Digital Voter Registration System

Third Semester Project, BDSA, Bachelor in Software Development
IT University of Copenhagen, Fall 2011

Lecturer: Joseph Kiniry

Dirty Lion

Christian Olsson (chro@itu.dk), Kåre Sylow Pedersen (ksyl@itu.dk)
& Henrik Haugbølle (hhau@itu.dk)

14th December 2011

Contents

Abstract

Registration of voters at a Danish election is normally done by pen and paper. A voter is registered when the ballot is handed out so a voter cannot vote twice. A digital voter registration system enables an election to be executed with more efficiency than with the original procedure.

Originally each polling table has a list of voters permitted to get a ballot from that table. By digitization of the lists a voter can attend any polling table and the look-up is performed instantly. Increased registration efficiency will require less staff and less waiting time for each citizen.

Our solution is separated into three main parts. The first part being the centrally managed generation of polling cards and voter registration lists both in digital and printable format, the second part being the registration of attending voters at each polling table at a polling venue, and the third part being the local handling of data registered at each polling venue.

Requirements

Mandatory Requirements

- Generate voter cards in printable format
- Generate registration lists for specific venues and polling tables in printable format
- Locally validate and register voter attendance digitally without Internet connectivity
- Validate a citizen by voting card and/or by social security number (CPR-no)
- Guarantee that exactly those in the data set provided can vote and no one can vote twice
- Include a graphical user interface for both the central management and local clients
- Facilitate server and multiple clients running on different computers at the same venue

Optional Requirements

- Generate bar-codes on voter cards for more efficient look-ups
- Facilitate local server redundancy

System Overview

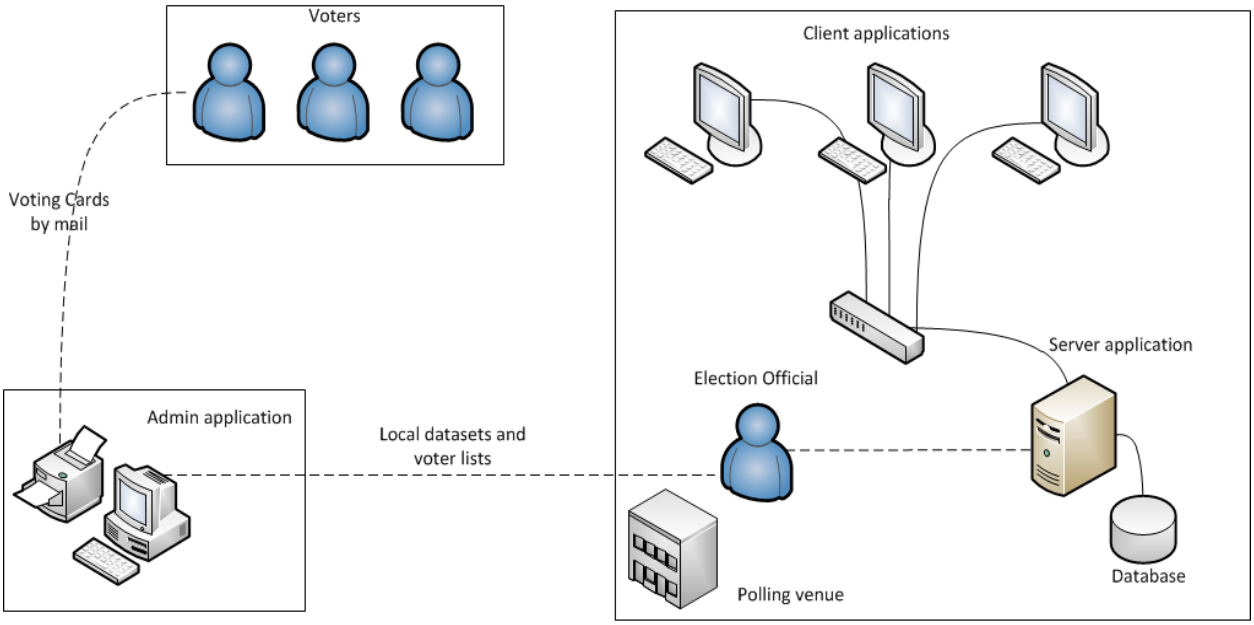


Fig 1. Illustrating the high level system architecture.

Early in our system analysis we discovered, that our solution could be organised into three main parts. A central management application, a client applications running at each polling table and a server application at each polling venue supporting the various number of clients.

The central application transforms a given data set into polling cards mailed to the voters. It also generates the polling venue data and voter lists used on each polling venue. The data to the polling venue is transported manually e.g. via an USB hard-drive.

The client applications are operated by the official election staff. The staff can look up a voter, see information about him and register that a ballot is handed out.

The server contains information about the voters associated with the polling venue and whether the voter has voted or not. The server can serve several clients simultaneous with no risk of double registration and thereby avoiding hand out of a ballot to the same voter twice.

The computers used at a polling venue are on a closed wired network with no Internet connection. In case of a system error or crash the staff can switch to the manual voter lists and continue to register voters.

Admin Application

The application is a stand alone program, that generates polling cards, voter lists and local data sets to the polling venues. The program loads all the voters supplied as an XML file by an external supplier and adds a unique voter id, before it divides them into polling cards, voter lists and data set for their respective polling venue. The polling cards and voter lists are exported as PDF files for easy printing. The local data set is exported as a CSV file which the server application can load.

Server Application

A server is placed at each polling venue running the server application. The server application relies on the Database and Network libraries for interacting with the SQL database and for communicating with the client applications at the polling venue. On start-up the server application is capable of removing data from the database and importing the data set meant for the specific polling venue. When the server is running it listens for request from one or more client applications on the local network, but it handles them sequential to avoid interference. Only one server can be present on a network. It is important to notice that the server ensures that a voter cannot be registered twice in the system.

Client Application

The application is used by the election officials to look-up and register voters. A voter can be looked-up by the barcode on their voting card or by their CPR number. The client application retrieves information and registers handing out of ballots to the local server. A local log is updated to give the possibility of undoing a previous action. The application warns the election official if a voter has been registered before. This could be due to a person trying to vote twice, or a client computer crashing at the worst possible time, which causes a retry at another client to fail. This is dealt with by showing the election official where and when the person was registered. The election official will be presented with the table name and the time at which the voter was registered and in case it was the place and time at which a client computer failed the election official can rightfully hand out the ballot.

Components

Database Library

The database library consists of two parts. The first being a connector for the SQL database and the second being a query builder. The connector acts as a driver/interface for connecting, disconnecting and executing queries towards the specific database type (in our case a MySQL database). The query builder is an utility on which you can specify a range of parameters to generate a query without writing any actual SQL yourself.

These two components decouples the rest of the application from the external data source, making it fairly easy to switch between various types of external data sources. If we were to switch to another kind of SQL database, we could simply correct the connector and query builder to fit the new requirements of the database. Because of the decoupling, the rest of the application would need no correction.

For system analysis and system design see attached disc “Documentation\BON\Database.bon” or Appendix B: Database.bon.

Entities Library

Generally the entities part of the project contains entity components reflecting real-world objects such as persons, polling venues and logs. Going a bit deeper one will find that it contains two kinds of components. The first kind of entity component has the ability to interact with the database; fetching, manipulating and deleting data, and is used to manipulate the database. The second kind of entity component does not have these abilities and is somewhat a “shallow” object. This latter kind of component is used in the parts of the application where no database connectivity is required or allowed.

For system analysis and system design see attached disc “Documentation\BON\Entities.bon” or Appendix B: Entities.bon.

I/O Library

The I/O library manages two tasks, importing and exporting data.

- The import task is to load raw data into the admin application. The raw data is provided by external suppliers as an XML file, with information about the voters and polling venues. The XML file is validated against an XML schema to prevent the program from crashing. If the XML file pass the validation, the information will be passed into polling venues and persons that the rest of the libraries can use. The XML schema is located in a separate XML file.
- The export part handles output of three types of data; polling cards, voter lists and voter data. All the generated files, are created in a folder with the same name as the polling venue. The folder is created as a subfolder in the decided path.

  The polling cards and voter lists are generated as PDF files and the voter data is a CSV file. The library takes a polling venue as input, and parses the voters into the polling tables they have been assigned. Each voter list is saved in separate PDF files for easy distribution on the polling venue. The CSV file and the polling cards file is saved as two large files containing all the voters.

For system analysis and system design see attached disc “Documentation\BON\IO.bon” or Appendix B: IO.bon.

Network Library

Consists of three layers of network communication:

- The bottommost layer is UDP multicast, without delivery guaranty. Two different ports are used, so only the server receives requests from clients and clients from the server. This is to enable testing a client instance and a server instance on the same computer.
- The middle layer is a request-reply protocol. The server blocks and listens for calls from clients. The client makes a request to the server and then blocks and awaits a reply. Missing reply from the server will make the client resend the request. A request with the same identifier will result in the server repeating the same reply. A timeout value can be set on both send and receive.

- The top layer is remote procedure call (RPC). Some specially defined methods for our digital voting system can be called by the client and be listened to by the server.

For system analysis and system design see attached disc “Documentation\BON\Network.bon” or Appendix B: Network.bon.

PdfGenerator Library

The library is divided into two parts, the polling card and the voter list. Each of them can generate a PDF document with various numbers of pages depending on the data set.

- The polling card is developed to look like a rough copy of the polling card used for danish national election. The polling card contains information about the election, polling venue, who it is from, and the voter it self. Polling cards can be added to the document as needed, and they are all saved to the same PDF file.
- The voter list is a simple graphical table where voters can be added and saved as a single PDF file. The list contains information about what table it is associated with.

Appendix E: Polling card
For system analysis and system design see attached
disc “Documentation\BON\PdfGenerator.bon” or Appendix B: PdfGenerator.bon.

Server

The server application described earlier relies on a server component utilizing several of the other component libraries, including the database, entity and network components.

For system analysis and system design see attached disc “Documentation\BON\Server.bon” or Appendix B: Server.bon.

Process Overview

We have mostly been working together in the same room throughout the project. Nonetheless we have begun everyday with a meeting about where we were and where we were heading. We have discussed all major design aspects and have worked as a whole. Due to this we have not had any special or fixed roles in the group.

Because the design and the interfaces was clear to everybody we were able to split the project into three main parts, which we have been working on quite independently. The three main parts were separated as listed below (the primary responsible person is stated in parenthesis):

- **Admin part** (Kåre Sylow Pedersen)
  Generation of polling cards and voter lists in PDF format, importing and exporting data sets and the admin application.

- **Server part** (Henrik Haugbølle)
  Database communication, entity objects, importing data to the database, generation of test data, and the server application.

- **Client part** (Christian Olsson)
  Network communication including UDP multicast, request/reply, RPC, and the client application.

We have implemented, documented and tested each part independently according to the design decisions made as a group.

Validating the System

To validate the correctness of our system we have written a series of unit test suites as listed below. These test suites cover some of the critical parts of the system such as the database connectivity, the correctness of response from the server, the internal workings of the network part and the PDF generators interaction with the surrounding file system.

- *Database*
  - ConnectorTestSuite (tests database connection)
  - QueryBuiderTestSuite (tests query building)
- *Entities*
  - EntityTestSuite (tests the correctness of entity interaction)
- *Generator*
  - PdfGeneratorTest (tests interaction with the computers file system)
  - VoterIdTest (tests uniqueness of generated voter id's)
- *IO*
  - IOTest (tests correct creation of files in the computers file system)
- *Network*
  - RPCTestSuite (tests internal workings of our RPC structure)

It is very important to notice that the database and entity test suites requires an instance of a MySQL database running on the computer when testing (see additional documentation in the comments of the ConnectorTestSuite) and that the IO test suite requires specific XML documents placed in the right directory to run. Both of these external dependencies is included in the project which should make the tests fairly easy to run.

After running PEX on various classes of our system we have chosen not to generate any PEX test suites because of the high degree of network communication and external data source/ external file system interaction in our system.

Listed below is our code coverage of the key components of the system as calculated by the DotCover Visual Studio plug-in:

Overall code coverage of our SmallTuba library 97% distributed as follows:

- Database 99%
- Entities 100%
- PDFGenerator 100%
- IO 97%
- Network 94%

Code coverage does of course only tell us how much of the code CAN be run without failures and not that the code can not fail. Failures may exist but is not triggered by our tests. Because correct execution is very important to us, we have tried to write as many code contracts as possible, so the code will fail in case it does not perform as expected or is used in a way that could put it in an illegal state. This is also a nice way of avoiding unnecessary long defensive programming statements.

Besides writing unit tests and contracts the system has been manually tested and debugged on multiple computers on a local wired and a wireless network.

Fulfilling the Requirements

We believe that we have accomplished to fulfill our mandatory requirements as well as partly fulfilling our optional requirements.

- We are able to generate printable polling cards as well as voter lists for specific polling venues and polling tables.
- We can validate and register attending voters locally at a polling venue while ensuring that no one can be registered twice in the system.
- The client, as well as the the admin application, can be operated using a graphical user interface.
- The server application can communicate with several clients running on different computers at a local network.

In addition to meeting the mandatory requirements as summarized above, we have managed to create scan-able unique barcodes on each polling card for more efficient look-up at the polling tables.

We have not had the resources to implement local server redundancy. We have been looking into this but since this is a voting system and we do not tolerate failures, we choose to stick with a version that is safe. We have managed to create installers for the client, server and admin application making them easier to distribute and setup. This could have been an optional requirement, as it is a very nice feature for the end-user to have, especially taking the installation of a MySQL database into account, which is included and automatically handled in our server installer.

Dictionary

Admin Application
The program that loads all the voters and generates voter lists, polling cards and voter data to the polling venues.

Ballot
A list of candidates the voter can vote for. The voter marks which candidate to vote for and submits it.

Client Application
The program where the voter can be looked up and registered on the polling venue.

Dirty Lion
The name of our group.

Election
Name of the election e.g. national election.

Election Officials
The personnel responsible for the local execution of the election.

Log
Information about the time and which client who handed out the ballot.

Person
A collection of personal information for a person like address, CPR number, which polling venue and municipality the person is associated with.

Polling Card
The letter send to the voter with information about the election type and where to vote etc. The polling card contains a unique voter id represented as a barcode that can be scanned by the election officials.

Polling Table
A table at the polling venue is operated by election officials. Any table can be chosen by the voter except in the case where the digital voter system is down in which case the table described on the ballot must be used. Each table has an individual voter list.

Polling Venue
The location where the voter can vote.

Server Application
The program running on a server on each polling venue serving the clients with information about the voters.

Small Tuba
The code name of the entire project and also the project name of the main class library.

Voter
A person with the right to vote.

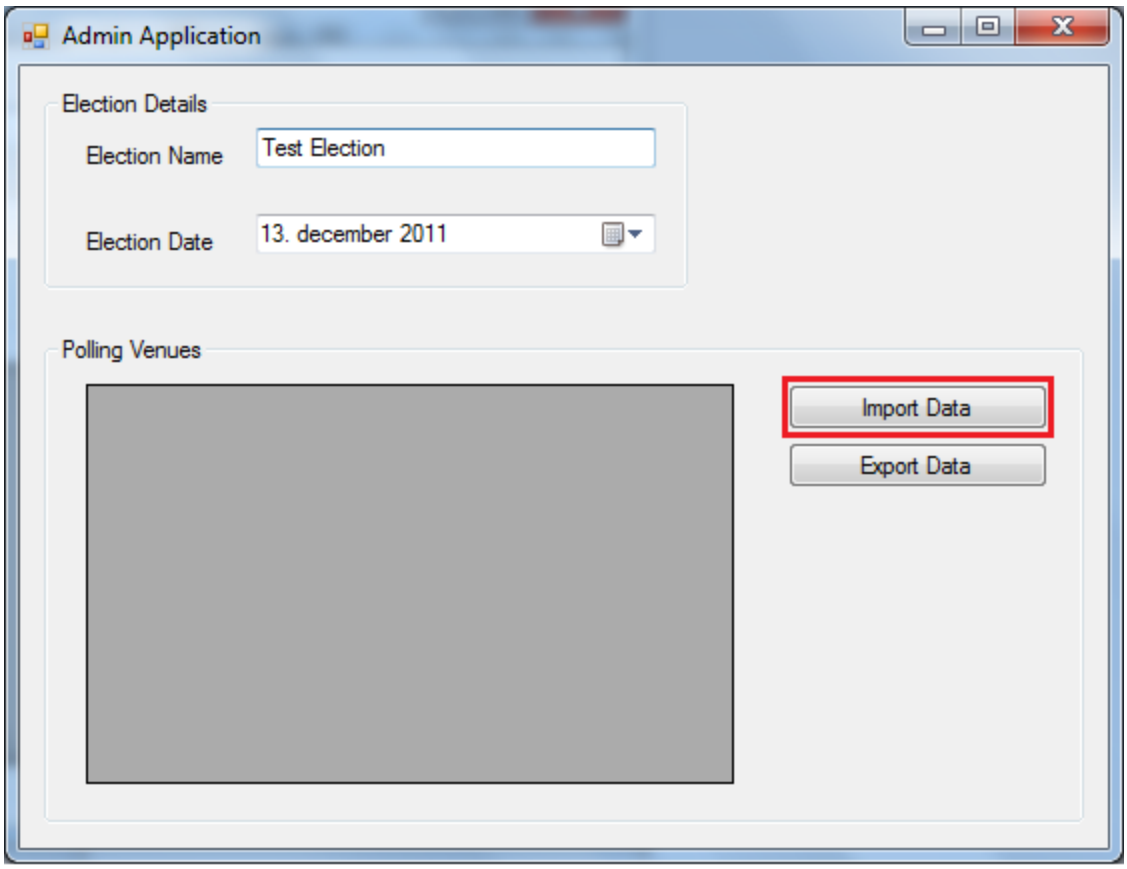
Voter id
An unique id number printed on each polling card.

Voter list
Each polling table at the polling venue has a voter list with all the voters information for that table if the digital system is down. The voter can be looked up and validated from the voter list.

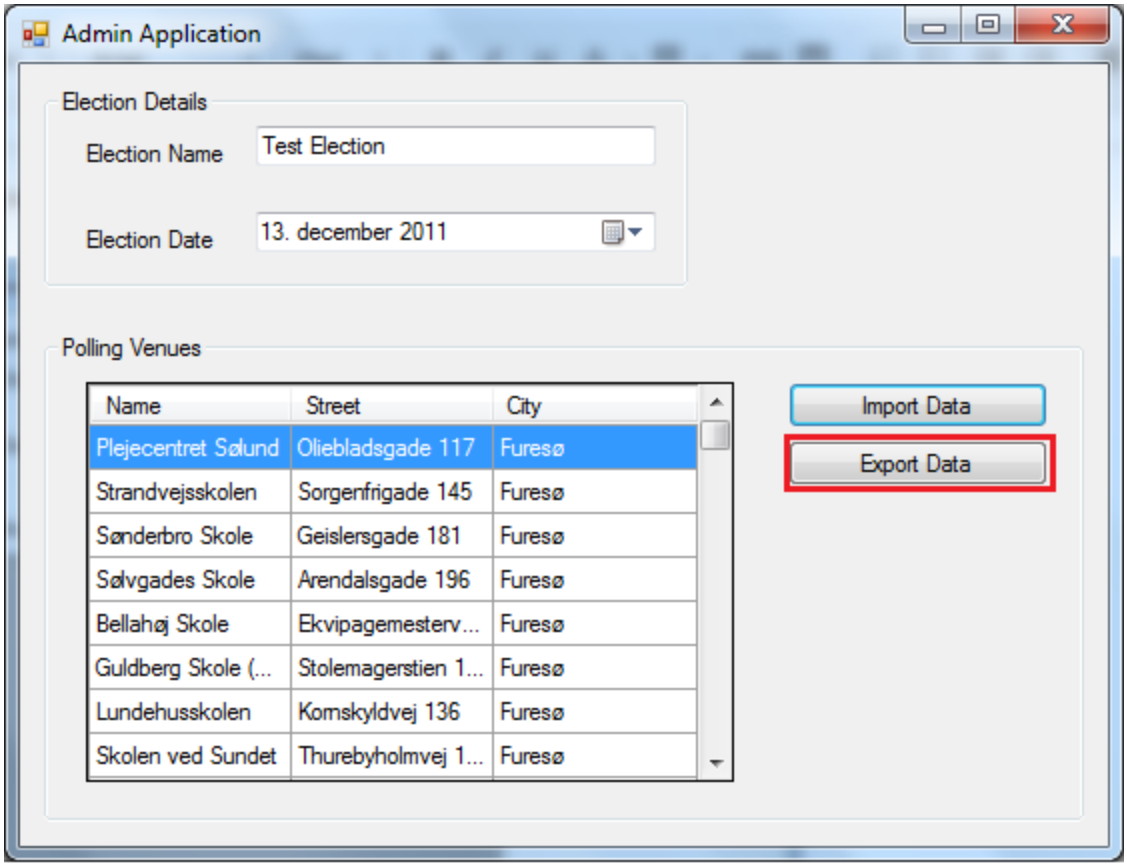
Manuals and Examples

Admin Application Manual

The admin application is launched via the AdminApplication.exe file made by the installer. At the top of the window is a text box and a date picker, where the main election information like name and date can be added. These information will be printed on all the polling cards and voter lists. In the middle of the window is a overview table and two buttons. To load data into the program choose the import button and navigate to the XML file with the file dialog that pops up.



If the file is in the correct format, all the polling venues will be listed in the overview, otherwise an error message will occur with a description of the problem. To export data for the selected polling venue click on the export data button.



Use the check boxes to decide what kind of data you want to export.

Click on the export data button, to choose a folder where the generated files will be saved in. The program will automatically create a new subfolder, with the name of the polling venue inside the folder you selected and put the files here.

When the files is saved, the export window will disappear and another polling venue can be selected for export.

Server Application Manual

Starting the Server

To start the server one will need to execute the batch file StartServerApplicationAndMySQL.bat. This can either be done by accessing the file through the directory the server application was installed in or by clicking the shortcut icon on the desktop created by the installer.

By running the StartServerApplicationAndMySQL.bat, both the server and an instance of a MySQL server will start as indicated by the name. This will result in two command prompts.

It is possible to start the server application and the MySQL server instance independently. To start up the server application independently run the ServerApplication.exe file. To Start up the MySQL server independently run the mysql.bat file.

Make sure that you do not have any other instances of MySQL servers running on the computer when executing the start-up file. If no instance or multiple instances of MySQL is running the server application will crash.

It is also possible to setup a MySQL server yourself. But in that case you need to make sure that it is running on the correct port and contains the correct data structure and a user with the correct privileges (see Appendix A: “Manual MySQL Installation Guide” for more information on manually running a MySQL server along with the server application).

Configuring and Running the Server

Once the server is running the user will be prompted with the setup message as shown below. The user can choose to clear the log or person data contained in the database or import a new data set of persons by pointing to a CSV file generated by the admin application. When the user is done setting up the server, it can be started with the simple command “start”.

```
Digital Voter Registration System
Server v1.8

Setup server:

Commands: import <<filetoimport.csv>>, clear <<person|log>>, start, cancel
```

And example or configuration is shown below (typed commands is underlined)

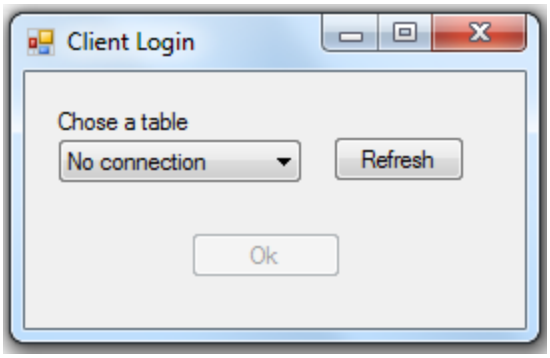
```
clear person
clearing of database succeeded
Commands: import <<filetoimport.csv>>, clear <<person|log>>, start, cancel
clear log
clearing of database succeeded
Commands: import <<filetoimport.csv>>, clear <<person|log>>, start, cancel
import C:/Users/SomeUser/Voters.csv
Import From: C/Users/SomeUser/Voters.csv
Imported: 1323627889
...
Imported: 1232627844
importing to database succeeded
Commands: import <<filetoimport.csv>>, clear <<person|log>>, start, cancel
start
Server is starting
Server is running
```

Closing the Server
Because of the architecture of the server, it is not possible to type a command in the prompt to shut it down. This means that the sever must be shut down by clicking the close-button in the upper right of the window or by killing the process named “ServerApplication.vshost.exe” using the Windows Task Manager.

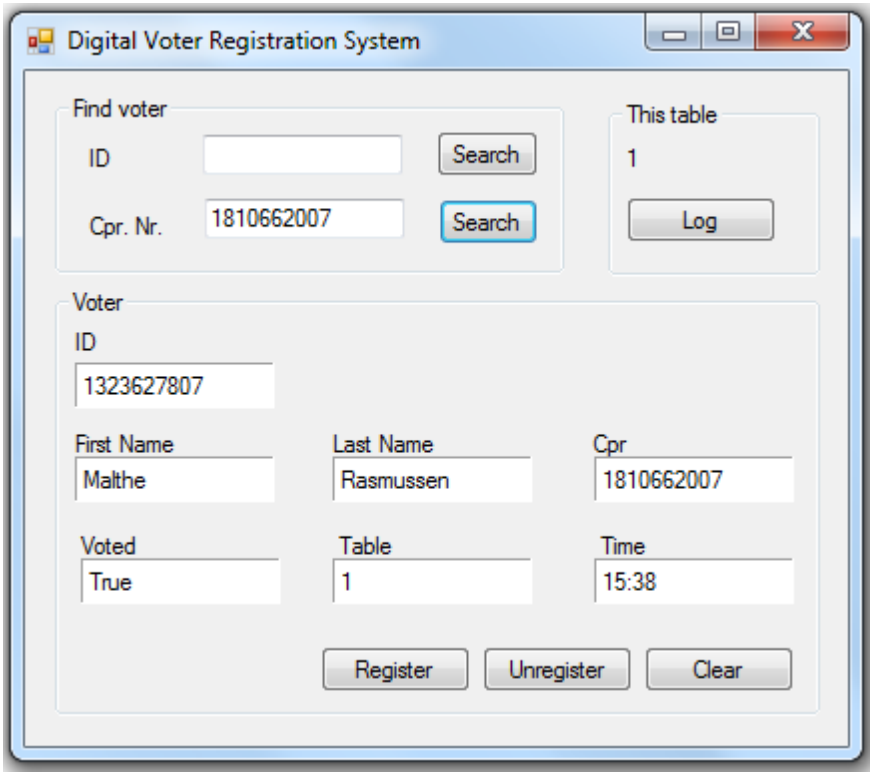
The MySQL server must by shut down by typing “close” in the MySQL server prompt. If the prompt is closed by pressing the close-button in the upper right corner, the MySQL server instance will still be running. If this happens, please kill the process “mysqld.exe” using the Windows Task Manager.

Client Application Manual

To start the client one will need to execute the ClientApplication.exe file made by the installer. This can either be done by accessing the file through the directory it was installed in or by clicking the shortcut icon on the desktop created by the installer. This will start the client application.



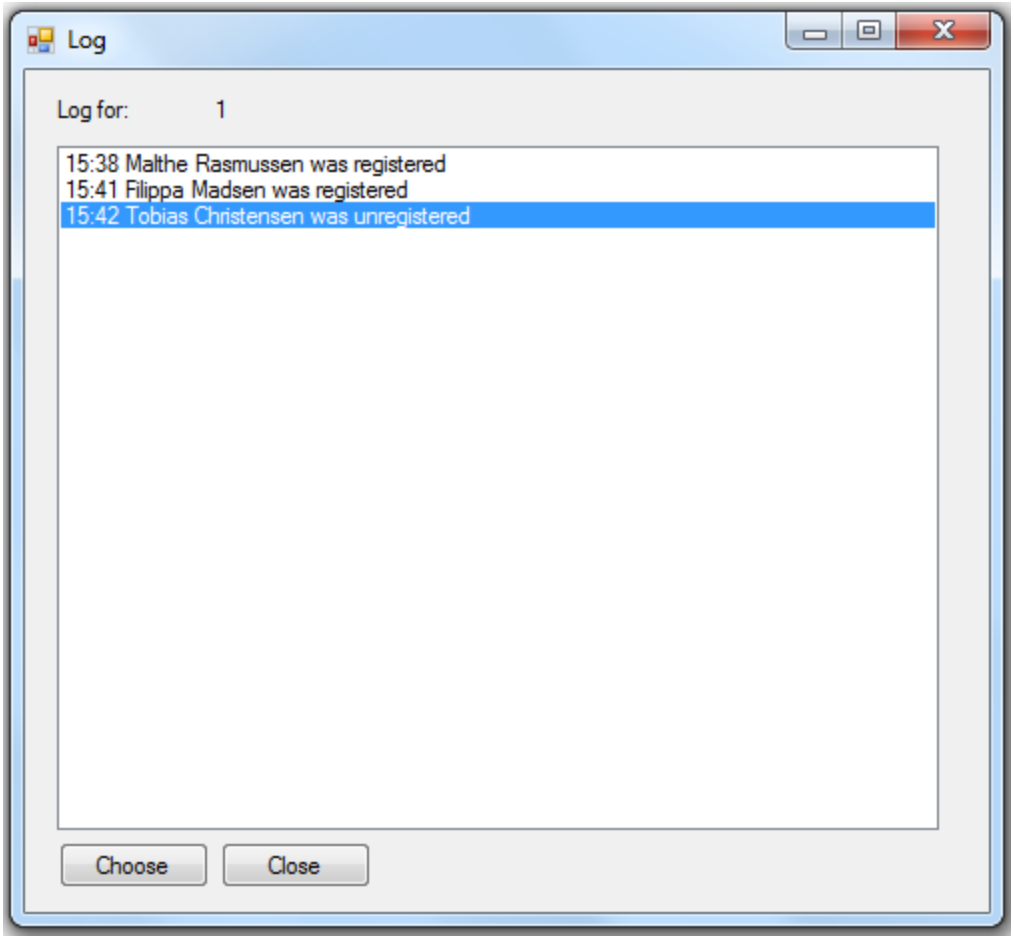
The client application consists of three windows. When the application is started, the user can choose a name for the client from a list of valid table names requested from the server. If there are no names to choose from and the only item in the drop-down menu is “No connection” then there is no server available on the network. Pressing the “Refresh” button will make the program try to connect to the server again.



After selecting a name, the user will be presented with the main window. In this window the user can look up voters from a CPR number or barcode id and register them. All information

about a voter will be displayed in the bottom part of the window. If a voter is selected the buttons “Register”, “Unregister” and “Clear” will be enabled. Register will try to register the voter at the server, unregister will unregister and clear will reset the window.

The user can open a log window from the main window, enabling an overview of the voters registered and unregistered at this table. The log and the unregister functions serve as an undo feature, if an official accidentally makes a mistake.



Bibliography

Distributed Systems: Concepts and Design, 5th edition - George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair, 2011.

Seamless Object-Oriented Software Architecture: Analysis and design of Reliable Systems - Kim Waldén, Jean-Marc Nerson, 1994.

Visual C#, Visual Studio 2010, http://msdn.microsoft.com/library/kx37x36 - Microsoft Developer Network.

C# Station, tutorials, http://www.csharp-station.com - Joe Mayo.

C# Corner, tutorials, http://www.c-sharpcorner.com - various authors.

Stack Overflow, FAQ community, http://www.stackoverflow.com - various authors.

NUnit Framework Documentation, http://www.nunit.org/index.php?p=documentation - unknown author.

Repository

Our repository is public available at GitHub. You will need to install the Git application to fork or clone the repository. Our commit history is also public available and reachable through the URL address listed below.

Public repository
https://github.com/dirtylion/smalltuba

Public commit history
https://github.com/dirtylion/smalltuba/commits/master

Appendix A: Install Guides

Admin Application Installation Guide

Requirements
Microsoft Windows 7 x86/x64 Edition
Preferable at least 2 GB RAM
Preferable at least 2.0 GHz Processor
40 MB storage

Description
The AdminSetup will install the administration application on your computer. Launch the provided CD and navigate to its contents using Windows Explorer, then follow the steps described below.

Step by Step

1. Double click the AdminSetup.msi to begin the installation.
2. When the setup start-up is shown press “Next”.
3. Choose which directory to install the application in and press “Next”.
   Note: The installer will choose the directory C:\Dirty Lion\Admin as default. If you choose the C:\Program Files\ folder as installation directory you may encounter some permission problems witch will result in application error. Therefore we recommend installation of the application in the default directory chosen by the installer.
4. To start the installation press “Next”. The application will now be installed.
5. If a pop-up window occurs prompting you to accept that the application will make changes on the computer, please press “Yes” to proceed with the installation.
6. When installation is completed press “Close” to exit the installer.
7. The admin application is now installed on your computer and you can access it either through the AdminApplication.exe file in the directory where you installed or through the shortcut placed on your desktop by the installer.

Further information
Please see the Admin Application Manual section of the Overview document for further information on use of the application.

Client Application Installation Guide

Requirements
Microsoft Windows 7 x86/x64 Edition
Preferable at least 2 GB RAM
Preferable at least 2.0 GHz Processor
40 MB storage

Description
The ClientSetup will install the client application on your computer. Launch the provided CD and navigate to its contents using Windows Explorer, then follow the steps described below.

Step by Step

1. Double click the ClientSetup.msi to begin the installation.
2. When the setup start-up is shown press “Next”.
3. Choose which directory to install the application in and press “Next”.
   Note: The installer will choose the directory C:\Dirty Lion\Client as default. If you choose the C:\Program Files\ folder as installation directory you may encounter some permission problems witch will result in application error. Therefore we recommend installation of the application in the default directory chosen by the installer.
4. To start the installation press “Next”. The application will now be installed.
5. If a pop-up window occurs prompting you to accept that the application will make changes on the computer, please press “Yes” to proceed with the installation.
6. When installation is completed press “Close” to exit the installer.
7. The admin application is now installed on your computer and you can access it either through the ClientApplication.exe file in the directory where you installed or through the shortcut placed on your desktop by the installer.

Further information
Please see the Client Application Manual section of the Overview document for further information on use of the application.

Server Application Installation Guide

Requirements
Microsoft Windows 7 x86/x64 Edition
Preferable at least 2 GB RAM
Preferable at least 2.0 GHz Processor
150 MB storage

Description
The ServerSetup will install the server application and create a copy of a MySQL installation on your computer. Launch the provided CD and navigate to its contents using Windows Explorer, then follow the steps described below.

Step by Step

1. Double click the ServerSetup.msi to begin the installation.
2. When the setup start-up is shown press “Next”.
3. Choose which directory to install the application in and press “Next”.
   Note: The installer will choose the directory C:\Dirty Lion\Server as default. If you choose the C:\Program Files\ folder as installation directory you may encounter some permission problems witch will result in application error. Therefore we recommend installation of the application in the default directory chosen by the installer.
4. To start the installation press “Next”. The application will now be installed.
5. If a pop-up window occurs prompting you to accept that the application will make changes on the computer, please press “Yes” to proceed with the installation.
6. When installation is completed press “Close” to exit the installer.
7. The admin application is now installed on your computer and you can access it either through the ServerApplication.exe file in the directory where you installed or through the shortcut placed on your desktop by the installer.

Further information
Please see the Server Application Manual section of the Overview document for further information on use of the application.

Manual MySQL Installation Guide

Description
The ServerSetup will setup MySQL automatically on your computer, but in case you experience trouble you can use the guide as described below.

The first part of the guide will instruct you on how to install a working instance of MySQL, the second part will guide you through the correct configuration of data on the MySQL server.

Installing MySQL
WAMP is a popular easy-to-install server environment including MySQL and provides an easy-to-use interface called phpMyAdmin for interacting with the MySQL server. Download the latest version from the URL below and run the installation.

http://www.wampserver.com/en/

You can also choose to install MySQL by directly downloading the latest version from the official MySQL web page, but this does not provide any user interface and you must therefore interact with the database using a command prompt. For this solution please download the latest version from the URL below and run the installation.

http://dev.mysql.com/downloads/

Configuration the MySQL server
To use the server application, the MySQL server needs to have a specific data structure and a specific user with the correct privileges.

If you have succeeded in installation MySQL or already have an instance running on the computer where you want the server application to run, you can easily create the data structure and user structure by executing the two SQL files listed below (these files are provided with the distribution in the subfolder called “MySQL”).

“mysql dump - structure and data.sql” (database, table and column structure)
“mysql dump - user.sql” (user structure and privileges)

You can either copy and paste the contents of the files or you can import them using the import functionality provided by MySQL.1

Important
The MySQL server ***must*** run at port 3306 for the server application to work.

Appendix B: BON

Database.bon

```
system_chart DATABASE_SYSTEM
 indexing
   author: "Henrik Haugbølle <hhau@itu.dk>";
   course: "BDSA";
   university: "ITU";
   created: "2011-11-30";
 explanation
   "Classes for connecting and querying a database."

   cluster DATABASE description "Classes for connecting and quering a database."
end

cluster_chart DATABASE
 indexing
   author: "Henrik Haugbølle <hhau@itu.dk>";
   created: "2011-11-30";
   keywords: "database, query builder, connection, sql";
 explanation
   "Classes ued for establishing database connection, executing queries and process result sets."

   class CONNECTOR description "Used for establishing a connection and executing queries against the database."
   class QUERY_BUILDER description "Able to take various parameters and to assemble a valid sql query from these."
end

class_chart CONNECTOR
 indexing
   author: "Henrik Haugbølle <hhau@itu.dk>";
   created: "2011-12-12";
   keywords: "database, connection, sql";
 explanation
   "Object used to establish and terminate database connection and execute queries by it. Only one object should be in play."
 query
   "Get the connector?",
   "What is the result of the execution of this fetching query?",
   "What is the result of the execution of this non-fetching query?",
   "Is the connection established?",
   "What is the number of rows fetched from last query?",
   "What was the total number of rows possible to fetch without limits and offset from last query?"
 command
   "Connect to the database!",
   "Disconnect from the database!"
end

class_chart QUERY_BUILDER
 indexing
   author: "Henrik Haugbølle <hhau@itu.dk>";
   created: "2011-11-30";
   keywords: "query builder, sql";
 explanation
   "Able of assembly given a range of various and optional parameters."
 query
   "What is the last executed query?",
   "Get the number of rows in the last resultset?",
   "Get the number of total rows found in the last query?",
   "What is the result of the assembling the query from the given parameters?",
   "What is the result of the assembling and execution of this fetching query?",
   "What is the last affected id of assembling and execution of this none fetching query?"
 command
   "Set the type of the query to be this!",
   "Use this table in the query!",
   "Use these columns in the query!",
   "Use these values in the query!",
```

```
  "Add this condition to the query!",
  "Add this condition with this following logical operator to the query!",
  "Add this condition with this following logical operator and this identifier index!",
  "Remove the condition with this given index!",
  "Set the limit of the result to this!",
  "Set the offset of the result!",
  "Set the group by of the query!",
  "Add this as the order of the result!",
  "Add this as the order of the result with this direction!",
  "Add this as the order of the result with this direction and this identifier index!",
  "Remove the order with this given index!"
end

static_diagram DATABASE
component
 cluster DATABASE
 component
  class CONNECTOR
   feature
    GetConnector : THIS
    Connect : VOID
    Disconnect : VOID
     require
      IsConnected = TRUE
     end
    IsConnected : BOOLEAN
    ExecuteQuery : ARRAY[VALUE] -> a_query : STRING
     require
      IsConnected = TRUE
     end
    ExecuteNoneQuery -> a_query : STRING
     require
      IsConnected = TRUE
     end
    GetCount : INTEGER
    GetCountTotal : INTEGER
     require
      IsConnected = TRUE
     end
  end

  class QUERY_BUILDER
   feature
    GetLastQuery : STRING
    GetCount : INTEGER
    GetCountTotal : INTEGER
    Assemble : STRING
    ExecuteQuery : ARRAY[ARRAY[VALUE]]
    ExecuteNoneQuery : INTEGER
    SetType : THIS -> type : STRING
     require
      type /= VOID and
      (type = "select" or type = "update" or type = "insert" or type = "delete" or type = "truncate")
     end
    SetTable : THIS -> table : STRING
     require
      type /= VOID
     end
    SetColumns : THIS -> columns : ARRAY[STRING]
     require
      columns /= VOID and columns.Size > 0
     end
    SetValues : THIS -> values : ARRAY[STRING]
     require
      values /= VOID and values.Size > 0
     end
    AddCondition : THIS
     -> condition : STRING
     require
      condition /= VOID
```

```
      end
    AddConditionWithBind : VALUE
      -> condition : STRING
      -> bind : STRING
      require
        condition /= VOID and bind /= VOID and
        (bind = "or" or bind = "and")
      end
    AddConditionWithBindWithIndex : THIS
      -> condition : STRING
      -> bind : STRING
      -> index : STRING
      require
        condition /= VOID and bind /= VOID and index /= VOID and
        (bind = "or" or bind = "and")
      end
    RemoveCondition : THIS -> index : STRING
      require
        index /= VOID
      end
    SetLimit : THIS -> limit : INTEGER
      require
        limit > 0
      end
    SetOffset : THIS -> offset : INTEGER
      require
        offset > -1
      end
    SetGroupBy : THIS -> groupby : STRING
      require
        groupby /= VOID
      end
    AddOrder : THIS
      -> order : STRING
      require
        order /= VOID
      end
    AddOrderWithDirection : THIS
      -> order : STRING
      -> direction : STRING
      require
        order /= VOID and direction /= VOID and
        (direction = "asc" or direction = "desc")
      end
    AddOrderWithDirectionWithIndex : THIS
      -> order : STRING
      -> direction : STRING
      -> index : STRING
      require
        order /= VOID and direction /= VOID and index /= VOID and
        (direction = "asc" or direction = "desc")
      end
    RemoveOrder : THIS -> index : STRING
      require
        index /= VOID
      end
  end
 end
end
```

Entities.bon

```
system_chart ENTITY_SYSTEM
 indexing
   author: "Henrik Haugbølle <hhau@itu.dk>";
   course: "BDSA";
   university: "ITU";
   created: "2011-11-30";
 explanation
   "Classes used to represent various logically encapsulated data in object form."

   cluster ENTITIES description "Classes used to represent various logically encapsulated data in object form."
end

cluster_chart ENTITIES
 indexing
   author: "Henrik Haugbølle <hhau@itu.dk>";
   created: "2011-11-30";
   keywords: "entity, value object, data access object";
 explanation
   "Entity classes reflects virtual and real-world objects."

   class ADDRESS description "A shallow object containing information about an address."

   class POLLING_VENUE description "A shallow object containing information about an polling venue."

   class PERSON description "A shallow value holder object of the person entity used where no external data connectivity is
required."
   class PERSON_ENTITY description "A person entity reflecting a person in real life with associated relevant information."
   class PERSON_VALUE_OBJECT description "The value object of the person entity"
   class PERSON_DATA_ACCESS_OBJECT description "The data access object of the person entity"
   class PERSON_RESOURCE description "The resource of the person entity, which is used to fetch one or more persons from the
external data source."

   class LOG description "A shallow value holder object of the log entity used where no external data connectivity is required."
   class LOG_ENTITY description "A log entity containing information on client events and time of execution."
   class LOG_VALUE_OBJECT description "The value object of the log entity"
   class LOG_DATA_ACCESS_OBJECT description "The data access object of the log entity"
   class LOG_RESOURCE description "The resource of the log entity, which is used to fetch one or more logs from the external data
source."

   cluster ABSTRACTS description "Abstracts for the entities."
end

cluster_chart ABSTRACTS
 indexing
   author: "Henrik Haugbølle <hhau@itu.dk>";
   created: "2011-11-30";
   keywords: "entity, value object, data access object, abstracts";
 explanation
   "Abstracts for entities, value objects, data access objects and resources."

   class ABSTRACT_ENTITY description "The abstract entity implements standard load, save and delete methods for all entities."
   class ABSTRACT_VALUE_OBJECT description "The abstract value object implements methods to set and get values from a
value object."
   class ABSTRACT_DATA_ACCESS_OBJECT description "Abstracts the standard load, save and delete methods for data access
objects."
   class ABSTRACT_RESOURCE description "Makes the standard methods for a resource available, including an abstract build
method for entity generation."
end

-- ABSTRACTS

class_chart ABSTRACT_ENTITY
 indexing
   author: "Henrik Haugbølle <hhau@itu.dk>";
   created: "2011-11-30";
   keywords: "entity, data object, virtual and real-world encapsulation.";
```

```
  in_cluster: "ABSTRACTS";
 explanation
  "An entity is a object encapsulating and making data within a specific logically scope manipulable."
 query
  "May I have an existing entity loaded with these parameters?",
  "May I have the id of this entity?",
  "Does this entity exists?"
 command
  "Save this entity!",
  "Delete this entity!"
end

class_chart ABSTRACT_VALUE_OBJECT
 indexing
  author: "Henrik Haugbølle <hhau@itu.dk>";
  created: "2011-11-30";
  keywords: "entity, value object";
  in_cluster: "ABSTRACTS";
 explanation
  "The entity value object is the holder for all of the entities values fetch from an external data source."
 query
  "May I have a new empty value object?",
  "May I have the value?",
  "May I have the values?"
 command
  "Set value!",
  "Set values!"
end

class_chart ABSTRACT_DATA_ACCESS_OBJECT
 indexing
  author: "Henrik Haugbølle <hhau@itu.dk>";
  created: "2011-11-30";
  keywords: "entity, data access object, query builder";
  in_cluster: "ABSTRACTS";
 explanation
  "The entity data access object handles the communication with the external data source through an entity's resource."
 query
  "May I have a new data access object?",
  "May I have an existing entity data set loaded with these parameters?"
 command
  "Save the associated entity given these data!",
  "Delete the associated entity given this id!"
end

class_chart ABSTRACT_RESOURCE
 indexing
  author: "Henrik Haugbølle <hhau@itu.dk>";
  created: "2011-11-30";
  keywords: "entity, resource, query builder";
  in_cluster: "ABSTRACTS";
 explanation
  "The abstract resource makes available the mmost often used methods of a resource and is able to build entities from data sets
fetched by the query builder."
 query
  "May I have a new resource?",
  "May I have the the real number of entities fetched?",
  "May I have the total number of entities which where possible to fetch without the offset and limit parameters?"
 command
  "Set the sorting order with this sorting direction!",
  "Set the limit of entities fetched!",
  "Set the offset of entities fetched!",
  "Set the group by of entities fetched",
  "Fetch and build the entities from the external data source by the current parameters set!",
  "Build the entities accordion to the parameters given!"
end

-- ENTITIES

class_chart ADDRESS
```

```
 indexing
   author: "Henrik Haugbølle <hhau@itu.dk>";
   created: "2011-12-12";
   keywords: "address";
   in_cluster: "ENTITIES";
 explanation
   "Represents an address."
 query
   "May I have a new address?",
   "May I have the name of the address?",
   "May I have the street of the address?",
   "May I have the city of the address?"
 command
   "Set the name to this!",
   "Set the street to this!",
   "Set the city to this!"
end

class_chart POLLING_VENUE
 indexing
   author: "Henrik Haugbølle <hhau@itu.dk>";
   created: "2011-12-12";
   keywords: "polling venue, voters, addresses";
   in_cluster: "ENTITIES";
 explanation
   "Represents a polling venue"
 query
   "May I have a new polling venue?",
   "May I have the persons who can vote here?",
   "May I have the address of the polling venue?",
   "May I have the address of the municipality the polling venue is in?"
 command
   "Set the persons who can vote here!",
   "Set the address of the polling venue?",
   "Set the address of the municipality the polling venue is in?"
end

class_chart PERSON
 indexing
   author: "Henrik Haugbølle <hhau@itu.dk>, Christian Olsson <chro@itu.dk>, Kåre Sylow Pedersen <ksyl@itu.dk>";
   created: "2011-12-12";
   keywords: "person, voter, shallow, value holder";
   in_cluster: "ENTITIES";
 explanation
   "Represents a real person, but with no external data connectivity."
 query
   "May I have a new person?",
   "May I have the Database id?",
   "May I have the Firstname?",
   "May I have the Lastname?",
   "May I have the Street?",
   "May I have the City?",
   "May I have the Cpr?",
   "May I have the Voter id?",
   "Which polling venue should the person vote at?",
   "Which polling table should the person vote at?",
   "Has the person voted?",
   "At which time did the person vote?",
   "At which polling table did the person vote?",
   "Does the person exists in the external data source?",
   "Is this person equal to this other person?",
   "Is this person equal to this object?",
   "What is the hash code of the person?",
   "How is the person represented as a string?",
   "How should I sort this person, given other persons, by name?"
 command
   "Set database id!",
   "Set firstname!",
   "Set lastname!",
   "Set street!",
```

```
  "Set city!",
  "Set cpr!",
  "Set voter id!",
  "Set at which polling venue the person should vote!",
  "Set at which polling table the person should vote!",
  "Set whether the person has voted!",
  "Set the time of which the person voted!",
  "Set which table the person voted at!",
  "Set whether the person exists in the data source!"
end

class_chart PERSON_ENTITY
 indexing
  author: "Henrik Haugbølle <hhau@itu.dk>";
  created: "2011-11-30";
  keywords: "person, voter";
  in_cluster: "ENTITIES";
 explanation
  "Represents a real person with associated CPR number, firstname, lastname, gender etc."
 inherit
  ABSTRACT_ENTITY
 query
  "May I have a new person?",
  "May I have a new person given these values?",
  "May I have the Firstame?",
  "May I have the Lastname?",
  "May I have the Cpr?",
  "May I have the Voter id?",
  "Which polling station should you vote at?",
  "Which polling table should you vote at?",
  "Have you voted?",
  "At what time did you vote?",
  "At what polling table did you vote?",
  "May I have all logs?",
  "May I have the most recent log?",
  "May I have a representation of this as a PERSON object?"
 command
  "Set firstname!",
  "Set lastname!",
  "Set cpr!",
  "Set voter id",
  "Set at which polling venue the person should vote!",
  "Set at which polling table the person should vote!"
end

class_chart PERSON_VALUE_OBJECT
 indexing
  author: "Henrik Haugbølle <hhau@itu.dk>";
  created: "2011-11-30";
  keywords: "person, voter, value object";
  in_cluster: "ENTITIES";
 explanation
  "The person value object inheriting from the abstract value object"
 inherit
  ABSTRACT_VALUE_OBJECT
 query
  "May I have a new person value object?"
end

class_chart PERSON_DATA_ACCESS_OBJECT
 indexing
  author: "Henrik Haugbølle <hhau@itu.dk>";
  created: "2011-11-30";
  keywords: "person, voter, data access object";
  in_cluster: "ENTITIES";
 explanation
  "The person data access object inheriting from the abstract data access object"
 inherit
  ABSTRACT_DATA_ACCESS_OBJECT
 query
```

```
   "May I have a new person data access object?"
end

class_chart PERSON_RESOURCE
 indexing
   author: "Henrik Haugbølle <hhau@itu.dk>";
   created: "2011-11-30";
   keywords: "person, voter, resource object";
   in_cluster: "ENTITIES";
 explanation
   "The person resource inheriting from the abstract resource"
 inherit
   ABSTRACT_RESOURCE
 query
   "May I have the persons given the parameters set?"
 command
   "The firstname must be like this value!",
   "The lastname must be like this value!",
   "The gender must be this value!",
   "The CPR number must be this value!",
   "The polling station must be this value!",
   "The polling table must be this value!",
   "The state of whether the person has voted must be this value!",
   "Set whether the barcode value must have been generated!"
end

class_chart LOG_ENTITY
 indexing
   author: "Henrik Haugbølle <hhau@itu.dk>";
   created: "2011-11-30";
   keywords: "log, client";
   in_cluster: "ENTITIES";
 explanation
   "Used to log the actions of a client computer."
 inherit
   ABSTRACT_ENTITY
 query
   "May I have the person id of which the log belongs to?",
   "Which command was executed?",
   "Which client executed the command?",
   "At which polling table was the command executed?",
   "At what time was the command executed?"
 command
   "Set the person id!",
   "Set the command!",
   "Set the executing client!",
   "Set the polling table at which the command was executed!",
   "Set the time of when the command was executed!"
 end

class_chart LOG_VALUE_OBJECT
 indexing
   author: "Henrik Haugbølle <hhau@itu.dk>";
   created: "2011-11-30";
   keywords: "log, client, value object";
   in_cluster: "ENTITIES";
 explanation
   "The log value object inheriting from the abstract value object"
 inherit
   ABSTRACT_VALUE_OBJECT
 query
   "Which command was executed?",
   "Which client executed the command?",
   "At what time was the command executed?"
end

class_chart LOG_DATA_ACCESS_OBJECT
 indexing
   author: "Henrik Haugbølle <hhau@itu.dk>";
   created: "2011-11-30";
```

```
   keywords: "log, client, data access object";
   in_cluster: "ENTITIES";
 explanation
   "The log data access object inheriting from the abstract data access object"
 inherit
   ABSTRACT_DATA_ACCESS_OBJECT
end

class_chart LOG_RESOURCE
 indexing
   author: "Henrik Haugbølle <hhau@itu.dk>";
   created: "2011-11-30";
   keywords: "log, client, resource object";
   in_cluster: "ENTITIES";
 explanation
   "The log resource inheriting from the abstract resource"
 inherit
   ABSTRACT_RESOURCE
 command
   "The timeinterval must be between these two timestamps!",
   "The client computer must be identified by this value!",
   "The action must be this value!"
end

-- BON, BON, FORMAL BON!

static_diagram ENTITY_CLASSES
component
 cluster ENTITIES
 component
   class ADDRESS
     feature
     Make : THIS
     GetName : STRING
     GetStreet : STRING
     GetCity : STRING
     SetName -> name : STRING
     SetStreet -> street : STRING
     SetCity -> city : STRING
   end

   class POLLING_VENUE
     feature
     Make : THIS
     GetPersons : ARRAY[PERSON]
     GetPollingVenueAddress : ADDRESS
     GetMunicipalityAddress : ADDRESS
     SetPersons -> persons : ARRAY[PERSON]
     SetPollingVenueAddress -> address : ADDRESS
     SetMunicipalityAddress -> address : ADDRESS
   end

   class PERSON
     feature
       Make: THIS
       GetDbId : INTEGER
       GetFirstName : STRING
       GetLastName : STRING
       GetStreet : STRING
       GetCity : STRING
       GetCpr : STRING
       GetVoterId : INTEGER
       GetPollingVenue : STRING
       GetPollingTable : STRING
       GetVoted : BOOLEAN
       GetVotedTime : INTEGER
       GetVotedPollingTable : STRING
       GetExists : BOOLEAN
       EqualsToPerson : BOOLEAN -> person : PERSON
       EqualsToObject : BOOLEAN -> a_object : OBJECT
```

```
    GetHashCode : INTEGER
    ToString : STRING
    GetNameComparator : COMPARATOR
    SetDbId -> db_id : INTEGER
    SetFirstName -> first_name : STRING
    SetLastName -> last_name : STRING
    SetStreet -> street : STRING
    SetCity -> city : STRING
    SetCpr -> cpr : STRING
    SetVoterId -> voter_id : INTEGER
    SetPollingVenue -> polling_venue : STRING
    SetPollingTable -> polling_table : STRING
    SetVoted -> voted : BOOLEAN
    SetVotedTime -> voted_time : INTEGER
    SetVotedPollingTable -> polling_table : STRING
    SetExists -> does_exists : BOOLEAN
end

class PERSON_ENTITY
  inherit ABSTRACT_ENTITY
  feature
  Make : THIS
  MakeWithValues : THIS -> values : ARRAY[VALUE] -- maybe ARRAY[STRING]
    require
      values /= VOID
    end
  GetFirstname : STRING
  GetLastname : STRING
  GetGender : INTEGER
  GetCpr : INTEGER
  GetBarcodeValue : NUMBER
  GetPollingStation : STRING
  GetPollingTable : STRING
  HasVoted : BOOLEAN
  GetVotedTime : INTEGER
  GetVotedPollingTable : STRING
  GetLogs : ARRAY[LOG]
  GetMostRecentLog : LOG
  GetPersonObject : PERSON

  SetFirstname -> firstname : STRING
  SetLastname -> lastname : STRING
  SetCpr -> cpr : STRING
  SetVoterId -> voter_id : INTEGER
  SetPollingVenue -> polling_venue : STRING
  SetPollingTable -> polling_table : STRING
end

class PERSON_VALUE_OBJECT
  inherit ABSTRACT_VALUE_OBJECT
end

class PERSON_DATA_ACCESS_OBJECT
  inherit ABSTRACT_DATA_ACCESS_OBJECT
end

class PERSON_RESOURCE
  inherit ABSTRACT_RESOURCE
  feature
    SetFirstname : THIS -> firstname : STRING
      require
        firstname /= VOID
      end
    SetLastname : THIS -> lastname : STRING
      require
        lastname /= VOID
      end
    SetCpr : THIS -> cpr : STRING
      require
        cpr /= VOID
```

```
      end
    SetPollingVenue : THIS -> polling_venue : STRING
      require
        polling_venue /= VOID
      end
    SetPollingTable : THIS -> polling_table : STRING
      require
        polling_table /= VOID
      end
    SetVoterId : THIS -> voter_id : INTEGER
      require
        voter_id > 0
      end
    Build : ARRAY[PERSON_ENTITY]
end

class LOG
  inherit ABSTRACT_ENTITY
  feature
    GetPersonId : INTEGER
    GetAction : STRING
    GetClient : STRING
    GetPollingTable : STRING
    GetTimestamp : INTEGER
    SetPersonId -> person_id : INTEGER
    SetAction : a_action -> STRING
    SetClient : a_client -> STRING
    SetPollingTable -> polling_table : STRING
    SetTimestamp -> timestamp : INTEGER
end

class LOG_VALUE_OBJECT
  inherit ABSTRACT_VALUE_OBJECT
end

class LOG_DATA_ACCESS_OBJECT
  inherit ABSTRACT_DATA_ACCESS_OBJECT
end

class LOG_RESOURCE
  inherit ABSTRACT_RESOURCE
  feature
    SetPerson : THIS -> person : PERSON_ENTITY
      require
        person /= VOID and person.Exists
      end
    Build : ARRAY[VALUE]
end

cluster ABSTRACTS
component
  deferred class ABSTRACT_ENTITY
    feature
      Load : THIS -> parameters : ARRAY[VALUE] -- maybe ARRAY[STRING]
        require
          parameters /= VOID and parameters.Size > 0
        end
      GetId : INTEGER
      Exists : BOOLEAN
      Save : VOID
      Delete : VOID
        require
          Current.Exists = TRUE
        end
  end

  deferred class ABSTRACT_VALUE_OBJECT
    feature
      Make: THIS
      GetValue : VALUE -> key : STRING
```

```
      GetValues : ARRAY[VALUE]
      SetValue -> key : STRING -> value : VALUE
      SetValues -> values : ARRAY[VALUE]
        require
          values /= VOID
        end
    end

    deferred class ABSTRACT_DATA_ACCESS_OBJECT
      feature
      Make : THIS
      Load : THIS -> parameters : ARRAY[VALUE] -- maybe ARRAY[STRING]
        require
          parameters /= VOID and parameters.Size > 0
        end
      Save : VOID -> values : ARRAY[VALUE] -- maybe ARRAY[STRING]
        require
          values /= VOID and values.Size > 0
        end
      Delete : VOID -> id : INTEGER
        require
          id > 0
        end
    end

    deferred class ABSTRACT_RESOURCE
      feature
      Make : THIS
      GetCount : INTEGER
      GetCountTotal : INTEGER
      SetOrder
        -> order : STRING
        -> direction : STRING
        require
          order /= VOID and direction /= VOID -- and
           -- (direction = "asc" or direction = "desc") -- BON compiler fails here, works perfectly fine in database.bon though (!?)
        end
      SetLimit -> limit : INTEGER
        require
          limit > 0
        end
      SetOffset -> offset : INTEGER
        require
          limit >= 0
        end
      SetGroupBy -> groupby : STRING
        require
          groupby /= VOID
        end
      deferred Build : ARRAY[OBJECT]
    end
  end
 end
end
```

I/O.bon

```
system_chart VOTER_MATERIAL_IO
 indexing
   author: "Kåre Sylow Pedersen <ksyl@itu.dk>";
   course: "BDSA";
   university: "ITU";
   created: "2011-11-30";
 explanation
   "Input/Output for the election data."

 cluster IO description "Imports and exports raw voter data, polling cards, voter lists and polling venue data."
end

cluster_chart IO
 indexing
   author: "Kåre Sylow Pedersen <ksyl@itu.dk>";
   created: "2011-11-30";
 explanation
   "Imports and exports raw voter data, polling cards, voter lists and polling venue data."

 class FILE_LOADER description "Loads a xml file, and parse the content to polling venues."
 class FILE_SAVER description "Saves polling cards, voter lists and local polling venue data."
end

class_chart FILE_LOADER
 indexing
   author: "Kåre Sylow Pedersen <ksyl@itu.dk>";
   created: "2011-11-30";
 explanation
   "Loads a xml file, and parse the content to polling venues."
 query
   "Can you load this xml file, and parse it to a list of polling venues?"
 constraint
   "The path must not be 'nothing'.",
   "The file must exists."
end

class_chart FILE_SAVER
 indexing
   author: "Kåre Sylow Pedersen <ksyl@itu.dk>";
   created: "2011-11-30";
 explanation
   "Saves polling cards, voter lists and local polling venue data."
 query
   "May I have a new file saver for this polling venue and this path?",
   "Can you devide these voters into voter lists, and save them to this location on the harddrive?",
   "Can you save the polling cards for this polling venue on the harddrive?",
   "Can you save these voters to a csv file seperated with a ';'?"
 constraint
   "The path must not be 'nothing'",
   "The polling venue name must not be 'nothing'",
   "The file must exists on the harddrive after it has been saved",
   "The name of the polling venue must be more than 0 characters"
end

static_diagram VOTER_MATERIAL_IO
component
 cluster IO
 component
   class FILE_LOADER
     feature
       GetPollingVenues : LIST[POLLINCARDS] -> path : STRING -> notifier : VALIDATIONEVENTHANDLER
         require  path /= void and notifier /= void and "file.exist(path)" and "file extension = .xml"
        end
   end

   class FILE_SAVER
```

```
    feature
      make: FILE_SAVER -> path : String -> pollingVenueName : STRING
        require path /= void and pollingVenueName /= void and pollingVenueNane.lenght > 0
        ensure "DIRECTORY.exists(path + pollingVenueName)"
        end
      SaveVoterList -> persons : LIST[PERSONS] -> electionName : STRING ->electionDate : STRING
        require persons /= void
        ensure "DIRECTORY.GetFiles(path).Length > 0"
        end
      SavePollingCards -> pollingVenue : POLLINGVENUE -> electionName : STRING ->electionDate : STRING
        require pollingVenue /= void
        ensure "FILE.exist(path)"
        end
      SaveVoters -> pollingVenue : POLLINGVENUE
        require pollingVenue /= void
        ensure "FILE.exist(path)"
        end
  end
 end
end
```

Network.bon

```
system_chart NETWORK_COMMUNICATION
 indexing
   author: "Christian Olsson <chro@itu.dk>";
   course: "BDSA";
   university: "ITU";
   created: "2011-11-30";
 explanation
   "This system handles the network communication in our polling system.\
   \On the top level voter request are handled, and underneath network\
   \communication and checks are made to ensure that the UDP multicast\
   \packets are delivered succefully"
 cluster NETWORK description "The network cluster"
end

cluster_chart NETWORK
 indexing
   author: "Christian Olsson <chro@itu.dk>";
   created: "2011-11-30";
 explanation "The network communication in our system"
 cluster RPC description "Cluster used for remote procedure calls from a voter client to a voter server"
 cluster REQUEST_REPLY description "Cluster used for request-reply with delivery garanty"
 cluster UDP description "Cluster for low level udp multicast"
end

cluster_chart RPC
 indexing
   author: "Christian Olsson <chro@itu.dk>";
   created: "2011-11-30";
 explanation "Cluster used for remote procedure calls from a voter client to a voter server"
 class VOTER_CLIENT description "The client side"
 class VOTER_SERVER description "The server side"
end

cluster_chart REQUEST_REPLY
 indexing
   author: "Christian Olsson <chro@itu.dk>";
   created: "2011-11-30";
 explanation "Cluster used for request-reply with delivery garanty"
 class CLIENT_FRONT_END description "The client front end"
 class SERVER_FRONT_END description "The server front end"
end

cluster_chart UDP
 indexing
   author: "Christian Olsson <chro@itu.dk>";
   created: "2011-11-30";
 explanation "Cluster for low level udp multicast"
 class UDP_MULTICAST description "Class used for multicasting data"
end

class_chart VOTER_CLIENT
 indexing
   author: "Christian Olsson <chro@itu.dk>";
   created: "2011-11-30";
 explanation "The client side of the network communication in our voting system.\
       \This is the top most level and communication is based on procedure calls."
 query
   "May I have a new client for the voter network with this name?",
   "What is the name of this client?",
   "May I have all information about the person with this cpr. nr.?",
   "May I have all information about the person with this ID?",
   "What are the valid tables for this server?",
   "Are you connected to a server?"
 command
   "Make this the new name of this client.",
   "Register that this voter has voted.",
```

```
   "Unregister that this voter has voted."
 constraint
   "Two clients must not have the same name.",
   "If a reply has not been received within 2 seconds, the method must return.",
   "If no reply is received, nothing will be returned.",
   "A request for registering/unregistering a voter, must contain a voter.",
   "The name must have a value.",
   "The name must not be SERVER."
end

class_chart VOTER_SERVER
 indexing
   author: "Christian Olsson <chro@itu.dk>";
   created: "2011-11-30";
 explanation "The server side of the network communication in our voting system.\
     \This is the top most level and communication is based on procedure calls."
 query
   "May I have a new server for the voter network with this name?",
   "Which function will be invoked when asked about a person from a cpr.nr.?",
   "Which function will be invoked when asked about a person from a barcode ID?",
   "Which function will be invoked  when asked about registering a user?",
   "Which function will be invoked  when asked about unregistering a user?",
   "Which function will be invoked  when asked about valid tables?"
 command
   "Invoke this function that returns a person when asked about a person from a cpr.nr.",
   "Invoke this function that returns a person when asked about a person from a barcode ID.",
   "Invoke this function when asked about registering a user.",
   "Invoke this function when asked about unregistering a user.",
   "Invoke this function when asked about valid tables.",
   "Listen for calls for this amount of time."
 constraint
   "Before calling listen, all handlers for replies must be set.",
   "If the time specified is 0, the server must wait forever.",
   "The timout value must not be negative.",
   "The name must have a value.",
   "The server must reply to ping requests from the client.",
   "Two servers must not have the same name."
end

class_chart CLIENT_FRONT_END
 indexing
   author: "Christian Olsson <chro@itu.dk>";
   created: "2011-11-30";
 explanation "This class keeps resending packets for the server if its unresponsive.\
      \This class only sends to and receives packages from the server not other clients."
 query
   "May I have a new client front end with this name?",
   "What is the result of the this request, with this timeout?",
 constraint
   "If a server is unresponsive the request must be repeated.",
   "If the time specified is 0, the server must wait forever.",
   "The timeout value must be not be negative.",
   "The request cannot be empty.",
   "Two clients must not have the same name.",
   "The name must have a value.",
   "The name must not be SERVER."
end

class_chart SERVER_FRONT_END
 indexing
   author: "Christian Olsson <chro@itu.dk>";
   created: "2011-11-30";
 explanation "This class listens for request for the server and replies.\
      \This class only receives request adressed to the server."
 query
   "May I have a new server front end?",
   "which function is invoked when a call is received?"
 command
   "Invoke this function when a call is received.",
   "Listen for calls for this amount of time."
```

```
 constraint
   "If the timeout is set to 0, then it will wait infinit.",
   "The timeout value must not be negative.",
   "Before calling listen, the request handler must be set.",
   "A function must consume a message and produce a message.",
   "If a request from a client is repeated with the same request ID, the response must be repeated.",
   "To servers must not exist in the same network."
end

class_chart UDP_MULTICAST
 indexing
   author: "Christian Olsson <chro@itu.dk>";
   created: "2011-11-30";
 explanation "Class used for multicasting data"
 query
   "May i have a new multicast client with this choice of server or client setting?",
   "May I have an object if one is in queue or arrives within this timeframe?"
 command
   "Send this object."
 constraint
   "If the timeout is set to 0, then it will wait infinit.",
   "Only two types of udp multicast clients can be made each listening and sending on different ports.",
   "The client must only listen to ip 224.5.6.7 port 5000.",
   "The client must only send to ip 224.5.6.7 port 5001.",
   "The server must only listen to ip 224.5.6.7 port 5001.",
   "The server must only send to ip 224.5.6.7 port 5000.",
   "An instance can only listen to either the server socket or the client socket.",
   "Objects may not arrive to all subscribers of this udp channel.",
   "A connection is needed to the recipients.",
   "Packets maybe lost if the buffers gets to full.",
   "If an incomming data is not a serialized object, the packet must be ignored.",
   "It is not possible to send nothing.",
   "If nothing is recieved within the timeout, nothing must be returned."
end

static_diagram CLASS_INTERFACES
component
 cluster NETWORK
 component
   cluster RPC
   component
     class VOTER_CLIENT
       feature
         make : VOTER_CLIENT -> name : STRING
         getPersonFromCPR : PERSON -> cpr : NATURAL
           -- ensure
             -- connected => result /= void and
             -- not connected => result = void
           -- end
         getPersonFromID : PERSON -> id : NATURAL
           -- ensure
             -- connected => result /= void and
             -- not connected => result = void
           -- end
         registerVoter : BOOLEAN -> person : PERSON
           require person /= void
           -- ensure not connected => result = false
           end
         unRegisterVoter : BOOLEAN -> person : PERSON
           require person /= void
           --- ensure not connected => result = false
           end
         validTables : ARRAY[STRING]
           -- ensure
             -- connected => result.length > 0 and
             -- not conneted => result = void
           -- end
         connected : BOOLEAN
         invariant
           name /= void and
```

```
        name /= "server" and
        name.length > 0
      end
    end
    class VOTER_NETWORK_SERVER
      feature
        make : VOTER_NETWORK_SERVER -> name : STRING
          require name /= void
        end
        CprToPersonRequest : FUNCTION[NATURAL, PERSON]
        IdToPersonRequest : FUNCTION[NATURAL, PERSON]
        RegisterVoteRequest : FUNCTION[PERSON, BOOL]
        UnRegisterVoteRequest : FUNCTION[PERSON, BOOL]
        ValidTableRequest : FUNCTION[ARRAY[STRING]]
        ListenForCalls -> timeOut : INTEGER
          require
            timeOut >= 0 and
            CprToPersonRequest /= void and
            IdToPersonRequest /= void and
            RegisterVoteRequest /= void and
            UnRegisterVoteRequest /= void and
            ValidTableRequest /= void and
            ListenForCalls /= void
          end
    end
  end
  cluster REQUEST_REPLY
  component
    class CLIENT_FRONT_END
      feature
        make : CLIENT_FRONT_END -> name : STRING
          require name /= void and name /= "server"
        end
        sendQuery : VALUE -> message : VALUE -> timeOut : INTEGER
          require
            message /= void and
            timeOut >= 0
          end
    end
    class SERVER_FRONT_END
      feature
        make : SERVER_FRONT_END
        RequestHandler : FUNCTION[VALUE, VALUE]
        listenForCalls -> timeOut : INTEGER
          require
            timeOut >= 0 and
            RequestHandler /= void
          end
    end
  end
  cluster UDP
  component
    class UDP_MULTICAST
      feature
        make : UDP_MULTICAST -> server : INTEGER
          require server = 0 or server = 1
        end
        send -> packet : VALUE
          require packet /= void
        end
        receive : VALUE -> timeOut : INTEGER
          require timeOut >= 0
        end
    end
  end
end
```

PdfGenerator.bon

```
system_chart VOTER_MATERIAL_GENERATOR
 indexing
   author: "Kåre Sylow Pedersen <ksyl@itu.dk>";
   course: "BDSA";
   university: "ITU";
   created: "2011-11-30";
 explanation
   "Generates polling cards and voterlist for a election"

 cluster GENERATOR description "Generates polling cards and voter lists with a unique id."
end

cluster_chart GENERATOR
 indexing
   author: "Kåre Sylow Pedersen <ksyl@itu.dk>";
   created: "2011-11-30";
 explanation
   "Generates polling cards and voter lists with a unique id."

 class POLLING_CARDS description "Generated pdf polling cards and save them to the harddrive."
 class VOTERLIST description "Generates voterlists and save them to the harddrive."
 class VOTER_ID_GENERATOR description "Generates a unique id number."
end

class_chart POLLING_CARDS
 indexing
   author: "Kåre Sylow Pedersen <ksyl@itu.dk>";
   created: "2011-11-30";
 explanation
   "Create and append polling cards to a pdf file and save it to the harddrive"
 query
   "May I have a new polling cards generator for this election?",
   "Can you save all the polling card on this location on the harddrive?"
 command
   "Create a polling card for this person!"
 constraint
   "The person must not be 'nothing'.",
   "The file is located on the harddrive when it is saved."
end

class_chart VOTERLIST
 indexing
   author: "Kåre Sylow Pedersen <ksyl@itu.dk>";
   created: "2011-11-30";
 explanation
   "Create voterlist to the polling stations used for backup, if the digital solution fails."
 query
   "May I have a new voting list for this election?",
   "Can you save the voting list to this location on the harddrive?"
 command
   "Add this person to the voting list."
 constraint
   "The person must not be 'nothing'",
   "The number of rows on the list have to be bigger than 20.",
   "The file is located on the harddrive when it is saved."
end

class_chart VOTER_ID_GENERATOR
 indexing
   author: "Kåre Sylow Pedersen <ksyl@itu.dk>";
   created: "2011-11-30";
 explanation
   "Generates a unique id number."
 command
   "Generate a new voter id!"
 constraint
```

```
  "A new voter id must be bigger than the previous."
end

static_diagram VOTER_MATERIAL_GENERATOR
component
 cluster GENERATOR
 component
  class POLLINGCARDS
   feature
    make: POLLINGCARDS -> electionName : STRING -> electionDate : STRING -> electionTime : STRING
     require
      electionName /= void and electionDate /= void and electionTime /= void
     end
    CreatePollingCard -> p : PERSON -> ADDRESS : sender -> ADDRESS : pollingVenue -> ADDRESS
      require p /= void
     end
    SaveToDisk -> path : STRING
     require path /= void
     ensure "FILE.exist(path)"
     end
  end

  class VOTERLIST
   feature
    make: VOTERLIST -> rows : NATURAL -> electionName : STRING -> date : STRING -> table : STRING
     require
      rows > 20 and electionname /= void and date /= void and table /= void
     end
    AddVoter -> p : PERSON
     require p /= void
     end
    SaveToDisk -> path : STRING
     require path /= void
     ensure "FILE.exist(path)"
     end
  end

  class VOTER_ID_GENERATOR
   feature
   CreateUniqueID : STRING
    ensure "return value > old(value)"
    end
  end
 end
end
```

Server.bon

```
system_chart SERVER_APPLICATION
 indexing
   author: "Henrik Haugbølle <hhau@itu.dk>";
   course: "BDSA";
   university: "ITU";
   created: "2011-12-08";
 explanation
   "Local server application used to serve the clients with data from the external data source upon request."

 cluster SERVER_PACKAGE description "The namespace containing the SERVER class."
end

cluster_chart SERVER_PACKAGE
 indexing
   author: "Henrik Haugbølle <hhau@itu.dk>";
   created: "2011-12-12";
   keywords: "server, data";
 explanation
   "The namespace containing the SERVER and SERVER_DATA classes."

 class SERVER description "Server class handling requests from the client application."
 class SERVER_DATA description "Assists the server in importing and removing data from the database."
end

class_chart SERVER
 indexing
   author: "Henrik Haugbølle <hhau@itu.dk>";
   created: "2011-12-08";
   keywords: "server, network, external data source, data";
 explanation
   "Server class handling requests from the client application."
 query
   "Create an instance of the server?",
   "Fetch a person given this cpr?",
   "Fetch a person given this voter id?",
   "Register that this person has voted?",
   "Unregister that this person has voted?",
   "Give all the polling tables?"
 command
   "Start the server!"
end

class_chart SERVER_DATA
 indexing
   author: "Henrik Haugbølle <hhau@itu.dk>";
   created: "2011-12-11";
   keywords: "commandline, external data source, data";
 explanation
   "Assists the server in importing and removing data from the database."
 query
   "Create an instance of the server data class?"
 command
   "Import data to the database!",
   "Clear table in the database!"
end

-- Helper types defined in other files
static_diagram CORE_TYPES
component
 class PERSON
end

static_diagram SERVER_APPLICATION
component
 cluster SERVER_PACKAGE
 component
```

```
  class SERVER
    feature
      Make : THIS
      Start : VOID
      CprToPerson : PERSON -> cpr : STRING
        require
          cpr /= VOID
        end
      VoterIdToPerson : PERSON -> voter_id : INTEGER
      RegisterVoter : BOOLEAN -> person : PERSON
        require
          person /= void
        ensure
          -- old VoterIdToPerson(person.VoterId).Exists = FALSE or old VoterIdToPerson(person.VoterId).Voted = TRUE ? Result =
false : Result = true
          TRUE
        end
      UnregisterVoter : BOOLEAN -> person : PERSON
        require
          person /= void
        ensure
          -- old VoterIdToPerson(person.VoterId).Exists = FALSE or old VoterIdToPerson(person.VoterId).Voted = FALSE ? Result =
false : Result = true
          TRUE
        end
      AvailableTables : ARRAY[STRING]
    end
  class SERVER_DATA
    feature
      Make : THIS
      Import : BOOLEAN -> arguments : ARRAY[STRING]
      Clear : BOOLEAN -> arguments : ARRAY[STRING]
    end
 end
end
```

Appendix C: Revision History

2011 November 21 - henrikhaugboelle - “first commit”
2011 December 05 - ChristianOlsson - “Udp, request-reply and RPC v1.0”
2011 December 07 - kaaresylow - “voterlist and pollingcards pdf generator is added”
2011 December 07 - henrikhaugboelle - “working prototype of database classes and entity classes”
2011 December 07 - ChristianOlsson - “Ping and Table added to RPC”
2011 December 07 - ChristianOlsson - “Client application v1.0”
2011 December 08 - kaaresylow - “admin gui almost done”
2011 December 08 - henrikhaugboelle - “entities update and server prototyping”
2011 December 08 - henrikhaugboelle - “server test suite and comments, plus a little persontate/person adding”
2011 December 09 - henrikhaugboelle - “fixes on server plus prototype of server installer”
2011 December 09 - kaaresylow - “Admin gui almost done, still no export function”
2011 December 09 - ChristianOlsson - “Network final adjustments”
2011 December 09 - kaaresylow - “XML schema validation is implemented”
2011 December 10 - kaaresylow - “export to csv is implemented in the admin gui, comments are added to the pdf polling card class”
2011 December 10 - kaaresylow - “new export window in the admin gui”
2011 December 10 - henrikhaugboelle - “data generator with gui interface”
2011 December 10 - ChristianOlsson - “Network test suite added”
2011 December 11 - kaaresylow “comments are added to voterlist generator, fileimport/export is moved to IO in smalltuba project”
2011 December 11 - henrikhaugboelle - “server cmdline interface, and import/clear functionality”
2011 December 12 - henrikhaugboelle - “entity test suite”
2011 December 12 - kaaresylow - “files for io test”
2011 December 12 - kaaresylow - “RC1”
2011 Deecmber 12 - henrikhaugboelle - “admin and client final installers”

Appendix D: Original Overview

Digital Voter Registration System

Authors

Henrik Haugbølle - hhau@itu.dk
Christian Olsson - chro@itu.dk
Kåre Sylow Pedersen - ksyl@itu.dk

Abstract

Registration of voters at a Danish election is normally done by pen and paper. A voter is registered when the ballot is handed out so a voter cannot vote twice. A digital voter registration system enables an election to be executed with more efficiency than with the original procedure.

Originally each polling table has a list of voters permitted to get a ballot from that table. By digitization of the lists a voter can attend any polling table and the look-up is performed instantly. Increased registration efficiency will require less staff and less waiting time for each citizen.

Our solution is separated into two main parts. The first part being the centrally managed generation of polling cards and voter registration lists both in digital and printable format. The second part being the digital registration of voter attendance at each electoral venue.

Mandatory Requirements

- Generate voter cards in printable format
- Generate registration lists for specific venues and polling tables in printable format
- Locally validate and register voter attendance digitally without Internet connectivity
- Validate a citizen by voting card and/or by social security number (CPR-no)
- Guarantee that exactly those in the data set provided can vote and no one can vote twice
- Include a graphical user interface for both the central management and local clients
- Server and multiple clients on different computers at the same venue

Optional Requirements

- Generate barcodes on voter cards for more efficient look-ups
- Facilitate local server redundancy

Appendix E: Polling Card

Test Election
13. december 2011

Medbring kortet ved afstemningen

Afstemningssted:
Plejecentret Sølund
Oliebladsgade 117
Furesø

Valgbord: **15**

Vælgernr.: **1323790116**

Afstemningstid: **09.00 - 20.00**

1323790116

Afsender:
Furesø Kommune
Strandøre 172
Furesø

Modtager:
Thea Thomsen
Rundholmen 96
Furesø

VALGKORT