DISTRIBUTED SYSTEMS

Server-client system for electric cars batteries management

Mandatory project within 3rd semester course in Computer Science
by Alexandra Kønig

Mentors:

Simon Kongshøj

Michael Holm Andersen

Gianna Bellé

Supervisors:

Simon Kongshøj

Class: dmaa0214

Group: 7

Submission date:

04/06-2015

ABSTRACT

This document is a description to the practical part of the 3rd semester exam project within Computer Science programm at UCN, Aalborg. The subject to the project is development of a client-server computer system to the problem of Electric Cars, as one of the given choices to work with. This documentation is presented in four major headings. The ”**Introduction**” introduces to the problem given and the framework of the solution. ”**Functional specification**” dicusses the means of its functionality in its enviorment, and answers to the questions of ”what” and ”why” of the project. ”**Technical specification**” goes into details of how the major technical problems were solved and answers to the ”how's”. ”**Evaluation**” briefly summarises my experience during this project.

TABLE OF CONTENTS

Glossary

Project – this project.

System – unless stated otherwise, the is the client-server system, the subject of the project.

Application – server side module of the system, composed of services and logic.

Location – a dedicated battery replacement station.

Location network – a set of locations connected to each other

Final user – user of the electric car, who uses the system.

Customer – the group of people, or company taking the system under deployement and utilization.

Manager – a person or a programm which manages the acquisition and replacement of the batteries at the locations

Administrator – a person, dedicated to administrating a network of locations.

Developer – producer of the system, or its parts.

WCF – Windows Communication Foundation

Price-to-value – comparative resources usage in implementation of a given scenario to the result it gives. High price-to-value indicates a goal not worth the resources spent. Low price-to-value indicates achieving a goal with minimum resources spent.

INTRODUCTION

Problem definition

Electric cars face a problem of a comparably short life of their batteries, which in turn makes long distance travels somewhat problematic. Among other solutions to this problem there emerges a concept of rapid replacement of battery units at dedicated battery replacement stations. Each station has a limited number of battery units available at given time. This project is aimed at developing an electronic system, which allows calculate optimal route for the traveler based on the location of battery stations and number of available battery units at each station and to book battery units at the estimated time of arrival. The system suggests extension to support of autonomic data management and to be accessible from internet and/or other mobile platforms later on.

Concept

The system purpose is to allow the final user to calculate an optimal route from the point of departure to point of arrival using the data about location network and availability of the batteries at the locations, stored within the system, and to book the batteries on that path. The system also supports registering acquired batteries and allows the administrators of the system to edit the network of the locations together with their links and batteries attached to them. The system can be applied to an isolated or global network. The access to the system is gained by the final user through a web browser, and by manager and administrator – through dedicated clients, which have an authentication function.

Domain model

This scematic illustration in form of domain model helps to understand what elements, with what properties and which internal relations the reality presents.

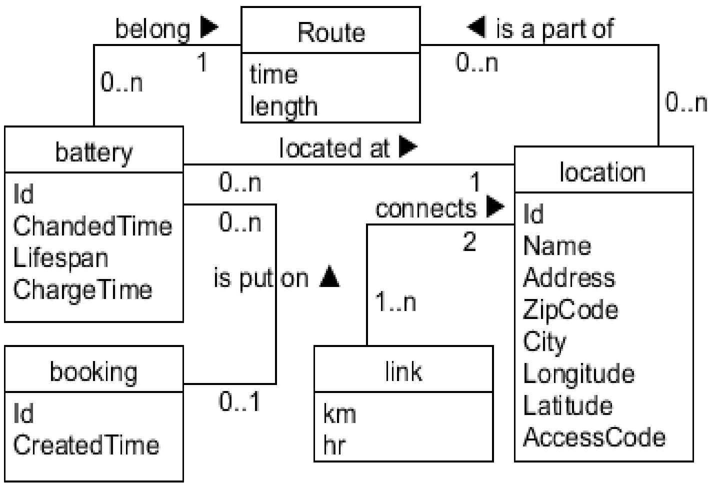

Illustration 1: Domain Model

There is a **car**, with a **battery**, which <u>*has*</u> a limited **lifespan**. The car takes a a **trip** from one **location** to another. The **trip** can <u>*include*</u> several **locations** on its way. Each **location** can <u>*have*</u> a **link** to several others.

Location _has_ several **batteries**. The **batteries** at **location** can be **charged** or not. It takes a certain **time to charge** a _battery_. There is a **car user**, who _plans_ a **trip**, and would like to know an optimal **route**, and if **route's length** exeeds **battery life** of her **car**, she is able to _change_ the **battery** at one of the **routes locations**, possibly a *couple of times*. The user would like to be sure there is a **charged battery** at the **location** when she arrives therefore will like to **book** charged **batteries**, or **batteries** which will be **charged by the time** of arrival forehand.

Manager who has **permission** to change batteries at this location[1] upon arrival of the car **user** will find the **booked battery**, _exchange_ it with the **battery** in the car.

Conditions and assumptions:

- The trip is calculated out of the assumption that the car will start moving immidiately after the booking is made.
- The booking has a limited time of validity

This model presents the real problem in scematic way and is not a code presentation yet, but it gives the overview over the conceptual classes existance and their interrelativity:

> It [domain model] illustrates noteworthy concepts in a domain. It ***can*** be source of inspiration for designning ***some*** software objects […]. […] "Domain model" mean a representation of ***real-situation*** conceptual classes, ***not software objects***. [In the agile approach]the purpouse of creating a domain model is to quickly understand and communicatie a rough approximination of the key comcepts. [..]There is no such thing as a single correct domain model.
>
> Craig Larman, "Applying UML and Patterns"

From this model we can see that the system shall operate at least with the illustrated objects and their attributes, and to present the shown relations. Beside that, we can see that there is brief role definition emerging from actors mentioned. Considering the case, which defines also a problem of autonomic managment of the data in the system, it is possible to define one more user role beside the given two (the user and the manager): administrator. The system is structured around this conceptual model.

1 Can be car user herself, a station personel, or an automatic terminal

FUNCTIONAL SPECIFICATION

Insight

This chapter is dedicated to describing the functionality of the system and why those functions are implemented. The chapter covers the subjects of:

- **Development** – information about what the system presents to its users and how the system is utilized, and what user can expect from the system;
- **Deployement** – covers information about how the system is applied, what technologies it uses, and what technologies are required for its deployement;
- **Administration** – covers information about what is required from the deployers of the system to maintain it;
- and **Performance** – covers information about outside factors that can affect the system and how the system handles it.

Development

USER BASE

Final user. The system is developed with Initial final user in mind being a Danish sitizen, who has an electric car as only long distance vehicle in possession and addresses the system when plans a trip on her electric vehicle, which cannot be performed on one battery charge. The system is accessed via a web-side, which presumes an internet connection. The user then will choose the closest to her trip start and end points locations, from two lists, provided by the system and will request the route. The system will present route if found, together with locations suggested for changing car's battery, and a list of batteries available for booking. The user will request booking of the presented batteries. The system then, if booking was successful, will return the booking code. If the booking was not successful, the system will suggest to the user to find an alternative route or cancel the seach.(See illustration 2).

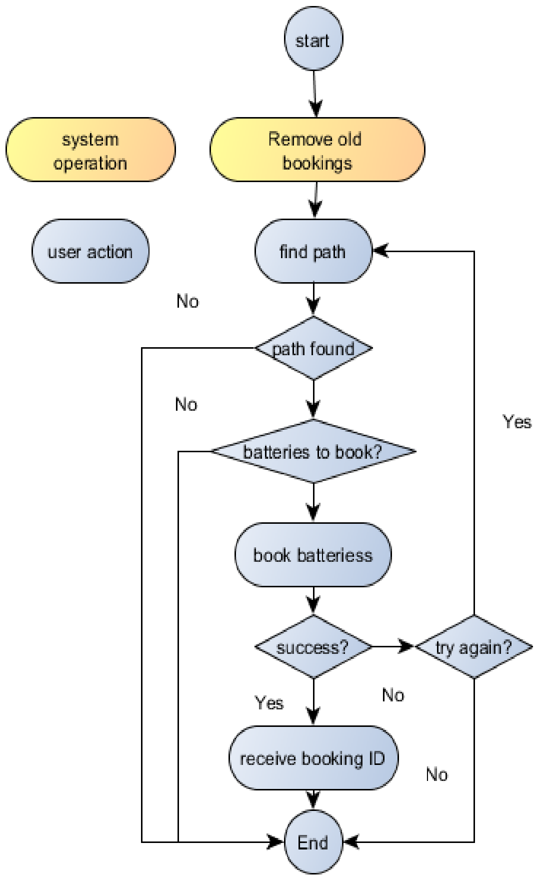

Illustration 2: Final user scenario

Management. The system requires a management unit presented by a human, operating a personal computer with an installed dedicated software, or a partially or fully automated terminal supplied with a dedicated software. Upon arival of the user, the manager will recive the booking code from the user, find the battery in the database, release the battery from booking and then will provide the user with charged battery, recieving in return the exhaused battery from the user's car, which will be placed on the station for recharging. The new database will recive a timestamp of when it was replaced. Comparing this time with configured in the system average recharging time, the system can use this battery in calculating traveling routes for other users, thus minimizing the downtime of the batteries. The manager system needs authentication in the system in order to functions. .

Administration. The system has a support for administration. The administratior has ability to modify existing location network by deleting, creating or editing locations, links between them and the batteries attached to those locations. In order to access those functions of the system, administrator has to provide authentication to the system with private login and password.

INTEROPERABILITY

The system is based on WCF application, which allows development of clients, that support the serialization provided by the technology.[2] This technology allows development and distribution additional clients, thus improving the user's and management's experiece. The peak of this oppotunity lies in automation management with terminals, which will allow the customer to optimize the running revenue of their stations.

UPGRADES AND UPDATES

Global changes

The system is currently aimed at a relatively small group of final users located in Denmark. Current final user base allows comparably low access conflict rate. However, with the growth of the user base, the system will have to consider more sophisticated means of dealing with the problem. The challange of the sophisticated versus size of the user base determines the price-to-value grade in the development and implementation.[3]

The growth of user base will indicate two points of big upgrades to the system, wich will affect the user client functionality and its interface.

Minor improvements

The system has a window to enchancing the final user's experience in the functionality and performance:

- existing modification of **base algorythm** is a subject to change – will not affect any other system's modules;
- **time search** – allowing to find a fastest route instead of the shortest. The improvement will require a changes in client. The older versions of clients still will able to function notmally;
- **log** monitoring – allowing to track access to the system on manager or admin level, and activity performed by with this access and fails the system confronts regulary;
- making **booking** possible ahead from several hours to several days, which will make it more practical to plan two-way travels;
- reconsideration of **address format**, in particular danish zip-code model;
- reconsideration of **linking model**, making multiple linking between two locations possible;

2 For details about WCF implementation see ”Deployement” (Page 8)

3 For details about scalability see ”Performance” (Page 10).

- **modification of SQL handler**, improving database communication. Will not affect any other tiers or modules.
- **statistic tools**, including those to track concurrency events frequency. The tool will affect administration clients.
- **services configuration settings** allowing to change the battery life time, battery charging timespan, and restrictions to deletion of booked batteries and locations, to which those booked batteries are assigned.

SECURITY AND AUTHENTICATION

The system provides three levels of data access and thus three levels of security, and separates the data access to different services distinct by the access permission and functionality.

User. Booking changes to system's data are free to the final user and do not require authorized access. The booking service has acess to creating a new booking and changing the batteries data about their booking if the batteries do not already have booking registered to them. The final user provides the batteries identification numers. The service will book the batteries with those identification numbers, if they do not have any booking already assigned to them. The booking will be cleared with next user access to data storage if the booking exeeds the maximum booking time, which is set by default to 24 hours. The maximum booking time can be canged by administrator[4].

Manager. Changes to booking before maximum booking time exceeds are allowed only through management service contract. The client shall provide the location identification number and the access code related to that location.

Admin. The changess related to adding, removing or editing locations, links between them and the batteries related to them are only aviable through the administration service. To access the service, the client shall provide an unique identification and access code attached to the administrator. The system supports mulltiple administrators and logs their activity[5]

Both management and administrator authentications use *Custom* authentication method due to expectance of high price-to-value implementation of other scenarios. Implementation of sophisticated schemes, such as Membership Provider would require a user base of administration and managment, which cannot be monitored manually. Malicios access on a low number of locations in location network, which can be estimated for Denmark does not reason with the scale of the implementation of such a system and can be monitored with logs manually or programically.

The deployer is responsibilie of the security of the system from outside influence if the system is deployed in the closed network, and redundancy.

Deployment

On the application level the system uses Windows Communication Foundation(WCF). The choice of the platform is determined by

4 Not implemented.

5 Managment identification is partially implemented, administration authentication is not implemented

- WCF interoperability with other applications and non-WCF using SOAP messaging, which allows expansion and cross-integration of the system with other projects.
- Unified API, allowing to various types of communication
- WCF clear separation of service interface, communication and logic tiers, allowing their reuse and independant modification, which are important for this system that estimates major upgrades in the future.

The applicattion is presented by three services unified by business (logic) level with a single databbase handler module. The separation is determined by the different authentication methods and functionality of the user roles (See Illustration [3]).

Protocols. Each service exposes one HTTP binding, the choice of which is determined by the communication means, and which are not limited to WCF platform. The this system the majority of communication is aimed at web applications. There are options of expanding the endpoints with TCP bindings on the management and administration services for the clients that can eventually be developed on the WCF platforms.

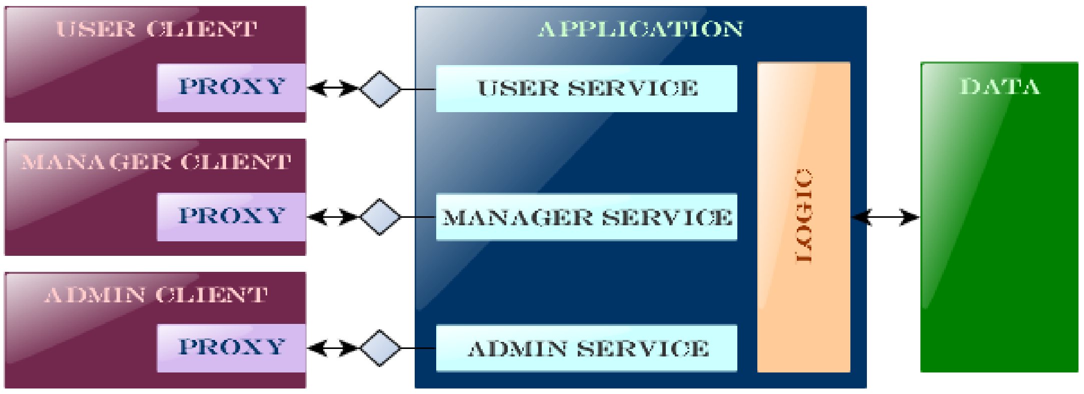

Illustration 3: System Design

Ports. At the current state the bindings use adresses with ports 8733-8736. The choice of the ports was determined by the test conditions. With current userbase the ports may presist. Running services with custom ports give the allowance of keeping the operating system in a user mode, which provides highter security to administrative mode. With a bigger user base and a higher spread of the system it is considerable to change HTTP bindings to their commonly recognized port 80, exposing thus security of the system, yet increasing the accessability of the system. This change shall be weighted with price-to-value considerations because it will require additional security prcausions.

Management and administration

Uphold. The system's deployement and uphold is responsobility of the customer. The system provides tools for monitoring and administration of the system as well as improvements of its initial components: clients, application and datastorage.

Error reports. The system provides extended failure tracking based on integrated fail-reports and logging of server errors on the application level[6]. The users recive notification about fails related to their activity on the client side. The client responses are generated by the client based on the messages recived from server.

Changes responsobilities. The customer also takes the responsobility of handlig the booking with supplied management clients and modification of the location network with administration client. The developer takes a responsibility of improving all the modules of the system. The system allows integration of third-side client software supported by service technology thus allowing expansion of the system with, for example, already suggested terminal managment solutions.

Performance

SCALABILITY

Performance of the system if deternined by final user base and locations network.

The current composition of the system is aimed at final user group consisting of people utilizing electric cars stationed in Denmark, the number of wich at the end of 2014 was 3000 units[7]. A large number of this amount is assumably owned by public and social workers and are never aimed at long distance travel. Thus, even with doble growth of the amount of cars in denmark every year, one can estimate the following load on the system closest years (See table [1]).

	2016	2017	2018
Maximum possible unique users	3000	6000	12000
Connections a day in consideration of one unique connection per year	10	20	30
Connections per hour in consideration of one unique connection per year	2,4	4,8	9,2

Table 1: Estimated maximum unique users of the system yearly in Denmark

Reasonably to suggest that season, weekend and holiday factor will change those numbers up or down. Its also reasonable to estimate that not every single private user of the cars will do long travels regulary, and that not everyone who does long travels will use the system. Of those who would, they probably will use system at least twice a year. Yet even with those estimations the numbers will not change greatly (See Table [2]).

One can estimate the numbers double or even triple in certain time of year, yet it is not enough to worry about a load high enough to concider multiple application or database servers to support the system. However a matter of concurrency may appear in its mild form, where booking conflicts have to be handled programmacally. The database load is handled by composed requests on logic level and a choice of booking scenario.

6 Logging planned byt not implented in this release

7 See Appendix B for details on stock of electric cars statistics.

	2016	2017	2018
Maximum estimated real connections per year	1000	2000	4000
Estimated real connections a day	2,7	5,4	10,8
Estimated real connections per hour	0,1	0,2	0,4

Table 2: Estimated real connections yearly in Denmark

The change of booking scenario may affect the load on database and change the concurrency conflict events frequency without a need to change logic. Compare initial scenario (Illustration 2, page 6) and alternative scenarios (Appendix C: 1). In the current scenario the user is given a choice to affect the database or not by activating booking operation, thus opening a time window for another user to ”steal” the batteries from her path. The other two scenarios modify data no matter, was the user going to book suggested batteries or not. The fewer users there are to book the batteries at the given moment, the less is the need for the shorter window between calculating the bath and booking. The more users there are searching for the path, the more data modifications are perfomed, and in the case of two alternative scenarios, independantly of the user's intention to actually book the suggeted batteries, thus increasing a load on database.

GROWTH ESTIMATIONS

The picture begins to change if the system is to be applied to a wider user base, for example the whole EU. Just Germany, France, Netherlands and Norway together will multiply the danish user base by ten. It is still not enough to get conserned about the load on server or database, but in season peaks may bring concurrency handling to a new level. In this case a next system upgrade will be nessesary.

TECHNICAL SPECIFICATION

Insight

This chapter is dedicated to describing the technical implementation of the system and its components, and their specifics. The chapter covers the subjects of:

- **Architecture** – the overview on the whole structure of the system;
- **Pathfinding algorithm** – goes into the details about the main feature of the business layer;
- **Data tier** – presents the implementation of data storage solution and the methods the system uses to operate the data;
- **Failure tolerance** – discusses possible points of failure within the sysem and by what means they are handled.

Architecture

The final solution presents a system in three tiers: Interface tier, Logic tier, and Data tier.

Interface tier includes three clients: two application windows forms clients for administration and managment purposes and one web client for the final user. Each client is dedicated to a specific role, and thus carries specific to this role functions. User web-application client carres a role of finding path and booking free batteries. Manager windows forms client carries a role of releasing the booked batteries, if a correct booking code is provided. And the administration windows forms clienf is dedicated to managing the location network

Logic tier is presented by a web-service build on WCF technology. There are four layers in this tier: Presentation, Models, Business and Data handlers. Models layer is presented by the number modular classes wich implement the abstract classes of domain model and are used to present data. Business layer in its majority present the pathfinder, and its minority include smaller handlers. Data handlers layer include a library of classes mastering the connection to database and datatransfer. Presentation layer performs a fuction of a connection between the presentation tier and logic tier in the same manner as Database Handler module performs the connection between the database and the logic tier. Presentation layer exposes tree endpoints for the service: user, manager and admin.

Data tier is presented by SQL databse with five data tables.

Logic tier and pathfinding algorithm

STRUCTURE

Path-finding is a core of the project and therefore one of the most important functions in the system.

The system uses Dijkstra's algorithms as a base to the pathfinder in the system. The choice of Dijkstra foundation for the pathfinder is determined by its outperformance on for the particular case of weighted undirected graph.

There were considered other choices. The primitive pathfinding methods of one step at a time or, also so-called "dont look ahead" are excuded on the breach as being insuffisient for complicated logic that was expected of them:

- find shortest possible path
- terminate if path does not exist
- be able to transverse indefinite depth integration with additional data

Among sophisticated methods, overlooking the graph ahead, there were considered common search methods:

- **Breadth First Search**, which is focused on layer scanning and good for finding a path, but does not ensure its optimal length;
- **Depth First Search**, which is focused on branch scanning, and thus is unfitting for the network of roads, for the reason of the network not nessesary being a tree;
- **Best First Search,** which is a variation if Breadth First search with implementation of of heuritics, and which make the algoryithm to work faster than Breadth First search, yet it still doesnt ensure the shortest path returned;
- **A*** - Similar to Best First Search uses heuritics to estimate how close is each next node to the goal. Unfitting for for our case for the reason of the nodes not having a shortest connection due to roads not going in a straght line;
- **Variations of A*:**
  - **D*** - dynamic, consideres changes in the graph under the search process;
  - **HPA*** - hierarcical, speeds up the algorithm by implementating several abstract search levels tracking different heuristic;
  - **IDA*** – improves algorithm's speed by combining the breadth search of A* with dephth search into iterative deepening;
  - **Djikstra** – a case of A* algorithm, where heuristics has same value for all node and implements tracking of nodes cost adjusting it with lowest found.

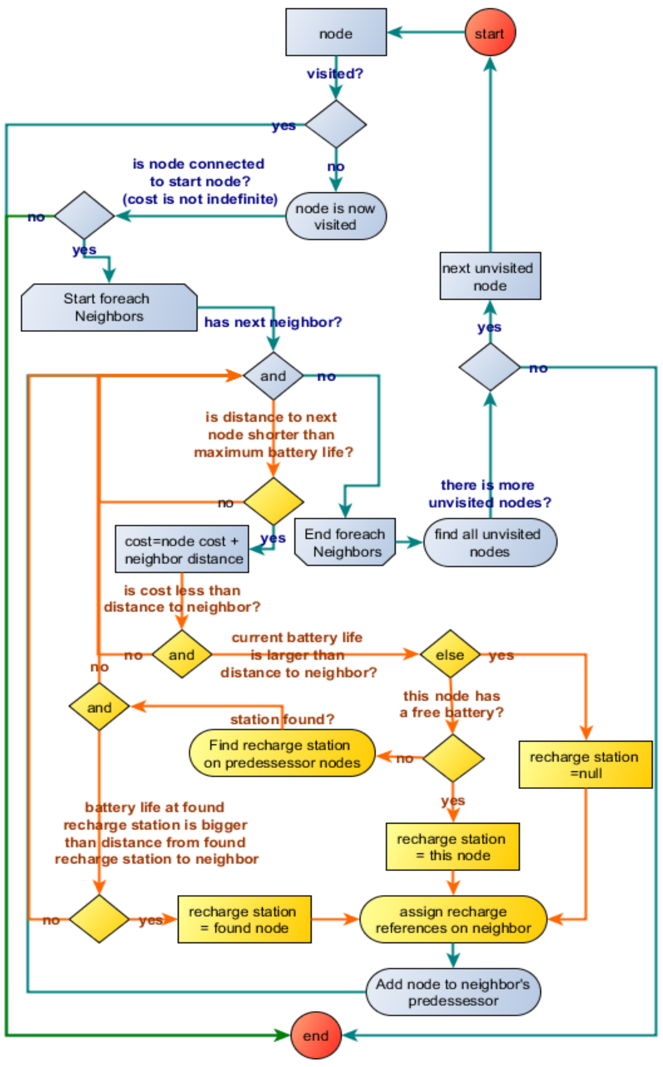
start
node
visited?
yes
no
node is now visited
is node connected to start node? (cost is not indefinite)
no
yes
Start foreach Neighbors
has next neighbor?
and
no
is distance to next node shorter than maximum battery life?
no
yes
cost=node cost + neighbor distance
End foreach Neighbors
find all unvisited nodes
there is more unvisited nodes?
yes
no
next unvisited node
is cost less than distance to neighbor?
no
and
current battery life is larger than distance to neighbor?
else
yes
no
and
station found?
Find recharge station on predessessor nodes
this node has a free battery?
no
yes
recharge station =null
recharge station = this node
battery life at found recharge station is bigger than distance from found recharge station to neighbor
no
yes
recharge station = found node
assign recharge references on neighbor
Add node to neighbor's predessessor
end

Illustration 4: Modified Dijkstra's Algorithm

With excluded heuristics and a track of the path length trough a counter, though it doesn't quite satisfy the demands of the problem, Dijkstra stands out from all other considered methods. On the Illustration[4] (and Appendix C: 2) it is possible to see how Dijkstra is modified to fit the basic requirements for the path-finding in this system.

Modification of the standart algorith brings a tight integration with modular classes and challanges of estimating its complexity and isolation from the project.

COMPLEXITY

Due to its nature, the nework of roads will never be a complete graph, and is very much likely to be a sparce graph rather than dense, depending on the area covered by the location network. Denamrk alone will tend to get closer to dense graph, but as the location network grows and system scales the graph becomes more and more sparse. This determines the choice of ajacency implementaton. The algorithm uses ajacency-list implementation. For representing edges in the graph.

Dijkstra's algorythm performs at the **O(V^2)** complexity where **V** is number of vertecies in the graph. With modification there was implemented logic, which will search the existing shortest path for an aviable battery when the factor of battery life will exceed the length of the total found path. The worst case for this operation will be a path length of total number of vertices where the start to end minus one is less than battery life and the iteration will have to process the whole path (total number of vertices in the graph), which will make the complexity become at worst **O(V^3)**.

The logic of the modification is a subject to improvement.

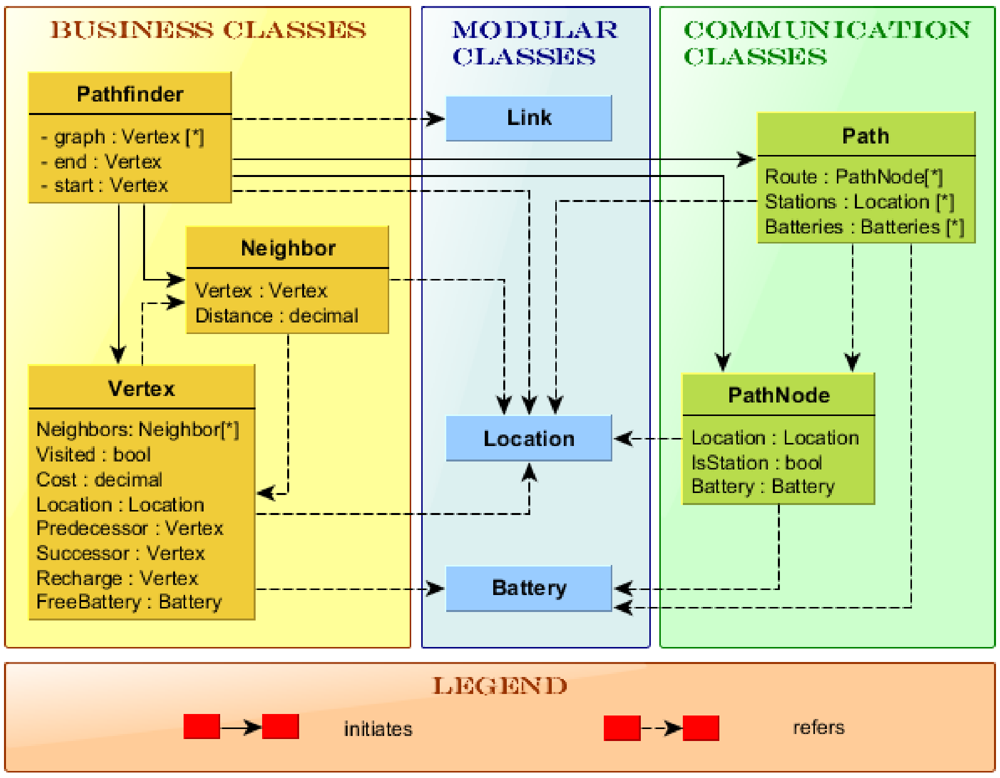

Illustration 5: Pathfinder class model

FUNCTIONALITY ISOLATION

Due to deep integration with the modular classes, imlementation of the algorythm logic and graph modularity confronts a question of its isolation from the business layer. In the context of the current scale of the project, where the pathfinding algorithm is the only function presented in the relation to the graph, and with the considerations of the uniquness of the algorithm due to its modification, isolation is considered inefficient due to estimated high price-to-value.

COMMUNICATION

The pathfinder class includes data and references to suffisticated classes built to answer the mechanics of the algorithm. Its final form is rather overweighted to pass to client. Additional classes are implemented to fulfill this role. Upon completing its task of finding the path, Pathfinder class translates the final data into user-friendly structure and presents the final result with two classes: Path and Pathnode. They hold information about the sequence of the pathnodes and the batteries, that has to booked. Those classes exclude all the data, which was used for the path calculation.

Data tier

IMPLEMENTATION

Data tier is presented by SQL databse on the platform of **MSSQL Server v11.0.2100.60**. Use of dedicated database is determined at first place by concurrency challange settling the system. Dedicated server allows assign much of the concurrency responsibility to database thus excluding many of delay events in communication between application and data that may affect the precision of data handling. Dedicated database server also has its own integrated exception handling and additional tools for optimizing data queries. MSSQL server is most compatible with other Microsoft tools and technologies used in implementing this system, thus defining its preference before other choices.

Data access from application is gained with a help of specifically designed for that data handler module. Arguments in favor of use of MS product for the choice of database server can make it obvious of continuing the trend and use ADO.NET technology for the communication of apllication with database. ADO.NET provides a large library allowing manlipulate data and is very efficient with LINQ selection queries. However, when it comes to processing multiple operations in one query, ADO.NET shows a lack of simple tools. Often a little more sophisticated query of more than one operation meets a nessesity of lengthy adjustment to the library and additional coding to implement the lacking functionality. One of examples of a high price-to-value cases can be a select scope identity query with ADO.NET, which is relatively common and useful operation in SQL enviorment and is used straight after insert query but was poorly implemented in this library. A little more sophisticaded queries often may end in a large scale adjustments. ADO.NET is good, when there is a need for a lot of various selec queries, but the task of this system differes, therefor the was taken a decition of developing a module, which will be compitable with the system design and will make any further improvements easy to implement.

STRUCTURE

The database for the system consist of five data tables, as defined in domain (See Illustration [1], on page 4), excluding Route, as it is remporary data, which is disposed after using, because it get quickly outdated.

There was added aditional model for ZipCode, as it is a strucured and reusable data, and is not defined by domain model. Location, Battery and Booking get an identification, making distinguishing and cross-referece possible.

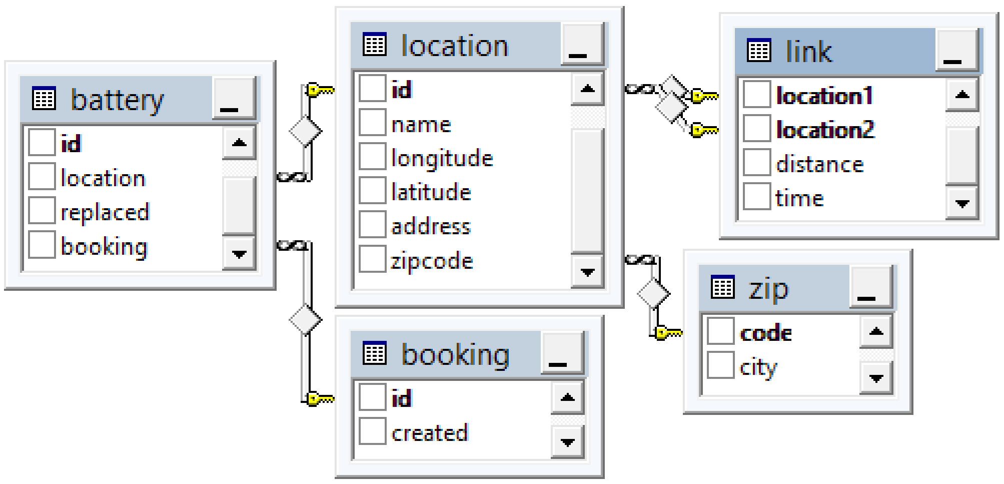

Illustration 6: Database model

Booking has an unique identifier, which with the dynamics of this data make the booking table more efficient. The booking id is also presented to the final user after booking is complete.

Zipcode is identified by common danish 4-digit number and has to be reconsidered, when system expands to other countries.

Link has a pair identifier, constrained to location Id. This identifier, however is also a subject to reconsideration due to specific of the location network, where can be more than one links between two locations of the different length and/or time. This reconsideration may affect the path-finding algorithm, making it more suffisticated and therefore was not implemented in first release.

SQL HANDLER

Structure

Sql handler module establishes connection between application and database. It is developed to answer the specifics of the project and to ensure fast integration of additional functions related to data.

The module consist of several classes which perform dedicated functions (See table [3]).

System operates with modular classes, which are presented by attributes and methods desclibing the modles, and Database classes, which present the collection of models. Business layer communicates with modular and database classes and they in turn communicate with SQL module, which handles operations with database (See illustration [7]).

Class	Function
Sql connection builder	Builds a standart connection string for accessing database based on giving parameters: server, password, username, database,. Allows a quick reconfiguration of database acess, Usefull for testing and further deployemnt, as well as for changing database, if such would be nessery
Sql Model	Standart interface for model classes. Used as a standart reference in other classes of the module
Sql Handler	Performs the creation of the model out of the data recived from database, and provides tools for transmitting the changes to database and handles the cache changes. The class provides various scenarious of the queries from common to specific.
SqlFormat	Static class providing transormation of data types from SQL to C# format
SqlData	Opposit to SqlFormat, provides simple interface to pass the data to Sql Handler which will use a query pattern to pass it further to database.
Sql Response	Passes the information about trasmissions between database and the logic layer
Caching	Handles caching

Table 3: Sql Handler module componets

Data request

When business layer requests data from database, it communicates with related database class. Database class then uses SqlHandler select methods to retrieve the data from database. If cache is emty, Sql Halndler will use SqlConnectionBuilder to establish connection to database, then for each retrieved row from the table it will initiate a new model class and call for an object builder within this class. Every failure within handler itself or with database is stored within the Response object, as well as report on final success. Object builder, with help of SqlFormat translates database datatypes to C# datatypes and builds the object complete according to its properties. If cache is not empty, Sql Handler will get the data from cache intstead, thus skipping extra request to database, and then will return a list of models to databse class. If response doesnt show fails with database operation, Database class will sort the list and retrive only objects conditioned by the method and return it to business layer.

Data modification

When business class requires to modify the data, it will initiate a model class, and call a method of desired interaction. The method passes object and conditions to related database class. Database class then checks the passed data for validity and if success, it will initiate an SqlData object with given data, which will parse and translate the data to Sql format, if needed, and will prepare data for Sql queries. If the data is not valid, it will initiate the SqlResponse object with respective response. Further on, the SqlData object will be passed to SqlHandler, which will build a respective to the request and conditions query with SqlData object and then connect to database and execute the query and remove related caching in case of successful operation. The results will be returned to Database class with SqlResponse.

Benifits and oppotunities

Use of SqlData class allows SqlHandler to be modified with sophisticated queries as it translates passed data to acceptable by Sql format datatypes, and prepares sub-queries according to the data types it knows, thus excluding the fails on sql syntax, if SqlHandler has to be edited with new queries. SqlData itself can be modified to recognize more data types, without interfering into the functions of the communication itself.

Thus this module is aimed at fast and fail-less modification and achieves the following result:

- separates model from database functionality;
- shares repeatable code,functions and queries;
- translates data from sql to c# and the other way without developer's concern;
- allows adding sufficticated queries fast and without any risk of syntax error;
- extracts sorting function from query and assign it to dedicated class, thus allowing to have an overview of the sort methods and their rapid modification;
- has an extend report on handlers and database operations, which can be used in other modules of the application;
- has inbuild connection string builder, allowing its quick modification and separating database connection from database operation functionality.

Although the module at the moment functions acceptable, it has a window for improvement, nessesety of which will grow as the intensity of the system usage scales.

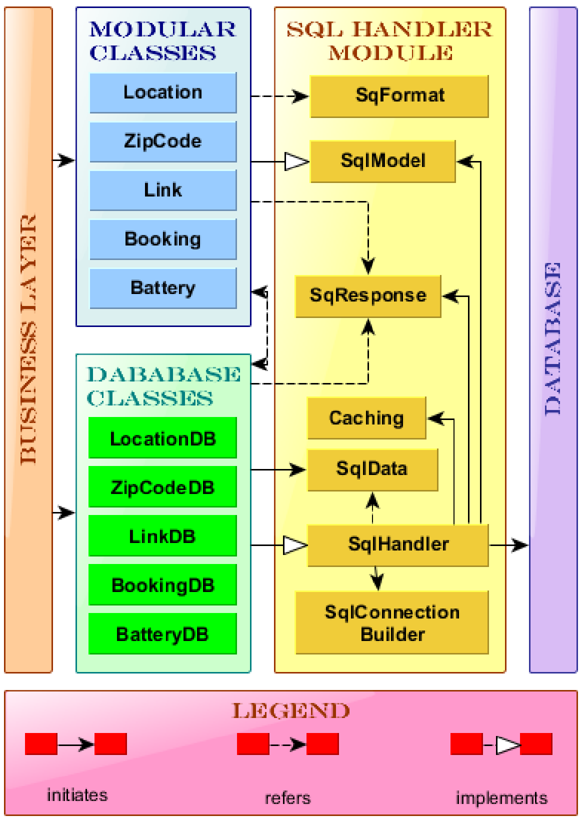

Illustration 7: SQL handler integration

CONCURRENCY

Definition

As it was already discussed, the system confronts a challange of concurrency. The problem accures when a final user tries to book batteries on earlier found path, while the suggested to her batteries were taken occuped by another booking. Intencity of this problem depends on user base scale. The problem can be solved on different levels and with different methods. Combination of those methods allows reguate their price-to-value implementation and adjust as needed with further updates.

Long before taking making any calculations, it is important to understand, that it is impossible to completely avoid any conflicts in booking. Even in the most grammatically sufficient solution to the problem errors will still accrue because time passed between the moment the pathfinder recieves the request from the user on finding the path, and to the moment the calculations are made is enough for another request to pass through the system and to make the booking on the batteries, which are listed in the data pathfinder operates with as available (See illustration [8]).

The frequency of this scenario accurance is dependant on the user base size. The more users are addressing the system at the given time the more frequient is the conflict.

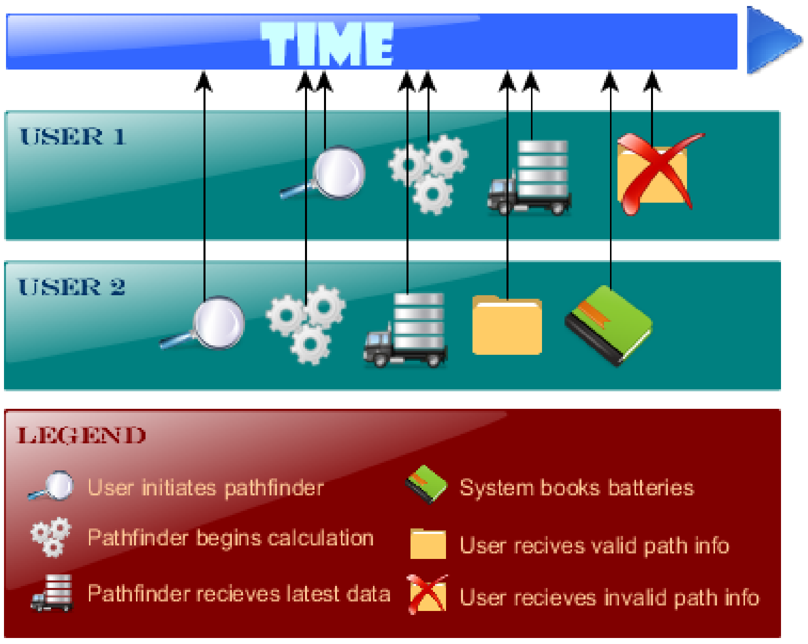
TIME
USER 1
USER 2
LEGEND
User initiates pathfinder
Pathfinder begins calculation
Pathfinder recieves latest data
System books batteries
User recives valid path info
User recieves invalid path info

Illustration 8: Concurrency problem scenario

There are several points where the conflict can be stopped and the ways it can be done:

- block any changes to database while pathfinder operates
- pathfinder keeps the track of datachanges while the calculations are made
- user recives a message after a booking attempt if the batteries were taken in the time between the path request and the actual booking process and makes the user to initiate another search
- implementing the maxumum automation of pathfinder process to minimize the second solution

Blocking database changes. Response scales with number of requests growing. Absolutely unnessesary in low request numbers and totally ineffecient for system performance in high request numbers.

Datachanges tracking would spot the problem as it appears under the calculation of the path. It will require implementation of a listener to data handlers events, or cache changes on a global level, outside the initiated application instance. It will be efficient for a very frequent accurence of the conflict, and will require a sufficient ammount of time to implement. The result will pay itself in the case of a very large user base, where several conflicts may accure more that once in a minute.

User re-initiation solution spots the problem on the booking of unavialiable batteries. It requires re-check of data stored in database before modifying the data. In the case of the conflict the user will recive a suggestion re-initiate the pathfinder request due to booking failure. Implementation of this solution is perofrmed on databse query level or on application level with transaction logic and is rated as a very low price implementation. This solution will be benificial only for very seldom cases of the conflic occurances.

Automation soltion minimizes the time between pathfinder request and booking request. It can be performed in two ways: one way will implement reservation with a short time of exparation if booking request is not recived and a straight booking. The implementation of this method arise a consern rising database queries, and therefore may become exreamly inofficient with a large user base. This solution fulfills most benefitially for user and the system at medium user base size.

Due to architecture of the system all three solutions are implementable. Currently, due to its user base aim, the system uses the re-initiation scenario. The system aimed at danish sitizen user base does not expect as many conflict to implement higher technology solutions due to their low price-to-value level.

Implementation

The system implements re-initiation solution at two levels: the application and database. The dedicated sql handler allows the system to perform suffusticated sql queries without altering the structure of the source. Therefore the system is able to execute multiple data modifications queries and recive an exception if any of the queries fail. This specific allows to catch an exception and role the request back and report to failure to the initiator.

Lets have a closer look at how it is implemented.

```
private string UpdateQuery(SqlData data)
{
    return String.Format(@"
        BEGIN TRY
             UPDATE {0} SET {1} WHERE {2}
                 IF @@ROWCOUNT = 0
             BEGIN  RAISERROR(-904, 10, 07) WITH LOG
             END
        END TRY
        BEGIN CATCH
                THROW 60000,'Row with {2} was not updated and I dont know why.', 1;
        END CATCH;",
        this.table, data.SqlPairs(), data.WhereClause
        );
}
```

This code shows a construction of an update query. It can take in any data, thanks to SqlData class, which is a part of SqlHandler molule_ and make an update on defined by modular database class condition. This method construcsts a string of sql query with a transaction logical statement, asking to update data only if the conditions given are met. If those conditions are not met and nothing was modified, database will return a critical error to the handler:

```
try
{
    command.CommandText = query;
    command.ExecuteNonQuery();

    // Attempt to commit the transaction.
    transaction.Commit();
    SqlCaching.Remove(this.table);
    this.Success = true;
}
catch (Exception commitEx)
{
        string exceptionType = String.Format("Commit Exception Type: {0}",
                commitEx.GetType());
```

```
            string exceptionMessage = String.Format("Message: {0}", commitEx.Message);
            this.AddResponse(commitEx.Message);
            // Attempt to roll back the transaction.
            try
            {
                transaction.Rollback();
            }
            catch (Exception rollbackEx)
            {
                this.AddResponse(rollbackEx.Message);
            }
        }
```

With this code the handler catches the response from database server on the query. The query can be multiple leveled, but if one of the levels throws the exception strong enough for the handler to consider it critical, it will stop the transaction and will try to roll back the changes that were made. All the exceptions and errors that appear during this operation are carefully stored in the responce object and are returned to the caller once the operation is complete.

This double-level precausing allows to catch and prevent a conflict on booking. However it is only efficient and tolerable at low user base level. Frequent requests of user to retry pathfinding may become user -unfriendly and thus lead to loss of users. The point at which this will happen shall be monitored and the relevant changes shall be implemented to the system in order to make the user explerience better.

Failure tolerance

This chapter takes a brief overview on possible fails that may appear in the system under deployement, utilisation and further developmnet. Errors and fails can accrue on several levels. In grand scheme of the whole system the fails may be separated on following categories: client side, application, database, platform and Hardware errors. Due to this projects goal not aiming the final deployement, this overvew does not cover the platform and the hardware failures.

CLIENT

Connection issues. Connection to service from client may be caused the missing internet connection, initial errors in configuration of the service connection, which is responsibility of the client developer, or firewall configuration declicning used by the server ports.

User input. User interface may return errors upon providing or requesting invalid data as well, as including authentication identification. Client developer takes a responsobility of providing final user with tools excluding, informing of, or masking those errors. The validity of the input is checked by application, thus does not require additional precausing.

Communication performance. Long responses and freeses of user interface are noticed upon testing, appearing the first connection with service. The developer of the client is taking responsibility of providing the separation of interface and connection processes to improve performace. Communication performance can as well be caused by server overload. In that case the deployment must be reconsidered.

Frequent concurrency cases. The final user may experience a frequent concurrency cases, when chosen path is taken when user attempts to book suggested batteries. With current release developers are suggested to provide user interface with tools to collect reports about such events. In later releases the tracking of such events may be implemented on server side and presented to the customer in form of statistics.

Datatype insufficency. The system is using wsHttp binding thus is based on SOAP 1.2 specification and shall be considered in client development.

Administration conflicts. Administrators may confront a problem of inability to delete booked batteries and locations to which those batteries are assigned. It is administration client developer responsibility to provde the administratior with information about this conflict. In further system upgrates there maybe implemented a configuation setting allowing to switch those restrictions on and off.

Authentication. Attempts to access data without authentication on manager and admin clients will end in no data responce from service. It is a responsobility of client developer to provide authentication interface and information about missing authentication for manager(if operated by human) and administration clients.

SERVICES

Authentication. Manager and admin services require authentication for the client to use the aviable functions of the services. All the errors are handled within the application and return the list with errors and their reasons to the client if such was not aqqured.[8]

Consistency. Final user service accepts session connections and stores temporary data within session and requires a consistent access to its functions. The service keeps a track of the data stored in the session and thus determines on which step of the process the connection stands. Any attempts to access the functions denied from that step will not have a response.

Empty objects. The services operatie with objects recived from client side therefore have to handle possible empty data before proceeding. The services automatically fill empty objects with default data without informing the client without the action.

BUSINESS

User input. The client input is not to be trusted therefore data validation is performed on Database Handler module level. The handler modifies data where it can to fit the requirements and passes the requiest with the modifications or silently denies it. Further data structure modifications would require extension of this handling.

Database connection. Stability and flexibility of the database connection is ensured by Database Handler module. Fail of connection would be cought and logged. The system will not provide any functionality on the failed database connection but will stay running until next request where attempt of establishing connection will be repeated.[9]

Database exceptions. All database exceptions are captured with dedicated tools, and can be logged in further development.

8 The functions is in development

9 Under implementation and testing

Pathfinder exceptions. Pathfinder returns empty path, if no connections were found. Case of not found connection may accure if 1) there is no suitable path, 2) inexisting stations id's were provided (returned graph was empty)

Logical erros. Due to it's structure the application stands a high risk of endless loops, especially in construction of modular objects, which have crossreferences with other objects, and shall be considered in futher development.

DATABASE

Syntax and names. The Database handler module is aimed at simplifying modification of modular level and its implementation with database. The module provides tools of automatic queries deneration and datatype recognition, requiring for it syncronization of modular class names with database objects.

Exported logic. The database model at its current state fails some basic integrated functions, which shall be handled ”manually” with application operations. The full list of operation requirements to be considered are presented In Appendix A.

EVALUATION

At the moment of the begining of this cource I was confident of majority of the subjects listed. I had a basic knowleadge of C# programming withing .NET framework, as well as extended knowledge of web producing and developing. This course helped me to understand new terms, such as distributed systems. The biggest struggle within these course was understanding of network. My brain refuses to understand its complexity but I have soaked the basics which shall give me a good base if I shall confront any tasks related. The biggest interest in this course for me was the study of graphs and algorythms, it was first time I have confronted the graphs concept, and it helped me to understand many problems that were earlier just a blur to me. I didnt learn much new about databases, I had a strong background before I started the course, but I have had fun playing with what I had, it was a good ballance to struggles with networks theory.

APPENDIX A: DATABASE

1. Operation requirements

Columns	Null	Default	Constrains	Insert/Update	Delete
Battery					
PK:id[int] , unique	NOT NULL	ai(1)		banned	
FK:location[int]	NOT NULL			Required exists(Location:id)	
replaced[date]	NOT NULL	getdate()		not required	
booking[guid]	NULL			not required	
Booking					
PK:id[guid], unique	NOT NULL	Ai (newid())		banned	
created[date]	NOT NULL	getdate()		not required	
Link					
PK,FK:location1[int]	NOT NULL		distinct: location2 pair(location1): unique	required distinct: location2 pair(location1): unique exists(Location:id)	
PK,FK:location2[int]	NOT NULL		distinct: location1 pair(location2): unique	required distinct: location1 pair(location2): unique exists(Location:id)	
distance[decimal]	NOT NULL			required	
time[time]	NOT NULL			required	
Location					
PK:id[int],unique	NOT NULL	ai(1)	Battery(location): *Delete[CASCADE]* *Update[CASCADE]* Link(location1): *Delete[DO NOTHING]* *Update[DO NOTHING]* Link(location2): *Delete[DO NOTHING]* *Update[DO NOTHING]*	banned	[Pre] manual cascade
name[string]	NULL			not required	
longitude[decimal]	NOT NULL			required	
latitude[decimal]	NOT NULL			required	
address[string]	NULL			not required	
FK: zipcode[int]	NULL			not required exists(Zip:iCode)	
Zip					
Code[int], unique	NOT NULL		Location(zipcode):	Required	

Columns	Null	Default	Constrains	Insert/Update	Delete
			Delete[SET NULL] *Update[CASCADE]*	unique	
City[string], unique	NOT NULL			Required unique	

APPENDIX B: STOCK OF ELECTRIFIED VEHICLES

1. Registration of all-electric vehicles in France by year (2010-2014)

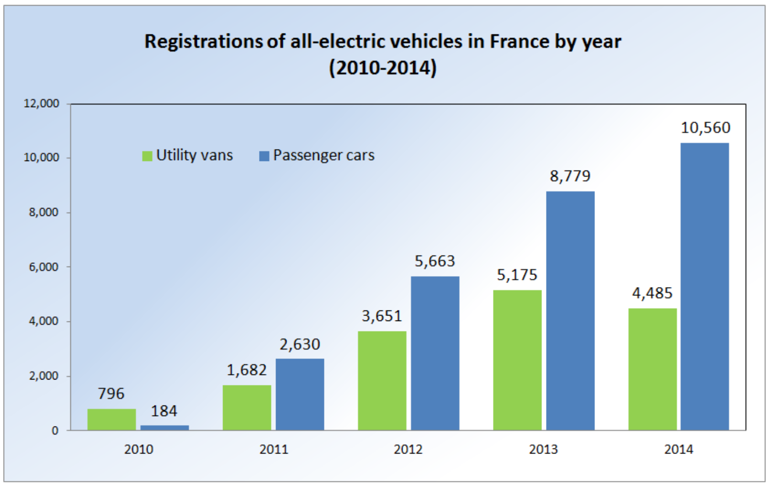

Illustration 9: Registration of all-electric vehicles in France by year

2. Sales of new electic vehicles in China by year (2011-2014)

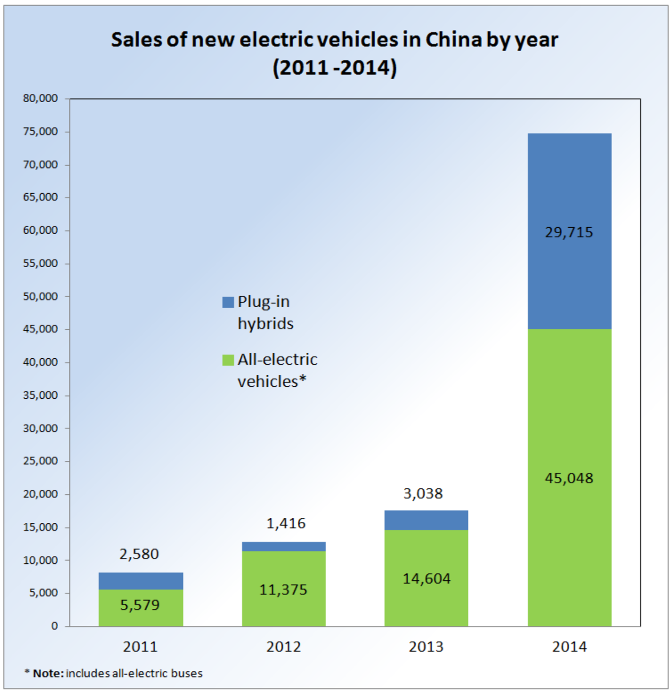

Illustration 10: Sales of new electic vehicles in China by year

3. Registration of plug-in electric cars in Germany by year (2010-2014)

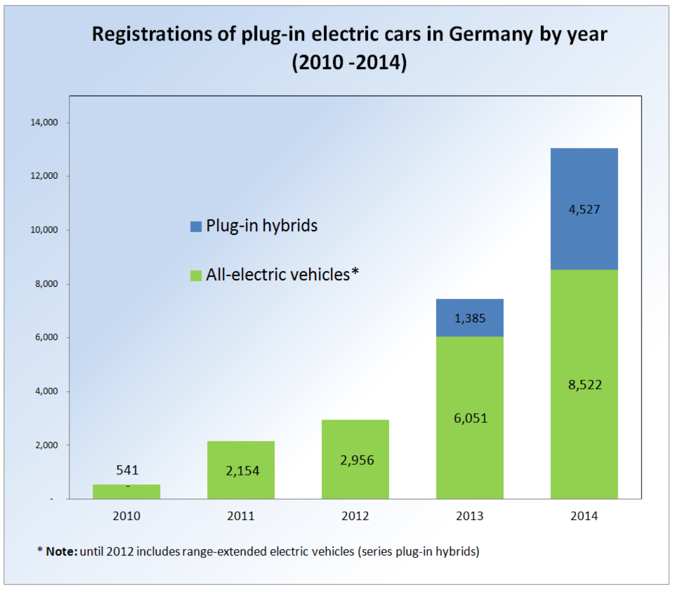

Illustration 11: Registration of plug-in electric cars in Germany by year (2010-2014)

4.Registration of plug-in electric vehicles in Sweden by year (2011-2014)

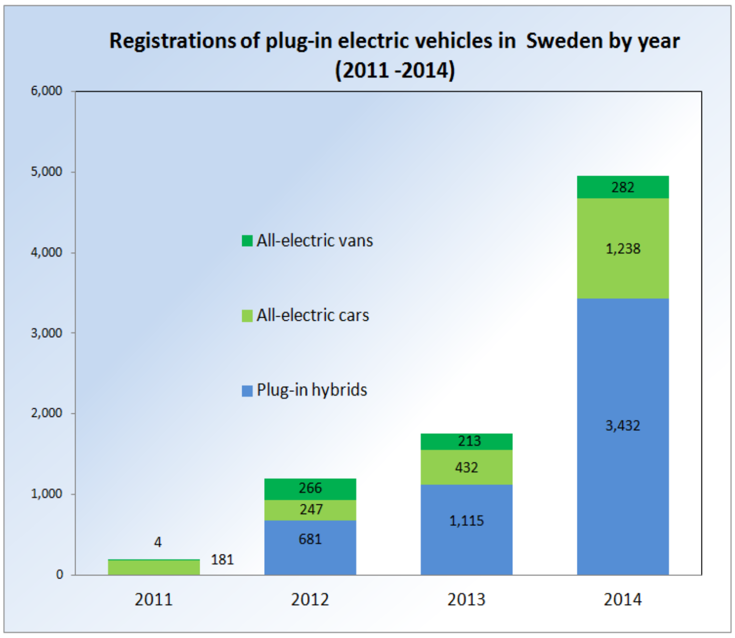

Illustration 12: Registration of plug-in electric vehicles in Sweden by year (2011-2014)

5. US cumulative sales of plug-in electric vehicles by year (2010-2014)

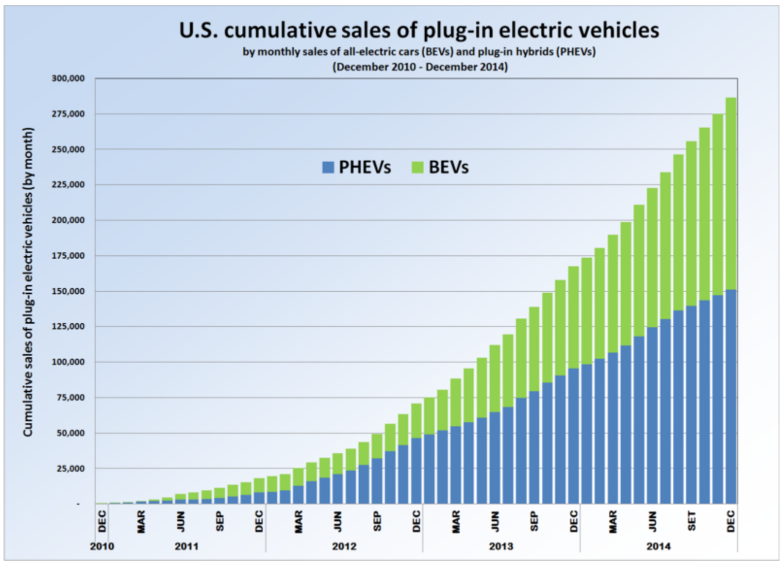

Illustration 13: US cumulative sales of plug-in electric vehicles

6. Sales and stock of electric cars in Denmark by year (2009-2014)

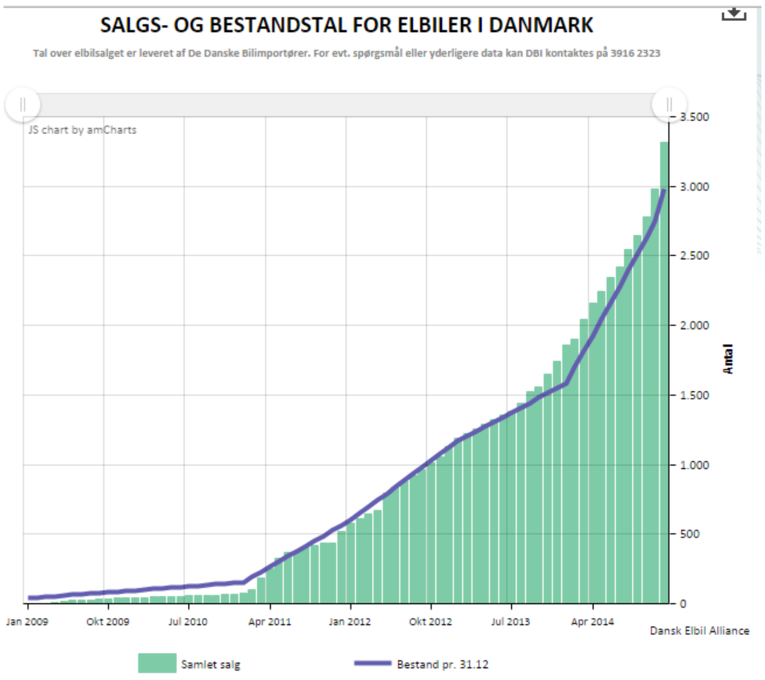

Illustration 14: Sales and stock of electric cars in Denmark by year (2009-2014)

APPENDIX C: MODELS

1. Final user scenarios

1.1 FINAL USER SCENARIO FOR SMALL SIZE USER BASE

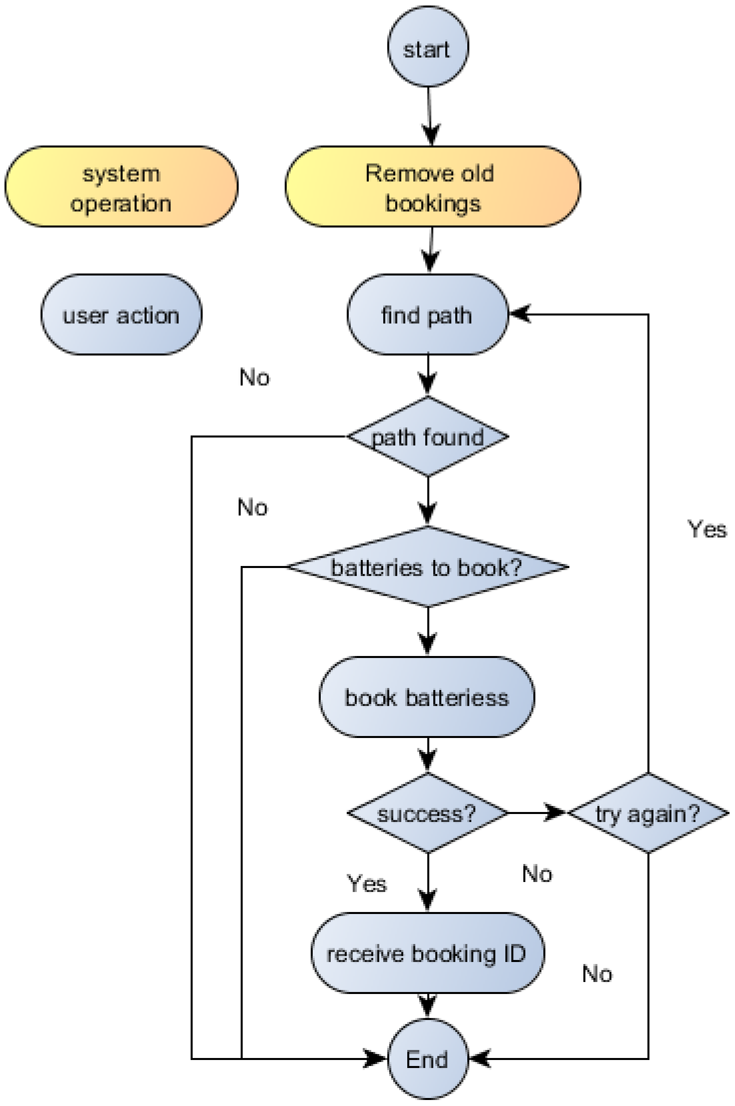

Illustration 15: Final user scenario for release

1.2 FINAL USER SCENARIO FOR MEDIUM SIZE USER BASE

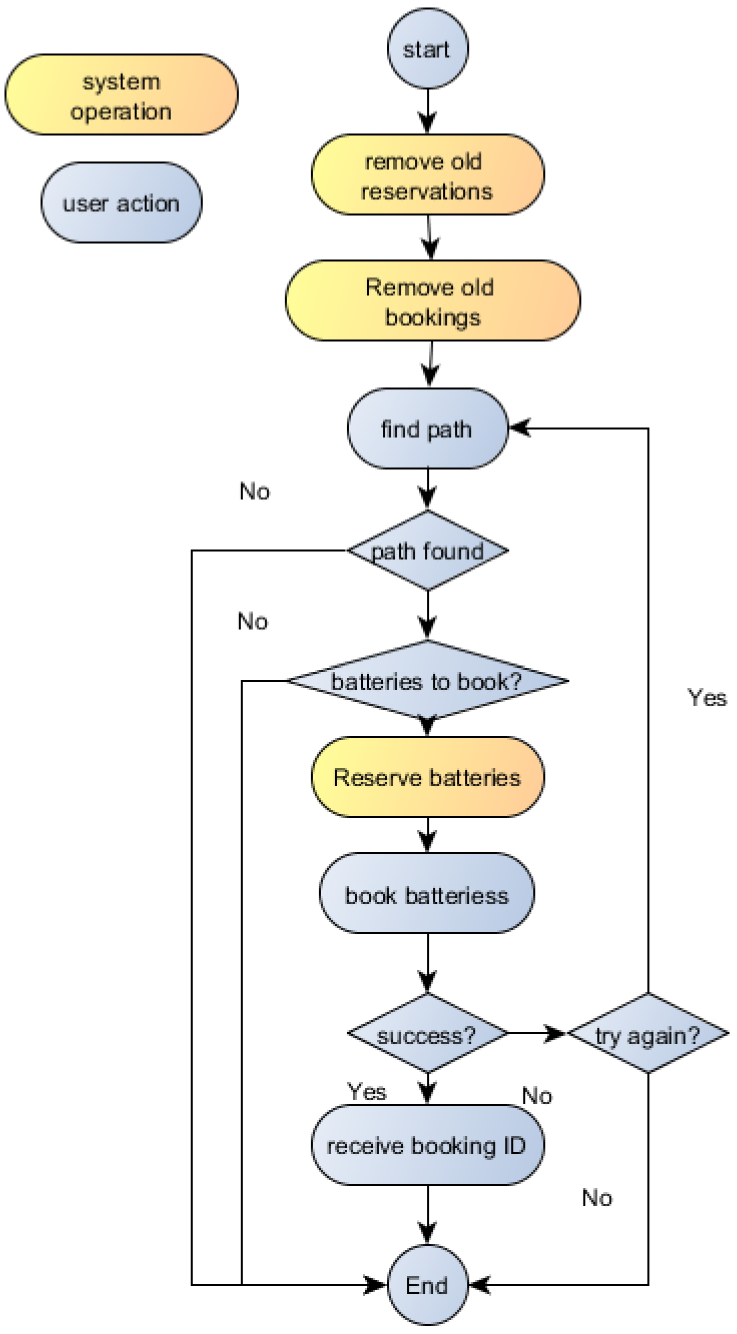

Illustration 16: Final user scenario for medium size user base

1.3 FINAL USER SCENARIO FOR LARGE SIZE USERBASE

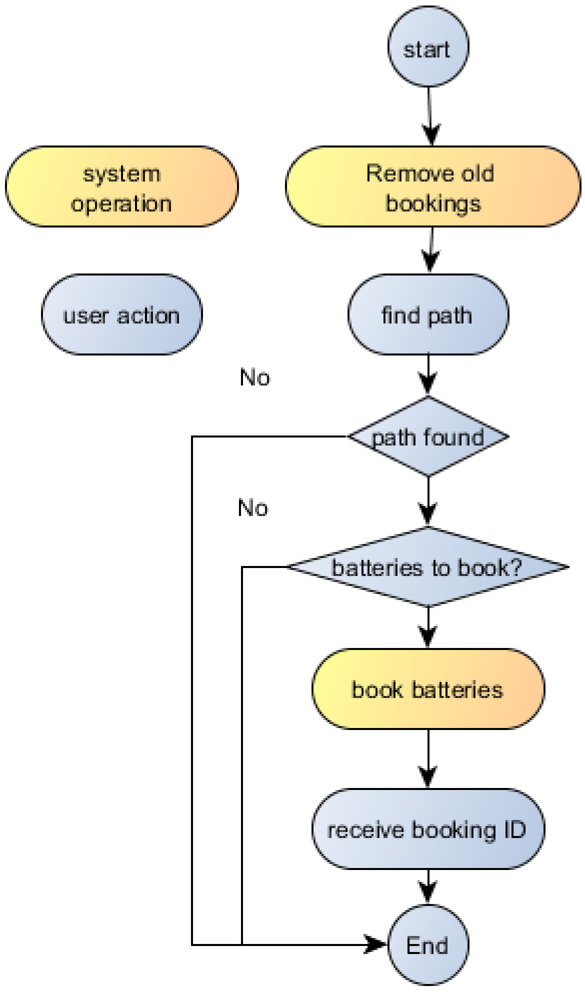

Illustration 17: Final user scenario for large size userbase

2. Modified Dijkstra Pathfinding Algorithm

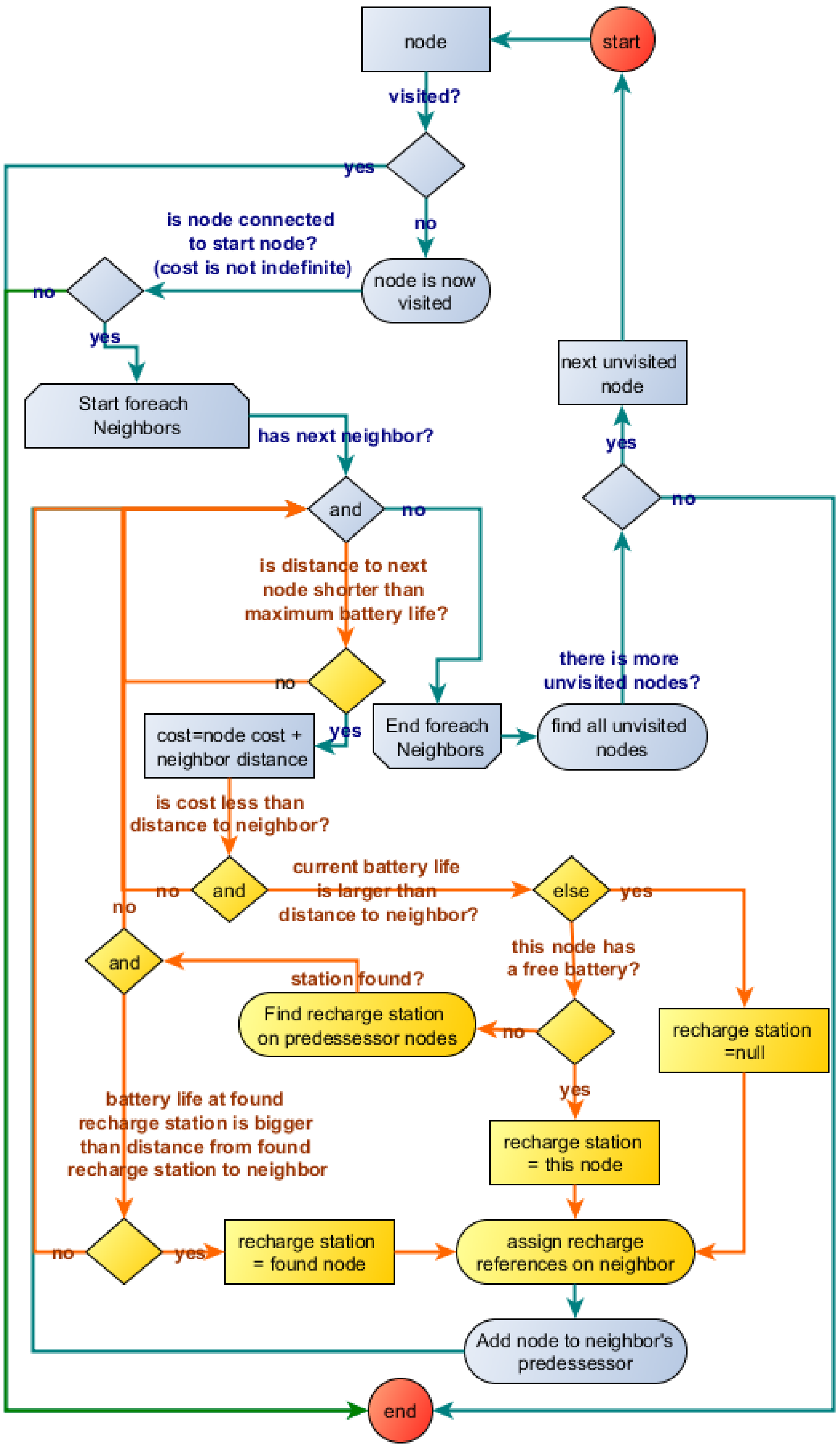
start
node
visited?
yes
no
node is now visited
is node connected to start node? (cost is not indefinite)
no
yes
Start foreach Neighbors
has next neighbor?
and
no
is distance to next node shorter than maximum battery life?
no
yes
cost=node cost + neighbor distance
End foreach Neighbors
find all unvisited nodes
there is more unvisited nodes?
yes
no
next unvisited node
is cost less than distance to neighbor?
and
no
current battery life is larger than distance to neighbor?
else
yes
no
and
station found?
this node has a free battery?
Find recharge station on predessessor nodes
no
yes
recharge station =null
recharge station = this node
battery life at found recharge station is bigger than distance from found recharge station to neighbor
no
yes
recharge station = found node
assign recharge references on neighbor
Add node to neighbor's predessessor
end

Illustration 18: Modified Dijkstra algorithm (Full size)

BIBLIOGRAPHY

[1] ——, ’Learning Goals’ (Article) <http://www.bigpicture.org/schools/learning-goals/> accessed 15 February 2015

[2] ——, ‘OSCOLA 2006 ‘(Standards Documentation) <https://www.law.ox.ac.uk/published/oscola/oscola_2006.pdf> accessed 15 February 2015

[3] Andrew Warfield, Yvonne Coady, Norm Hutchinson University of British Columbia , 2001, 'Identifying Open Problems in Distributed Systems ' (Article) <http://www.cs.ubc.ca/~andy/papers/ERSADS2001-UBC.pdf> accessed 21 May 2015

[4] Craig Larman, ”Applying UML and Patterns.Introduction to object-oriented analysis and design and Iterative development” 3rd Edition, 2012.

[5] Peter R. Egli, Windows Communication Foundation (Presentation) <http://www.slideshare.net/PeterREgli/windows-communication-foundation-wcf-11426496> accessed 25 May 2015.

[6] Saurabh Gupta, “Performance comparation in Windows Comminication Foundation (WCF) with existing Distributed Communication Technologies” (Article at Mictrosoft Developer Network Library, 2007) <https://msdn.microsoft.com/en-us/library/bb310550.aspx> accessed 25 May 2015.

[7] H. M. Deitel, P. J. Deitel, D. R. Choffnes, "Operating Systems", 3rd edition.

[8] H. M. Deitel, P. J. Deitel, ”Visual C#: How to programm“, 4th edition, 2010