result in province result wins the ranking, and all others loose. The pre condition for the algorithm is that there is a draw issues that must be resolved, and the post condition is that there is not a draw issue that must be resolved.

ResolveStep4Draw(int rank, party pres, province pr)
31. for all parties in all provinces
32. if quotients.ranking = rank and (party != pres or province != pr)
33. subtract 1 from province.additional_seats_taken
34. subtract 1 from party.additional_seats_taken (national level)
35. subtract 1 from party.additional_seats (province level)
36. remove from div list the last div and set ranking of last div to 0
37. subtract 1 from additional_seats_taken

The algorithm above resolves the draw issue by making all parts lose the draw except for 1 party-province pair. This means that there are no draws left after this issue, and the next transition from 'Step 4 Done' is 'allocate additional seats to provinces'. The if-statement in line 32 ensures that all rankings are found, except for the winning party-province pair that the algorithm receives as parameters. This means that if the draw involves 3 party-province pairs and this method is called, then the method removes all but what corresponds to the parameters received; i.e. 2 party-province pairs. Line number 37 is therefore called twice in this working example. And when the main algorithm eventually resumes, then the ranking in line 5 is set to a number which is 1 higher than it was before.

There's no reason to suggest that ResolveStep4Draw doesn't terminate.

It has already been shown, that when there are no draws then all additional seats are allocated on the provincial level. When draws are involved this is also the case. A possible draw slows down the process since it needs to be resolved before the algorithm is resumed. After resolving one issue, the 'additional seats taken' gets closer to 40 by at least 1. It has been shown that ResolvStep4Draw completely resolves a draw. Putting

these two things together implies that eventually all draw issues are resolved. This is exactly what the post conditions state, and it must therefore be concluded that the algorithm is correct. It has already been shown that the algorithms will terminate.

It is important to note that the last pre condition says that all passing parties get votes in all provinces. If this pre condition is removed, then all parties might run in only some provinces or just one. The law says explicitly that if there are parties that run in only one or two provinces then their additional seats should be allocated before the others to ensure that the province is not arrested before party's quotient becomes the highest in the main algorithm. This is assuming that there are only a few passing parties that don't run in all provinces and allocating the additional seats to these parties effectively resolves the issues. However, if all parties don't run in all provinces then there is a good chance that the situation mentioned above occurs anyway. This can be shown with a counterexample[57]:

> *All provinces have 45 constituency seats and 13 or 14 additional seats allocated. 27 parties are running. They all get the same number of votes and they all get 5 constituency seats each. The parties are only running in 1 province. In computation step 3, the parties get allocated 6 or 7 seats total. This means that they get 1 or 2 additional seats. Since all get the same number of votes their fractions are all the same and therefore lots must be drawn. The lots provide all parties in one province with 2 additional seats. This means e.g. that 18 seats belong to parties in this province. The province has only 13 or 14 additional seats, and therefore the province gets arrested before all the additional seats of the parties running in the province are allocated. In the main algorithm the while-loop will only go through 2 iterations before producing a draw. In the first, 27 additional seats get allocated. This results in 14 parties getting arrested, since their only additional seat is taken, but no provinces are arrested. In the second, the remaining 13 parties become involved in a draw issue. This issue gets resolved by drawing lots with only one winner, the ResolveStep4Draw is called, and the main algorithm is resumed. The result is, that 15 parties are arrested and 12 parties are involved in a draw. Again this issue gets resolved, and it goes on. At some point the province gets arrested, as well as all parties in the other provinces. This means that line 4 will not find any highest quotient, and the if-statement on line 16 becomes true. At this point there are no draws, since TooManySeatsTaken returns false. But there exists on the national level a few parties that haven't received their additional seats yet on the provincial level. This violates the post condition that all seats should be allocated.*

Therefore the pre condition stating that all parties must get votes in all provinces is kept (cf. Appendix F).

allocate additional seats to constituencies

The pre conditions for this transition are:

- All 40 additional seats have been allocated to the parties on the provincial level
- For all parties in all constituencies all additional seats have been initialized to 0
- The inner state must be 'Step 4 Resolved'

[57] [Schumacher01] p.16

The first point states that there should be no unresolved draws left from the previous transition and that all additional seats are allocated. The second one just mentions proper initializations.

The post conditions for this transition are:

- For each party on the constituency level there are at least as many seats taken that the party on the provincial level allows (for the constituencies belonging to the province)
- If all parties on the provincial level receive additional seats that corresponds to what's allocated, then all candidates in all parties in all constituencies are arranged by descending total vote count
- The inner state changes to 'Step 5 Done' or 'Step 5 Resolved' (if there are no draws)

In order to verify the correctness we need to look at the algorithm:

1. for each passing party on the provincial level
2. ranking := 1
3. while additional seats taken < additional seats for parties on the provincial level do
4. find highest quotient among constituencies that belong to province
5. locate highest quotient among divs in these constituencies and for each quotient found and set the ranking of div equal 'ranking' and add one to div list
6. accumulate ranking and additional seats taken on the provincial level by number of divs found on line 5
7. if additional seats taken = additional seats allocated for all parties on the provincial level then
8. sort all candidates in all parties in all constituencies descending by total vote count

In line 4 it will always be possible to find the highest value for the same arguments in transition 'allocate constituency seats' p. 66. This implies that each iteration of the while-loop will add at least 1 to the additional seats taken, allowing the while-loop to terminate at some point. The last iteration of the while-loop might find several highest quotients. Therefore there might be more seats taken on the provincial level than allocated. This satisfies the first post condition.

The second post condition is for when there are no draws. This follows from lines 7 and 8. The post condition is therefore satisfied.

The while-loop will always terminate because the algorithm will always find a highest quotient (line 4).

resolve constituency result

This transition is used when the previous transition produces draws between constituencies for certain parties with regards to where the additional seats are allocated. The method receives three parameters: an integer ranking, a party result, a constituency result. The semantics of the parameters is that for this party result the constituency wins the ranking and all other constituencies with same ranking are moved down one place.

The pre conditions for this transition are:

- Ranking must be greater than 0
- Party result parameter must exist in the constituency result parameter
- Party result parameter must have that ranking in the div list
- For the party result the ranking also exists in another constituency result within same province
- The inner state must be 'Step 5 Done'

The first pre condition says that the ranking set by the previous transition is always larger than 0. The remaining points argue that there should be a relationship between the party result, the ranking, and the constituency result. If this is not the case, then calling this method would result in inconsistencies.

The post conditions for this transition are:

- For the party result in the same province there exist no other constituency result with the same ranking
- If party result in another constituency is pushed down, and that ranking number is higher than the number of seats taken, then the seat is taken from the constituency and the number of seats taken on the provincial level is decremented by 1
- If all parties on the provincial level have additional seats taken, and that number corresponds to the number of seats allocated, then all candidates in all parties in all constituencies are arranged by descending total vote count
- The inner state changes from 'Step 5 Done' to 'Step 5 Resolved' (if there are no draws)

In order to verify the correctness we need to look at the algorithm:

```
1.  for province and party that corresponds to parameters
2.      for party in constituencies within province
3.          if constituency doesn't correspond to parameter but ranking does
4.              push down ranking by 1
5.              if new ranking value > number of seats on the provincial level then
6.                  eliminate seat from party (both on constituency and provincial level)
9.  if  additional seats taken = additional seats allocated for all parties on the provincial level  then
7.      sort all candidates in all parties in all constituencies descending by total vote count
```

In line 3 all other constituencies are found. If ranking in line 4 corresponds to the parameter, then the ranking is pushed down by 1. Lines 5 and 6 make sure that ranking numbers greater than the number of additional seats on the provincial level are eliminated. This corresponds exactly to the first two post conditions.

The last post condition follows from lines 7 and 8, which are exactly the same as in the previous transition.

There is no reason to suggest that the algorithm will not terminate.

select candidates

The pre conditions for this transition are:

- All candidates in all parties in all constituencies are arranged by descending total vote count
- On the constituency level the number of elected for parties is 0
- The total number of seats for parties on the constituency level equals constituency seats and additional seats added together
- In all constituencies the parties have more candidates than they get seats
- The party votes have been distributed to the candidates correctly according to the list organization that the party uses
- The inner state must be 'Step 5 Resolved'

The first three pre conditions argue about proper initializations. The first point, however, is a consequence of the post conditions in the previous two transitions. The fourth pre condition states, that DiVS currently doesn't handle the issue of a party getting more seats in a constituency than it has candidates (cf. subsections 2.0.1 and 3.5.3). The responsibility of the fifth pre condition has the electoral management board in each district; its correctness is therefore assumed.

The post conditions for this transition are:

- On the constituency level for each party the number of elected is equal or greater than the total number of seats allocated to the party
- The inner state changes to 'Step 6 Done' or 'Step 6 Resolved' (if there are no draws)

In order to verify the correctness we need to look at the algorithm:

1. for all party results on the constituency level
2. set the x first candidates to elected, where x = total number of seats for party in constituency
3. the next candidates having the same total number of votes as an elected one are also elected
4. accumulate the number of elected for actions on line 2 and 3

Line 2 ensures that the number of candidates elected equals the number of seats. There might, however, exist a candidate with the same number of votes further down the list. Line 3 ensures that this candidate is also elected. The total number of elected candidates, which is updated on line 4, then corresponds to the post condition; hence it is verified.

There is no reason to suggest that the algorithm will not terminate.

resolve candidates

This transition is used when the previous transition produces draws between candidates of the same party in a constituency. The method receives three parameters: a candidate result, a party result, and a constituency result. The semantic of the parameters is that candidate result running for party result has lost the draw for a seat in constituency result.

The pre conditions for this transition are:

- Candidate result parameter must be elected
- In the constituency result parameter the party result parameter must have more elected candidates than the party result has seats allocated
- The candidate result parameter must be a candidate that belongs to the party result parameter
- The party result parameter must exist in the constituency result parameter
- The inner state must be 'Step 6 Done'

The first two pre conditions state that there should be a reason to invoke the method. The third and fourth pre conditions state that there should be a relationship between candidate and party as well as between party and constituency.

The post conditions for this transition are:

- The candidate is no longer elected
- The party has 1 less elected candidate than before
- The inner state changes from 'Step 6 Done' to 'Step 6 Resolved' (if there are no draws left)

In order to verify the correctness we need to look at the algorithm:

1. locate candidate among candidates of the party in the constituency
2. set candidate to un-elected
3. decrement number of candidates elected by 1

Line 2 ensures that the candidate is no longer elected. Since the pre condition states that the candidates can be found, the first post condition is hereby satisfied.

The number of elected candidates is decremented on line 3. This satisfies post condition number 2.

The number of draws present is finite after the previous transition. For every call of this method there will be 1 less draw to resolve, and eventually all draws will be resolved. There is no reason to suggest that the algorithm will not terminate.

4.2.3 Conclusion of the Outlined Proof

The outer state machine depends upon two instances of the inner state machine, preliminary and final result. More explicitly, it depends upon the ability to come from 'Before Computing' to the final inner states 'No Passing Parties', 'Step 5 Resolved' or 'Step 6 Resolved'; the last two are for the preliminary and final result instances respectively. The discussion of the inner state machine has shown that these final states are reachable, since all transitions will terminate. The issue arising in the fourth step of the computation is resolved by adding the extra pre condition: all parties must get votes in all provinces.

It must therefore be concluded that the program correctness is verified.

5 Test

This section documents the implementation testing. The program correctness has already been verified in the previous chapter.

The test strategy is that if the tests are not failing then the system is assumed to function correctly. This strategy, however, depends on the code coverage in these tests. Every effort has made to reach as much code coverage as possible in order to increase the level of confidence that one can have in the system.

The Java classes can be categorized as either simple classes or non-simple classes. The non-simple classes are DatabaseGateway, Election, and ElectionResult, hence all other classes are categorized as simple.

This chapter will document the tests of the simple classes, tests of the non-simple classes, and lastly describe the level of confidence one can have in the system.

5.1 Method

The tests in this project are JUnit tests. They have been generated by Eclipse by adding a JUnit Test Case and selecting all public methods including the constructor or constructors. The `setUp` method has been added for most classes in order to initialize the tests. The test classes are named after the class that is under test prefixed by `Test`, and all test classes reside in the `test` source folder.

5.2. Simple Classes

Initialization of primitive data types and one object of the class under test are made in the setUp method. The primitive data types are never initialized to 0. The constructor and getter methods are then tested in order to show that the parameterized constructor is using the parameters correctly. For setters methods the set method is called with a local variable, whereas the corresponding getter method is called in order to see if the "setter" methods functions correctly.

5.3 Generator Framework

In order to test the non simple classes a generator framework has been implemented. It resides in the `generator` source folder. The framework consists of a `generator` package, a `print` package, and a `persistence` package.

The framework generates and persists ballots. The ballots have several indicators that are based on random generation. These indicators are voter turnout, ratio between party and personal votes, number of personal and party votes, and the ratio of the votes a party gets on the national level. The code coverage produced by the testing framework could be more, since the number of parties currently is fixed, and that the ratio of votes for these parties is more or less fixed to a certain level.

Not enough time is used to develop the framework. Creating special test cases therefore depends largely on manual editing of ballots and ballot journals.

5.3.1 Generator Package

The generator package consists of:

- GeneratorDriver — contains a main method
- GenerateBallots — contains methods that generate normal ballots and special ballots
- GenerateElectionStates — contains methods to that produce either Election and ElectionResult in a certain state

5.3.2 Persistence Package

The persistence package consists of a class that can persist ballots as well as fetch them again.

5.3.3 Print Package

The print package has the PrintResult class consisting of methods that print the result after the 6 calculation steps. This is useful as long as the DiVS project doesn't have any GUI to display the results graphically.

5.4 Non-simple Classes

Shown below is the documentation of the three non-simple classes.

5.4.1 DatabaseGateway

The test class uses both setUpBeforeClass and tearDownAfterClass. The former is used because the fetching of the password of the database in the properties file only needs to be done once. The latter is used because the tests involve altering data in the database and therefore the altering must be undone. The setUp method resets the state of the database to what it was before the tests started by deleting all results and changing the open state of all polling stations to 'Not Open'.

During the testing, it is assumed that the DML script in appendix D.2 has been loaded into the database. The appendix shows only portions of this script. The complete script can be found by checking out the project[58]. The script is located on the relative path '/sql/data.sql'. The following is assumed:

- there are polling stations in the database; two per district
- the number of parties should be at least 1
- polling stations have an open state that is 'Not Open' when no results are registered
- the counting of total valid or invalid votes is 0 when no results are registered
- the total number votes on the constituency level for a certain party should equal the country total as well as the total number of votes on the provincial level for a certain party should equal the country total

[58] [SVN]

- The total number of votes for the candidates in a party in a constituency must equal the total number of votes for that party.

The tests and the generator framework reflect these assumptions.

5.4.2 ElectionResult

The test class uses both setUpBeforeClass and tearDownAfterClass for the same reason as above. The same assumptions above also apply here.

The computation of the results through the ElectionResult class must complement the outlined verification in the previous chapter, both for the preliminary and the final computations. This means that all the states in the inner state machine must be reachable and all draws resolved. Test data has been created by the generator framework in order to cover the most common cases; i.e. cases where there are no issues to resolve. Moreover, test data has also been created to cover the special cases, where there are issues to resolve. This has been created for all situations except for the computation in step 2, where one possible outcome is that no parties pass the threshold.

The tests of ElectionResult use the generator framework that generates at runtime random results. There are, however, a few special cases, which are hard to produce runtime. In order for the tests to cover these special cases the persistence framework has been used; all 184 ballots are persisted and fetched only for the purpose to provoke situation where there are draws involved.

Registration of parties and candidates are not included in this project. In addition one of the pre conditions to computation step 4 is that all parties run in all provinces. This means that it has not been possible to test two special situations: no parties pass the threshold and situation after computation step 4 is irresolvable.

5.4.3 Election

The test class uses both setUpBeforeClass and tearDownAfterClass for the same reason as in subsection 5.4.1, as well as maintaining the same assumptions.

The test of the election class must complement the outlined verification in the previous chapter. The states in the outer state chart must be reachable. The exact same test data as above has been used in order to cover both the common and the special cases (e.g. draw issues).

The generator framework has been used in the same when way testing the Election class as described in the previous subsection.

5.4 Test Results

The results show that both simple and non-simple classes pass the tests. Below are test result outlined for the special cases when draws are involved:

Step	No	Description	Result

1	1	Draw involves at least two parties	Ok
1	2	Draw involves at least two independent candidates	Ok
1	3	Draw involves both parties and candidates	Ok
3	1	Draw involves at least two parties	Ok
4	1	Draw involves at least two parties in the same province	Ok
4	2	Draw involves one party in several provinces	Ok
4	3	Draw involves several parties in several provinces	Not tested
5	1	Draw involves several constituencies	Ok
6	1	Draw involves several candidates	Ok

5.5 Setup Tests

From the repository's website a zip file containing all test cases can be downloaded. The tests have been generated by the generator framework. The file must be unzipped to a location, and the Persistence class must be informed where the files are in order to reproduce the tests.

5.5.1 Naming of Persisted Ballots

The persisted files are named inside the Persistence class. The names follow a certain pattern:

- File has a leading "f" or "t". "f" corresponds to the final computations phase and "t" to the preliminary (before preliminary computations phase got its name it was called 'temporal')
- File extension is ".ser" (short for serialized)
- Last part of file name is polling station id with leading zeros, e.g. "041" represents polling station number 41
- An "_<inner status>_" is between the "f" or "t" for the special cases
- "f_4_077.ser" represents the ballot from the final counting of polling station 77. Its content, together with the other persisted files for inner state 4, will produce a draw after state 4 ('Step 4 Done')

5.6 Level of Confidence

The level of confidence that one can have in the system is high. The outlined proof alone does not give much confidence. The tests and the outlined proof combined, however, provide a lot of confidence, since most the scenarios have been tested and they pass the tests.

Conclusion

This report documents the transitions from the Danish electoral law through BON to a fully functional Java program. The transition from the law into informal BON is sound, except for parts of the electoral law deemed irrelevant. The transition from informal BON into formal BON is sound, except for the limitations. The runtime checker of the Eclipse plug-in Beetlz has made sure that the formal BON and the Java implementation match, excluding the quantified expressions, which have been checked by hand. This means that the transition from formal BON into Java is sound. It can therefore be concluded, that DiVS is a sound model of the Danish electoral law.

A thorough outlined proof has been carried out in order to verify DiVS' program correctness. The results of the proof is that the correctness has been verified, since all transitions in both the outer and inner state machines are sound and complete. The transitions are sound because all the post conditions are verified. The transitions are complete because all transitions will terminate and, given the pre conditions, the final state is always reachable.

There are, however, some issues with the Danish electoral law that need to be dealt with. It has been shown that a situation can arise where no parties pass the threshold; which implies that there are only 135 seats representing Denmark in the Folketing. It has also been shown that if all parties are not running in all provinces a situation can occur where not all parties get their additional seats allocated to the provincial level. Although these situations are unlikely they must be dealt with. These issues along with others are listed in the appendix F.

Tests have been carried out in order to complement the verification of DiVS. All the scenarios listed in informal BON have been tested. A few more tests must be carried out in order to be complete. The tests results show that one can have a high degree of confidence in DiVS, especially since most special cases have been tested. The confidence can be higher when the tests that are left out are included.

When all limitations are implemented and a GUI has been developed it should be possible for the Ministry of the Interior to use DiVS to control the election procedures instead of their current non open source software. Section 3.5 and appendix G accounts for how this should be done. Using DiVS provides transparency that the current program in the Ministry of the Interior doesn't have.

The source code as well as the API can be found on the following URL:

http://code.google.com/p/danishvotingsystem/

By checking out the code or by downloading the API, one can see what a verified implementation of the Danish Voting System looks like.

Bibliography

[DEL] Danish Electoral Law,
https://www.retsinformation.dk/forms/r0710.aspx?id=135719

[PESD] Parliamentary Electoral System in Denmark, published by the Ministry of the Interior
http://elections.ism.dk/parliament-elections/Documents/Parlelectsys.pdf

[ESD] The Electoral System in Denmark, published by the Ministry of the Interior
http://elections.sm.dk/parliament-elections/Documents/FTelections.pdf

[CAD] The Constitutional Act of Denmark
http://www.grundloven.dk/

[Barrett08] Reasoning About Java Persistence
UCD, School of Computer Science and Informatics, February 2008

[Waldén&Nerson94] Seamless Object-oriented Software Architecture
http://bon-method.com/index_normal.htm

[KindSoftware] http://kind.ucd.ie/

[Cochran&Kiniry10] Vótáil: PR-STV Ballot Counting Software for Irish Elections
Dermot Cochran & Joseph R. Kiniry; IT-University of Copenhagen, Denmark, 2010,

[Kiniry et.al06] Soundness and Completeness Warnings in ESC/Java2
Joseph R. Kiniry, Alan E. Morkan & Barry Denby; UCD 2006; ISBN: 1-59593-586-X

[Darulová09] Beetlz – BON Software Model Consistency Checker for Eclipse
UCD, School of Computer Science and Informatics, May 2009

[Schumacher01] Chapter Zero, Fundamental Notions of Abstract Mathematics, 2nd Edition
Carol Schumacher; Addison Wesley Longman 2001; ISBN: 0-201-43724-4

[Rubin02] Security Considerations for Remote Electronic Voting over the Internet
AT&T Labs – Research, Florham Park, NJ, 2002

[Sørensen&Lewinski08] Trust in E-voting Systems
IT-University of Copenhagen & Copenhagen University, March 2008

[Version2] Elektroniske valg lader vente på sig, November 2007

http://mobil.version2.dk/artikel/4760

[Computerworld] Unix-veteran kinder valgresultatet før alle andre, November 2007
http://www.computerworld.dk/art/42588/unix-veteran-kender-valgresultatet-foer-alle-andre

[Sørensen04] En Introduktion til Sandsynlighedsregning, 5. udgave
Michael Sørensen; Afdeling for anvendt matematik og statistik, Københavns Universitet; ISBN: 87-7834-599-5

[Hunt&Thomas99] The Pragmatic Programmer
Andy Hunt & David Thomas; Addison-Wesley 1999; ISBN: 0-201-61622-X

[Cadle&Yeates] Project Management for Information Systems, 4th Edition
James Cadle & Donald Yeates; Prentice Hall 2004; ISBN: 0-273-68580

[Vahid&Givargis] Embedded System Design – A Unified Hardware/Software Introduction
Frank Vahid & Tony Givargis; John Wiley & Sons, Inc. 2002; ISBN: 0-471-38678-2

[Fowler05] UML Distilled, 3rd Edition, A Brief Guide to the Standard Object Modeling Language
Martin Fowler; Pearson Education, Inc. 2004; ISBN: 0-321-19368-7

[Kurose&Ross08] Computer Networking, 4th Edition, A Top-Down Approach
James F. Kurose & Keith W. Ross; Pearson Education, Inc. 2008; ISBN: 0-321-51325-8

[Gronback09] Eclipse Modeling Project, A Domain-Specific Language (DSL) Toolkit
Richard C. Gronback; Addison-Wesley 2009; ISBN: 0-321 53407-7

[Deitel&Deitel09] Internet & World Wide Web, 4th Edition, How to Program
P. J. Deitel & H. M. Deitel; Pearson Education International, Inc. 2009; ISBN: 0-13-603542-6

[Adams05] The Hitchhiker's Guide to the Galaxy
Douglas Adams; Aschehoug 2005; ISBN: 87-11-29012-9

[Ben-Ari01] Mathematical Logic for Computer Science, 2nd Edition
Mordechai Ben-Ari; Springer 2001; ISBN: 1-85233-319-7

[Sommerville07] Software Engineering, 8th Edition
Ian Sommerville; Addison-Wesley 2007; ISBN: 0-321-31379-8

[ShashaBonnet03] Database Tuning, Principles, Experiments, and Troubleshooting Techniques
Dennis Shasha & Philippe Bonnet; Morgan Kaufmann 2003; ISBN: 1-55860-753-6

[Lewis&Loftus05] Java Software Solutions, 4th Edition, Foundations of Program Design
John Lewis & William Loftus; Pearson Education International, Inc. 2005; ISBN: 0-321-31246-5

[Andrews00] Foundations of Multithreaded, Parallel, and Distributed Programming
Gregory R. Andrews; Addison Wesley Longman, Inc. 2000; ISBN: 0-201-35752-6

[Møller&Schwartzbach06] Introduction to XML and Web Technologies
Anders Møller & Michael Schwartzbach; Addison Wesley 2006; ISBN-13: 978-0-321-26966-9

[Elmasri&Navathe07] Fundamentals of Database Systems, 5th Edition
Ramez Elmasri & Shamkant B. Navathe; Addison Wesley 2007; ISBN: 0-321-41506-X

[Cormen et.al.03] Introduction to Algorithms, 2nd Edition
Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest & Clifford Stein; The McGraw-Hill Book Company 2001; ISBN: 0-07-013151-1

[Jensen&Høholdt00] Grundlæggende Matematik
Helge Elbrønd Jensen & Tom Høholdt; Institut for matematik, DTU 2000; ISBN: 87-88764-67-2

[Molich03] Brugervenligt Webdesign, 2. Udgave
Rolf Molich; Ingeniøren bøger 2003; ISBN: 87-571-2481-7

[Patterson&Hennessy07] Computer Organization and Design, 3rd Edition, The Hardware/Software Interface
David A. Patterson & John L. Hennessy; Morgan Kaufmann 2007; ISBN: 978-0-12-370606-5

[Russell&Norvig03] Artificial Intelligence, 2nd Edition, A Modern Approach
Stuart Russell & Peter Norvig; Prentice-Hall 2003; ISBN: 0-13-080302-2

[Chothewitz94] Gud kaster ikke terning, En bog om Albert Einstein
David Chotjewitz; Gyldendal 1995; ISBN: 87-00-21346-2

Appendix A – Introduction to the BON Method

BON is a language with a concrete syntax. Its semantics are used primarily to describe a structured object oriented system. BON has three key properties (or major ideas): seamlessness, reversibility, and contracting[59]. It is seamless because one can model everything from the analysis to the design using the same language. This is done in a continuous process ensuring a direct mapping between a problem and its software solution[60]. It is reversible because everything you express in BON can be refined to a software system. Changing the software will change the BON and vice versa. It fits together semantically, but there are also tools that do this[61]. And it is contract-based because the BON (and the software) will contain assertions that define the semantics of a class. The assertions are pre- and post-conditions on features and invariants on classes.

BON has both a graphical and a textual syntax. The textual syntax allows the language to generate chart-based notation, which is usable to non-programmers.

There are two levels of BON specifications: informal BON and formal BON.

Informal BON allows you to specify primary concepts within the system (a class dictionary) plus descriptions on how the concepts interact dynamically (scenarios, events, and creation chart). Much of the informal BON is expressed using a natural language (possibly in a structured way). This allows non computer scientists to participate in the process. The informal BON can be thought of as requirement analysis or requirement specifications. The concepts contain usually three things: queries, commands, and constraints. Queries can be thought of as questions you ask the created objects and having them answer without changing their state. E.g. a class 'Candle' may be asked: "What is your color?" And the 'Candle' will answer: "White" (and not change state). A command is when you ask a class to do something that does change its state. E.g. "Light!" You add constraints to a class to ensure certain properties. In this working analogy, a 'Candle' can have the constraint: "Each candle has at least one wick".

Formal BON is in the world of types. The class dictionary gets specified formally. Specified classes at the formal level typically contain two things: features and invariants. From the informal level queries and commands become features in the formal level whereas constraints become invariants. Commands never return anything.

Both informal and formal BON have clusters where related concepts are grouped together. Clusters can have nested clusters.

Let it be assumed that the software system is written in Java. There exists a tool called Beetlz that can translate formal BON into Java. Moreover Beetlz features a runtime checker that ensures that the formal BON and the Java code match. Current version (April 2011) does not translate quantified expression, meaning that the BON contracts are not translated into Java Modeling Language (JML) contrasts on the Java methods and vice-versa; the runtime checker doesn't check the quantified expressions.

[59] [BB] p. xiii
[60] [BB] p. xiii
[61] [HSE] #4 – 0:02:00

Java implementations that contain JML contracts can be checked by static checkers for inconsistencies. The inconsistencies can be invariants and pre and post conditions are in conflict. But they can also be that the post conditions cannot be verified assuming the pre conditions taking into account method bodies.

Since the checking can be done the Java methods don't have to be programmed defensively; i.e. check all input for ranges and throw exceptions when input is illegal. This implies that a Java method will accordingly to the BON method just contain what is should do, and nothing else, because the pre conditions can simply be assumed.

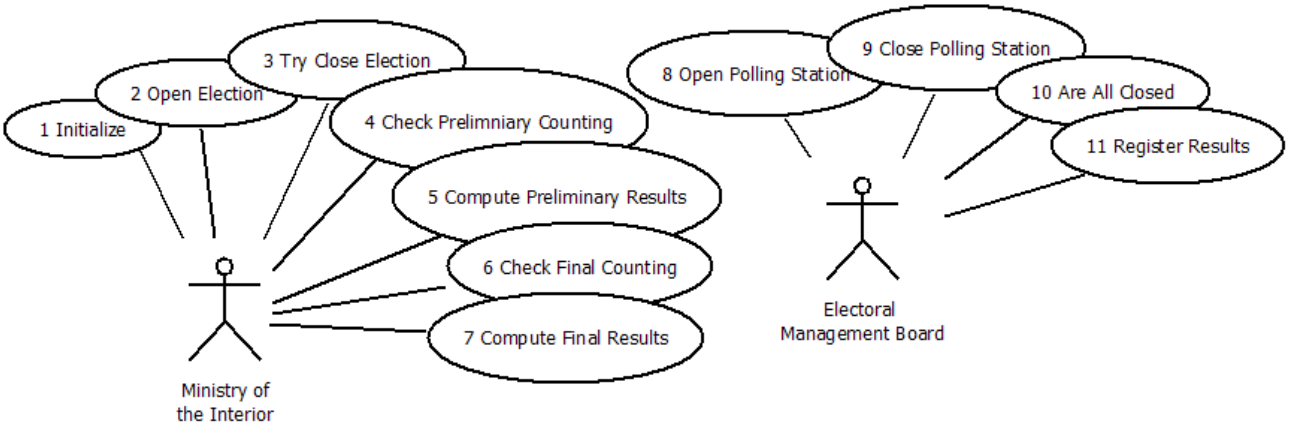
Appendix B – Classic Requirements Specifications

The main report contains requirements using the BON method. Readers can use this section to compare and contrast between the BON method and the classic requirements specifications.

The architecture introduced in section 1.3 is assumed, meaning that DiVS has two interfaces two the outside world – namely through a GUI and through a web server.

B.1 Use cases diagram

There are many actors of an election. Subsection 1.1.3 shows a complete overview of them. The actors using DiVS are just the Ministry of the Interior and electoral management board. The use cases diagram can be seen below.



Certain aspects of the election are not included in this project. These are e.g. the nomination of candidates within each party, projections, and more. Login and logout features have been left out for simplicity reasons.

B.2 Use Cases

The use cases are listed here.

Name	1 Initialize
Summary	Before the election the system has to be initialized
Actors	Ministry of the Interior
Preconditions	System is running. All necessary data is available in the database. Election status is 'Before Election'
Basic course of events	1. Ministry of the Interior presses the 'Initialize Election' button.
Alternative path	n/a
Post conditions	DiVS changes state to 'Initialized'
Notes	n/a
Author	ÓK

Name	2 Open Election
Summary	Ministry of the Interior wants to enable polling stations to open
Actors	Ministry of the Interior
Preconditions	System is running. Election status is 'Initialized'
Basic course of events	1. Ministry of the Interior presses the 'Open Election' button.
Alternative path	n/a
Post conditions	DiVS changes state to 'Election opened'
Notes	n/a
Author	ÓK

Name	3 Try Close Election
Summary	Election day has come to a close and the counting of votes should commence. Ministry of the Interior wants to close the election, but this can only be done when all polling stations are closed.
Actors	Ministry of the Interior
Preconditions	System is running. Election status is 'Election Open'.
Basic course of events	1. Ministry of the Interior presses the 'Close Election' button. 2. DiVS changes status to 'Election Closed' if all polling stations are closed
Alternative path	n/a
Post conditions	Election status is 'Election closed' or the same as before
Notes	n/a
Author	ÓK

Name	4 Check Preliminary Counting
Summary	Ministry of the Interior wants to check if all preliminary results have arrived from the polling stations.
Actors	Ministry of the Interior
Preconditions	System is running. Election status is 'Election Closed'.
Basic course of events	1. Ministry of the Interior presses the 'Check Preliminary Counting' button. 2. If all results have arrived from the polling stations then system changes election status to 'Preliminary Counting Competed'
Alternative path	n/a
Post conditions	Election status is 'Preliminary Counting Competed' or the same as before. If the former is the case, then the preliminary results are loaded from the database
Notes	n/a
Author	ÓK

Name	5 Compute Preliminary Results
Summary	Ministry of the Interior now wants to compute the result of the election

Actor	Ministry of the Interior
Preconditions	System is running. Election status is 'Preliminary Counting Competed'.
Basic course of events	1. Ministry of the Interior presses the 'Compute step 1' button. 2. DiVS computes step 1 3. Ministry of the Interior presses the 'Compute step 2' button. 4. DiVS computes step 2 5. Ministry of the Interior presses the 'Compute step 3' button. 6. DiVS computes step 3 7. Ministry of the Interior presses the 'Compute step 4' button. 8. DiVS computes step 4 9. Ministry of the Interior presses the 'Compute step 5' button. 10. DiVS computes step 5
Alternative path	2 a) If there are draws then lots are drawn by Ministry of the Interior. Afterwards Ministry of the Interior informs DiVS of the results of the draws 4 a) If there are no passing parties then further computation is put to rest 6 a) If there are draws then lots are drawn by Ministry of the Interior. Afterwards Ministry of the Interior informs DiVS of the results of the draws 8 a) If there are draws then lots are drawn by Ministry of the Interior. Afterwards Ministry of the Interior informs DiVS of the results of the draws 10 a) If there are draws then lots are drawn by Ministry of the Interior. Afterwards Ministry of the Interior informs DiVS of the results of the draws
Post conditions	Election status is 'Preliminary Results Computed'.
Notes	n/a
Author	ÓK

Name	**6 Check Final Counting**
Summary	Ministry of the Interior wants to check if all final results have arrived from the polling stations
Actors	Ministry of the Interior
Preconditions	System is running. Election status is 'Preliminary Results Computed'.
Basic course of events	1. Ministry of the Interior presses the 'Check Final Counting' button. 2. If all results have arrived from the polling stations then system changes election status to 'Final Counting Competed'
Alternative path	n/a
Post conditions	Election status is 'Final Counting Competed' or the same as before. If the former is the case, then the final results are loaded from the database
Notes	n/a
Author	ÓK

Name	7 Compute Final Results
Summary	Ministry of the Interior now wants to compute the final results of the election
Actor	Ministry of the Interior
Preconditions	System is running. Election status is 'Final Counting Competed'.
Basic course of events	1-10 are the same as in use case 5 and therefore not repeated here. 11 Ministry of the Interior presses the 'Compute step 6' button. 12 DiVS computes step 6
Alternative path	12 a) If there are draws then lots are drawn by Ministry of the Interior. Afterwards Ministry of the Interior informs DiVS of the results of the draws
Post conditions	Election status is 'Final Results Computed'.
Notes	n/a
Author	ÓK

Name	8 Open Polling Station
Summary	Electoral management board in a district wants to notify to the Ministry of the Interior that some of its polling stations have opened
Actors	Electoral management board
Preconditions	System is running. Election status is 'Election Open'.
Basic course of events	1. Electoral management board logs on to the national level system through a web browser 2. DiVS' web server shows a list of the polling stations that belong to the district of the electoral management board 3. Manager presses the 'Open Polling Station' button next to the polling stations shown
Alternative path	n/a
Post conditions	Polling stations change status from 'Not Open' to 'Open'.
Notes	n/a
Author	ÓK

Name	9 Close Polling Station
Summary	Electoral management board in a district wants to notify to the Ministry of the Interior that some of its polling stations have closed
Actors	Electoral management board
Preconditions	System is running. Election status is 'Election Open'.
Basic course of events	1. Electoral management board logs on to the national level system through a web browser 2. DiVS' web server shows a list of the polling stations that belong to the district of the electoral management board 3. Manager presses the 'Close Polling Station' button next to the polling stations shown
Alternative path	n/a
Post conditions	Polling stations change status from 'Open' to 'Closed'.

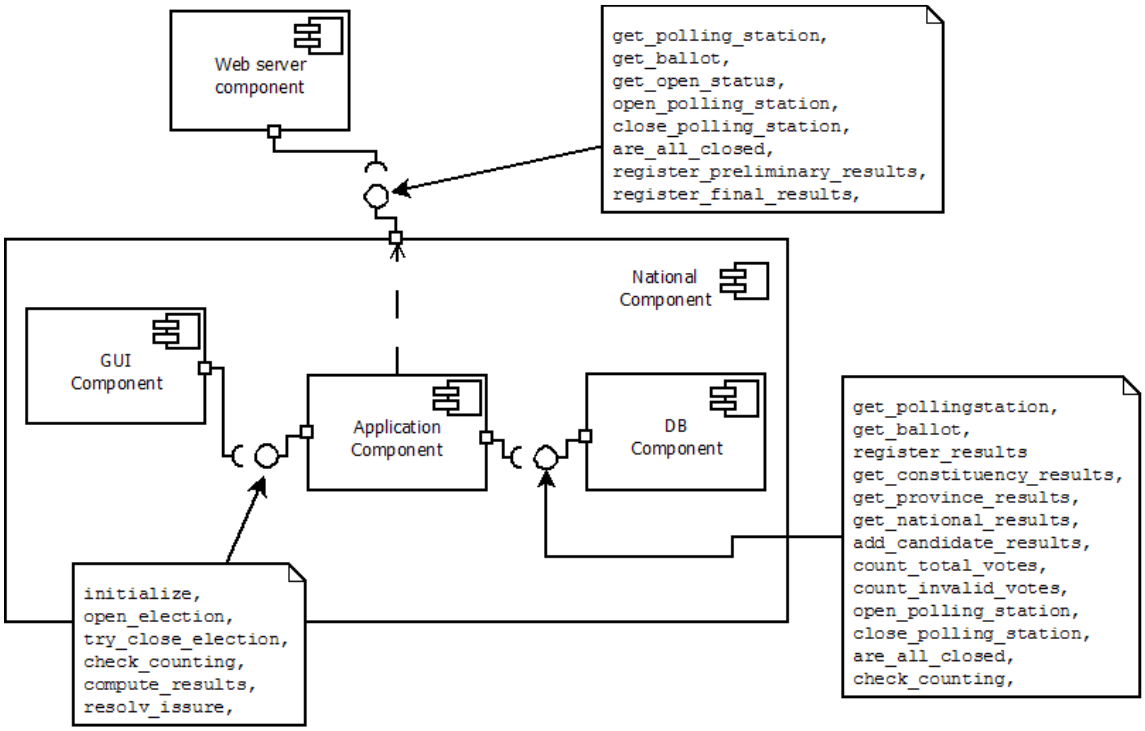
Notes	n/a
Author	ÓK

Name	**10 Are All Closed**
Summary	Electoral management board in a district wants to know if all polling stations have closed, since they must not initiate the counting before all polling stations in the country are closed
Actors	Electoral management board
Preconditions	System is running. Election status is 'Election Open' or 'Election Closed'
Basic course of events	1. Electoral management board logs on to the national level system through a web browser 2. DiVS' web server shows a red or green text saying either "Counting must not commence" or "Counting may commence" respectively dependant on the status
Alternative path	n/a
Post conditions	n/a
Notes	n/a
Author	ÓK

Name	**11 Register Result**
Summary	Electoral management board has received the results from its polling stations and now wants to register the results.
Actors	Electoral management board
Preconditions	System is running. Open status of the polling stations is 'Closed'
Basic course of events	1. Electoral management board logs on to system at polling station 2. Electoral management board presses the 'Register Result' button 3. For each polling station electoral management board types in the results on forms that the web server provides 4. Electoral management board presses the 'Save' button 5. Step 2-4 is repeated for each polling station
Alternative path	n/a
Post conditions	The results are stored in the database through DiVS
Notes	The results can be preliminary results and final results
Author	ÓK

B.3 UML Component Diagram

DiVS consists of several components. Their relation is shown in the diagram below.


Web server component
get_polling_station,
get_ballot,
get_open_status,
open_polling_station,
close_polling_station,
are_all_closed,
register_preliminary_results,
register_final_results,
National Component
GUI Component
Application Component
DB Component
get_pollingstation,
get_ballot,
register_results
get_constituency_results,
get_province_results,
get_national_results,
add_candidate_results,
count_total_votes,
count_invalid_votes,
open_polling_station,
close_polling_station,
are_all_closed,
check_counting,
initialize,
open_election,
try_close_election,
check_counting,
compute_results,
resolv_issure,

Appendix C – Electoral map of Denmark

C.1 Provinces

No	Name	Constituency seats	Additional Seats
1	Metropolitan Copenhagen	38	11
2	Sealand- Southern Denmark	51	15
3	Northern and Central Jutland	46	14

C.2 Multi-member constituencies

No	Province	Name	Constituency seats
1	1	Københavns Storkreds	15
2	1	Københavns Omegns Storkreds	11
3	1	Nordsjællands Storkreds	10
4	1	Bornholms Storkreds	2
5	2	Sjællands Storkreds	17
6	2	Fyns Storkreds	18
7	2	Sydjyllands Storkreds	16
8	3	Østjyllands Storkreds	21
9	3	Vestjyllands Storkreds	14
10	3	Nordjyllands Storkreds	11

C.3 Districts

No	Constit.	Name
1	1	Østerbro
2	1	Sundbyvester
3	1	Indre By
4	1	Sundbyøster
5	1	Nørrebro
6	1	Utterslev
7	1	Brønshøj
8	1	Valby
9	1	Vesterbro
10	1	Falkoner
11	1	Slots
12	1	Tårnby
13	2	Gentofte
14	2	Lyngby
15	2	Gladsaxe
16	2	Rødovre
17	2	Hvidovre
18	2	Brøndby
19	2	Taastrup
20	2	Ballerup
21	3	Helsingør
22	3	Fredensborg
23	3	Hillerød
24	3	Frederikssund
25	3	Egedal
26	3	Rudersdal
27	4	Rønne
28	4	Aakirkeby
29	5	Lolland
30	5	Guldborgsund
31	5	Vordingborg
32	5	Næstved
33	5	Faxe
34	5	Køge
35	5	Greve
36	5	Roskilde
37	5	Holbæk

38	5	Kalundborg
39	5	Ringsted
40	5	Slagelse
41	6	Odense Øst
42	6	Odense Vest
43	6	Odense Syd
44	6	Assens
45	6	Middelfart
46	6	Nyborg
47	6	Svendborg
48	6	Faaborg
49	7	Sønderborg
50	7	Aabenraa
51	7	Tønder
52	7	Esbjerg By
53	7	Esbjerg Omegn
54	7	Varde
55	7	Vejen
56	7	Vejle Nord
57	7	Vejle Syd
58	7	Fredericia
59	7	Kolding Nord
60	7	Kolding Syd
61	7	Haderslev
62	8	Århus Syd
63	8	Århus Vest
64	8	Århus Nord
65	8	Århus Øst
66	8	Djurs
67	8	Randers Nord
68	8	Randers Syd
69	8	Favrskov
70	8	Skanderborg
71	8	Horsens
72	8	Hedensted
73	9	Struer
74	9	Skive
75	9	Viborg Vest
76	9	Viborg Øst
77	9	Silkeborg Nord
78	9	Silkeborg Syd
79	9	Ikast
80	9	Herning Syd
81	9	Herning Nord
82	9	Holstebro
83	9	Ringkøbing
84	10	Frederikshavn
85	10	Hjørring
86	10	Brønderslev
87	10	Thisted
88	10	Himmerland
89	10	Mariagerfjord
90	10	Aalborg Øst
91	10	Aalborg Vest
92	10	Aalborg Nord

Appendix D – Database of the System

This appendix contains DDL for creating the database tables. The ER diagram can be found in section 2.3.1.

D.1 DDL

The DDL below is used to create all tables including foreign keys in the database.

```
drop table dvs.Voter;
drop table dvs.InvalidVotes;
drop table dvs.Votes;
drop table dvs.BallotJournal;
drop table dvs.PollingStation;
drop table dvs.PartyList;
drop table dvs.Candidate;
drop table dvs.Ballot;
drop table dvs.District;
drop table dvs.Constituency;
drop table dvs.Province;
drop table dvs.Party;

create table dvs.Province (
 province_id int not null,
 name char(50) not null,
 constituency_seats int not null,
 additional_seats int not null,
 primary key(province_id)
);

create table dvs.Constituency (
 constituency_id int not null,
 province_id int not null,
 name char(50) not null,
 constituency_seats int not null,
 bornholm smallint not null,
 primary key(constituency_id),
 constraint prov_const Foreign Key
(province_id) REFERENCES Province
(province_id)
);

create table dvs.District (
 district_id int not null,
 constituency_id int not null,
 name char(50) not null,
 primary key(district_id),
 constraint const_distr Foreign Key
```

```
create table dvs.PartyList (
 pl_id int NOT NULL GENERATED ALWAYS AS
IDENTITY (START WITH 1, INCREMENT BY 1),
 party_id int not null,
 constituency_id int not null,
 candidate_id int not null,
 primary key(pl_id),
 constraint party_pl Foreign Key
(party_id) REFERENCES Party (party_id),
 constraint const_pl Foreign Key
(constituency_id) REFERENCES
Constituency (constituency_id),
 constraint cand_pl Foreign Key
(candidate_id) REFERENCES Candidate
(candidate_id)
);

create table dvs.BallotJournal (
 bj_id int NOT NULL GENERATED ALWAYS AS
IDENTITY (START WITH 1, INCREMENT BY 1),
 ballot_id int not null,
 candidate_id int,
 party_id int,
 standing int,
 primary key(bj_id),
 constraint ballot_bc Foreign Key
(ballot_id) REFERENCES Ballot
(ballot_id),
 constraint cand_bc Foreign Key
(candidate_id) REFERENCES Candidate
(candidate_id),
 constraint party_bc Foreign Key
(party_id) REFERENCES Party (party_id)
);

create table dvs.Votes (
 bj_id int not null,
 ps_id int not null,
 temp_final int not null,
```

```
(constituency_id) REFERENCES
Constituency (constituency_id)
);

create table dvs.PollingStation (
 ps_id int not null,
 district_id int,
 name char(50) not null,
 no_of_reg_voters int not null,
 open_state int not null,
 primary key(ps_id),
 constraint distr_ps Foreign Key
(district_id) REFERENCES District
(district_id)
);

create table dvs.Party (
 party_id int not null,
 name char(50) not null,
 letter char(2) not null,
 standing int not null,
 primary key(party_id)
);

create table dvs.Ballot (
 ballot_id int not null,
 district_id int not null,
 primary key(ballot_id),
 constraint distr_ballot Foreign Key
(district_id) REFERENCES District
(district_id)
);

create table dvs.Candidate (
 candidate_id int not null,
 name char(50) not null,
 cpr char(11) not null,
 position char(25),
 address char(50),
 party_id int,
 primary key(candidate_id),
 constraint party_cand Foreign Key
(party_id) REFERENCES Party (party_id)
);
```

```
 party_votes int not null,
 personal_votes int not null,
 primary key (bj_id, ps_id, temp_final),
 constraint res_ps Foreign Key (ps_id)
REFERENCES PollingStation (ps_id),
 constraint bj_res Foreign Key (bj_id)
REFERENCES BallotJournal (bj_id)
);

create table dvs.InvalidVotes (
 ballot_id int not null,
 ps_id int not null,
 temp_final int not null,
 votes int not null,
 primary key (ballot_id, ps_id,
temp_final),
 constraint ps_invv Foreign Key (ps_id)
REFERENCES PollingStation (ps_id),
 constraint bal_invv Foreign Key
(ballot_id) REFERENCES Ballot
(ballot_id)
);

create table dvs.Voter (
 voter_id int not null,
 name char(50) not null,
 ps_id int not null,
 primary key (voter_id),
 constraint voter_ps Foreign Key (ps_id)
REFERENCES PollingStation (ps_id)
);
```

D.2 DML

The DML below contains partially the electoral map, parties, ballots, candidate, and ballot journals. This has been used in order to create test data. Notice that polling stations are not real polling stations. There for every district exactly 2 fictional polling stations, which makes the total number of polling stations 184. The actual number of polling station in the country is over 1.600. The relationship between districts and polling stations being 1-to-many and not 1-to-1 is better for testing, since it could be a source of errors.

```
insert into Province values (1, 'Metropolitan Copenhagen', 40, 11);
insert into Province values (2, 'Sealand- Southern Denmark', 49, 15);
insert into Province values (3, 'Northern and Central Jutland', 46, 14);

insert into Constituency values (1, 1, 'Københavns Storkreds', 17, 0);
insert into Constituency values (2, 1, 'Københavns Omegns Storkreds', 11, 0);
insert into Constituency values (3, 1, 'Nordsjællands Storkreds', 10, 0);
insert into Constituency values (4, 1, 'Bornholms Storkreds', 2, 1);

insert into Constituency values (5, 2, 'Sjællands Storkreds', 17, 0);
insert into Constituency values (6, 2, 'Fyns Storkreds', 16, 0);
insert into Constituency values (7, 2, 'Sydjyllands Storkreds', 16, 0);

insert into Constituency values (8, 3, 'Østjyllands Storkreds', 21, 0);
insert into Constituency values (9, 3, 'Vestjyllands Storkreds', 14, 0);
insert into Constituency values (10, 3, 'Nordjyllands Storkreds', 11, 0);

insert into District values (1, 1, 'Østerbro');
insert into District values (2, 1, 'Sundbyvester');
insert into District values (3, 1, 'Indre By');
…
insert into PollingStation values (1, 1, 'Østerbro', 19456, 0);
insert into PollingStation values (2, 1, 'Østerbro II', 14456, 0);
insert into PollingStation values (3, 2, 'Sundbyvester', 33456, 0);
insert into PollingStation values (4, 2, 'Sundbyvester II', 27456, 0);
insert into PollingStation values (5, 3, 'Indre By', 18496, 0);
insert into PollingStation values (6, 3, 'Indre By II', 14496, 0);
…
insert into Ballot values (1, 1);
insert into Ballot values (2, 2);
insert into Ballot values (3, 3);
…
insert into Party values (1, 'Socialdemokraterne', 'A', 1);
insert into Party values (2, 'Det Radikale Venstre', 'B', 1);
insert into Party values (3, 'Det Konservative Folkeparti', 'C', 1);
insert into Party values (4, 'Centrum Demokraterne', 'D', 1);
insert into Party values (5, 'Socialistisk Folkeparti', 'F', 1);
insert into Party values (6, 'Kristen Demokraterne', 'K', 1);
insert into Party values (7, 'Dansk Folkeparti', 'O', 1);
insert into Party values (8, 'Venstre', 'V', 1);
insert into Party values (9, 'Liberal Alliance', 'Y', 1);
insert into Party values (10, 'Enhedslisten', 'Ø', 1);
…
insert into Candidate (candidate_id, name, cpr, party_id) values (1,'Pia
Bang','xxxxxx-yyyy',1);
```

```
insert into BallotJournal (ballot_id, party_id, candidate_id, standing) values
(1,1,null, -1);
insert into BallotJournal (ballot_id, party_id, candidate_id, standing) values
(1,null,1,2);
insert into BallotJournal (ballot_id, party_id, candidate_id, standing) values
(2,null,1,2);
insert into BallotJournal (ballot_id, party_id, candidate_id, standing) values
(3,null,1,1);
insert into BallotJournal (ballot_id, party_id, candidate_id, standing) values
(4,null,1,2);
insert into BallotJournal (ballot_id, party_id, candidate_id, standing) values
(5,null,1,2);
insert into BallotJournal (ballot_id, party_id, candidate_id, standing) values
(6,null,1,2);
insert into BallotJournal (ballot_id, party_id, candidate_id, standing) values
(7,null,1,2);
insert into BallotJournal (ballot_id, party_id, candidate_id, standing) values
(8,null,1,2);
insert into BallotJournal (ballot_id, party_id, candidate_id, standing) values
(9,null,1,2);
insert into BallotJournal (ballot_id, party_id, candidate_id, standing) values
(10,null,1,2);
insert into BallotJournal (ballot_id, party_id, candidate_id, standing) values
(11,null,1,2);
insert into BallotJournal (ballot_id, party_id, candidate_id, standing) values
(12,null,1,2);
```

Appendix E – Election Results Computation Examples

Albert Einstein once said, "Learning by example isn't the best way to learn. It's the only way to learn"[62].This appendix therefore contains a simplified example computation of the Danish electoral law. The examples are heavily inspired by the existing downloadable PDFs with examples provided by the Ministry of the Interior[63]. The six computation steps are demonstrated below.

E.1 Allocate Constituency Seats

For each constituency the computation looks like this. Note that the quotients use the divisor method d'Hondt (1, 2, 3, etc.):

Constituency X								
Parties	A		B		C		D	
	Seat no		Seat no		Seat no		Seat no	
Votes		12000		21000		33000		45000
Quot. 1	7.	12000	4.	21000	2.	33000	1.	45000
Quot. 2		6000	10.	10500	5.	16500	3.	22500
Quot. 3				7000	9.	11000	6.	15000
Quot. 4						8250	8.	11250
Quot. 5								9000
Total	1		2		3		4	

Initially there is one quotient per party. These are compared and party D gets the first seat. This means that party D gets a new quotient (22500); the second divisor is 2 according to the d'Hondt method. The next seat goes to party C, and so forth.

E.2 Determine Threshold

In order to pass the threshold a party must pass one of these three tests:

a) The party receives at least one constituency seat
b) The party gets on the provincial level, at least the number of votes that corresponds to the average votes / seats ratio. This must be obtained in at least two out of the three provinces
c) The party receives at least 2.0 percent of the valid votes on the national level

E.3 Allocate Additional Seats on National Level

Before the computation may start the quota must be calculated as the total number of valid votes for passing parties divided by 175.

quota: 2502500 / 175 = 14300

[62] [Chotjewitz95]
[63] [PESD], [ESD]

On the national level, the computations look like this:

	Party votes	Quotient	Highest fraction	Total seats	Constituency seats	Additional seats
Total	2502500	175.000		175	135	40
Party A	900000	62,937	1	63	52	11
Party B	750000	52,448		52	43	9
Party C	550000	38,462	1	39	27	12
Party D	302500	21,154		21	13	8

The quotients are calculated on basis of party votes / quota. All parties get the integer number of the quotient. When this has been done there are 2 seats left. The party with the highest fraction gets the first seat: party A with fraction 937. The party with the second highest fraction gets the second seat: party C with fraction 462.

E.4. Allocate Additional Seats on the provincial level

The computations look like this. Note that the quotients use the divisor method Sainte Laguë (1, 3, 5, etc.):

	Metropolitan Copenhagen. 14 constituency seats and 2 additional seats							
Party	A		B		C		D	
	Seat no		Seat no		Seat no		Seat no	
Votes		3000		1450		22100		29000
Quot. 1	1.	3000		X		X		X
Quot. 2		1000		X		X		X
Quot. 3				X		X		X
Quot. 4				2071		X		X
Quot. 5						X		X
Quot. 6						2009		X
Quot. 7							2.	2231
Quot. 8								1933
	Sealand and Southern Denmark. 19 constituency seats and 5 additional seats							
Party	A		B		C		D	
	Seat no		Seat no		Seat no		Seat no	
Votes		11000		16500		21500		36500
Quot. 1		X		X		X		X
Quot. 2		X		X		X		X
Quot. 3	3.	2200		X		X		X
Quot. 4		1571		X		X		X
Quot. 5				1833		X		X
Quot. 6				1500	8.	1955		X
Quot. 7					10.	1654		X
Quot. 8						1433		X
Quot. 9							4.	2147
Quot. 10							9.	1921
Quot. 11								1738
	Northern and Central Jytland. 17 constituency seats and 3 additional seats.							

Party	A		B		C		D	
	Seat no		Seat no		Seat no		Seat no	
Votes		9500		14000		18500		35000
Quot. 1		X		X		X		X
Quot. 2		X		X		X		X
Quot. 3		1900		X		X		X
Quot. 4			7.	2000		X		X
Quot. 5				1556	6.	2056		X
Quot. 6						1682		X
Quot. 7								X
Quot. 8								X
Quot. 9							5.	2088
Quot. 10								1868

There are, in this example, 10 seats that need to be allocated to the provincial level. First, the parties get quotients crossed out that corresponds to the number of constituency seats received in the province. Then the quotients are compared. The first two seats go to Metropolitan Copenhagen, which is then put to rest from the computations, since it has all its seats allocated. After 7 seats are allocated, Northern and Central Jytland get put to rest as well for the same reasons. Seat number 10 gets allocated to party C although the quotients of party B and D are higher. But these parties have already been put to rest, since they have received all their seats.

E.5. Allocate Additional Seats on the constituency level

The computations for parties on the provincial level look like this. Note that the quotients use the Danish divisor method (1, 4, 7, etc.):

	Sealand		Funen		Southern Jytland		Total
	Seat no		Seat no		Seat no		
Party D							
Constituency seats		3		2		3	10
Total votes		14200		10000		12300	36500
Quotient 1		X		X		X	
Quotient 3		X		X		X	
Quotient 5		X	1.	1429		X	
Quotient 7	2.	1420		1000		1230	
Quotient 9		1092					

There are, in this example, 2 additional seats that need to be allocated to the constituencies; total number of seats is equal to 10 and the three constituencies have in total 8 constituency seats. First, all the constituencies get quotients crossed out that corresponds to the number of constituency seats received in the constituencies. Then the remaining quotients are compared. The first seat goes to Funen, since 1429 is greater than both 1420 and 1230. A new quotient is created 10000/10 = 1000 to Funen. The second seat goes to Sealand, since 1420 is greater than 1000 and 1230.

E.6 Select Candidates

The selection of candidates depends on the list organization that the party uses in a constituency. If the party uses standing-by-district, then the party nominates a candidate in every district. This means that all the party votes in districts where the candidate is nominated are added to the personal votes that the candidate receives. If the party uses standing-in-parallel, then all the party votes in a district are distributed to the candidates in the same ratio that the candidate receives personal votes.

Below is an example of standing-by-district usage. Note that the * marks the nominated candidate in the district:

Party B	Fjordby	Sundby	Aaby	Færgeby	Broby	Personal votes	Party votes	Total
Andersen	180	150	60	120	250*	760	1600	2360
Hansen	20	60*	20	20	10	130	1700	1830
Jensen	200	300	550*	300	360	1710	1650	3360
Nielsen	650*	75	50	110	120	1005	2400	**3405**
Pedersen	50	15	20	50*	10	145	1400	1545
Pers. votes in constituency	1100	600	700	600	750	3750		
Party votes	2400	1700	1650	1400	1600		8750	
Total constituency votes	3500	2300	2350	2000	2350			12500

Nielsen is elected with 3405 votes.

Below is an example for when standing-in-parallel is used.

	Fjordby	Sundby	Aaby	Færgeb	Broby	Personal votes	Party votes	Total
Andersen	393	425	141	280	534	760	1773	2533
Hansen	44	170	47	47	21	130	329	459
Jensen	436	850	1297	700	768	1710	4051	**5761**
Nielsen	1418	213	118	257	256	1005	2262	3267
Pedersen	109	42	47	116	21	145	335	480
Pers. votes in constituency	1100	600	700	600	750	3750		
Party votes	2400	1700	1650	1400	1600		8750	
Total constituency votes	3500	2300	2350	2000	2350			12500

All candidates receive the share of the party votes that their personal votes ratio in each district provides. Jensen gets elected.

Appendix F – Recommendations and Comments to Danish Electoral Law

Analysis of the Danish electoral law revealed several unnecessary and missing items. This appendix contains a list of recommended modifications and comments to the law.

F.1 Recommendations

- Article §77 does not deal with the case that no parties pass the threshold. The implication of no passing parties is that the Folketing will only have 139 members. That number is not in conflict with the Constitutional act stating that the Folketing should at most have 179 members[64], but it is in conflict with the electoral law that states there are 175 members elected in Denmark[65].
  It is hereby recommended that article §77 should deal with it.
- Article §77 stk. 5 is redundant. The article states that if some parties after the recalculation have more seats than before, then the results must be recalculated. It has been shown that this can never occur since any recalculation will at most give a party the number of seats as in the first calculation plus one (cf. subsection 4.2.2 on p. 74).
  It is hereby recommended that the stk. 5 of article §77 be removed.
- Article §78 mentions nothing of the possibilities for ties between parties and/or provinces. Article §79, however, mentions this possibility together with the possibility of ties between provinces and constituencies, and that lots should be drawn in case of ties. It is shown (cf. 4.2.2 on p. 79) that an irresolvable situation may arise when not all additional seats get allocated to the parties on the provincial level.
  It is hereby recommended that article §78 explicitly state that lots should be drawn to resolve ties between involved parts. It is also recommended that § 78 deals with the situation described above.

F.2. Comments

- One way to resolve the situation when no parties pass the threshold is to state in the electoral law that there are at most 175 seats representing Denmark. Alternatively a second threshold should be defined and used when no parties are passing the threshold.
- A pragmatic approach to resolve the situation with regards to article §78 is to enable the calculation to allocate the seats to the parties in their respective provinces. This would be a violation on how the seats are allocated to the electoral map, which uses principles from the Constitutional Act. It would, however, make the situation resolvable.

[64] [CA] § 28
[65] [DEL] § 7

Appendix G – Using DiVS

At the time of this writing (April 2011) DiVS is just the model of a larger system, or, one may say, an API. If the Ministry of the Interior at some point chooses to use DiVS instead of the current system, there is need for a graphical user interface (GUI). This appendix explains the usage of DiVS to the GUI developer.

It is assumed that the Eclipse IDE is used, and that the reader is familiar with that environment. The reader is encouraged to view figure 4 'Model of DiVS' before reading this appendix.

G.1 Controller

This section deals with the controller of DiVS.

G.1.1 Add Necessary Libraries

After creation of a new Java project two external jars must be added to the project: derby.jar and divs.jar. The former will enable a connection to the database. The latter will enable using DiVS.

G.1.2 Create Database

No database is shipped with the jar file, and since DiVS doesn't create it automatically, it must be created manually. Follow the instructions in subsection 3.4.4. Remember to add the properties file.

C.1.3 Implementation

Create top package 'controller' and inside it the Controller class. The responsibility of the Controller class is to serve as the controller of the model, DiVS. The Controller class instantiates the Election class. The Controller class should be singleton.

G.2 View

G.2.1 GUI

In the same project, at the top level, create a package 'view' with GUI classes inside. The GUI classes must be able to use the Controller class aforementioned.

Note that the semantics of the `resolve` methods is that the parameters sometimes win and sometimes lose the draw.

G.2.2 Web Application

Make sure that the web application is able to use the Controller class.