Софийски университет
“СВ. КЛИМЕНТ ОХРИДСКИ”
Факултет по математика и информатика

Катедра “ИНФОРМАЦИОННИ ТЕХНОЛОГИИ”
Специализация “Разпределени системи и мобилни технологии”

Дипломна работа

Тема: **Интернет приложение за локализация и управление на динамична географска карта**

Дипломант:
Тихомир Здравков Лазаров
ФN: M21329

Дипломен ръководител:
доц. д-р Боян Бончев

София, 2005 г.

Съдържание

1. Въведение....5
1.1. Мотивация....5
1.2. Цел на дипломната работа....5
1.3. Структура и начин на представяне на изложението....6
2. Изисквания към системата и сравнение с подобни решения....7
2.1. Функционални изисквания към системата....7
2.2. Нефункционални (технически) изисквания към системата....8
2.3. Сравнение с подобни решения....8
3. Избор на технологични средства за проектиране на решението....11
3.1. J2SE....11
3.2. Java Servlets API....11
3.3. Избор на сървър за бази данни....11
3.4. HTTP. Уеб сървър....11
3.5. Клиенти....12
3.6. Среда за разработка на приложението....12
3.7. Тестове и внедряване....12
3.8. Изготвяне на техническа документация....12
4. Архитектура....13
4.1. Общо описание....13
4.2. Инструменти за инсталиране на географски карти и разпознаване на пътища…
14
4.3. Инструменти за поддръжка и разглеждане на географската карта....17
4.3.1. Потребителски интерфейс....17
4.3.2. Модел на работа....18
4.4. Обектно и физическо представяне на кокмпонентите в системата....23
4.4.1. Карта....23
4.4.2. Обекти....29
4.4.3. Граф от пътища....34
4.4.4. Комуникация в клиент-сървър среда....38
4.4.5. Достъп до базата от данни....40
4.4.6. Конфигурация....41
4.5. База от данни....42
4.5.1. Таблици, описващи географската карта, пътищата и обектите по нея….44
4.5.2. Таблици, описващи категориите в системата....46
4.5.3. Таблица за глобална конфигурация....46
4.5.4. Таблица за текущо регистрираните потребители-редактори....46
5. Алгоритми....47
5.1. Инсталиране на растерно приложение във вид подходящ за системата....47
5.2. Разпознаване на пътищата по картата....50
5.2.1. Намиране на точките, които се намират в центъра на пътя....53
5.2.2. Създаване на граф от точките, намиращи се в центъра на пътя....55
5.2.3. Алгоритъм на Дийкстра....59
5.3. Навигация и мащабиране по картата....61
5.4. Управление на обекти – добавяне, изтриване, премахване....62
5.5. Управление на събитията за обектите на картата....63
5.6. Търсене измежду обектите по картата....63
5.7. Намиране на най-кратък път между два обекта....64
6. Инсталация....65
7. Указания за работа....66

7.1. Инсталация на нова карта в системата.....66
7.2. Разпознаване на пътищата от инсталирана карта.....66
7.3. Работа с картата от гледна точка на краен потребител.....68
7.3.1. Навигация.....68
7.3.2. Търсене и локализация на обекти.....69
7.4. Управление на обектите по картата като потребител-редактор.....71
8. Тестове.....74
9. Възможни бъдещи подобрения.....76
10. Използвани материали.....77

Използвани фигури

Фиг. 1. Общ преглед на системата....13
Фиг. 2. Инструмент за добавяне на географска карта в системата....15
Фиг. 3. Инструмент за разпознаване на пътищата....16
Фиг. 4. Клиент-сървър модел на комуникацията в инструмента за поддръжка и работа с географската карта....20
Фиг. 5. Потребител-редактор и краен потребител....21
Фиг. 6. Сесията, съхраняваща информацията за състоянието на картата за всеки потребител....22
Фиг. 7. Функционалност на класа MappedImage....24
Фиг. 8. Схема на класа MappedImage и класовете, с които си взаимодейства....25
Фиг. 9. Класът Map и класовете, с които си взаимодейства....26
Фиг. 10. Динамичната карта и класовете, с които си взаимодейства....27
Фиг. 11. Опростена схема на йерархията на представянето на карта в системата....27
Фиг. 12. Област и структура на атрибутите й....30
Фиг. 13. Обект или област от картата с атрибути и събития....30
Фиг. 14. Разположение на области по картата....31
Фиг. 15. Областен мениджър....32
Фиг. 16. Структура и представяне на граф върху географската карта....34
Фиг. 17. Функция на класа DBRoadGraph....35
Фиг. 18. Взаимодействие между класовете описващи графа от пътища....36
Фиг. 19. Трансформация на данните през MapRequest....38
Фиг. 20. Класове за комуникация на клиента със сървъра....39
Фиг. 21. Взаимодействие между конфигурационният файл на уеб-сървъра (web.xml) с Config класа. Взаимодействие на класове от системата с Config класа....41
Фиг. 22. Класът lib.cfg.Config....41
Фиг. 23. Модел на базата от данни....42
Фиг. 24. Инсталиране на географската карта и трансформирането й във вид използван от системата....48
Фиг. 25. Потребителски интерфейс и стъпки за разпознаването на пътищата по картата....51
Фиг. 26. Намиране на точките от центровете на площите от пътя....53
Фиг. 27. Изходно изображение, чийто път трябва да бъде разпознат....57
Фиг. 28. Потребителят е посочил няколко точки от пътя и околностите на цветовете на тези точки са оцветени в отличителен цвят....58
Фиг. X. Процес на разпознаване на пътя: След прилагането на грубия филтър....58
Фиг. 29. Процес на разпознаване на пътя: След изчистване на шума....58
Фиг. 30. Резултат от разпознаването на пътя....59
Фиг. 31. Разпознаване на пътища: Навигиране по картата докато се намери подходящ изглед на пътя....66
Фиг. 32. Разпознаване на пътища: След селекция на няколко точки от пътя....67
Фиг. 33. Разпознаване на пътища: Резултат от разпознатите пътища....67
Фиг. 34. Навигация по картата....68
Фиг. 35. Информация за обект върху картата....69
Фиг. 36. Намиране на най-кратък път....70
Фиг. 37. Влизане в режим на редактиране....71
Фиг. 38. В готовност за редактиране на обекти....72
Фиг. 39. Добавяне на нов обект....73

1. Въведение

1.1. Мотивация

В наши дни сме свидетели на много продукти, които предоставят услуги, свързани с географски карти. Информацията относно обекти и пътища по картата е трудна за поддържка, тъй като в много от случаите картата представлява само едно растерно изображение, а векторизирането на информацията върху нея е сложна задача. Приложения, които решават проблеми от този тип са тежки и скъпи.

В множество от продуктите на пазара липсва възможност за преминаване от една карта в друга, или т. нар. наличие на под-карти, всяка от които е мащабируема карта.

Всички тези изброени фактори са мотивация за разработка на малко и високо функционално приложение, което решава тези проблеми.

1.2. Цел на дипломната работа

Текущата работа реализира уеб-базирано приложение, което дава възможност за навигация и мащабиране на интерактивна и динамична географска карта и локализиране на обекти върху нея.

Потребителят разполага с удобно грфично приложение, с което навигира върху картата и разглежда обектите. Поддръжката на обектите по картата се извършва от потребител-редактор през административно графично приложение. В настоящата работа е използвана географска карта на България.

Важен принос на текущата работа е да улесни въвеждането на информация за локализацията на пътищата по географска карта като предлага универсален начин за разпознаването им и асоцииране им към дадена карта.

Предимство на разработката е сравнително малкият обем на кода и лесната инсталация. Текущата версия поддържа съхраняването в системата на неограничен брой карти с неограничен брой обекти по тях и теоретически произволен брой нива на мащабиране. Създава възможност за преминаване от една карта в друга (т.нар. под-карти). По този начин лесно може да се създаде верига от различни карти, всяка една от които е с висока детайлност – карта на света с под-карта, която е карта на континент, след това карта на държава, карта на град, карта на квартал, карта на сграда, карта на стая; и всяка една от тези може да бъде обогатена с наличието на

обекти по нея.

1.3. Структура и начин на представяне на изложението

Изложението на дипломната работа следва описание на системата, започвайки от общи данни и стигайки до детайлен преглед на алгоритми, използвани в нея.

Следващите няколко глави запознават читателя с допълнителна информация относно подобни продукти и сравнителна характеристика. Споменати са подробно средствата, използвани за изготвянето на настоящата работа.

Следва описание на архитектурата на компонентите в системата и как те си взаимодействат. Информацията е подкрепена с графични фигури. Архитектурата е разгледана в последователност отгоре-надолу – подсистеми, йерархия на обекти във всяка една от тях, комуникация между обекти, начин на записване на информацията за обектите, структура и дизайн на базата данни.

След запознаване със системата се отделя по-специално внимание на това как точно е реализирана работата на отделни компоненти на ниво алгоритъм.

Следват инструкции за инсталация и конфигуриране на системата. Вървейки в логическата последователност, по-нататък са описани инструкции за работа със системата и инструментите в нея. Добавен е план на тестване на продукта.

Тъй като настоящата тема е богата на идеи за по-нататъшно развитие, авторът е отбелязал какви подобрения могат да бъдат направени в бъдеще. Накрая е отбелязан списък с използвани материали.

2. Изисквания към системата и сравнение с подобни решения

2.1. Функционални изисквания към системата

1. Възможност за инсталация на неограничен брой географски карти, които са растерни изображения в PNG, JPG, GIF формат.
2. Навигация и мащабиране на географската карта. Навигацията трябва да бъде възможна както с щракане на бутони за посока, така и с щракане с мишката върху съответна област. Мащабирането трябва да може да става на последователни стъпки, както и с посочване на произволно ниво на мащабиране.
3. Добавяне, редактиране, премахване на обекти от картата, всеки от които съдържа детайлна информация, включваща име, описание, положение върху картата, уеб адрес (който крайният потребител може да достигне чрез щракване върху обекта на картата), категория, географска карта (под-карта), към която да насочват. Категорията се визуализира чрез икона, предварително предназначена за избраната категория.
4. Редактирането на обекти и публикуването промените по картата трябва да може да бъде извършено само от потребител-редактор, който е оторизиран чрез име и парола.
5. Възможност не само за мащабиране на текущата карта, но и разглеждане на подробни под-карти на градове в страната, върху които също трябва да може да се локализират обекти. Възможност за неограничено влагане на карти една в друга.
6. Предоставяне на възможност за търсене на обекти по картата. Резултатът от търсенето е автоматична локализация на намерените обекти. Търсенето се осъществява по един от следните критерии или по всички тях:
    1. текст, които трябва да се съдържа в името, описанието или уеб адреса
    2. категория
7. Визуализиране на информация за обекти по картата в създадения потребителски интерфейс
8. Възможност при действие върху обект от картата да се отвори прозорец на уеб-браузър, в който да се зареди уеб-адреса, който е приложен за обекта
9. Разпознаване на пътища по зададени цветове по време на инсталационния процес на географската карта
10. Възможност една карта да има няколко категории от пътища (напр. първокласни,

второкласни)

11. Търсене и визуализиране на оптимален път между обекти върху картата
12. Възможност за посочване на приблизителна дължина в "км" на най-краткия път между два обекта
13. Публикуване на добавената или редактираната информация от потребителя-редактор на адрес, видим за външните потребители

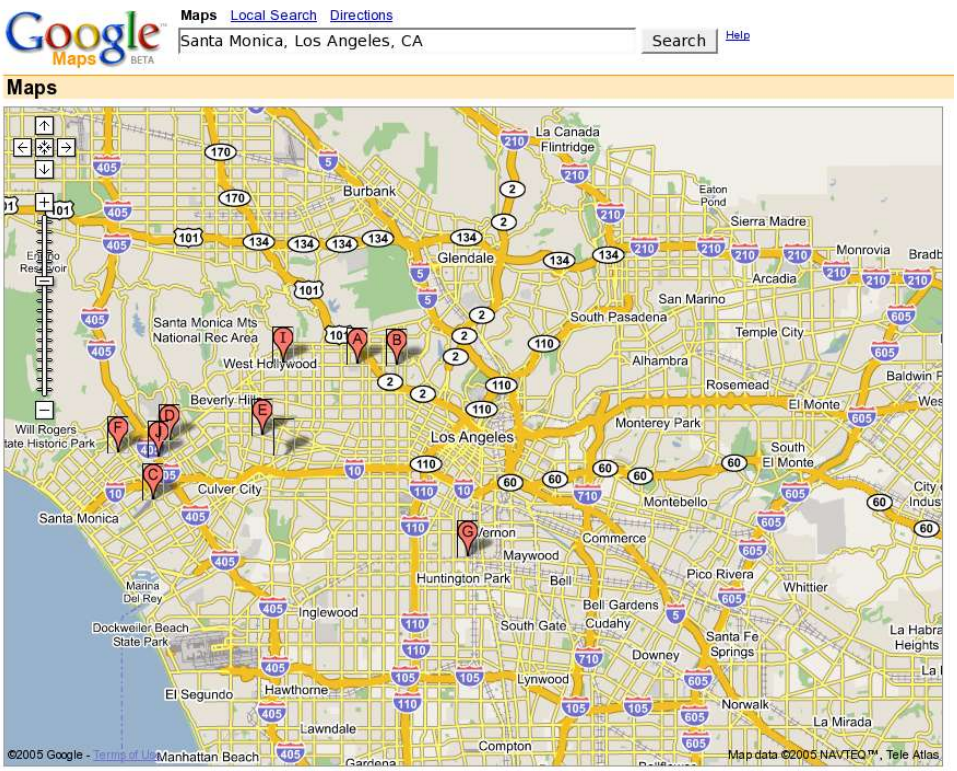
2.2. Нефункционални (технически) изисквания към системата

1. Системата трябва да бъде лесна за инсталация и конфигуриране
2. Добре организирани графични ресурси, които са използвани в нея
3. Оптимална скорост на действие
4. Възможност за разширяване на действията, които могат да се извършват върху обектите
5. Възможност за глобална конфигурация на системата
6. Възможност за смяна на големината на визуализиращия прозорец само през HTML файловете или през глобалната конфигурация
7. Удобен и лесно разбираем потребителски интерфейс, който умее да се предпази и да предпази системата от грешки на потребителя
8. Оптимално ползване на оперативна памет независимо от големината на географската карта

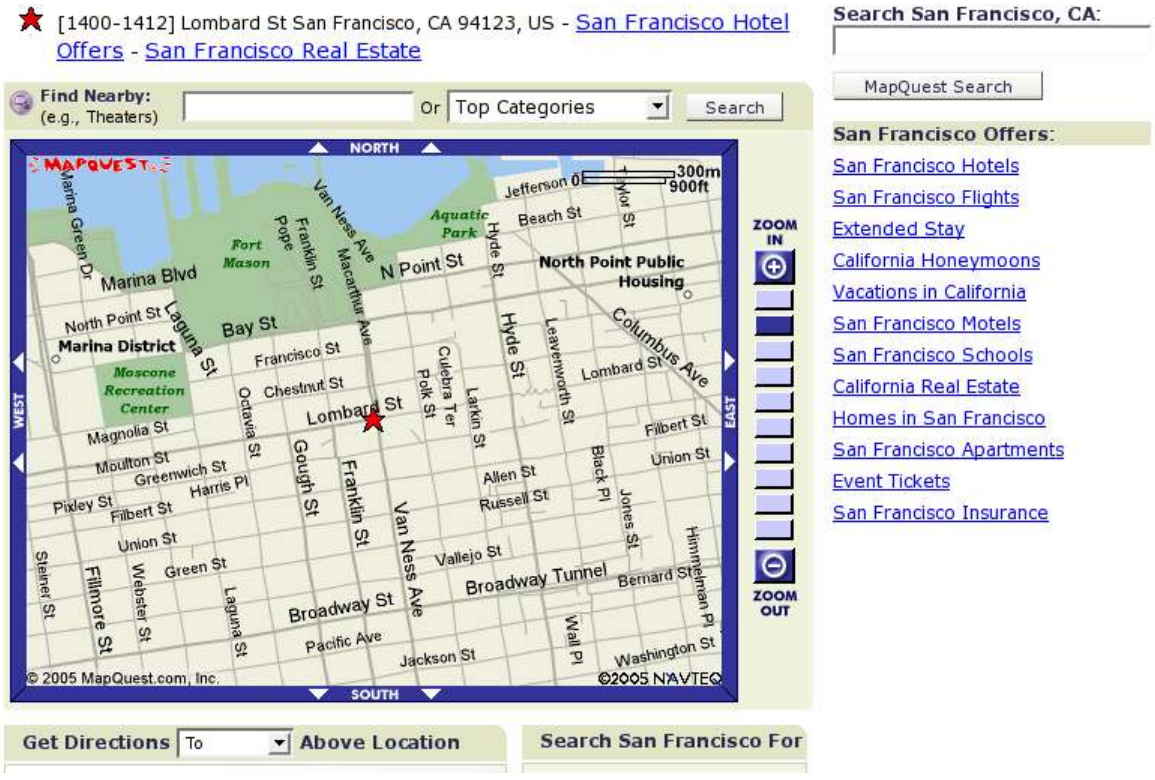
2.3. Сравнение с подобни решения

В момента на пазара на информационните технологии започват да излизат качествени продукти, които предлагат услуги за работа с географска карта. Ще се спра накратко върху сравнителна характеристика с два от най-нашумелите продукти – Google Maps и MapQuest.

Google Maps (http://maps.google.com) – интерактивни карти, които могат да се гледат в два режима – сателитни или опростени. Възможност за търсене на най-кратък път между две точки и детайлно описание на пътя. Потробна информация за улици, пътища, магистрали. Удобна навигация и добър потребителски интерфейс.



MapQuest (http://mapquest.com) – интерактивни карти. Търсене на пътища. Подробно описание на пътя между две точки. Удобна навигация.



Функционалност	***GoogleMaps***	***MapQuest***	***Настоящата работа***
Навигация	Да	Да	Да
Мащабиране с 1 стъпка	Да	Да	Да
Мащабиране на произволно ниво	Да	Да	Да
Мащабиране чрез щракване върху картата	Не	Да (само еднопосочно)	Да – мащабиране в двете посоки
Подробни под-карти на участъци	Не	Не	Да
Автоматизация на разпознаването на пътищата	Няма данни	Няма данни	Да
Java клиент	Не	Не	Да
HTML клиент	Да	Да	Не
Показване на версия за отпечатване	Да	Да	Не
Намиране на най-кратък път между две точки	Да	Да	Да
Текстово описание на път между две точки	Да	Да	Не
Търсене на обекти	Да	Да	Да
Възможност за посещение на уеб-адрес за обект	Да	Не	Да
Филтър по категории от обекти	Да	Да	Да

3. Избор на технологични средства за проектиране на решението

3.1. J2SE

Езикът, който е използван за изработването на софтуерното приложение за текущтата дипломна работа е Java. Платформата, която е използвана е J2SE (Java 2 Standard Edition). Приложенията, които са реализирани са няколко типа – аплети (клиенти), конзолни приложения и сървлети (уеб услуги). Графичните контроли в клиентските приложения са базирани на Swing технологията, която придава по-добър естетически вид на приложението и дава възможност за много подобрения в изгледа на контролите. Инсталацията на клиентските приложения е лесна и самите те са компактни.

3.2. Java Servlets API

Настоящата работа е базирана на клиент-сървър комуникационен модел. Услугите, които се предлагат от сървъра са изработени, използвайки Java Servlets технология и предлага богат набор от функции за комуникация в мрежа. Конфигурацията и инсталацията на сървлетите е лесна и имат добра скорост на изпълнение.

3.3. Избор на сървър за бази данни

Информацията се складира в база от данни MySQL 4.0. Не е използван XML поради трудността на поддръжка и възможност за много грешки при ръчно редактиране на файловете, докато при базата данни може да се подсигури цялост на данните чрез ограничения, указани към структурата на таблиците (уникални ключове, типове данни). За текущите цели и езикът, на който е базирана работата е използвана библиотека за работа с бази данни MySQL Java Connector 3.1.8.

3.4. HTTP. Уеб сървър

Протоколът, който е използван за комуникация между клиента и сървъра е HTTP. Предимството на този протокол се състои в множеството богати програмни класове, които предоставят лесна работа с данните, които се изпращат и приемат. Не трябва да се пренебрегва фактът, че поддръжката на сесия в сървърната част се реализира най-лесно с предаване на т. нар. cookie.

Формат на данните, с които се извършва комуникацията са HTTP форматирани параметри или двоичен поток от данни. Използвана е възможността,

която предлага езика за сериализиране на данни от сървлета към аплета. По този начин програмистът не се обременява с използването на допълнителни формати за комуникация като XML, но има възможност да се интегрира с такъв вид данни, което е добра опция за интегрирането с клиентски приложения, написани на други езици.

Уеб-сървърът, който е използван е Apache Tomcat 5.0.25, поради лесната му интеграция с клиент-сървър приложения, реализирани на Java.

3.5. Клиенти

Клиентите са аплети, които се стартират в уеб-браузър. За целта са използавни два от най-разпространените браузъри – Internet Explorer, Mozilla Firefox.

3.6. Среда за разработка на приложението

Средата за разработка е Eclipse 2.1, който е много удобен продукт за създаване на Java приложения, интегрирани в посочения уеб сървър.

3.7. Тестове и внедряване

Приложението е тествано и внедрено в Linux (Slackware 10.1) и Windows операционни системи. Внедряването му протече без особени проблеми, поради портабилността на използвания език за програмиране.

3.8. Изготвяне на техническа документация

Настоящата документация е подготвена с OpenOffice 1.1.4; софтуер за диаграми и графики – Kivio; Poseidon for UML Community Edition – UML диаграми; дизайн на базата от данни – DB Designer 4.

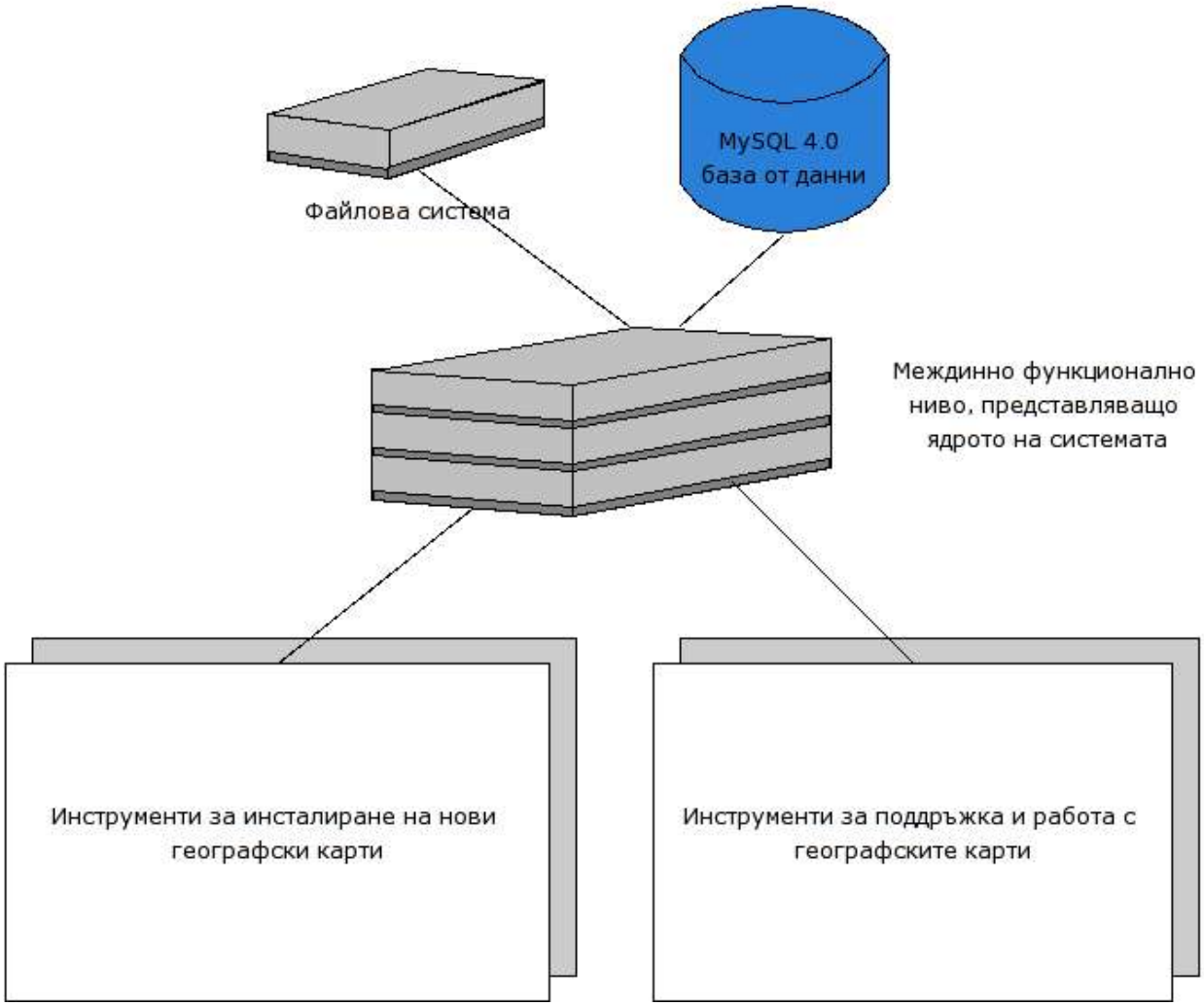
4. Архитектура

4.1. Общо описание

Приложението се състои от няколко елемента, всеки от които е използван в даден момент от "живота" на една географска карта, която е в системата, а именно: инструменти за инсталиране на нови географски карти в системата и инструменти за поддръжка и работа с картите. Допирната точка до тези две групи е базата от данни, където се складира информацията относно картите (фиг. 1) и файловата система.

Тези две групи от инструменти не взаимодействат директно с базата от данни и файловата система, а използват предлагана функционалност, която скрива детайлите по складирането на информацията от инструментите.

Фиг. 1. Общ преглед на системата



4.2. Инструменти за инсталиране на географски карти и разпознаване на пътища

Инструментите за инсталиране на нови географски карти са два:

1. Инструмент, който добавя нова карта в системата и записва информацията за нея в базата данни
2. Инструмент, който се занимава с разпознаването на пътищата върху картата и записването им в базата данни

Всяка карта може да бъде разглеждана от крайния потребител след като бъде само добавена в системата, използвайки първия инструмент. Информацията за разпознатите пътища е допълнение към функционалността на картата. То се извършва по решение на потребителя редактор, който инсталира картата. Процесът по разпознаване може да се извърши многократно във времето на "живота" на една инсталирана географска карта.

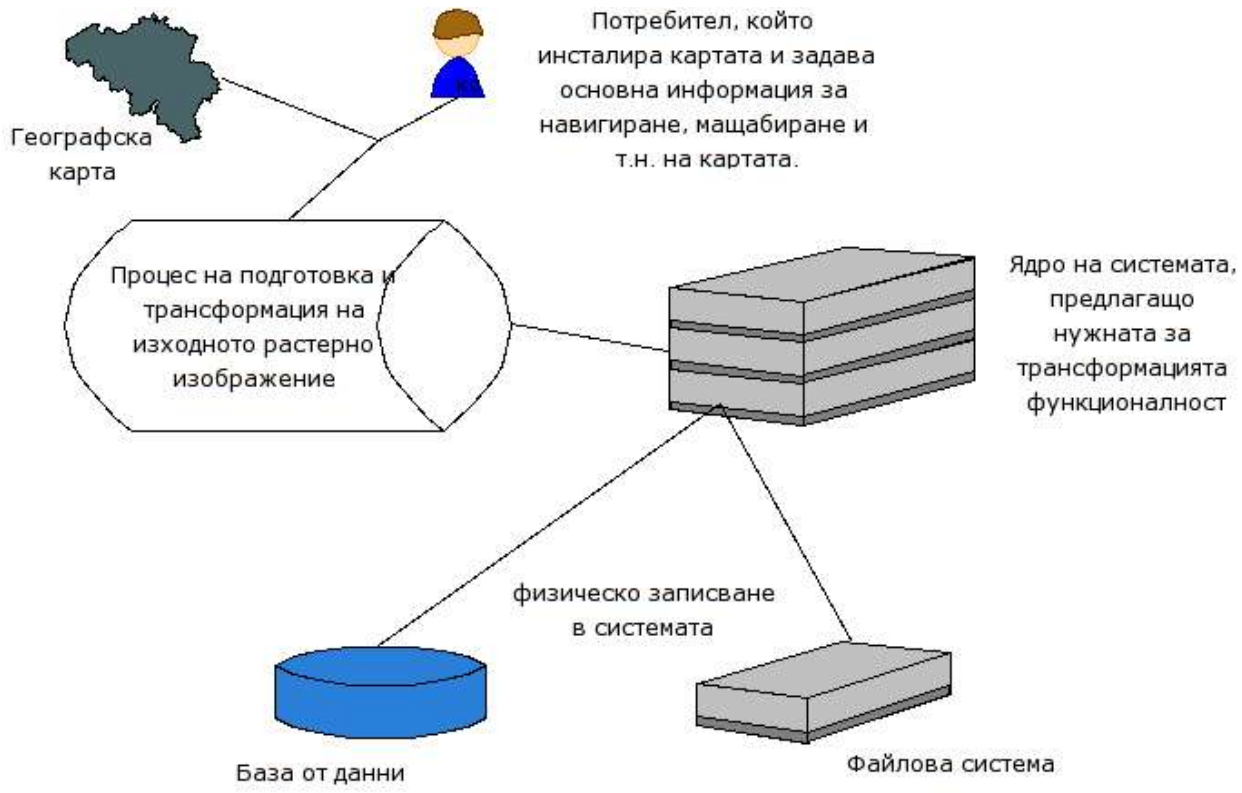
Добавянето на карта в системата се извършва чрез Java конзолно приложение, което на вход очаква информация относно:

- пътя до оригиналното растерно изображение на картата, която ще се добави в системата;
- брой нива на мащабиране (най-голямото ниво на мащабиране мащабът на оригиналната карта, а най-малкото е "най-отдалеченият" изглед на картата. Потребителят-редактор трябва да съобрази броят на нива на мащабиране с големината на прозореца на графичното приложение, което ще визуализира картата. Размерите на най-отдалеченят изглед се пресмятат като размерите на оригиналната карта се разделят на броя на нива на мащабиране. В такъв случай броят на нива на мащабиране трябва да е такъв, че размерът на най-отдалеченият изглед да не е по-малък от размера на визуализиращия прозорец);
- име на картата, с което тя ще се визуализира на потребителите;
- кратко описание на картата;
- мащаб – целочислено разстояние в метри, което отговаря на 1 пиксел от оригиналния размер на картата;
- флаг дали картата да бъде избрана като основната, карта, която ще се визуализира по премълчаване;
- информация за базата от данни – сървър, където се намира базата; потребител за достъп до базата – име и парола; име на базата от данни;

- местоположение на физическата файлова система, където да се разположат обработени части от подадената на вход оригиналната географска карта;

Резултатът от изпълнението на програмата е инсталирана географска карта, която предоставя пълна функционалност за мащабиране, навигация и поставяне на обекти по нея (фиг. 2).

Фиг. 2. Инструмент за добавяне на географска карта в системата

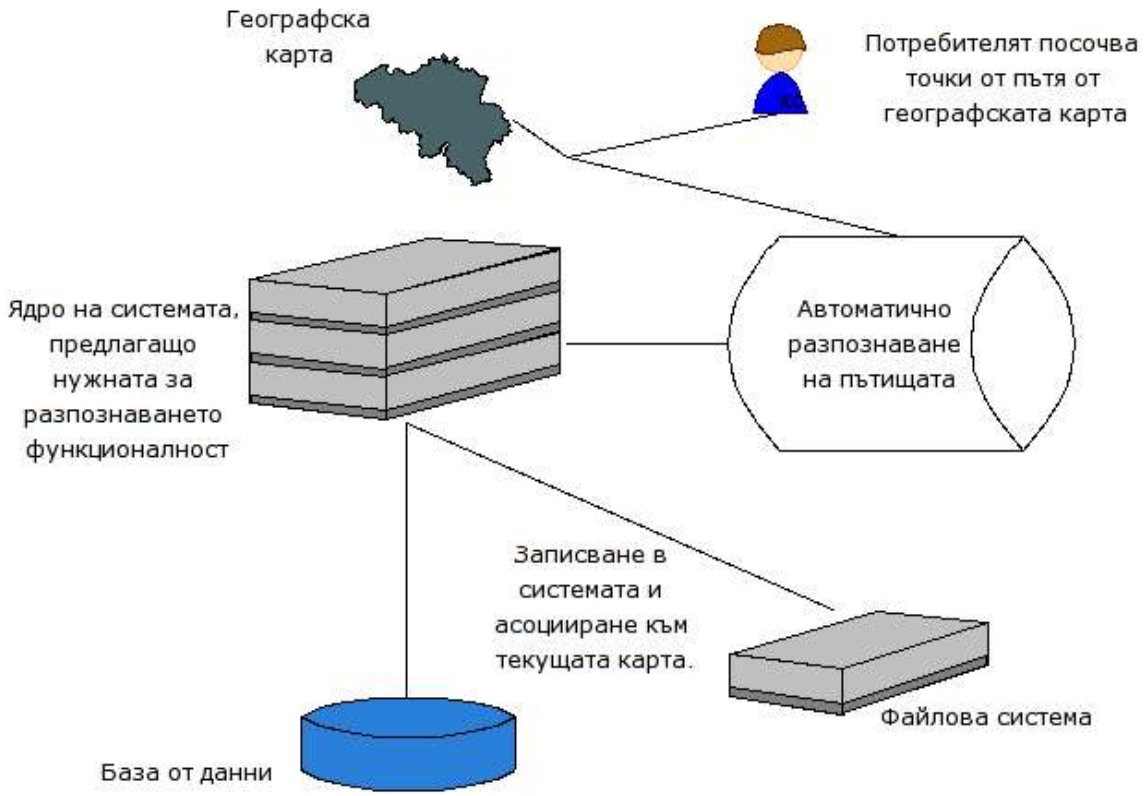


Разпознаването на пътищата по картата е полу-автоматично. Процесът на разпознаване очаква потребителски вход от данни. Тези данни представляват посочени точки от един или няколко различни пътища върху картата (точки от растерното изображение). Използвайки тази информация, инструментът за разпознаване на пътища създава граф на всички пътища, които удовлетворяват въведените от потребителя данни.

Този инструмент представлява графично приложение, което дава възможност на потребителя да избере карта, чиито пътища трябва да бъдат разпознати. Приложението позволява навигация и мащабиране на картата, за да може да се видят

детайлно кои точки принадлежат от пътищата. Потребителят посочва чрез щракване на мишката точки от текущата видима част от картата, които принадлежат на пътища. Следващата стъпка е предварителен резултат от това, което е въведено като данни от потребителя, т.е. визуализира се граф от пътищата върху текущата видима част от изображението. Ако резултатът е одобрен се пристъпва към следващия момент – прилагането на разпознавателния алгоритъм върху цялата карта и публикуването на резултатите в базата от данни. Инструментът използва изцяло предоставена му функционалност от ядрото на системата, а самият той само предлага удобен за потребителя интерфейс за взаимодействие с картата (фиг. 3).

Фиг. 3. Инструмент за разпознаване на пътищата



4.3. Инструменти за поддръжка и разглеждане на географската карта

4.3.1. Потребителски интерфейс

Поддръжката на географската карта е процес на управление на обектите по картата – добавяне, редактиране, премахване и публикуването на промените, т.е. записването на промение в системата. Поддръжката се извършва от потребители-редактори, който разполагат с уникална комбинация от име и парола, чрез които се оторизират от системата за работа по редакция на картата.

Разглеждане с картата наричаме действията, които включват навигация, мащабиране, търсене, отваряне на под-карти.

Тъй като поддръжката и разглеждането на картата са тясно свързани операции от гледна точка на потребителя (и потребителя-редактор), то за него те изглеждат като едно приложение. То дава възможност за влизане в режим на редакция на картата при указване на валидна комбинация от име и парола.

Потребителят-редактор разполага с цялата функционалност, с която разполага и крайният потребител, като към нея е добавена и административна част.

Контролите, които са достъпни за разглеждане на картата са посоки за навигация върху картата – наляво, надясно, нагоре, надолу, както и контроли за мащабиране на картата – по-голям, по-малък или произволен мащаб. Потребителят може да избира картата, която да разглежда в момента, избирайки от предоставен му списък с текущите такива в системата.

Друга възможност, която се предлага на крайния потребител е търсене. Търсенето се прилага върху обектите по картата. То се извършва по два критерия – въведен текст и категория. Резултатът от търсенето е текущата видима част от картата с изобразени върху нея обектите, които отговарят на зададените критерии.

Важна функционалност в търсенето е намирането на най-кратък път между два обекта. След като потребителят посочи обектите, между които трябва да се намери въпросния път се изпраща заявка до сървъра и резултатът е растерно изображение с видимата част на картата с визуализиран отличително най-кратък път между двата обекта.

Потребителят-редактор може да влезе в режим на редактиране на картата след въвеждане на име и парола, с които е регистриран в системата. Той може да

навигира върху картата, също както крайния потребител и в зависимост от работата, която трябва да извърши по картата – добавяне, редактиране, премахване на обекти - да намери място върху картата, където да приложи това действие. Предварително той указва режим на работа, в който ще работи – добавяне, редактиране, премахване на обекти. В съответствие с режима на работа графичното приложение се държи по съответен начин.

Промените, които се правят върху картата не се отразяват върху това, което вижда външният потребител докато не се публикуват от потребителя-редактор и докато външният потребител не зареди наново променената в системата карта. *Не е възможна едновременна работа на двама потребители-редактори върху една карта*.

4.3.2. Модел на работа

Процесът на поддръжка и работа с географската карта е реализиран като клиент-сървър модел. Клиентът е Java аплет, който предлага графичен интерфейс на потребителя да извършва манипулации с картата. Самата обработка на данните по картата и управлението й се извършват на отдалечен компютър, които връща резултати от действията на потребителя обратно на клиентското приложение. То само визуализира тези резултати.

Сървърната част се изпълнява от Java сървлети, които работят върху картата с функционалността предлагана от ядрото на системата. Те приемат заявки по HTTP протокол и отговарят по същия протокол. Данните, които идват от клиента са кодирани в UTF-8 формат. Резултатът от работата на сървлетите е или поток от байтове, които представляват растерно графично изображение или поток от данни, които представят сериализиран Java обект, който съдържа допълнителна информация за картата, за обектите по нея и за състоянието й в момента.

За всеки потребител, който работи с географска карта се пази сесия на сървъра, която съдържа състоянието на текущо разглежданата карта. В състоянитето на картата се пази и указател към инстанция на обект, който осигурява връзката до базата от данни. Всички сървлети, които представляват услугите на сървъра наследяват един базов, който се грижи за създаване на нова сесия, ако потребителят няма такава и за запазването на текущото състояние на картата там, както и за извличането на такова в случай, че вече има създадена сесия. По този начин за всеки потребител се пази само една постоянна връзка към базата от данни, докато сесията

стане невалидна.

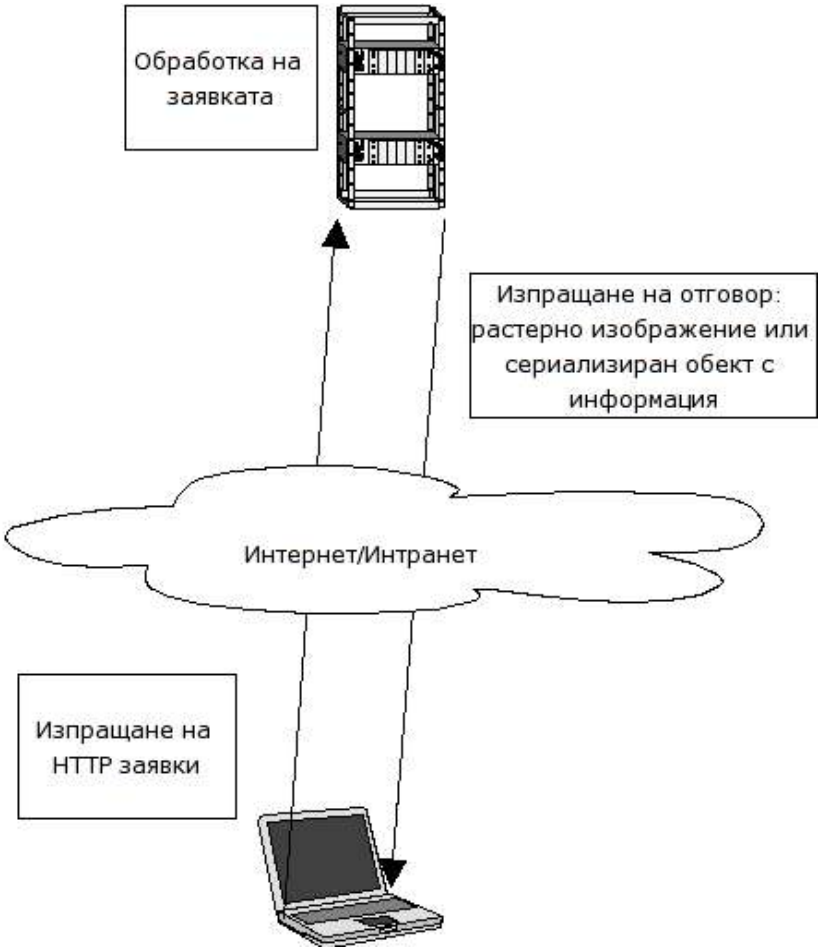
Достъпът до сесията е по уникален ключ, който се предава между клиента и сървъра във вид на т. нар. бисквитка (cookie). При начален достъп до сървъра се създава сесия и се праща ключа към нея до потребителя във вид на cookie. При действие от страна на клиента той го връща обратно на сървъра заедно с информация за съответното действие от негова страна.

Сесията на сървъра пази само информация за състоянието на картата, но не и самото растерно изображение в паметта, тъй като това би изтощило паметта на сървъра в случай, че има много потребители. Състоянието на картата се определя от няколко фактора:

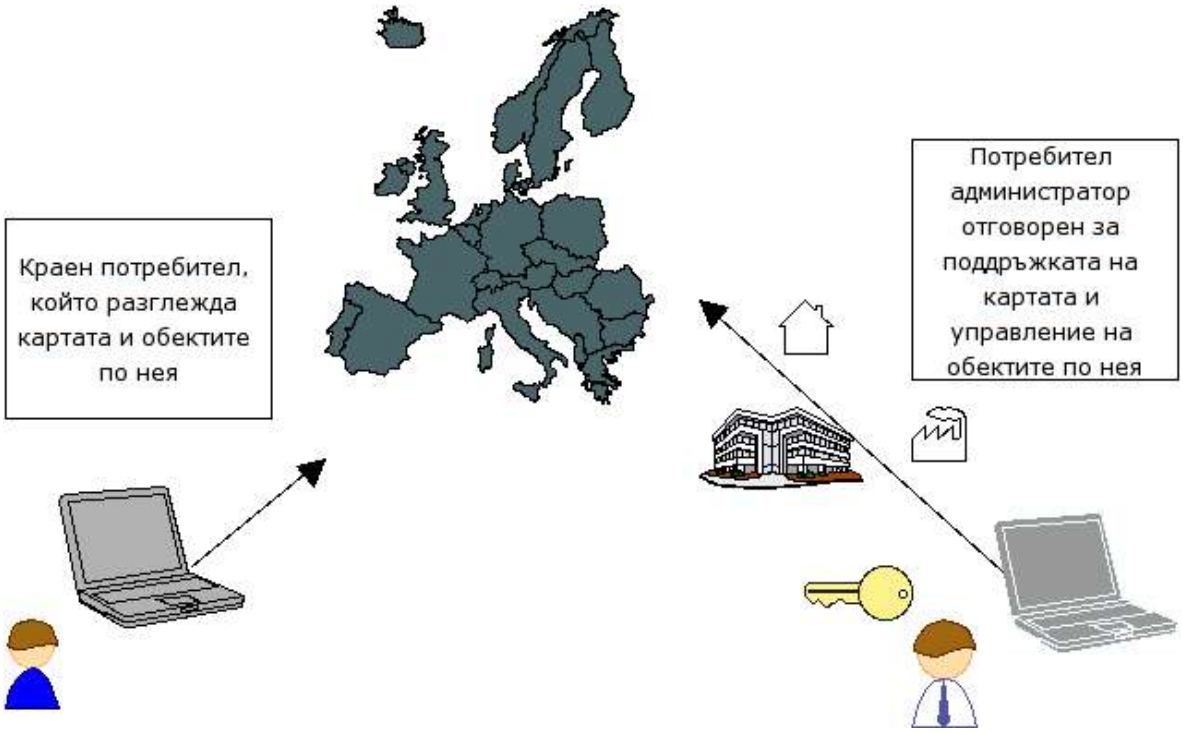
1. Позицията на правоъгълна част от карата, която е в момента видима за потребителя
2. Граф от точки за всички разпознати пътища по картата
3. Информация за всички обекти по картата – позиция, име, описание, уеб адрес и др.

От информацията, която се складира в сесията на сървъра най-обемни са данните за всички обекти по картата. Позицията на видимата правоъгълна част от картата е 4 числа (да речем 32 битови цели числа). Графът от точки представлява множество от точки, всяка от които има позиция (2 числа) и списък от индекси, към точки, с които е свързана. Обикновено една точка е свързана с две други. Ясно е, че тази информация при 10,000 точки е грубо пресметнато 50 килобайта, затова можем да си позволим да я държим в сесията. Информацията за обектите по картата също не е толкова голяма, че да не можем да си позволим да я държим в паметта. Информацита за един обект (име, описание, уеб-адрес, положение върху картата, категория) е в редки случаи 200 байта. Ако имаме 1000 обекта на картата, то паметта, която ще бъде заета от тази информация ще бъде само 200 килобайта. Разбира се, в тези калкулации не се взема предвид паметта, която ще бъде заделена от самата виртуална машина на Java за представяне на тези данни.

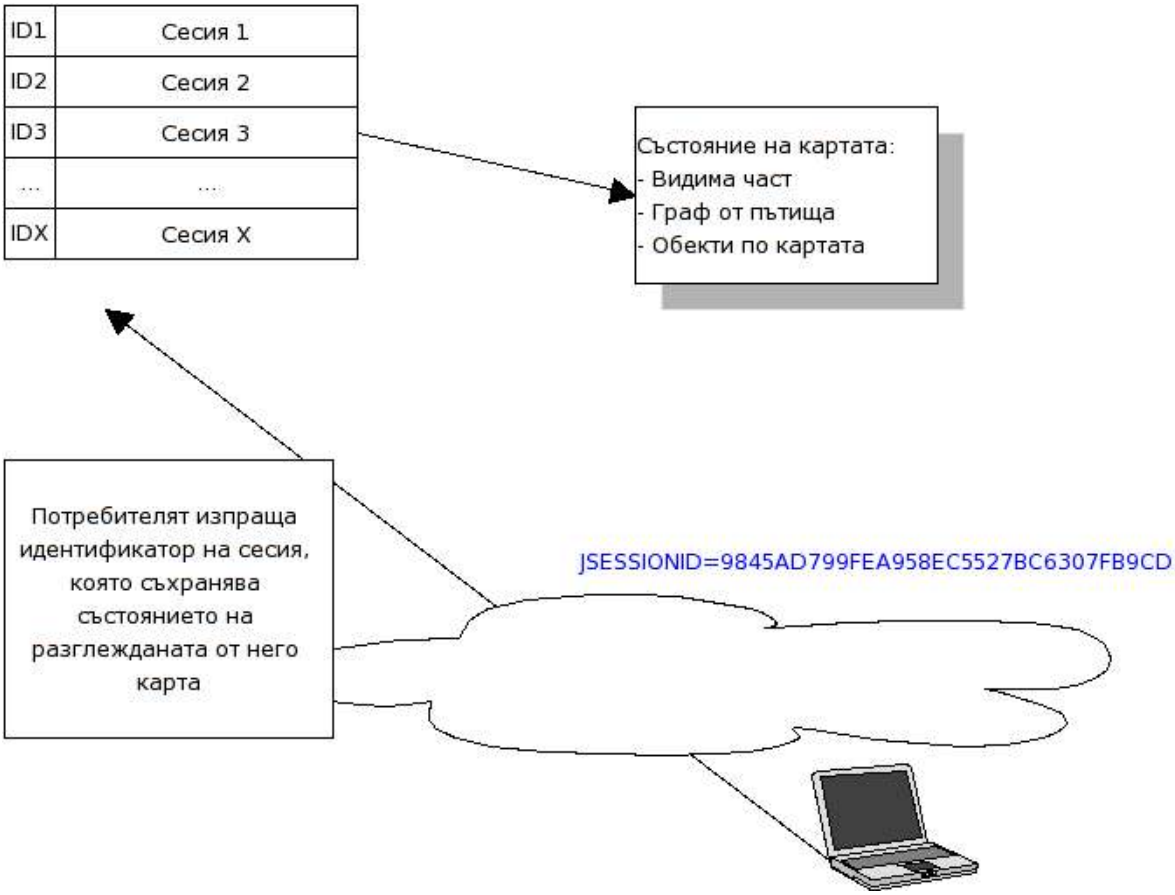
Фиг. 4. Клиент-сървър модел на комуникацията в инструмента за поддръжка и работа с географската карта



Фиг. 5. Потребител-редактор и краен потребител



Фиг. 6. Сесията, съхраняваща информацията за състоянието на картата за всеки потребител



4.4. Обектно и физическо представяне на кокмпонентите в системата

4.4.1. Карта

Понятието “карта” в текущата работа е изцяло обектно понятие в смисъла на обектно ориентираното програмиране. Картата има своите свойства и методи за достъп до предлагана от нея функционалност.

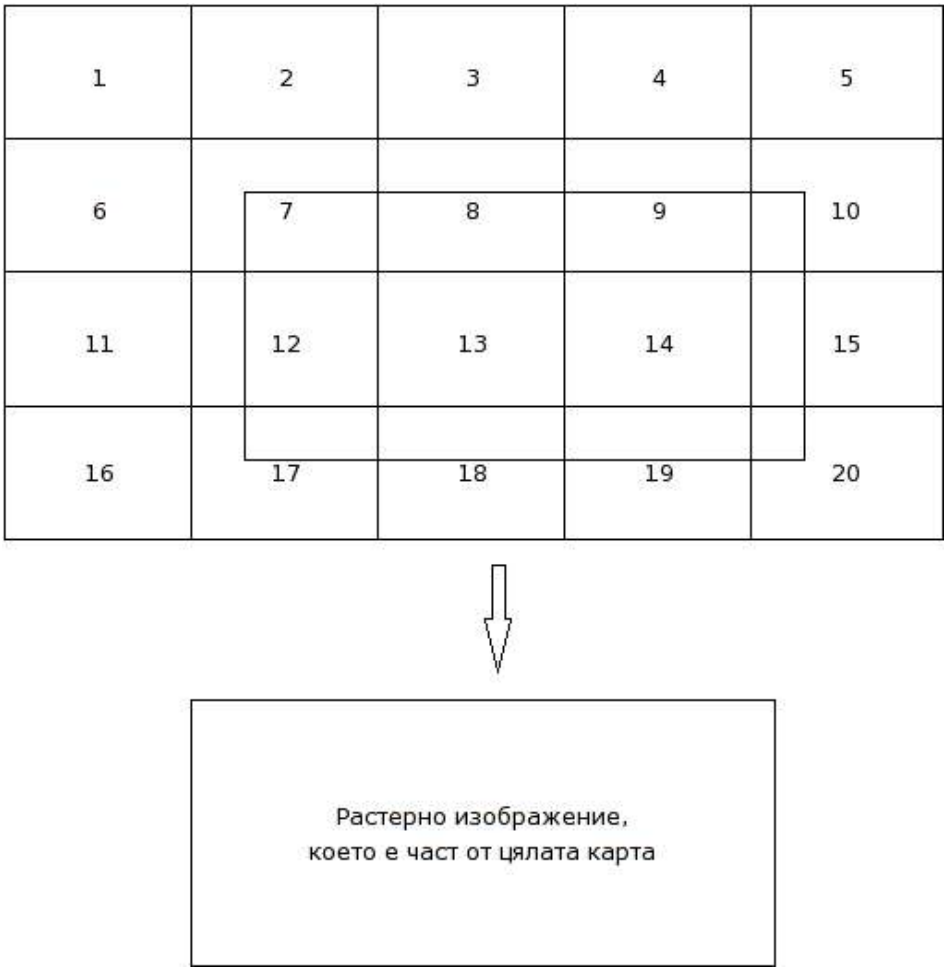
Поради сложността на операциите, които могат да се извършват върху географска карта, обектното понятие “карта” е разбито на 3 последователни нива (Java класове). Всяко следващо допълва и обогатява функционалността на предходното, или това е така нареченото “наследяване”.

Най-базовият обект е наречен MappedImage. Той представя функционалност за разглеждане правоъгълни на части от големи (като размерност в двумерното пространство) изображения без това да изтощава ресурсите на операционната система. Всяка инсталирана карта представлява множество от карти от вида MappedImage. Всяка една от тях се отличава от другата по това, че е с различно ниво на мащаб. Постигането на лекотата и бързодействието с големи изображения е постигнато чрез разделянето на всяко от мащабираните изображения на множество еднакви по размер правоъгълници. Всеки от тях представлява отделен физически файл. Обектът борави чрез тези физически файлове, за да предостави необходимата функционалност с необходимата скорост и ефективност на изпълнение. Този обект предлага два основни методи за работа:

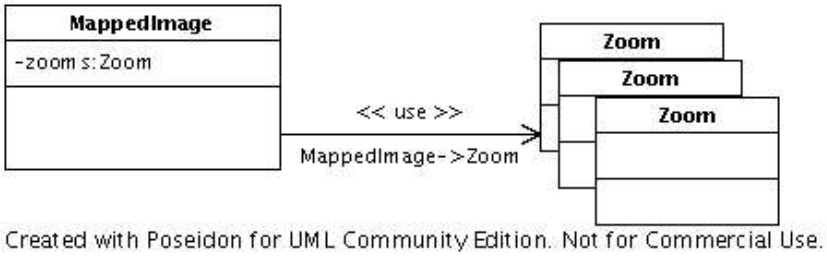
1. Трансформиране на растерно изображение (географска карта) до изображение, във вид, който е удобен за работа за клас. Този метод се използва от инструмента за добавяне на карти в системата.
2. Предоставяне на правоъгълна част от голямата карта във вид на стандартен Java обект за работа с растерни изображения. На вход този метод получава координати и размер на правоъгълник спрямо размера на текущия мащаб на картата.

Всеки един от правоъгълниците, на които е разделена картата се представя като инстанция на отделен клас - Zoom. Това е тривиален клас, по-скоро структура, която складира няколко свойства на правоъгълника, измежду които са позиция спрямо картата в пълния й размер, дължина и ширина.

Фиг. 7. Функционалност на класа MappedImage



Фиг. 8. Схема на класа MappedImage и класовете, с които си взаимодейства



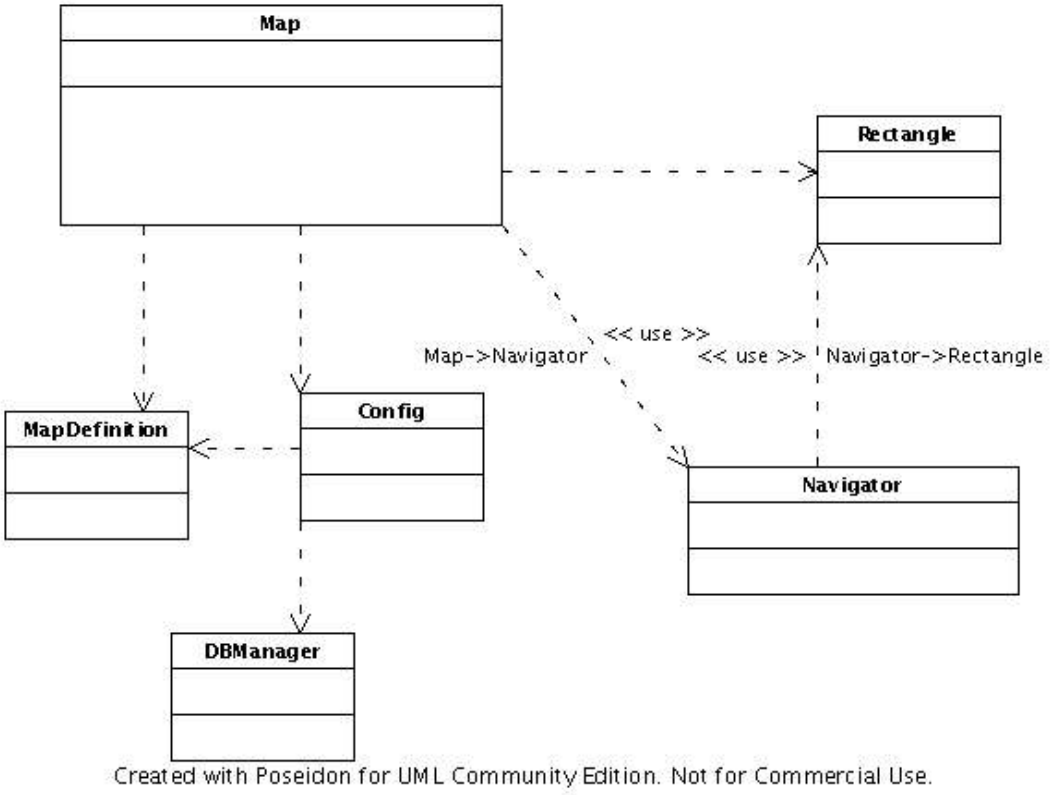
Следващото ниво от понятието "карта", или наследникът на базовия клас MappedImage е клас с името Map. Той разширява функционалността на MappedImage като добавя методи за навигация по картата, мащабиране и информация за състоянието й. Този клас се нуждае като вход от размер на правоъгълен участък, който се предполага да съдържа видимата част от картата (видим участък). Навигацията и мащабирането върху картата са всъщност движението на този правоъгълен участък върху голямо растерно изображение (обект MappedImage). Мащабирането върху картата изисква зареждането на съответен MappedImage обект за определено ниво на мащаб. Обектът предлага информация за текущия видим участък, както и функции за калкулация на размера и позицията на видимия правоъгълен район спрямо съответно ниво на мащаб. Основният метод в обекта се отнася към генериране на изображение, представляващо видимия участък във вид на стандартен Java обект BufferedImage.

Самото движение на правоъгълната видима част и калкулациите свързани с преизчисляването на координатите й при мащабиране са задача на обект, който се нарича Навигатор. Той е тясно свързан с работата на обекта Map.

На това ниво от йерархията е осъществено зареждането на мета-информация за картата. Това е направено тук, поради причината, че базовият обект се занимава основно с файлове по файловата система. Текущото ниво има за задача да укаже на базовия обект кои групи файлове (правоъгълници) отговарят за даден мащаб на географската карта. Информацията за картата се складира в базата от данни. Записването и зареждането на тази информация се извършват от клас, който се нарича Config. Той единствен от йерархията на класовете, описващи географската карта, има пряк достъп до базата от данни относно мета-информацията за картата. Самите мета-данни се складират в паметта под формата на инстанция на обект от

клас MapDefinition.

Фиг. 9. Класът Map и класовете, с които си взаимодейства



Следващият клас в йерархията наследява функционалността на предходните и разширява възможностите на картата с предоставяне на възможност за манипулиране на обекти върху нея. Не случайно идва и името на този обект – DynamicMap или динамична карта. Информацията, която се съдържа в този обект е разделена на следните групи:

1. Мета-данни за картата и за средата, в която се намира (графични ресурси, които използва – икони за обектите; конфигурация на системата и др.)
2. Данни за пътищата върху картата (ако има разпознати такива)
3. Информация за обектите върху картата

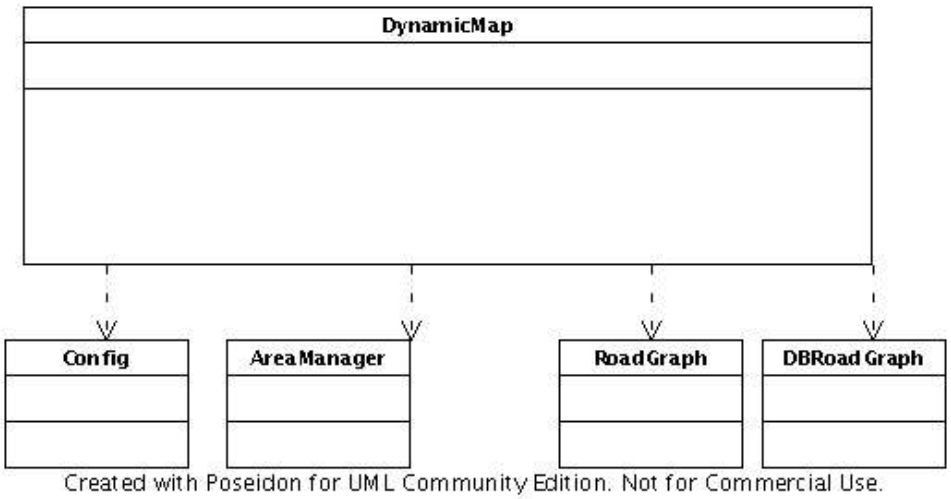
Всяка една от тези групи се управлява от отделен клас.

Мета-данните за картата и за средата са изцяло задача на клас, който складира текущата конфигурация на системата.

Обектите се управляват от клас наречен AreaManager. Той се занимава с добавянето, редактирането, премахването и локализацията на обектите и управляване на техните свойства.

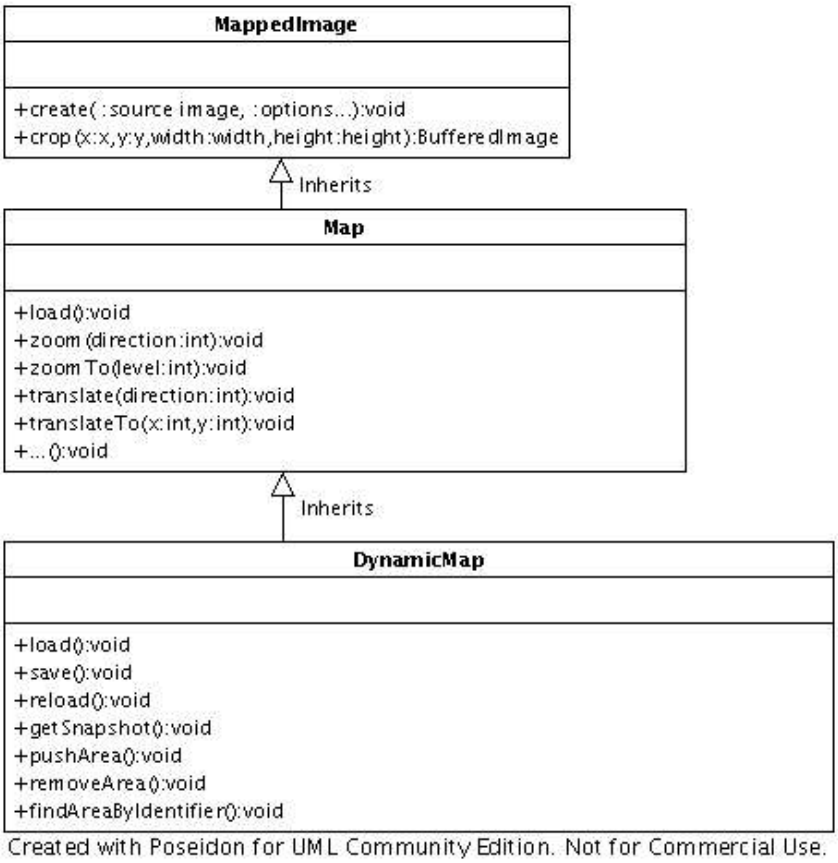
Класът за работа с пътищата по картата се нарича RoadGraph. Той осигурява информацията за най-кратките пътища между точки по картата.

Фиг. 10. Динамичната карта и класовете, с които си взаимодейства



Следващата фигура показва общ изглед на йерархията от обекти, които определят понятието "географска карта" в системата. Обърнете внимание, че не всички методи от класовете са показани, с цел добиване на обща представа за йерархията на класовете.

Фиг. 11. Опростена схема на йерархията на представянето на карта в системата.



4.4.2. Обекти

Следва да разгледаме класа, покриващ функционалността на понятието “обект върху географската карта”.

Разсъждавайки върху функционалността на обектите върху картата и способността им да имат свойства (име, описание, т.н.), стигаме до следната обща структура за всеки един обект:

1. Всеки обект има свойства, които в текущата документация ще наричам *атрибути*
2. При взаимнодействие с даден обект се изпълнява операция (напр. щракване с мишката при задържан клавиш SHIFT върху обект отваря асоциирания към обекта уеб адрес). Ще наречем взаимодействието *събитие*, а операцията – *действие*.

Този модел прилича много на представянето на HTML елементи в документ, всеки от които има свои *атрибути* и *събития*. Именно с такава семантика са представени обектите върху географската карта. Тези обекти представляват правоъгълни региони от картата, които притежават своите *атрибути* (позиция, размери, име, описание, уеб-адрес, т.н.) и *събития* (щракване с мишката, мащабиране, т.н.). Поради естеството на представяне на обектите, в текущата документация те са наричани доста често *области*.

Тъй като това представяне е доста универсално и гъвкаво, семантиката му е използвана при реализирането и на навигирането и мащабирането на картата. Всяка инсталирана в системата географска карта притежава една специална *област*, която покрива цялата карта и притежава *събития* за *придвижване (наляво, надясно, нагоре, надолу, на определена позиция)* и *мащабиране* (*по-голям, по-малък или произволен мащаб)*. В зависимост от това, кое събитие е настъпило се изпълнява съответното му действие.

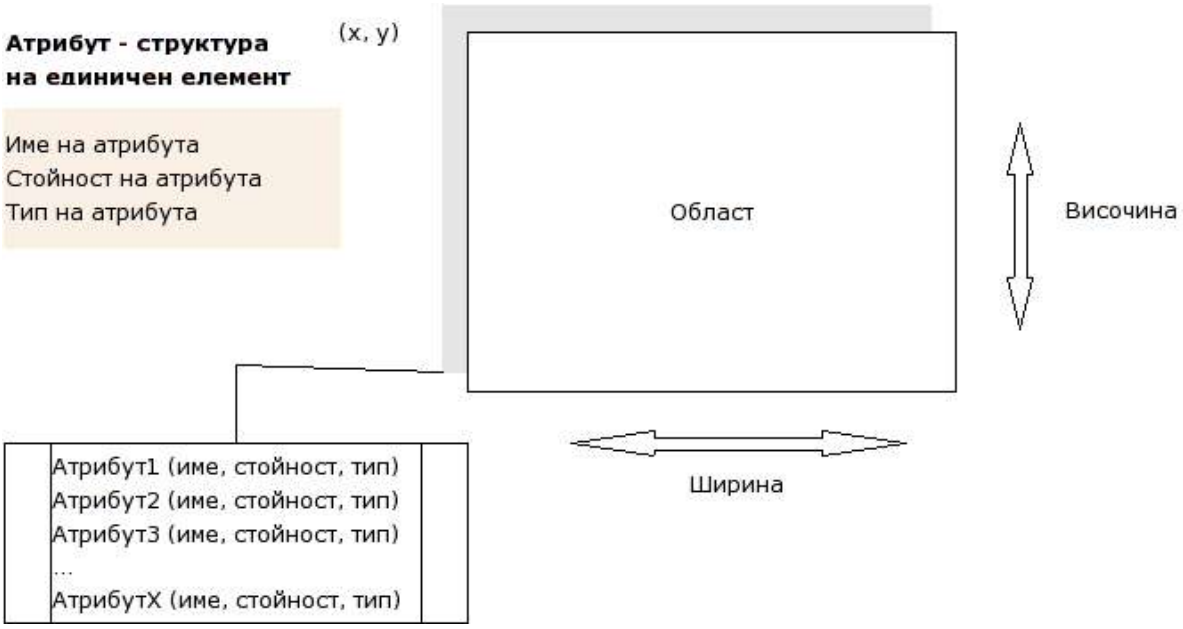
Всяка област има атрибути, които са видими за потребителя и атрибути, които са специални, или вътрешни за самата област. Специалните атрибути са *политика*, *местоположение по Z оста* и *тип* на атрибута. Семантиката на атрибута *политика* се съдържа в това дали дадената област може да бъде изтривана от потребител-редактор или не. Областта, която покрива цялата карта и отговаря за събитията по навигацията и мащабирането е с *политика* “*забранена за изтриване”*.

Областите са подредени не само в двумерното пространство на географската карта, но и по Z оста, т.е. могат да се припокриват или да бъдат една върху друга. Настъпилите събития върху дадена точка от картата се придвижват от най-горната (по Z оста) област към най-донлата, и за която от тях се намери събитие със същото

име се изпълнява действието, асоциирано към него.

Тъй като събитията са доста близки като представяне до атрибутите, то нотацията за събитие и атрибут е еднотипна: *име = стойност* (за атрибути); *събитие = аргументи за извършване на действие* (за събитие). Затова всеки атрибут си има и *тип,* което го отличава от това дали е атрибут, носещ информация или е събитие, водещо към някакво действие.

Фиг. 12. Област и структура на атрибутите й



Фиг. 13. Обект или *област* от картата с *атрибути* и *събития*

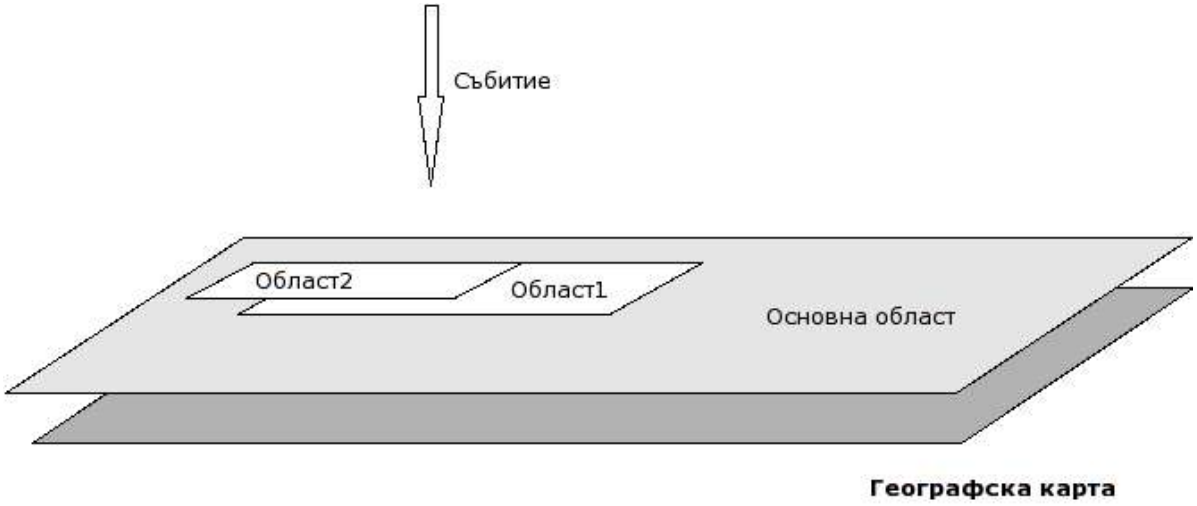
Информация за областта

	Публична информация - позиция (x, y) - размери (дължина, ширина) - име - описание - уеб-адрес - категория - под-карта Специална - Ниво по Z оста - Политика

Събития

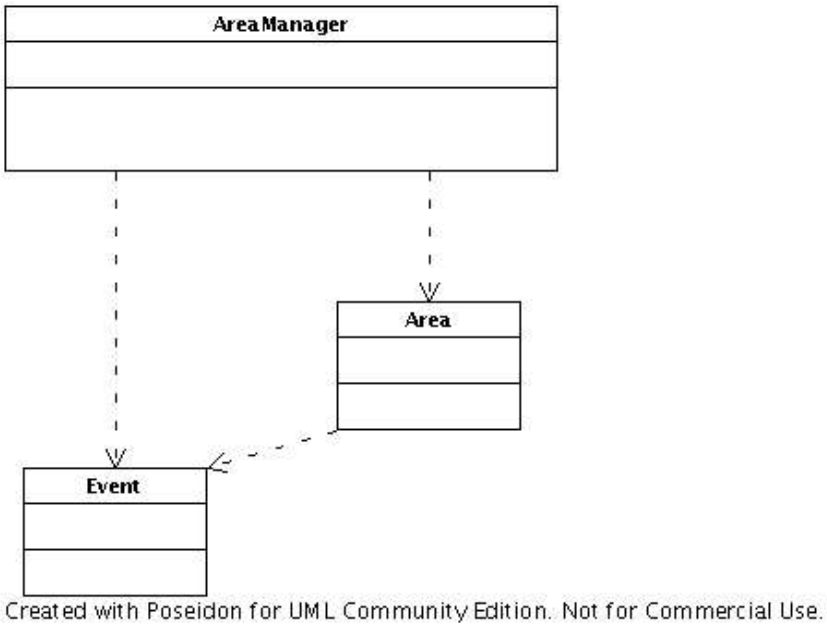
	- Щракване с мишка - Мащабиране (по-голям, по-малък мащаб) - Транслация (наляво, надясно, нагоре, надолу) - Транслация и мащабиране (комбинирано) - Стигане на максимален мащаб - Стигане на минимален мащаб	

Фиг. 14. Разположение на области по картата



Обектите/областите се управляват от така наречения областен мениджър (AreaManager). Той предлага методи за добавяне, редактиране и премахване на области. Други методи, които предлага са търсене из атрибутите на областите; търсене на област, която е засегната от някакво събитие; търсене на област по координати; търсене на област по уникален идентификатор. Областният мениджър се занимава с поставянето на събития за дадена област и следенето за изпълнението на действия при възникнали събития.

Фиг. 15. Областен мениджър



4.4.3. Граф от пътища

Пътищата върху карата са представени като граф от множество точки свързани с ребра. Гъстотата на точките е достатъчна за осигуряване на плавно и точно следване на формите на пътя върху картата.

Графът се генерира от съответен инструмент при процес на инсталиране на картата. Точките и ребрата между точките се складират в база от данни и са асоциирани към определена карта.

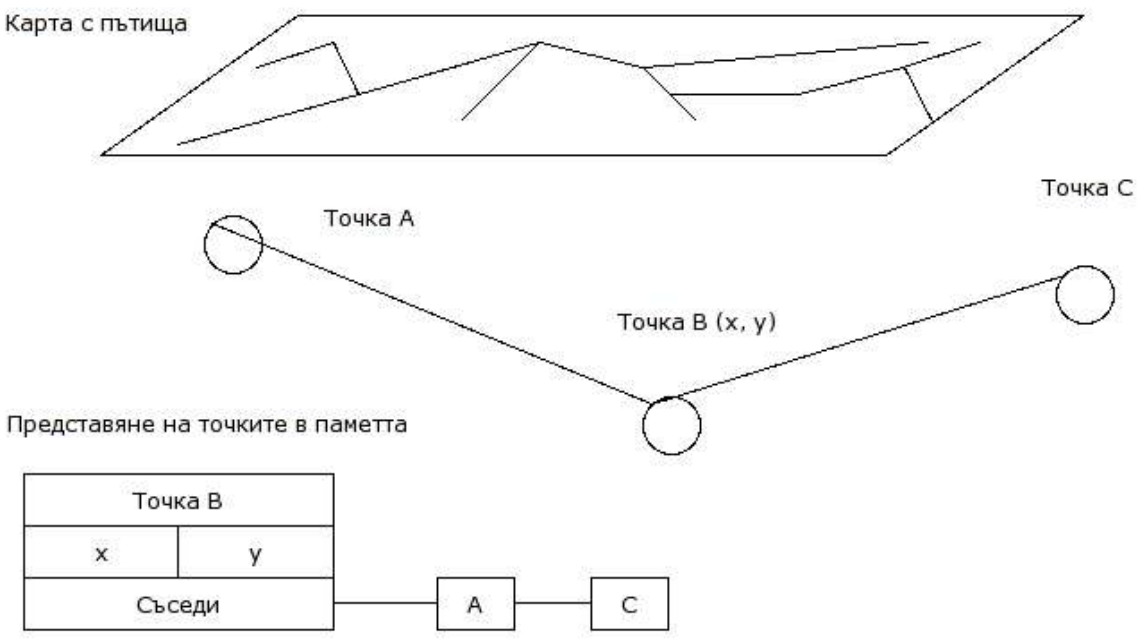
Всяка точка от графа (RoadPoint) има координати спрямо оригиналния размер на картата и списък от съседни точки (ConnectionPoint), с които е свързана. Графът е неоргиентиран – свързването между точките от графа е двупосочно, т.е. ако има ребро от точка А до точка Б, то има и ребро от Б до А.

Всяка карта може да разполага с един или повече графи от пътища. Когато потребителят търси най-кратък път между две точки на картата, системата търси най-краткия път между две от точките на определен граф от картата.

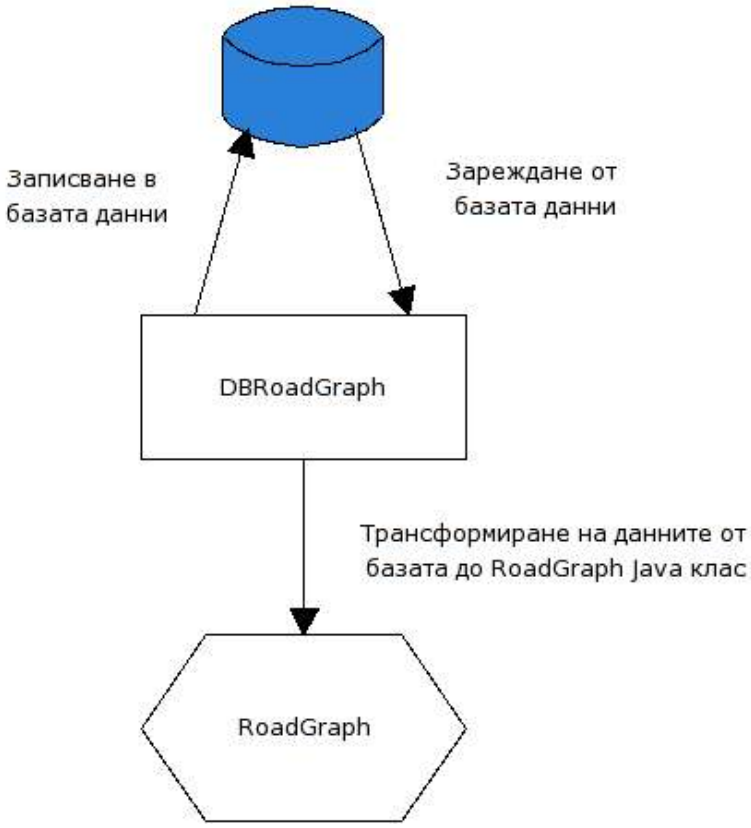
Съществува клас, който осъществява записването на графа в базата и зареждането му от там. Функционалността по записването на графа в базата се използва при инсталиране на карта. При зареждане за разглеждане (от страна на потребителя) на нова карта се зареждат и графите за тази карта (в случай, че има налични такива). Този клас е наречен DBRoadGraph.

Инструментът за разпознаване на пътищата по картата работи с тези класове, но тъй като те представят само данните от пътищата и складирането им в базата, то е нужен клас, който се занимава със самия процес на разпознаване - RoadRectognition. Информацията, която инструментът му подава е във вид на друг Java клас – RoadRecognitionInfo. След завършване на разпознаването RoadRecognition връща като резултат инстанция на обект от тип RoadGraph. Този обект е в удобен вид за запазване в базата данни и при преценка на потребителя-редактор разпознатият граф се записва в нея, асоциирайки го към дадената карта.

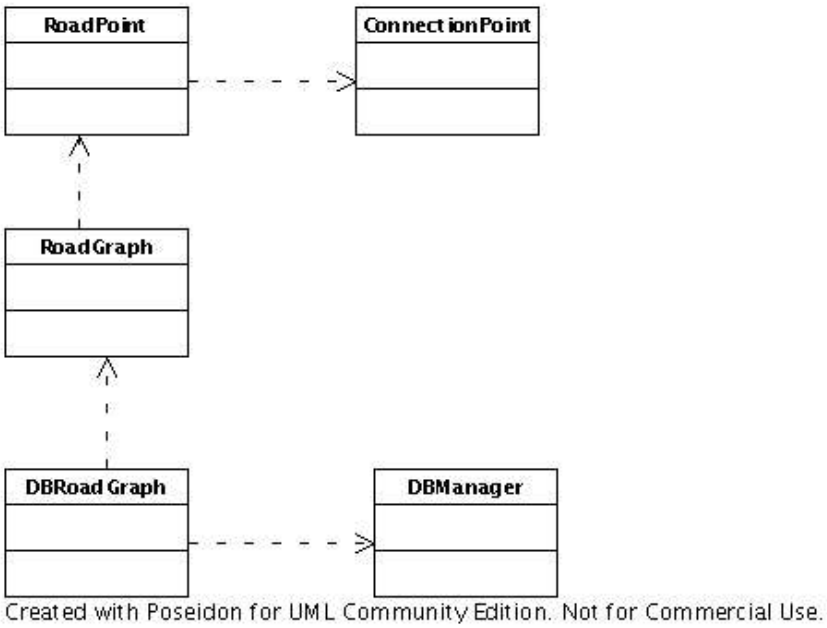
Фиг. 16. Структура и представяне на граф върху географската карта



Фиг. 17. Функция на класа DBRoadGraph



Фиг. 18. Взаимодействие между класовете описващи графа от пътища



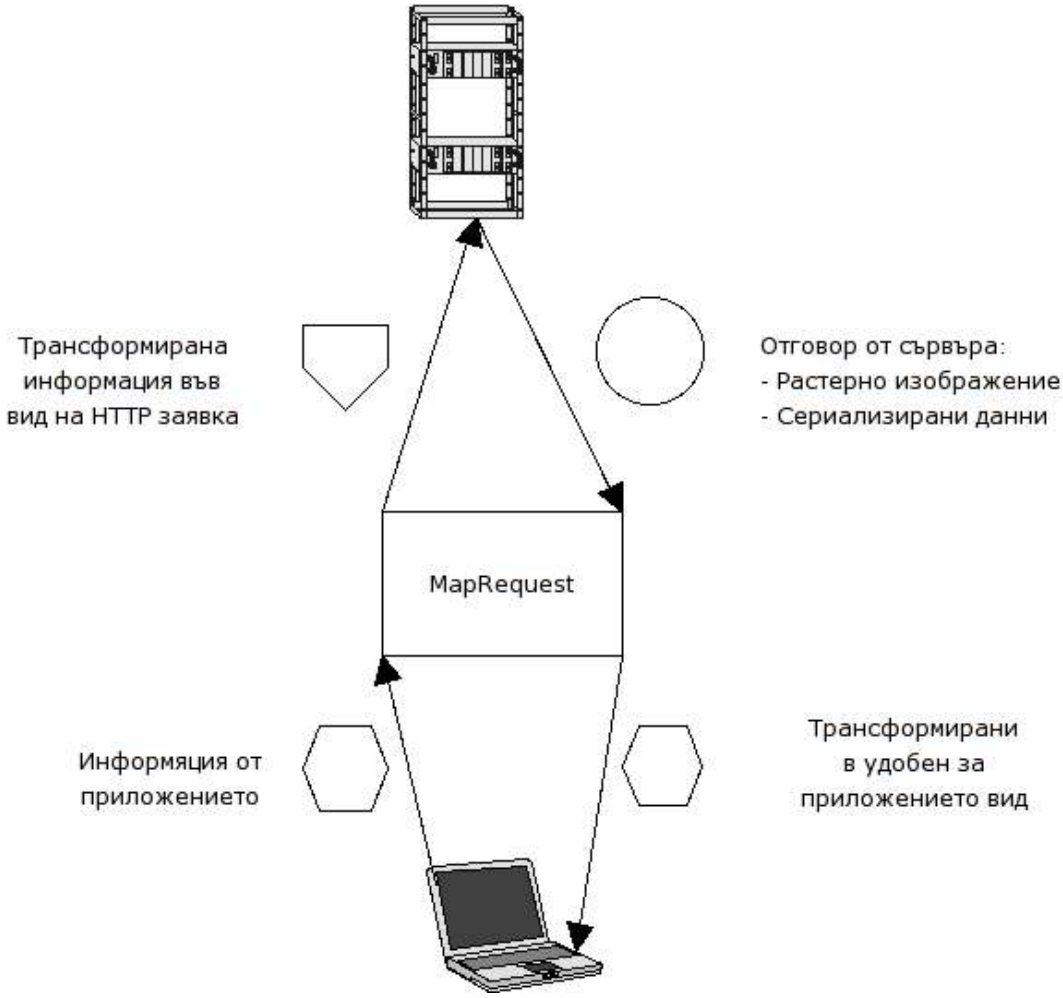
4.4.4. Комуникация в клиент-сървър среда

Комуникацията на клиента със сървъра се осъществява през клас, наречен MapRequest. Той е натоварен със задачата да изпраща заявки до сървъра и да приема отговори. Входните данни, които получава са от графичното приложение и са във вид “разбираем” за графичното приложение. Тези данни се трансформират до формат, в който да се пратят до сървъра. Отговорът на сървъра е или растерно изображение или сериализиран данни във формата на Java обект. И в двата случая отговорът се трансформира до вид, който е удобен за работа за графичното приложение.

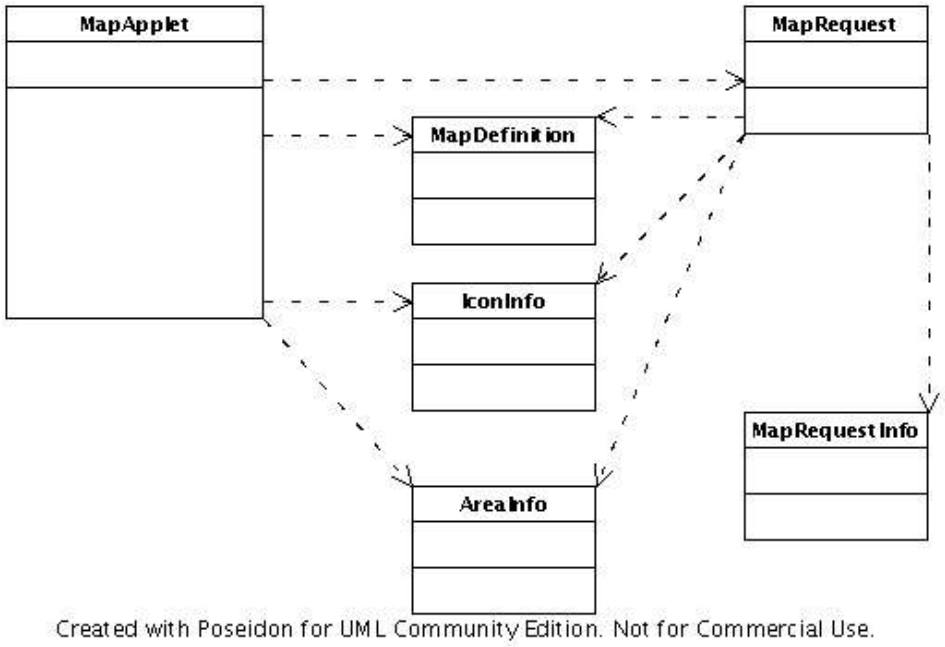
Класът изпраща два типа заявки. Единият тип очаква като резултат растерно изображение и е наречен картова заявка (map request). Другият тип заявки очакват като резултат данни за картата, за обектите по нея и т.н. В първия случай сървърът отговаря с поток от данни, който представлява растерно изображение. В другия случай се изпраща сериализиран клас от данни от тип MapRequestInfo. В този сериализиран клас от данни може да се съдържа информация за: обектите по картата (сериализируем клас AreaInfo); информация за самата карта, или картите в системата (класът MapDefinition); информация за текущите категории от обекти/области (IconInfo, GroupInfo); информация за видимата част на картата.

Тук е моментът да спомена нещо повече за категориите от обекти или области. Всяка област принадлежи към определена категория. Тази категория е атрибут на областта. Категорията се идентифицира с икона, която се използва като главен графичен способ за визуализиране на местоположението на обекта върху картата. Категориите са разделени на групи. Това не указва влияние върху представянето на категорията в областта. Групирането е използвано за улеснение на потребителя-редактор и крайния потребител. Информацията за категориите в системата се използва само в приложението за управление и работа с географската карта и се доставя чрез класа MapRequest.

Фиг. 19. Трансформация на данните през MapRequest



Фиг. 20. Класове за комуникация на клиента със сървъра



4.4.5. Достъп до базата от данни

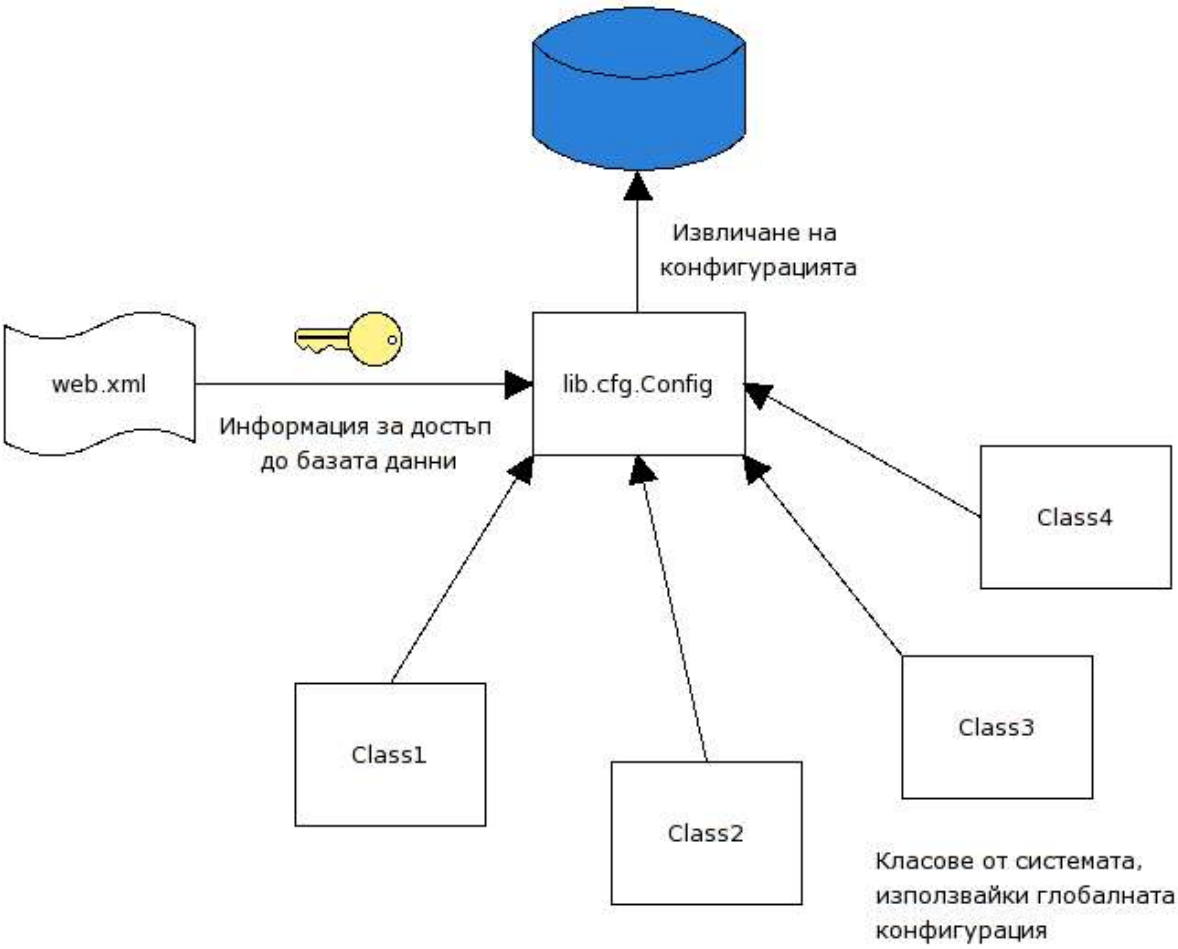
Достъпът до базата от данни се осъществява чрез специално създаден за това клас - DBManager. Той позволява свързване към даден MySQL сървър и отваряне на посочена база. Дава възможност за изпълняване на заявки към базата. Използван е стандаартна библиотека за работа с MySQL база от данни през Java програмен интерфейс.

Класовете, които имат достъп до базата данни (не директно, а чрез този клас) са такива, които зареждат данни от нея в Java обектите описани досега – класове представящи картата; класове, които се занимават с обектите по нея, класове за представяне на пътищата по картата. Всички тези не работят директно с DBManager, а са съсредоточени да вършат операции върху съответните данни – растерното изображение, обектите по него, графите. Създадени са помощни класове, които се грижат за записването и зареждането на данните от базата в съответните инстанции на обекти. Те комуникират с DBManager и играят ролята на делегати.

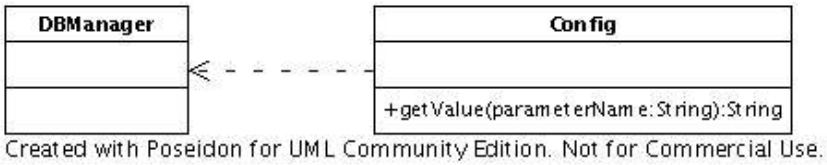
4.4.6. Конфигурация

Конфигурацията на системата се държи изцяло в базата от данни. Класът lib.cfg.Config изпълнява ролята на този, който представя конфигурацията от базата данни във вид удобен за останалите класове. За да стигне този клас до тази информация, е нужно на самия него да е посочено как да има достъп до базата от данни. Тази информация се държи в текстов формат, в конфигурационните файлове на уеб-сървъра. Веднъж свързал се към базата от данни, класът за конфигурация разполага с всичко необходимо като глобална информация в помощ за останалите класове.

Фиг. 21. Взаимодействие между конфигурационният файл на уеб-сървъра (web.xml) с Config класа. Взаимодействие на класове от системата с Config класа.



Фиг. 22. Класът lib.cfg.Config



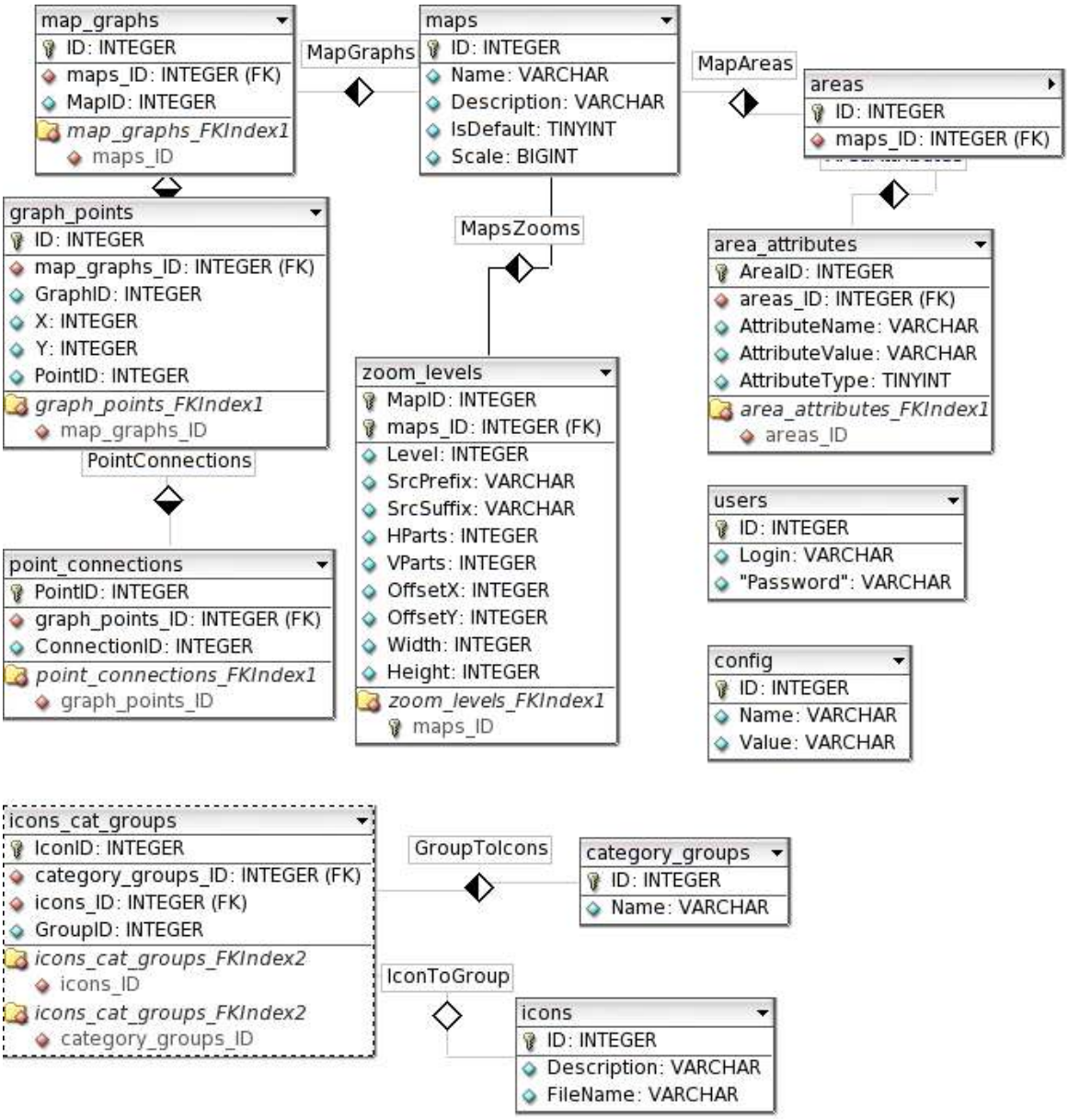
4.5. База от данни

Базата от данни складира всичко необходимо за работата и поддръжката на географската карта. Има четири основни групи в схемата на базата:

1. Таблици, които описват географската карта – области, графи, имена на физически файлове, които представят използваните растерни изображения
2. Таблици, които описват текущите категории в системата – групи от категории, категории/икони
3. Таблица с глобалната конфигурация
4. Таблица с текущите регистрирани потребители-редактори

Следващата фигура описва релациите между таблиците във всяка една от групите.

Фиг. 23. Модел на базата от данни



4.5.1. Таблици, описващи географската карта, пътищата и обектите по нея

1. **maps** – таблицата, която съдържа списък от всички карти в системата
   a) ID – идентификатор на картата (индекс и първичен ключ)
   b) Name – име на географската карта, което ще вижда потребителят
   c) Description – описание на картата
   d) IsDefault – флаг, който определя дали картата ще бъде избрана като текуща ако потребителят не е посочил определена карта
   e) Scale – мащаб на картата - метри отговарящи на 1 пиксел от оригиналния размер
2. **zoom_levels** – съхранява информация за определено ниво на мащаб – физически имена на файловете, които го образуват, размери и т.н. Релацията с таблицата **maps** е едно към много (една карта, много мащаби). Връзката е по идентификатор на карта.
   a) MapID – идентификатор за принадлежност към карта (индекс в таблицата)
   b) Level – ниво на мащаб. Най-детайлната карта е с най-голямо ниво на мащаб
   c) SrcPrefix – префикс, с който се образува името на файла, съдържащ едно от многото правоъгълни растерни изображения, на които са разбити всяко едно от мащабните изображения
   d) SrcSuffix – суфикс, с който се образува името на файл, съдържащ правоъгълно изображения от тези, на които са разбити мащабните изображения
   e) HParts – брой хоризонтални правоъгълни изображения в текущото ниво на мащаб
   f) VParts – брой вертикални правоъгълни изображения в текущото ниво на мащаб
   g) OffsetX – хоризлонталното разстояние между началата на две съседни правоъгълни изображения (ширина на единична правоъгълната разбивка)
   h) OffsetY – вертикалното разстояние между началата на две съседни правоъгълни изображения (височина на единична правоъгълната разбивка)
   i) Width – ширината на цялата карта в текущото ниво на мащаб
   j) Height – височина на цялата карта в текущото ниво на мащаб
3. **areas** – съдържа идентификатори на всички области разделени по принадлежност към карта. Релацията с таблицата **maps** е едно към много (една карта, много области). Връзката е по идентификатор на карта.
   a) ID – идентификатор на областта (първичен ключ, индекс)

b) MapID – идентификатор на карата, към която принадлежи областта (индекс)

4. **area_attributes** – атрибутите на областите. Разделени са по област и по тип на атрибута. Релацията с таблицата **areas** е едно към много (една област, много атрибути). Връзката е по идентификатор на област.
   a) AreaID – идентификатор на областта към която принадлежи атрибута (индекс)
   b) AttributeName – име на атрибута
   c) AttributeValue – стойност на атрибута
   d) AttributeType – тип на атрибута – мета-информация, дефиниция на събитие
5. **map_graphs** – съдържа идентификаторите на графи от пътища по картата. Разделени са по принадлежност към карта. Релацията с таблицата **maps** е едно към много (една карта, много графи от пътища). Връзката е по идентификатор на карта.
   a) ID – идентификатор на граф (първичен ключ, индекс)
   b) MapID – идентификатор на картата, към която принадлежи графът (индекс)
6. **graph_points** – съдържа всички точки от графите, разпределени по идентификатор на граф. Релацията с таблицата **map_graphs** е едно към много (един граф, много точки). Връзката е по идентификатор на граф.
   a) ID – идентификатор на точка от графа (първичен ключ, индекс)
   b) GraphID – идентификатор на граф, към който принадлежи (индекс)
   c) X – абсолютна позиция по X оста спрямо оригиналния размер на картата спрямо горния й ляв ъгъл
   d) Y – абсолютна позиция по Y оста спрямо оригиналния размер на картата спрямо горния й ляв ъгъл
   e) PointID – уникален идентификатор на точка. Използва се вътрешно от класа, който управлява точките от графа за референции към останали точки.
7. **point_connections** – съдържа връзките между точките в графите. Релацията с таблицата **graph_points** е едно към много (една точка, много връзки). Връзката е по идентификатор на точка.
   a) PointID – идентификатор на точка от графа. Това е връзката с колоната **graph_points.ID** (индекс)
   b) ConnectionID – идентификатор на съседна на точката, посочена от PointID. Съдържа идентификатор измежду **graph_points.ID**.

4.5.2. Таблици, описващи категориите в системата

1. **icons** – съдържа всички категории в системата. Всяка една от тях е асоциирана с икона, която е растерно изображение.
   a) ID – идентификатор на иконата/категорията (първичен ключ, индекс)
   b) Description – име на категорията
   c) FileName – файл, който съдържа изображението за тази категория
2. **category_groups** – групи, на които са разделени категориите
   a) ID – идентификатор на групата (първичен ключ, индекс)
   b) Name – име на групата
3. **icons_cat_groups** – съдържа релацията между групите и категориите. Релацията е едно към много (една група, много категории).
   a) IconID – идентификатор на категория/икона от таблицата **icons**
   b) GroupID – идентификатор на групата от таблицата **category_groups** (индекс)

4.5.3. Таблица за глобална конфигурация

1. **config** – съдържа двойки от типа име-стойност, които представляват глобални параметри за системата
   a) ID – идентификатор на параметъра (първичен ключ, индекс)
   b) Name – име на параметъра
   c) Value – стойност на конфигурационния параметър

4.5.4. Таблица за текущо регистрираните потребители-редактори

1. **users** – съдържа информация за текущо регистрираните потребители-редактори
   a) ID – идентификатор на потребителя (първичен ключ, индекс)
   b) Login – уникално име на потребителя, чрез което се идентифицира (уникален ключ)
   c) Password – парола на потребителя, която заедно с името определят ключа, с който се оторизира потребителя за редакторска дейност. Паролата е кодирана чрез вградената в MySQL функция PASSWORD().

5. Алгоритми

В следващите страници на тази секция са описани някои от по-сложните алгоритми, използвани в системата. Авторът не се е спирал за подробни обясненияя върху тривиалните алгоритми.

5.1. Инсталиране на растерно приложение във вид подходящ за системата

Процесът на добавяне на произволно растерно изображение в системата е задача за генериране на няколко мащаба на изображението и разделяне на всеки от тях на еднакви правоъгълни части.

На вход се подава броя на мащабите, на които може да бъде разглеждана картата. Изчисляването на размерите най-малкия (най-отдалечения изглед на картата) мащаб е:

размерите_на_оригиналната_карта / брой_на_мащабите.

Броят на мащабите се задава в зависимост от размера (ширина, височина) на оригиналното изображение и размерите на видимата част от картата в клиентското графично приложение. Ако най-малкият мащаб е по-малък от размерите на прозореца в графичното изображение, показващ видимата част от картата, то този мащаб ще се изрисува в горния ляв ъгъл на прозореца. Размерите на всяко ниво на мащаб се изчисляват по формулата:

*(размерите_на_оригиналната_карта/брой_на_мащабите)*текущ_мащаб*

След като се изчислят размерите на мащабираната карта за текущо ниво на мащаб, тя се свива от изходни до размерите на това ниво. Следва разделяне на мащабираното изображение на множество от еднакви по размер правоъгълници, всеки от които е със зададени константни размери W и H. Броят на хоризонтално разположените правоъгълници е изчислен по формулата – *текуща_ширина_на_мащаба / W*.

Аналогично за вертикално разположените – *текуща_височина_на_мащаба / H.*

Всеко едно от мащабните изображения се “нарязват” на правоъгълници, които се записват като отделни файлове по унифициран формат:

<префикс>_<ширина>x<височина>_<x_индекс>_<в_индекс>_<суфикс>,

където

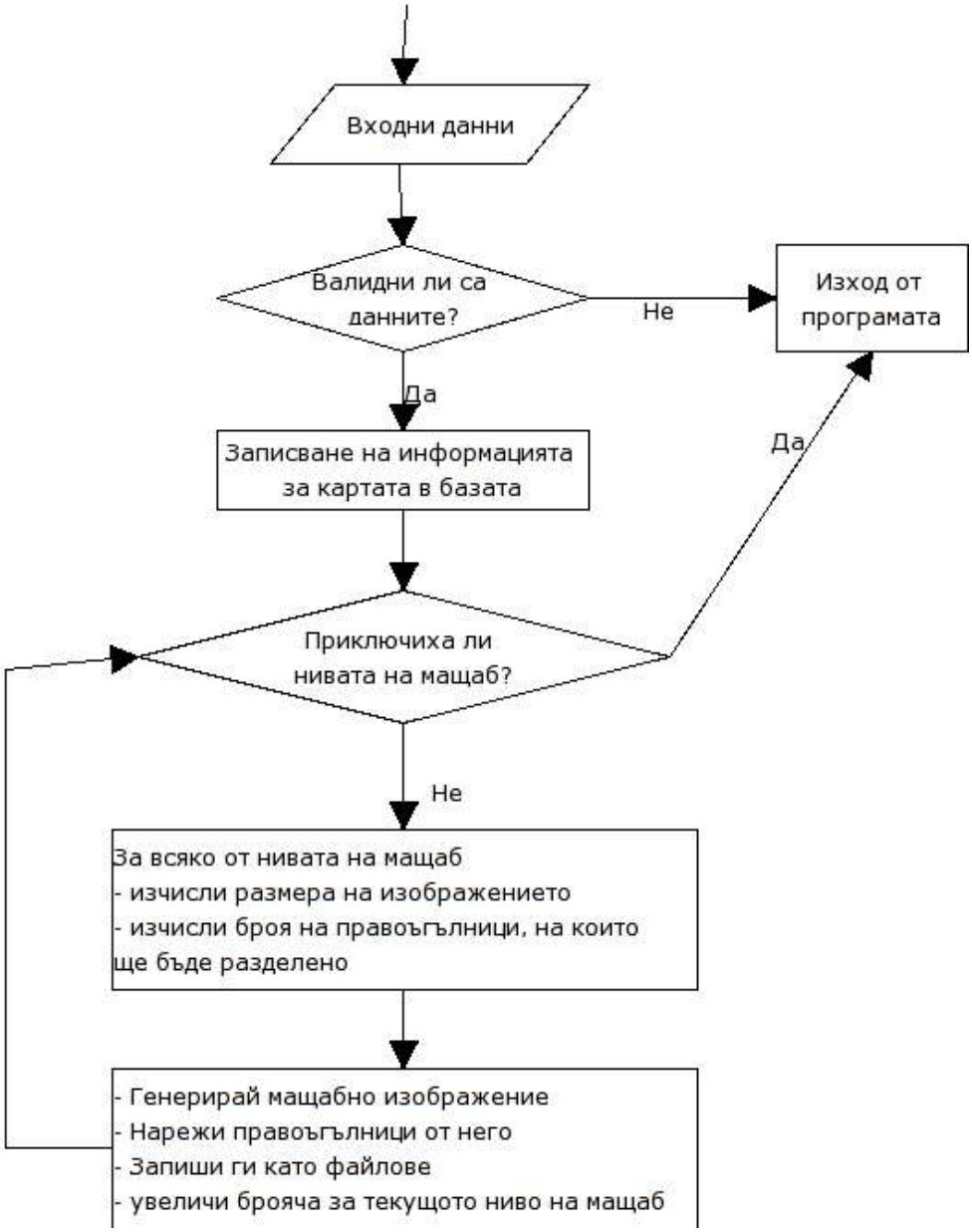
- *префикс* – символен низ. Във версията, по времето на която е писана текущата документация е използван префиксът zoom_<ниво на мащаб>
- *ширина* и *височина* – ширината и височината на текущото мащабно изображение
- *х_индекс* и *в_индекс* – хоризонтален и вертикален индекс на правоъгълника върху цялото мащабно изображение
- *суфикс* – обикновено разширението на файла (.png, .jpg, .gif, т.н.)

Файловете се записват на файловата система в папка, указана от входните параметри. Пътят до тази папка трябва да бъде указан в глобалните за системата параметри. Картата се добавя в базата данни, както и информация за разбитите на правоъгълници мащабирани изображения. Пътят до правоъгълно изображение е образуван от основната зададена папка, идентификаторът на текущата карта и името на файла, който отговаря на текущия правоъгълник.

Картата се добавя в базата от данни, използвайки още няколко входни параметри – име, описание, мащаб (метри, отговарящи на 1 пиксел от картата), флаг дали да е основна карта по премълчаване.

Класът, който осигурява валидни входни данни се нарича MapBuilder. Той валидира данните и ги подава на MappedImage, който трансформира картата в описания по-горе вид.

Фиг. 24. Инсталиране на географската карта и трансформирането ѝ във вид използван от системата.



5.2. Разпознаване на пътищата по картата

Процесът на разпознаване на пътищата по картата е полу-автоматичен, т.е. обвързан е с потребителски входни данни. Той може да се раздели на две основни части:

1. Удовлетворяване на потребителя, показвайки му примерно разпознати пътища върху малка част от картата, използвайки данните, които е въвел
2. Стартиране на процес за разпонаване на пътища върху най-детайлния мащаб на картата.

Процесът във втората част е същият, който се използва и в първата, с тази разлика, че в първата се прилага върху малко изображение, а във втората – върху картата в оригиналния й размер.

Входните данни, които са нужни за разпознаването на пътищата са списък от цветове на пиксели, които са част от пътища върху картата. Разпознаването е на базата на околности на цветовете, определени от тези точки.

След като потребителят посочи точка от екрана, всички точки, които се намират в околност на избрания цвят биват оцветени по отличителен начин, указвайки кои от точките ще се вземат предвид при процеса на разпознаване. След това отново се дава възможност на потребителя да избере още точки. Това продължава, докато той е удовлетворен от точността на покритата площ от отличително оцветените точки.

Цветовете на точките от изображението се вземат чрез функции за обработка на растерни изображения в Java (java.awt.image.*). Те са в RGB цветово пространство. В това пространство е трудно да се определи околност на определен цвят, затова потърсих превръщания в други цветови пространства, в които може лесно да се определи някаква околност на цвят. Направих опити с HSV (Hue, Saturation, Value/Brightness) и YCrCb (Luminnance, Chrominance). Използването на HSV цветовото пространство даде добри резултати (“добър резултат” ще наричаме големия брой близки по цвят точки разпознати от използването на околност на даден цвят), но YCrCb даде по-добри резултати. Факт е, че много системи за разпознаване на човешко изображение работят в YCrCb цветовото пространство.

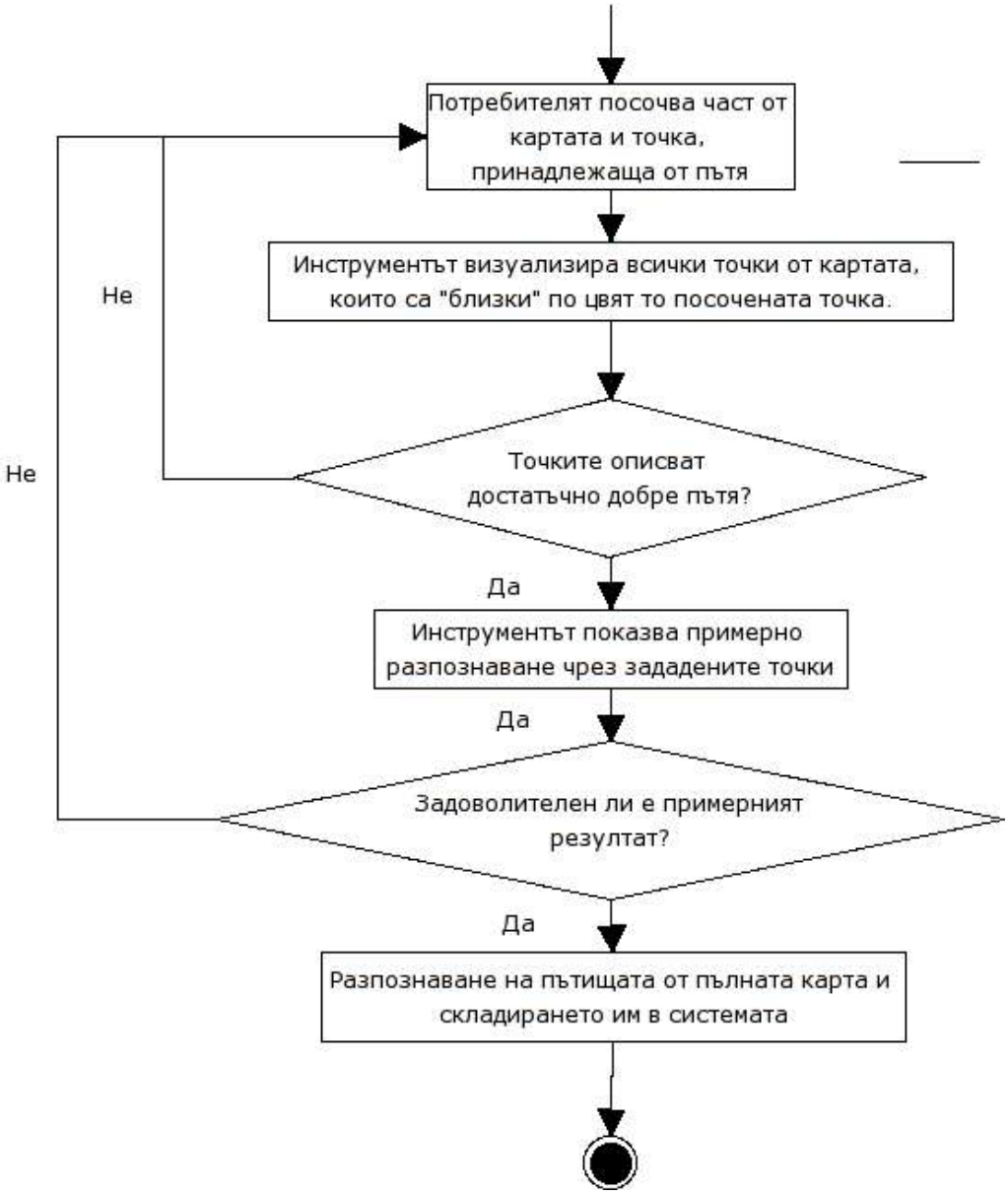
Превръщането на RGB в YCrCb става по формулата:

```
y  = (0.257 * red)+(0.504 * green)+(0.098 * blue)+16
cr = (0.439 * red)-(0.368 * green)-(0.071 * blue)+128
cb = -(0.148 * red)-(0.291 * green)+(0.439 * blue)+128
```

Дефинирам *околност на цвят на една точка* като три интервала на стойности на цвят в YCrCb цветовото пространство (всеки един от интервалите е определен за един от трите компонента на цвета в пространството), така че компонените на текущият цвят в YCrCb да е в средата на интервала. След неколкократни опита видях, че добри резултати дават следните дължини на интервали: 20 за Luminance (Y), 10 за Cr, 10 за Cb.

Дефинирам, че *една точка е от път в дадена карта* ако стойностите на компонентите на цвета й в YCrCb цветовото пространтво са в околност на цвят на някоя от точките от пътя, които потребителят е посочил.

Фиг. 25. Потребителски интерфейс и стъпки за разпознаването на пътищата по картата.



Самият алгоритъм[1] за разпознаване на пътищата върху картата и създаването на граф от точките върху пътя се състои от две стъпки:

1. Намирането на точките, които се намират в центъра на пътната ивица (или пътните ивици).
2. Създаване на граф от тези точки.

[1] Алгоритъмът за разпознаване на пътища е изцяло по идея на разработчика на текущата дипломна работа

5.2.1. Намиране на точките, които се намират в центъра на пътя

Процесът на намиране на въпросните точки се състои от три стъпки:

1. Грубо филтриране на изображението
2. Изчистване на шума от филтрираното изображение
3. Намиране на точките от центъра на пътя от резултатното изображение.

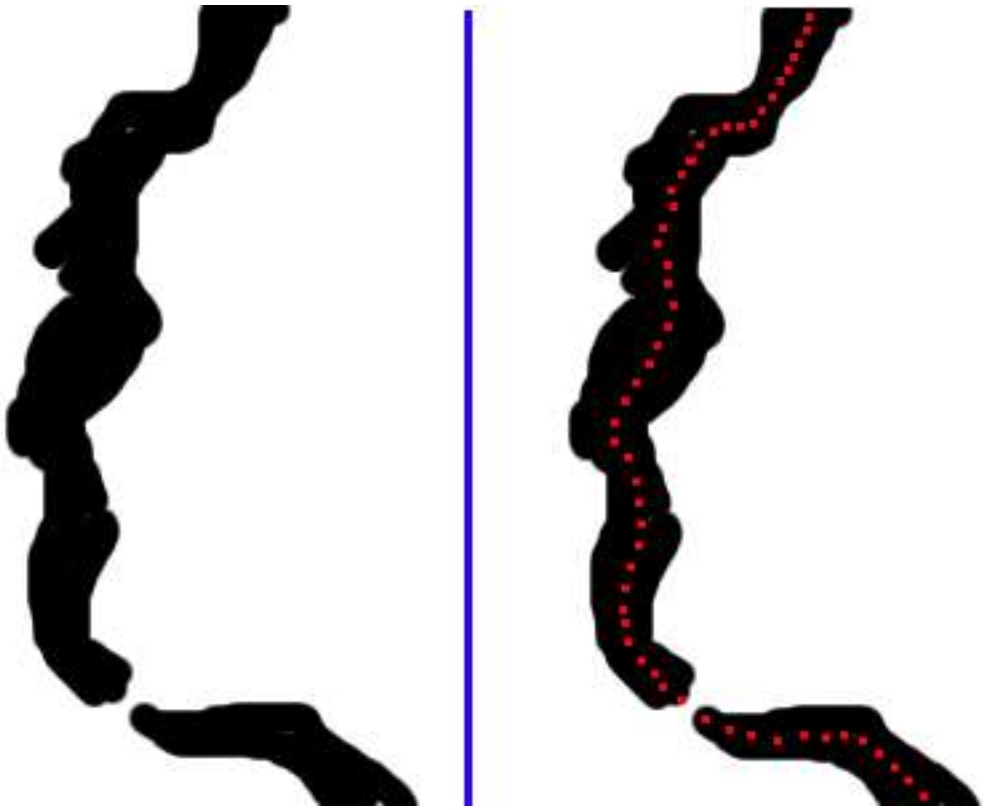
Резултатът от грубото филтриране на изображението е черно-бяло растерно изображение, при което черните точки са точки от пътя, а белите – не. Черно-бялото изображение е клас ByteImage, който представлява матрица, всеки от елементите на която може да има една от три стойности. Класът предлага методи за управление на елементите на матрицата (пикселите върху изображението). Всеки елемент от матрицата отговаря на точно един определен пиксел от изходното изображение. Причината за създаването на този обект е с цел по-бързата обработка на черно-бяло изображение пред RGB такова.

Резултатното черно-бяло изображение има много шум, т.е. черни и бели точки, които се срещат на места, където са заобиколени от противоположния цвят и би следвало да не присъстват в този цвят. Нужно е изчистването на този шум. Това се прави по следния начин: Черно-бялото изображение се обхожда от квадратна матрица с фиксиран размер. Стъпката, по която се обхожда изображението е размерът на матрицата. На всяка позиция се изчислява “енергията” на матрицата, т.е. броят на черните пиксели в нея. Ако “енергията” е по-голяма от фиксирано число, площта, на която е матрицата се оцветява в черно, в противен случай – в бяло.

Резултатното изображение има “плътни” участъци от черни пиксели, които принадлежат на пътища. Тези участъци са в общия случай разпокъсани на места. Нужно е намирането на точките от центровете на тези участъци, или по друг начин казано – точките от ивицата, която е в центъра на площите.

Следващото изображение показва примерен участък и желаният резултат от обработката.

Фиг. 26. Намиране на точките от центровете на площите от пътя.



Намирането на тези точки се състои в три стъпки:

1. Определянето на оптимален размер на квадратна матрица, която наложена в центъра на всеки един от участъците попада в него поне с фиксиран процент от площта си.
2. Обхождане с намерената матрица. Ако "черната площ", която се покрива е по-голяма от фиксиран процент от броя на клетките в матрицата, то точката, която е в центъа на матрицата се взема за точка от центъра на площтта (и се отбелязва с цвят различен от черно или бяло).
3. Точките, които са с различен от черен и бял цвят се отделят и се предполага, че са точки от пътя. Тези точки се използват по-нататък за векторизиране на пътя, т.е. построяването на графа от точките.

Определянеото на размерите на въпросната матрица става чрез сканиращ лъч по хоризонталата и по вертикалата. Ще разгледам само единият случай (по вертикалата), тъй като другият е аналогичен чрез транспониране на матрицата от черно-бели точки.

Нека предположим, че сме на колона X по хоризонталата. Започваме обхождане по вертикалата докато стигнем до черна точка. Запомняме положението на черната точка и продължаваме до стигането на бяла точка или до края на колоната. Запомняме дължината на черната ивица. След като изчерпим всички колони и приложим същия алгоритъм по редове, изчисляваме средната дължина на ивици от този тип. Средната дължина се взема като ширина/височина на матрицата, с която обхождаме на следваща стъпка.

5.2.2. Създаване на граф от точките, намиращи се в центъра на пътя

Досегашните трансформации не са били векторно ориентирани. Текущата трансформация също няма да бъде векторно ориентирана, а по-скоро многократно изчисляване на дължини между съседни точки и прилагане на алгоритъма на Дийкстра за намиране на най-кратък път между две точки в граф.

На вход алгоритъмът получава списък от точки, които са от същия тип като точките на графа. На изход имаме граф от тези точки.

Разстояние между две точки A(x1, y1) и B(x1, y1) ще наричам резултата от израза $\sqrt{(x1-x2)^2+(y1-y2)^2}$.

Дължина на път в граф наричам сумата на разстоянията между съседните точки от пътя.

Въвеждам понятието *околност на точка от пътя*, което дефинирам като множеството от всички точки, които са на разстояние не повече от фиксирана константа.

Дефинирам константа D, която ще е ограничителна мярка за намирането на околността на една точка от пътя.

Дефинирам константа LD (local distance), която е мяксималното разстояние между две точки, между които може да има ребро (D >= LD).

Ето и алгоритъмът, който ще приложим за фиксирана константа за разстояние D и променлива LD, която пробягва стойностите от 1 до D с някаква стъпка:

1. Намираме околностите на всяка една точка, използвайки константата D като мярка
2. За всяка точка от множеството на всички точки (нека я наречем “точка от пътя”):
    1. За всяка точка от околността на точката от пътя (да я наречем “околна точка”):

1. Свързваме точката от пътя с околната точка с ребро ако са налице всичките следващи условия:
   a) Разстоянието между съответните “x” и съответните “y” координати е не по-голямо от LD (тази проверка се прави, тъй като е целочислена и е по-бърза от изчисляване на разстоянието по горе-посочената формула. Ако тази проверка върне негативен резултат, продължаваме с изпълнението на цъкила без да продължаваме със следващите проверки)
   b) Няма ребро между тези две точки
   c) Разстоянието между двете точки е не по-голямо от LD
   d) Поне едно от следващите е изпълнено за най-краткия път между двете точки:
      1. Няма път между двете точки
      2. Съществува най-кратък път между пътната точка и околната точка и разстоянието между пътната точка и някоя от точките в намерения път, което е по-голямо от 2 * LD.
3. Извършване на оптимизация на графа. Оптимизацията се състои в премахването точка от графа, в случай че има точки, които са разположени “почти” на една линия. Изчислява се косинусът на ъгъла между всеки две ребра с общ връх и ако косинусът клони към -1 (с някаква константа за близост) тогава се премахва общата точка и двете ребра и се построява ново ребро, което е с краища – краищата на двете премахнати ребра.
4. Преизчисляване на околните точки на графа
5. Увеличаване на LD. Ако LD не е надвишило D се връщаме на 1.

Верността на алгоритъма се състои в това, че в никоя околност LD на точка от графа не може да има цикли. Това е и целта на алгоритъма – да свърже точките във вектори, които следват плавно линията на пътя. Доказателството на това твърдение, че няма цикли в дадената околност е по индукция.

1. Нека LD е 1 и няма път между никои две от точките. Условието е изпълнено
2. Допускаме, че на стъпка K, стойността на LD е LD и няма цикли в тази околност.
3. Нека сме на стъпка K + 1 и да допуснем, че сме добавили ребро между две точки и се е създал цикъл между тях. Тъй като точките от цикъла са в една околност, а ние обхождаме точките по двойки - всяка с всяка от една околност, то със сигурност на дадена стъпка е било добавено ребро между някои две точки и е бил създаден път между двете точки, който е бил по-кратък от 2x тогавашната

стойност на LD, която е била по-малка от текущата. Ако в този момент е било добавено ребро, то е било добавено, защото разстоянието между точките е било по-малко от разстоянието между тях по намерения най-кратък път, но това очевидно е невъзможно, тъй като ако разстоянието е по-кратко, то това ребро е трябвало да бъде добавено преди това и тогава най-краткият път не би бил текущо намереният. С това противоречие на допускането, че може да има цикъл в околност 2*LD е невярно, с което твърдението е доказано.

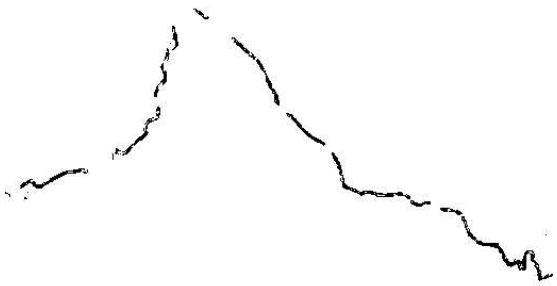
Когато се прилага алгоритъма за разпознаване на пътища върху най-детайлния мащаб на картата, се обхожда изображението като се работи с обекта MappedImage и не се зарежда цялата карта в паметта. За улеснение и оптимизация на обхождането е създаден обект MappedBufferedImage, който предлага методи за вземане на цвета на даден пиксел или за промяна на цвета на даден пиксел от най-детайлната карта. Обектът пази последно заредена правоъгълна част от картата и докато алгоритъмът не поиска точка от картата, която е извън последно запазената се работи върху текущия правоъгълник. По този начин се спестява памет от това да държи цялата карта в оперативната памет. Друга оптимизация, която е направена е, когато се обхожда голямото изображение и се прави грубия филтър (който резултира в ByteImage). При това обхождане не се минава от 0 до ширината на картата или от 0 до височината, ами се работи на "правоъгълник по правоъгълник", на които е разделена мащабната карта. По този начин се спестява постоянно четене от хард-диска за зареждане на нови правоъгълници.

Следващите няколко фигури показват последователните стъпки на разпознаване на пътя и части от черно-белите междинни резултати.

Фиг. 27. Изходно изображение, чийто път трябва да бъде разпознат

Фиг. 28. Потребителят е посочил няколко точки от пътя и околностите на цветовете на тези точки са оцветени в отличителен цвят.

Фиг. X. Процес на разпознаване на пътя: След прилагането на грубия филтър



Фиг. 29. Процес на разпознаване на пътя: След изчистване на шума

Фиг. 30. Резултат от разпознаването на пътя

5.2.3. Алгоритъм на Дийкстра

Използван е алгоритъмът на Дийкстра за намиране на най-кратък път между две точки. Сложността е сведена до nlog(n), тъй като се търси първия най-кратък път между две точки, а не всички най-кратки пътища.

1. На вход получаваме две точки (изходна и целева) и граф от точки
2. На изход получаваме списък с точки, които свързани в дадената последователност описват най-краткия път между изходната и целевата точка.

Ето и самият алгоритъм:

1. Следните структури от данни са налице:
   a) Хеш таблица с елементи, ключовете на които са точките от графа, а стойностите са минималните разстояния от изходната точка до тях. Нека тази структура я наречем *distances*. В началото всички стойности са "безкрайност", освен разстоянието от изходната точка до самата себе си, което е 0
   b) Списък от "свободни" точки (unsettledNodes), които все още се считат за недостигнати от изходната точка. В началото всички елементи без изходната точка присъстват в този списък.
   c) Списък от "намерени" точки – множеството от всички точки, които не са от списъка на "свободните". Този списък се състои само от изходната точка.
   d) Хеш таблица, на която ключовете и са точки от графа, а стойностите са точки от графа, които са предходни на ключчовете в най-краткия намерен път. В началото тази таблица е празна.
2. Докато има елементи в списъка със "свободните" точки
   a) Намира се елементът от списъка със "свободни" точки, който има най-малка стойност от структурата *distances* и се премахва от списъка със "свободни" точки
   b) Точката се добавя към списъка с намерени точки. Ако тази е целевата точка, излизаме от цикъла
   c) В противен случай преизчисляваме разстоянията в *distances*
      1. За всички съседни на намерената минимална точка, които са в списъка със "свободни" точки
         1. Ако разстоянието до съседна точка, според дефиницията на структурата *distances* е по-голямо от *разстоянието до минималната точка + разстоянието от нея до текущата съседна (*)*, тогава разстоянието до съседната точка в *distances* се преизчислява и се замества с (*). Предходната за точката от съседство се оставя да бъде текущата минимална точка.
3. Ако сме достигнали до крайната точка се тръгва назад по таблицата от предходниците и се връща списъка с точки от пътя в обратна последователност на обхождането по таблицата с предходниците.

5.3. Навигация и мащабиране по картата

Навигацията и мащабирането са действия на преизчисляване на координатите на правоъгълна част спрямо текущия мащаб на географската карта. Правоъгълната част отразява текущата част от картата, видима за потребителя.

Преди това ще се спра на алгоритъма, по който работи класа MappedImage за да достави поискана правоъгълна област, свързвайки няколкото от правоъгълните си разреза в едно.

На вход получава координати и размери на правоъгълна област (видима област) спрямо размерите на текущото мащабирано изображение. На изход трябва да върне обект от тип BufferedImage, който е отрязъка от текущия мащаб с посочените размери и позиция. Проверява се върху кои от правоъгълниците "стъпват" краищата на видимата област. Нека областта да е с координати и размери (X, Y, W, H), където X, Y са координатите на горния ляв ъгъл, а W, H са съответно ширината и височината.

1. Проверка дали поисканата видима област е в рамките на цялата карта и ако не е се променят така, че да се покаже максимална част от картата по малка или равна като площ от поисканата за областта.
2. Изчисляват се индексите (като колона и ред) на правоъгълниците, които са "застъпени" от точките (X, Y), (X+W, Y+H) по следната формула:
    1. Инкдекс по "x" = X (или X+W) целочислено разделен на <ширината на един правоъгълник>
    2. Аналогично за индекса по "y"
3. Зареждат се файловете с растерни изображения, които са за изчислените индекси (виж функцията MappedImage.generateZoomPartFileName) и се свързват да се получи едно правоъгълно изображение
4. Отрязва се частта, която е поискана от потребителя чрез зададените (X, Y, W, H), като преди това се преизчисляват само (X, Y) спрямо полученото от правоъгълниците изображение, т.е.:
    1. X = X - <X позицията на левия горен ъгъл на първия правоъгълник спрямо пълното мащабирано изображение>
    2. Y = Y - <Y позицията на левия горен ъгъл на първия правоъгълник спрямо пълното мащабирано изображение>

Това е операцията, която се прилага всеки път след преизчисляване на координатите на видимата част от картата.

За самата правоъгълна видима област са възможни за прилагане следните операции – транслация и преизчисляване на позицията на видимата част след мащабиране. Операцията по транслиране е тривиална. При нея се преизчислява само позицията на горния ляв ъгъл и се прави проверка дали правоъгълникът не излиза от размерите на цялото изображение.

При мащабиране са нужни координатите на правоъгълника в стария мащаб и размерите на текущата карта в новия мащаб. Точката, чиито координати се калкулират е центърът на видимата част. Новата позиция на тази точка върху картата от поискания мащаб се изчислява от старата позиция на централната точка така:

x = (<стара X>*<ширина на картата в текущия мащаб>)/<ширина на картата в стария мащаб>

y = (<стара Y>*<височина на картата в текущия мащаб>/<височина на картата в стария мащаб>

5.4. Управление на обекти – добавяне, изтриване, премахване

Управлението на обектите, включващо добавяне, изтриване и премахване на обекти са операции, които добавят, премахват или променят елементи от списък. Списъкът съдържа всички обекти за текуща карта. При редактиране на обект, операцията по нанасянето на промените на място в списъка е реализирана като премахване на стария обект и добавяне на нов, който има свойствата на стария с нанесени корекции по някои от атрибутите му.

Особена част в управлението на обектите е добавяне на обект, тъй като ако не е указана позиция по оста Z, трябва този обект да се постави на най-горно ниво, т.е. “върху” всички останали “под” него. За целта се обхождат всички обекти от картата, които “застъпват” областта на обекта, който се добавя. Те се сортират по позицията им по оста Z. Позицията на новия обект се изчислява като *<максималната позиция + 1>*.

Един обект се застъпва с друг обект, когато правоъгълните области, определени от двата обекта имат общи точки: проверява се дали някоя от страните на единия правоъгълник пресича другия или се съдържа в него. Ако поне едно от тези условия е изпълнено, тогава двата правоъгълника се “застъпват”.

5.5. Управление на събитията за обектите на картата

Събитията са атрибути от определен тип, асоциирани към даден обект. Когато настане събитие, то се идентифицира по два критерия – име на събитието и позиция, където става то.

Когато възникне събитието, се намират всички обекти, за които точката на събитието е точка от правоъгълната им област и се подреждат по позиция по оста Z. Обектите се обхождат по реда им по оста Z, започвайки от този с най-голяма стойност до този с най-малка. При първия обект, който има асоциирано събитие със същото име се спира. Изпълнява се действието, определено за събитието с аргументите, които са стойност на атрибута за събитие (в случай, че са нужни допълнителни аргументи).

Пример: Област/обект върху картата може да има събитие, което се казва "когато се достигне максималното ниво на мащаб". Действието, което се изпълнява при такова събитие е зареждане на нова под-карта. Идентификаторът на картата, която трябва да бъде заредена е записан като стойност на атрибута за това събитие.

5.6. Търсене измежду обектите по картата

Търсенето измежду обектите по картата е линейно и се извършва в паметта, тъй като са заредени всички обекти за дадена карта. Не се правят заявки до базата данни, тъй като в този момент може потребител-редактор да публикува промени по текущо разглежданата карта и да се получат аномалии в клиентското приложение на потребителя, който работи с картата.

Има два критерия за търсене:

1. Търсене по текст
2. Търсене по списък от категории

Търсенето по текст представлява търсене измежду стойностите на атрибутите на текущите обекти.

Търсенето по списък от категории е обхождане на всички обекти и показването само на тези, които имат категория измежду посочен спитък от идентификатори на катеогирии (икони).

Потребителският интерфейс предлага функционалност за търсене в групи от категории, но всъщност в сървърната част това представлява търсене по списък от идентификатори на няколко категории. Единствено и само клиентското приложение "знае" за съществуването на групи от категории. Когато потребителят избере група

от категории, клиентът предава на сървъра списък от идентификатори. Ако потребителят е избрал една категория се изпраща само един идентификатор.

5.7. Намиране на най-кратък път между два обекта

Намирането на най-кратък път между два обекта се състои в намирането на най-близките до двата обекта точки от графа. След това се извършва търсене на най-кратък път между намерените такива. *Най-близки точки* дефинирам като такива, на които разстоянието от центъра на правоъгълната област на обекта до тях е най-късо.

Намереният път се визуализира на картата чрез отличителен цвят.

6. Инсталация

Изискването за инсталация на текущата дипломна работа изисква уеб-сървър Apache Tomcat 4+, MySQL 3+, JRE1.5.0 и уеб-браузър, който има инсталиран плъгин за Java.

Заедно с текущата документация идва и носител на информация, който съдържа проекта. Следните стъпки трябва да се направят за инсталацията му:

1. Създава се база от данни, в която се зареждат данните, които се намират във файла Database.sql
2. Копира се цялото дърво на проекта в мястото в уеб-сървъра, където стоят други уеб-проекти (напр. webapps/DynaMap)
3. Редактира се файла за конфигурация на проекта – WEB-INF/web.xml и се описва начина, по който може проектът да се свърже към базата данни
4. Редактират се стойностите на таблицата “config”, които имат следното значение:
    1. MapsDirectory – абсолютен път до директорията, където се записват физически добавените географски карти
    2. ViewerWidth – ширината по премълчаване на видимата част на картата
    3. ViewerHeight – височината по премълчаване на видимата част на картата
    4. IconsDirectory – абсолютен път до директорията, където се намират иконите за категориите
    5. IconsWebDirectory – не се използва в тази версия
    6. PreferredAreaWidth – ширина на област (обикновено същата като ширината на една икона)
    7. PreferredAreaHeight – височина на област (обикновено е същата като височината на икона)

Файлът в главната директория “index.html” съдържа аплета, който позволява работа с картата.

7. Указания за работа

7.1. Инсталация на нова карта в системата

Инсталацията на нова карта става чрез инструмент, който се изпълнява в конзолна среда. За целта отидете в главната директория на проекта и изпълнете:

За Linux: *java tools/MapBuilder <списък от параметри>*

За Windows: *java tools\MapBuilder <списък от параметри>*

За указания какви параметри трябва да се подадат изпълнете командата без параметрите и ще бъде изведена информация за това.

7.2. Разпознаване на пътищата от инсталирана карта

Инструментът, който се използва за разпознаване на пътища се стартира с:

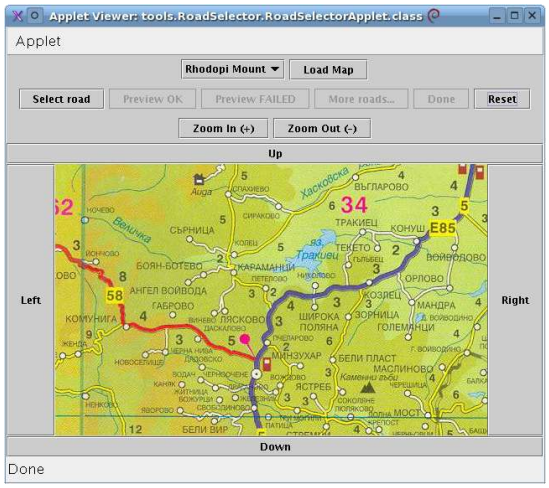
appletviewer selector.html

Файлът selector.html се намира в главната директория. Преди стартиране трябва да се редактира този файл и да се попълнят настройките за свързване към базата данни. След като се стартира аплета следвайте следните стъпки:

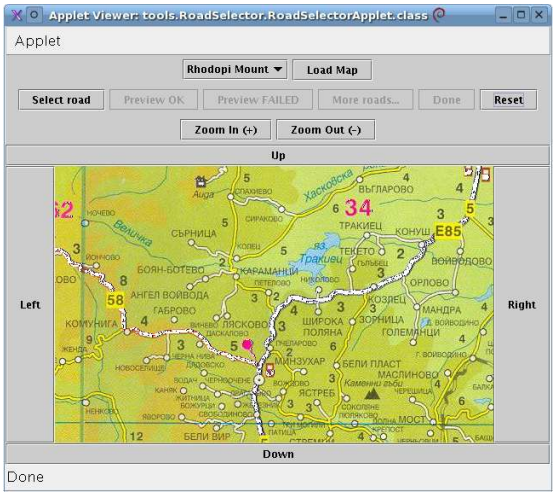
1. Навигирайте върху картата докато се намери положение, където ясно се виждат пътища на картата с отличителни цветове
2. Щракнете върху пътя и след секунди приложението ще отбележи и други точки, които са с "близък" до посочения от вас цвят. Ако се щракне погрешно върху цвят, който не е от пътя това може лесно да се върне като се щракне с десния бутон върху картата.
3. Повторете стъпката с посочването на точки от пътя, докато пътят бъде добре покрит от отличителното оцветяване.
4. Щракнете върху бутона "Select road" и изчакайте
5. Ако резултатът е задоволителен щракнете "Preview OK", в противен случай – "Preview Failed"
6. След като резултатът е задоволителен щракнете бутона "Done" и би трябвало до няколко минути (зависи от размера на картата и гъстотата на пътищата по нея) резултатите от пътищата да бъдат записани в базата от данни.

ВНИМАНИЕ: Всички стари пътища, асоциирани към тази карта ще бъдат изтрити от базата данни!

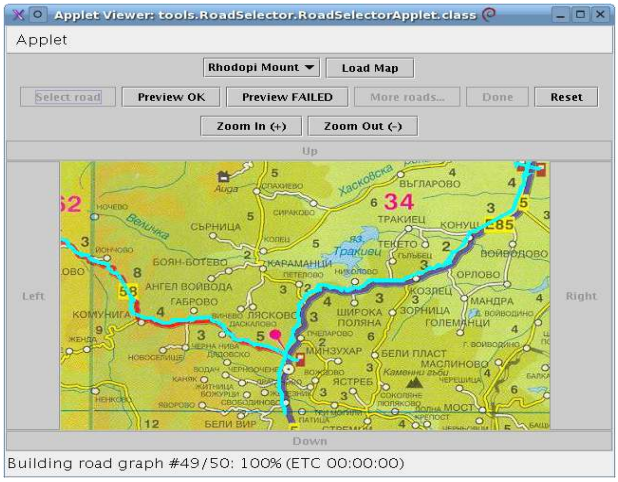
Фиг. 31. Разпознаване на пътища: Навигиране по картата докато се намери подходящ изглед на пътя



Фиг. 32. Разпознаване на пътища: След селекция на няколко точки от пътя



Фиг. 33. Разпознаване на пътища: Резултат от разпознатите пътища.



7.3. Работа с картата от гледна точка на краен потребител

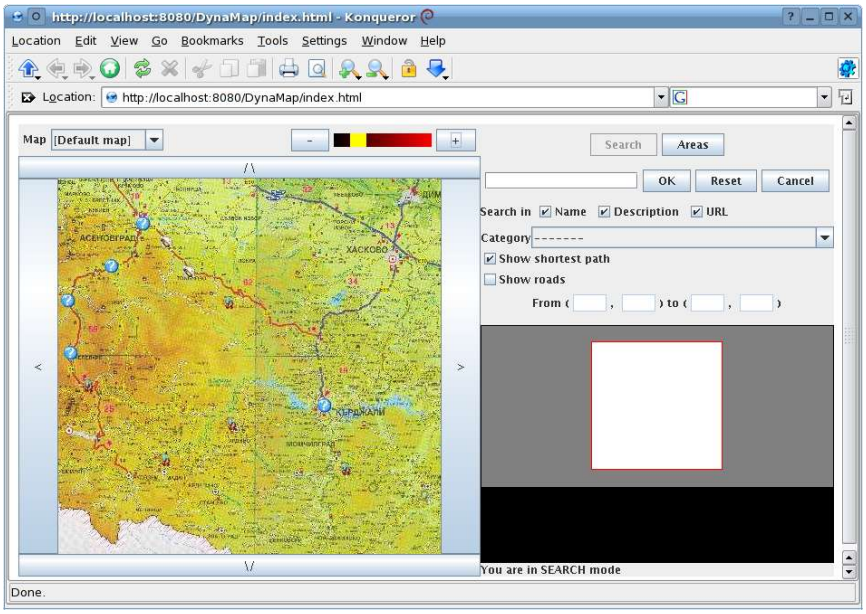
Стартирайте следния адрес в уеб-браузър, който поддържа Java:

http://<сървър>/<път_до_инсталирания_проект>/index.html

Пример за такъв адрес е http://localhost:8080/DynaMap/index.html.

След отваряне на този адрес ще се зареди аплет, който дава възможност за навигация по картата и локализиране на обекти.

Фиг. 34. Навигация по картата



Показаният аплет съдържа 3 основни панела с контроли – панел за навигация и мащабиране; панел за избиране на карта; панел с контроли за търсене. Под панела за търсене стои изображение, което показва къде се намира видимата част на картата спрямо текущо видимия мащаб. Под това изображение стои надпис указващ в какъв режим работи аплета в момента. На посочената фигура се намираме в режим "търсене".

7.3.1. Навигация

Навигацията по картата се осъществява по няколко начина – чрез контролните бутони или чрез мишката. При натискане на бутоните за посока видимата част на картата се измества в съответната посока. При натискане на бутоните за мащабиране ("+" и "-") се превключва към съответен на текущия мащаб

– следващия по-детайлен или по-малко детайлен. Оцветената площ между двата бутона за мащабиране е направена за удобство на потребителя ако има нужда от минаване на произволно ниво на мащаб.

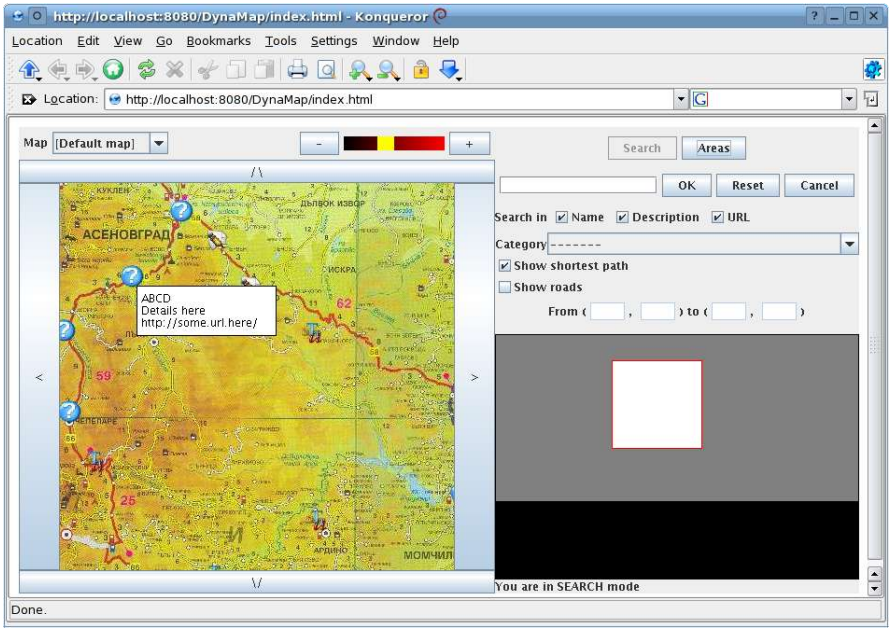
При щракване с левия бутон на мишката някъде върху картата се извършва мащабиране до следващото по-детайлно ниво и центърът на видимата част се придвижва върху позицията, която е била щракната (в случай, че центърът може да бъде разположен в средата на видимата част, освен ако не е щракнато някъде по краищата на картата). Аналогично при щракване с десния бутон се минава на следващото по-малко детайлно ниво и центърът на видимата част се придвижда върху щракнатото място.

Текущата карта може да бъде сменена като се посочи друга карта от списъка от текущите географски карти в системата, който е наличен в горния ляв ъгъл на аплета.

7.3.2. Търсене и локализация на обекти

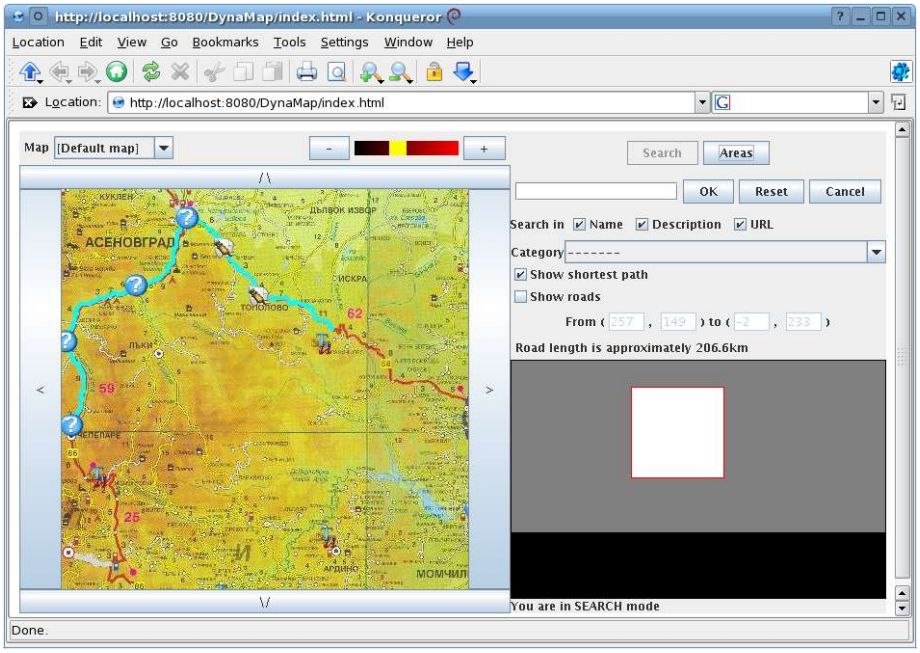
Върху картата има поставени обекти от потребител-редактор. При движение с мишката върху някои от тях се появява правоъгълник с информация за обекта – име, описание, уеб-адрес. За отваряне на уеб-адреса трябва да се щракне върху съответния обект със задържан SHIFT клавиш. Адресът се отваря в нов прозорец на браузъра.

Фиг. 35. Информация за обект върху картата



Намирането на най-краткия път между два обекта (ако има такъв) става с щракването върху съответните два обекта с натиснат клавиш CTRL. Аплетът ще направи заявка към сървъра, който ще върне резултат с най-краткия път между точките. Координатите на изходната и целевата точка се изписват в полетата предназначени за това. След изобразяването на пътя системата връща и съобщение с приблизителната дължина на пътя в километри.

Фиг. 36. Намиране на най-кратък път



Търсенето на обекти става, използвайки контролите в дясната част на аплета. Търсенето е по два критерия – текст и категория, които трябва да бъдат удовлетворени едновременно. Търсенето по текст може да се стесни до търсене само в определени атрибути на обектите – име, описание, уеб-адрес. Търсенето по категория може да бъде и търсене по група от категории. За целта трябва да се посочи групата от категориите, които трябва бъдат намерени и системата ще покаже само тях. Ако не е посочен текст за търсене или не е посочена определена категория, то този критерий няма да се използва при търсене.

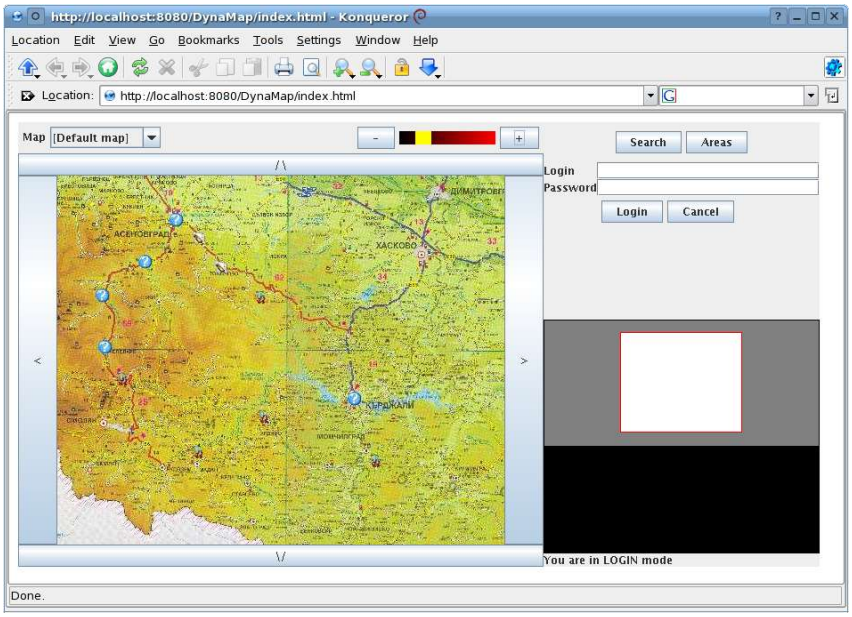
Същетсвуват обекти, които имат определена т. нар. под-карта. Тези обекти имат леко променена иконка, която е отличителна от другите. Достигането на под-картата "минвавайки през обекта" става като се увеличи мащаба на картата до максимален размер и се щракне върху обекта, който е с под-карта. Ще се отвори

под-картата. При обратно мащабиране от най-малко детайлното ниво на под-картата се връщаме в най-детайлния мащаб на изходната карта, като в центъра на видимата част стои обектът, през който сме "минали" за достигане на тази под-карта.

7.4. Управление на обектите по картата като потребител-редактор

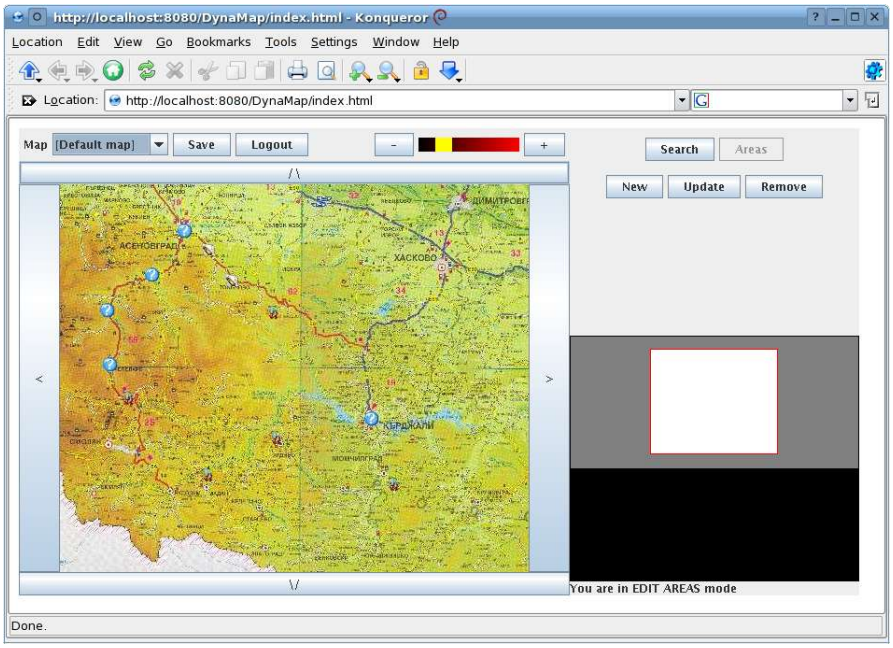
Отново както при работата с картата като краен потребител трябва да се зареди адреса с въпросния аплет. За да се редактират обектите трябва да се влезе в режим за редактиране. За целта трябва да се щракне бутона "Areas". Ще се появи прозорец с полета, в които трябва да се напише име и парола, които се намират в таблицата "users" в базата данни.

Фиг. 37. Влизане в режим на редактиране



След въвеждане на коректна комбинация от име и парола потребителят-редактор е допуснат до режим на редактиране на обектите. Появяват се още няколко бутона - "Save", който записва промените в базата данни (публикуване); "Logout" за излизане от режим на редакция; на мястото на полетата за име и парола се появяват три бутона - "New", "Update", "Remove", които определят трите режима за редактиране на обектите.

Фиг. 38. В готовност за редактиране на обекти.



Добавянето на нов обект става като първо се влезе в режим “добавяне”, т.е. с натискане на бутона “New”. Появяват се полета, които трябва да съдържат данните, с които да се добави обекта. Полетата за координати на обекта се попълват автоматично, когато се щракне някъде върху картата. Послетата за име, описание и уеб-адрес се попълват от потребителя-редактор. Избират се категория и под-карта. Изборът на под-карта не е задължителен. Обърнете внимание, че една карта може да бъде под-карта на точно един обект. След като се въведат всичките необходими данни се натиска бутона “ОК”, след което обектът се появява върху картата. Ако потребителят-редактор не публикува промените, то при затварянето на прозореца на браузъра промените няма да се запишат в базата.

Редактирането на съществуващ обект се извършв по подобен начин. Натиска се бутонът “Update”, за да се влезе в режим на редактиране. След това с мишката се щраква върху съществуващ обект. След това процедурата е същата като при добавянето. Щракване на мишката някъде върху картата променя координатите на обекта. След попълване на всички необходими полета се натиска бутона “ОК”.

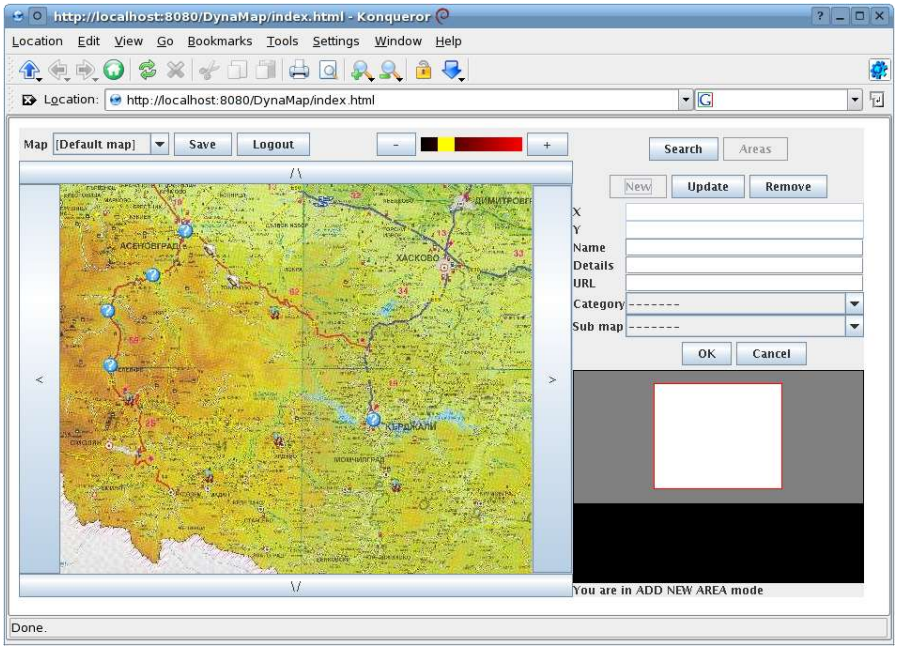
За изтриване на сущетсвуващ обект трябва да се щракне бутона “Remove” за влизане в режим “изтриване”. Щраква се върху обекта, който трябва да бъде изтрит и след това се натиска бутон “ОК”. Иатриването е направено нарочно “неудобно”, за

да не се случва неволно.

Излизането от всеки един от тези режими – добавяне, редактиране, изтриване става с натискането на бутона “Cancel”.

След свършването на редакцията по обектите трябва да се публикуват промените с натискането на “Save”.

Фиг. 39. Добавяне на нов обект



8. Тестове

Тестовете на ситемата са разделени на две групи:

1. Тестове при инсталация на картата – добавяне и разпознаване на пътища
   a) Добавяне на картата
      1. Проверка дали при невалидни или липсващи входни данни приложението ще покаже необходимаата за това грешка и ще прекрати работа
      2. Проверка дали генерираното изображение е способно да бъде разглеждано при навигация (използвайки аплета като тестова постановка).
      3. Проверка дали липсват граничните части на картата
      4. Проверка дали картата е записана на разбитите си правоъгълници в директория, чието име е идентификаторът на картата в базата от данни
   b) Разпознаване на пътища
      1. Проверка дали разпознаването на пътища работи по процедурата описана по-горе.
      2. Проверка дали разпознатите пътища не се влияят от прекъсвания - етикети с номера на пътя или магистралата, други знаци по картата, които са върху пътищата
      3. Проверка дали най-кратък път между два обекта може да бъде намерен след като предварително е видяно, че този път е разпознат от системата (използване на аплета)
2. Тестове на работата по картата – навигация и управление на обекти
   a) Навигация и търсене по картата
      1. Проверка дали навигацията по картата стига само до границите и не продължава по-нататък
      2. Проверка дали границите на картата са видими при навигиране по нея
      3. Проверка дали при мащабирането средната точка на видимата част остава на едно и също положение върху картата
      4. Проверка дали мащабирането, използвайки областта за избиране на произволно ниво на мащабиране работи коректно
      5. Проверка дали мащабиране, използвайки мишката (ляв и десен бутон) работи коректно – средната точка на видимата част трябва да бъде преместена там, където е било щракнато с мишката, освен в случай, че е стигната границата на картата и това не е възможно
      6. Проверка дали е успешно отварянето на уеб-адреси на обекти при валиден

уеб адреса

7. Проверка дали намирането на най-кратък път работи по посочения начин и дали това наистина е най-краткият път. Дължината на пътя трябва да бъде указана след намирането на пътя.
8. Проверка дали изображението, показващо положението на видимата част от картата спрямо цялата карта е коректно
9. Проверка дали търсенето само по текст работи успешно в зададените му атрибути на обекта
10. Проверка дали търсенето само по категория работи по зададена категория или група от категории
11. Проверка дали търсенето работи коректно при зададени текст и категория или текст и група от категории

b) Управление на обектите по картата

1. Проверка дали при невалидно име и парола потребителят е допуснат до режим на редакция
2. Проверка дали при валидно име и парола потребителят-редактор може да редактира обекти
3. Проверка дали бутонът “”Logout” извежда потребителя-редактор от режим на редактиране
4. Проверка дали при добавяне и редактиране на обект може да бъде добавен обект, на когото липсват някои от полетата (изключение прави под-картата)
5. Проверка дали обектът се добавя, когато се натисне бутона “ОК”.
6. Проверка дали добавянето на обект с под-карта работи коректно и след това тази под-карта е видима през този обект.
7. Проверка дали обектите с под-карти имат по-различна иконка
8. Проверка дали редактирането на обект не добавя нов обект, вместо да редактира същетсвуващ
9. Проверка дали изтриването на обект изтрива точно обектът, който е посочен
10. Проверка дали при добавяне на обекти един върху друг може да се премести няй-горния

9. Възможни бъдещи подобрения

1. Разработване на HTML версия
2. Добавяне на повече карти
3. Добавяне на повече икони
4. Възможност за наличието на различни по клас пътища
5. Възможност за описване на пътя – през кои други обекти минава
6. Редактор за потребители в системата
7. Редактор за конфигурация на таблицата с глобалните променливи
8. Редактор за управление на категориите и групите от категории
9. Възможност за групиране на картите
10. По-удобна навигация
11. Анимиране на местенето на видимата част на картата при навигация и мащабиране
12. Възможност за показване на версия за принтиране

10. Използвани материали

1. http://en.wikipedia.org/wiki/YCbCr – дефиниция на цветотово пространство
2. http://renaud.waldura.com/doc/java/dijkstra/ - алгоритъм на Дийкстра за намиране на най-кратък път
3. http://java.sun.com/ - документация на функции за обработка на растерни изображения, за рисуване по контроли, управление на контрол.