Софийски Университет "Св. Климент Охридски"
Факултет по математика и информатика

Дипломна работа
на
Панайот Пламенов Даскалов

PCMT, M21934

Ръководител
проф. Христо Кабакчиев

Тема:
Метод за обработка на данни от навигационни устройства в мрежа на мобилен оператор

- София -
2007

СЪДЪРЖАНИЕ

СЪДЪРЖАНИЕ на фигурите

Въведение

Настоящата дипломна работа описва система за комуникация, контрол и управление на навигационни устройства. Системата обработва и съхранява данните, получени от тях по различни комуникационни канали, като параметрите на тези процеси се определят в интерактивен режим пряко от потребителя в зависимост от конкретните нужди на неговото приложение. Практически системата е реализирана и се използва в Web базираната услуга, наречена **Follow Me on the Web** на фирма Мултипроцесорни Системи ООД, България.

Транспортът винаги е бил жизнено важен аспект от живота на човека, както в миналото така и в настоящето. Хора, стоки и суровини непрестанно имат нужда от него, за да достигнат от едно място на друго. Бързината и високите разходи са основните проблеми на всеки тип транспорт, а за да бъдат те преодоляни, е необходима строга и много прецизна организация, както и мощни средства за контрол и управление. Съвременните транспортни фирми и организации са огромна и сложна смесица от хора, техника, товари и др., която човек трудно може да осмисли, а още по трудно да контролира и следи. Нуждата от средства за контрол и управление на превозни средства стои не само пред транспортните фирми. От това се нуждаят и охранителни фирми, малки и големи предприятия притежаващи авто-парк, който желаят да контролират ефективно. Тези проблеми пораждат и нуждата от достъпна и ефективна

система за централизиран контрол и управление на транспортните средства.

Такива системи са комбинация от вградени в автомобилите устройства за позициониране и контрол и компютърни системи обработващи данните идващи от тях с цел да ги съхранят и впоследствие да ги предоставят във вид лесен за възприемане и обработка от човек или друга компютърна система осъществяваща някакъв вид дейност базирана на тези данни.

За да изпълнят високите изисквания на днешното време към тях, компютърните системи за контрол и управление на превозни средства (КУПС), се нуждаят от постоянен, надежден и високоскоростен канал и метод за обмен на данни с навигационните устройства инсталирани в превозните средства. Този канал е важна и неизменна част от една такава система, служещ освен за събиране на данни, но и за дистанционен контрол на целия автомобил чрез навигационното устройство. За да бъде правилно оценена неговата роля в една такава система, е необходимо да се познава нейната същност и проблемите, който възникват при изграждането й. Затова в тази глава съм си позволил да опиша накратко значението на компютърните системи за КУПС и технологиите които те използват, както и нуждата от усъвършенстване на съществуващите такива системи.

Дипломната работа е структурирана по следния начин –

Въведение – описва проблема, който се решава; структурата на системи за КУПС и начините за комуникация и позициониране използвани в тях.

Цели и задачи – поставя изискванията към метода за обмен на данни, за да може той да реши проблемите споменати в предходната глава; описва задачите, които се поставят в отговор на тези изисквания.

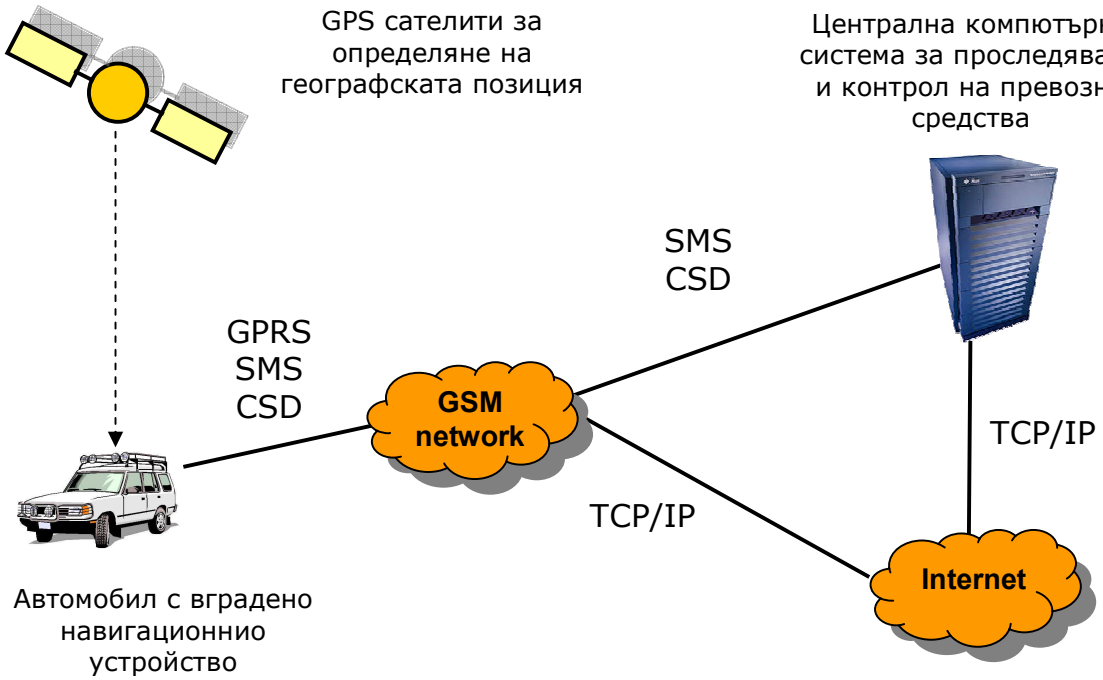
Структурно описание – представя процеса на проектиране на всеки един компонент от софтуерния модул реализиращ метода за обмен на данни.

Описание на имплементацията – описва конкретната реализация на софтуерния модул направена следвайки проекта от предходната глава.

Компютърни системи за управление на флот от превозни средства

Глобализацията в 21-ви век е процес, който бавно, но сигурно променя облика на нашия свят и начина, по който живеем. Едно от малкото добри неща, които тя носи след себе си, е възможността човек да получава все по-добри стоки и услуги, покриващи неговите индивидуални предпочитания и изисквания. Зад всичко това обаче стои една огромна машина от хора, компании, стоки, превозвачи и др., които трябва да намерят общ език и бърз и ефективен начин за сътрудничество и организация, за да осигурят на хората това, от което те се нуждаят. В технологичната ера, в която живеем, какъв по-добър начин за организация и контрол на превозните средства би могло да има от компютрите и системите, които могат да се изградят с тяхна помощ?

Системите за управление на флот от превозни средства не са нещо фундаментално ново. Напротив. Като всяко друго нещо те са еволюирали от трудемкото и доста неточно записване на ръка на маршрути, изминати километри, разход на гориво и т.н до Интернет базираните системи за контрол на превозни средства 24 часа в денонощието 7 дни в седмицата, които виждаме днес.



Фиг. 1 Примерна схема на Интернет базирана система за КУПС

Една такава примерна система би изглеждала по начина показан на фиг.1. Географската позиция на автомобила се определя чрез GPS сателити и вграден в навигационното устройство GPS приемник. След това тези данни могат да се предадат веднага към централния сървър на системата по един от показаните канали на GSM мрежата, чрез вграденият в устройството GSM модул. Друга възможност е данните да се натрупат в устройството и след това да се пратят накуп към централния сървър, вариантите са много и зависят от изискванията на приложението. От своя страна сървърът също може да изпраща данни към устройството, представляващи някакви команди. Данните могат да се обменят под формата на текстови съобщения, по един или няколко канала паралелно.

Компютърните системи за контрол и управление на превозни средства са в състояние да покрият високите изисквания за бързина, точност и простота на използване, които съвременните условия на живот им поставят. Някой от техните ключови предимства са:

• **централизираност** – с използване на някой от позициониращите методи, разгледани в следващата точка и възможностите за предаване на данни, които съвременните GSM мрежи предлагат, една компютърна система за КУПС дава възможност за съхранение и обработка на данните за практически неограничен брой превозни средства на едно единствено място.

• **точност на отчитане и драстично намаляване на раходите по транспорта** – вградените устройства за позициониране, освен възможност за снемане на местонахождението и предаването на тези данни на централна компютърна система, могат да включват и възможност за точно аналогово или цифрово измерване и контрол върху различни показатели на превозното средство (изминати, километри, разход на гориво, ниво и налягане на маслото, температура на двигателя и т.н). Отчитането и съхраняването на тези данни се прави без каквато и да било човешка намеса, което елиминира възможността за недоброжелателното им модифициране, като по този начин спестява изключително много средства на работодателите.

• **увеличаване на ефективността** – знаейки точното местонахождение и състояние, по всяко време, на всички превозни средства една компютърна система за КУПС дава възможност за извършването на прецизна логистика, а следователно и по-ефективна работа на превозните средства и повече доволни клиенти.

• **достъпност** – ползването на Интернет дава възможност на доставчиците и разработчиците на подобни услуги да ги предоставят за ползване и на малки и средни фирми, които не се занимават с транспорт, но имат нужда от евтино и ефективно решение за контрол на техния авто парк. Интернет елиминира нуждата от закупуването и инсталирането на скъпоструващ софтуер, който доскоро е бил приоритет само на големите транспортни фирми.

Компютърните системи за КУПС обаче не решават само проблемите на съвременния транспорт, но могат да се използват също толкова ефективно и в други сфери на живота. Например, знаейки географската позиция на дадено превозно средство и имайки постояннен канал за обмен на данни с него, тази компютърна система веднага може да бъде превърната в система за охрана на превозни средства. Тя би позволила на една охранителна фирма лесно, бързо, ефективно и без големи разходи да открива откраднати коли и да ги контролира дистанционно. Броят на възможностите за приложение е буквално неограничен.

Технологии за позициониране

При една съвременна система за контрол на транспортни средства, да знаеш географската позиция на всяко едно от тях във всеки един момент от времето е жизнено важно, тъй като цялата работа на една такава система се базира на тази информация. Без нея контролът и логистиката на транспорта не биха били възможни.

Избор от технологии за позициониране в днешно време не липсва. Има два основни начина за позициониране, достъпни на пазара в момента, това са:

- позициониране базирано на GPS системата (или на руския й еквивалент GLONASS – Global Navigation Satellite System, както и на новосъздадения европейски проект Galileo)

- позициониране базирано на мобилните телефонни мрежи

Съществуват и комбинации от тях, които увеличават точността на позициониране и намаляват крайната цена за потребителя, но за нашите цели нека само се спрем накратко на всяка една, за да добием представа как функционира.

Global Positioning System (GPS)

GPS системата е създадена от американското военно министерство и е пусната в действие през 1994г. Тя се състои от 24 сателита и 5 основни базови станции, като е планирана подмяната на съществуващите с 21 сателита от ново поколение. От самото начало е било предвидено сигналът да може да бъде използван, както от военни така и от цивилни потребители, но точността на цивилните приемници е била по-лоша от военните поради умишленото въвеждане на така наречената "избираема наличност" (Selective Availability - SA). През 2000г. с решение на президента SA е премахната, вследствие на което цивилните приемници драстично са увеличили своята точност *(El-Rabbany, 2002)*. Въпреки това, никога не е била давана абсолютна гаранция, че в ситуации, които американското правителство счете за извънредни, то няма да изключи достъпа на останалите до GPS сателитите. Поради тази причина Русия отдавна има своя навигационна система работеща по подобен начин, наречена GLONASS, а Европейския Съюз планира скорошно пускане на своя сателитна система наречена „Галилео.

GPS сателитите предават на две основни носещи честоти – L1 на 1.575GHz и L2 на 1.2GHz. Определянето на позицията става по следния начин. Сигналът от GPS сателитите се приема на Земята от специални GPS приемници, след което знаейки позицията на сателита в космоса (защото орбитата му и времето са известни) и разстоянието от приемника до сателита (защото времето за достигане на сигнала от сателита до Земята е известно) приемника може да изчисли така наречената дъга на разстоянието от него до сателита. Имайки дъгите на разстоянието за поне четири GPS сателита, приемник е в състояние да определи пресечната точка на четирите, която представлява и позицията му на земната повърхност *(El-Rabbany, 2002)*. Чрез използване на сложни математически алгоритми за коригиране на грешките при тези изчисления, този метод позволява не само определяне на позицията с висока точност, но и измерване на времето, скоростта и надморската височина на обектите.

GPS технологията е най-честият избор във всяко едно приложение, което изисква позициониране, най-вече защото ползването и е безплатно. Естествено тя не е съвършена. GPS сигналът е доста е слаб, което го прави трудно различим от шумовете и отразените от околните предмети и препятствия сигнали. От тук идва и един от големите недостатъци на GPS системата, а имменно лошата, а понякога дори невъзможна, работа в затворени помещения, тунели, сенки от високи

сгради и др. . Въпреки това, в днешно време GPS приемниците са доста усъвършенствани и с помоща на високо чувствителни антени и алгоритми за корекция на грешките успяват да увеличават точността на позициониране до голяма степен.

При изграждане на компютърна система за КУПС, използването на GPS като технология за позициониране е особено подходящо, заради безплатното му ползване, задоволителна висока точност и не на последно място огромния избор на хардуер на пазара подържащ GPS технологията.

Позициониране чрез клетъчната мрежа (Network ID based positioning)

Друг популярен начин за позициониране на обекти е чрез използване позицията на клетките в мобилна телефонна мрежа, като отправни точки за определяне на географската позиция.

Четирите най-популярни технологии, използвани за изграждане на мобилни телефонни мрежи са: GSM (Global System for Mobile Communications), CDMA (Code Division Multiple Access), WCDMA (Wideband CDMA), TDMA (Time Division Multiple Access). Независимо коя технолгия се използва, архитектурата на мобилната мрежа е винаги клетъчна, т.е територията, която се покрива от мрежата е разделена на региони с определен радиус наречени клетки. Всяка клетка има секционирана антена (най-често на три части от по 120 градуса), които осъществяват комуникацията между мобилните терминали (най-често мобилни телефони) и базовата станция, която след това предава данните към фиксираната телефонна мрежа или към Интернет. За да функционира правилно мобилната мрежа трябва да знае постоянно в коя клетка се намиран даден мобилен терминал, за да може да пренасочва коректно данните идващи за него. Затова мобилният терминал се регистрира всеки път, когато премине от една клетка в друга. Този преход се записва в така наречения HLR (Home Location Register) или VLR (Visitor Location Register) на мобилната мрежа, като скоростта на обновяване на тези данни понякога е необходимо да е не повече от милисекунда *(Добош, 2005)*.

Знаейки по всяко време, в коя клетка и в кой нейн сектор се намира даден мобилен терминал и нейния уникален номер CGI (Cell Global Identity) мобилният оператор е в състояние да определи позицията на терминала с точност от 100м до 35км, в засимост от радиуса на покритие

на клетката. Очевидно е, че тази точност в повечето случай се оказва недостатъчна, затова тя се комбинира с техниката на предварително синхронизиране (Timing Advance), която позволява да се определи разстоянието между мобилният терминал и базовата станция на базата на времето необходимо на сигнала да достигне от единия до другия. При тази комбинация точността на позициониране става средно около 500м *(Добош, 2005)*. Но за последното е необходимо специално оборудване, което не всички клетки и мобилни оператори притежават.

Позиционирането чрез клетъчната мрежа е подходящо за използване в приложения, предлагащи информация за местоположението в големи градове, където гъстотата на клетките е много голяма, следователно и точността на позициониране се увеличава. Тогава клиентите не е необходимо да закупуват допълнителен хардуер или софтуер, а могат просто да ползват стандартните си мобилни телефони. Ниската точност на повечето места и честото нежелание на мобилните оператори да разкриват точното местоположение на своите клетки, прави тази технология неподходяща за използване в компютърни системи за КУПС, където точността на позициониране е от решаващо значение.

Методи за обмен на данни в мобилна мрежа

При разглеждането на компютърните системи за КУПС, видяхме, че една такава система се нуждае от специално навигационно устройство, инсталирано в превозното средство, което да осигури информация за позицията и дистанционен контрол на различни параметри по автомобила. За тази цел то трябва да е в състояние да обменя данни по някакъв канал на мобилната мрежа с централна система за контрол и проследяване. Данните, които се обменят са текстови съобщения в специфичен формат различен за всеки вид устройство, който се определя от фирмата производител.

В случая ни е необходим канал и метод за обмен на ASCII текстови съобщения, съдържащи позицията на превозното средство определена чрез GPS приемник и предавана във формат NMEA (National Marine Electronics Association), както и за обмен на команди към устройството и за контрола резултатите от изпълнението на такива команди. Подобен вид комуникация, независимо в каква среда се извършва, се нарича Machine to Machine (M2M).

В средата на мобилна телфонна мрежа можем да използваме следните три канала, по които да обменяме необходимите ни текстови съобщения:

- SMS (Short Message Service)
- GPRS (General Packet Radio Service)
- CSD (Circuit Switched Data)

Повечето действащи компютърни системи за КУПС, използват SMS като основен канал и понякога CSD като резервен. Ще разгледаме накратко всяка една от трите възможности като изтъкнем предимствата и недостатъците й за нашата цел.

Short Message Service (SMS)

Една от най-популярните услуги, които всеки един мобилен оператор предлага е възможността за обмен на кратки текстови съобщения между мобилни телефони - SMS. Едно SMS съобщение има максимален размер от 256 символа, но стандартът позволява и изпращане на съобщение с по-голям размер, което след това се разцепва на няколко по-малки така, че да се вмести в ограничението на единично съобщение.

Както споменахме, за да функционира една компютърна система за КУПС е необходим метод и канал за обмен на данни във формата на текстови съобщения между навигационните устройства и централния сървър на системата. Освен географски данни за позицията на обекта, от GPS приемниците може да се получи и друга полезна информация като брой сателити, към които е фиксиран приемника в момента, аларма за навлизане или напускане на определен от потребителя географски регион и много други, които се добавят като изходни данни към NMEA съобщението с позицията. В NMEA, съобщението съдържащо минимума информация необходим за точното определяне позицията на даден обект се нарича $GPRMC и има дължина от 60-70 символа, което означава, че в едно SMS съобщение може да се предадат 3-4 $GPRMC съобщения. Останалите съобщения от NMEA стандарта също имат големина до 80 символа. При подходящ дизайн на протокола на съобщенията между устройството и централния сървър, потребителските команди, които то поддържа също могат да се пращат за изпълнение към него, като съобщения с дължина по-малка от 256 символа.

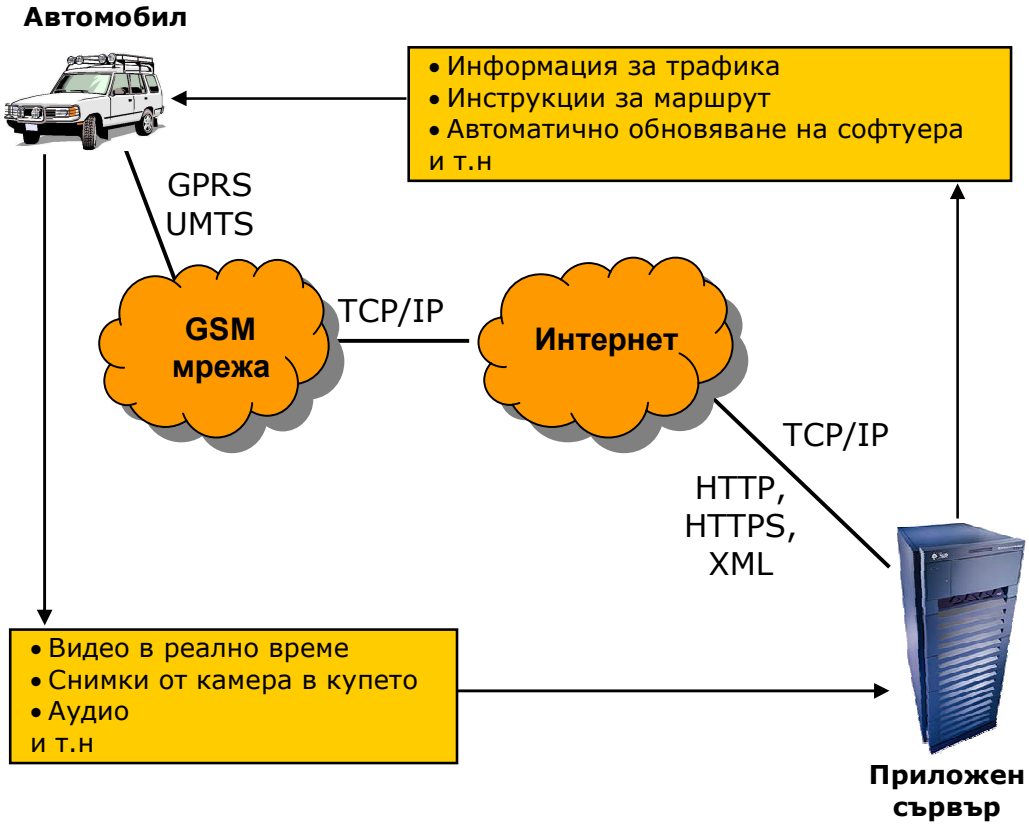
От казаното дотук стана ясно, че SMS услугата може спокойно да се ползва като метод за двустранна комуникация между навигационните устройства и централната система за КУПС. В зависимост от изискванията към нея обаче, може да се наложи превозното средство да бъде следено на кратки интервали от време от порядъка на 10-20сек. Това означава изпращане на SMS съобщение с позицията на интервали по-малки от 1мин., а изпращането на толкова много съобщения за кратко време, често се оказва непосилна задача за мобилната мрежа. В този случай цената на услугата също се вдига доста, тъй като броя на изпратените SMS съобщения става много голям. Тези, както и някой други важни причини, които ще спомена в следващата точка, карат доставчиците на подобни системи да търсят по-усъвършенствани методи за предаване на данни, какъвто е например GPRS.

General Packet Radio Service (GPRS)

GPRS услугата се появява за пръв път в Япония през 1999г., предлагана от японския телекомуникационен гигант NTT DoCoMo. Тя е част от второто поколение GSM мрежи (2G) и е първата от няколкото запланувани стъпки по пътя към предлагането на услуги за високоскоростно пакетно предаване на данни. GPRS е дълго очаквана от мобилните оператори технология, на която те разчитат, за да могат да предлагат на потребителите си нови интерактивни услуги и високоскоростен безжичен достъп до Интернет. Днес GPRS и неговият наследник UMTS, са много добре възприети и използвани навсякъде технологии, както от доставчиците на мобилни услуги така и от обикновенните потребители.

Компютърните системи за КУПС, могат да се възползват по невероятен начин от новите възможности, които GPRS технологията предлага. Пакетното предаване на данни предоставя възможност на мобилните устройства поддържащи GPRS или UMTS, да използват за комуникация протоколи от високо ниво като TCP/IP (Transmission Control Protocol / Internet Protocol) или UDP (User Datagram Protocol) и скорости на обмен на данни до 56.6Kbps (за GPRS-а) *(Bates, 2003)*. Това означава, че функциите на навигационните устройства могат да се вдигнат на едно по-високо стъпало от чистото предаване на съобщения с данни за позицията. TCP/IP протоколът може да бъде използван за предаването на разнообразна информация, като изображения с географски карти, данни за трафика по пътищата, прогноза за времето, инструкции за маршрут, видео и т.н от и към навигационното устройство. Новите възможностти са неизчерпаеми.

Един от най-сериозните проблеми, които разработчиците срещат при внедряването на GPRS, като основен канал за комуникация са IP адресите, които устройствата получават. В GPRS средата IP адресите най-често са динамични и задължително нереални. Т.е всяко едно устройство получава различен IP адрес всеки път, когато се обърне към мрежата и стои зад firewall-а на GPRS средата без да има възможност да получава директно заявки от други външни компютри. Това от своя страна означава, че централният сървър на една система за КУПС, не може да се обръща директно към едно устройство по своя инициатива и да изпраща или изисква данни от него. Инициаторът на връзката трябва винаги да бъде устройството. Има различни подходи към решаването на този проблем – софтуерни, хардуерни или комбинация от двете. Системата **Follow Me on the Web** също притежава собствено авторско решение на този проблем, описано по нататък в настоящата дипломна работа.



Фиг. 2 Нови възможности в системите за КУПС при използване на GPRS

GPRS технологията решава някой от проблемите на проследяването, с които SMS съобщенията не се справят успешно, тъй като по TCP/IP

няма проблем да се пращат много съобщения с произволна честота. С увеличаването на популярността и потреблението на GPRS услугата цената й неизбежно ще пада постоянно, а това още повече ще затвърди позицията й на заместник на SMS съобщенията, като основен метод за комуникация в компютърните системи за КУПС.

Circuit Switched Data (CSD)

CSD е методът, който първоначално е разработен за предаване на данни в GSM мрежи. По същността си представлява връзка модем към модем в мобилна мрежа. Той използва един единствен времеви отрязък, който GSM клетката му дава, през който може да предава данни със максимална скорост 9.6Kbps или 19.2Kbps *(Добош, 2005)*, като този времеви отрязък стои зает през цялото време докато една от двете страни не прекрати връзката. Това го прави доста неефективен метод за комуникация, тъй като ресурсите на мрежата не могат да бъдат използвани в моменти, в които данни между двата модема не се предават. Времето за установяване на връзка с другата страна също е много голямо, тъй като трябва да се набере номера на отсрещния модем и да установи диалог с него, което отнема минимум 1мин.

Тези два недостатъка са едни от основните причини CSD да не бъде използван като метод за комуникация в системите за КУПС, тъй като например за да се получат данните за 10 автомобила трябва да се наберат 10 номера, а според казаното по-горе това би отнело най-малко 10мин за данни, които реално е необходимо да се съберат за максимум минута. В днешните системи за КУПС, където CSD се използва то това е главно като резервен канал за комуникация или в случай, че е необходима обратна съвместимост с някой по стар хардуер, например.

Обобщение

Нуждата от усъвършенстване на компютърните системи за контрол и управление на превозни средства несъмнено съществува. Изискванията към транспорта, охраната и други сфери на живота ползващи автомобили, камиони и т.н постоянно растат и стават все по-разнообразни. За да ги покрият, доставчиците на системи за КУПС, трябва да усвоят някои нови технологии, като GPRS и да вградят нови функции в навигационните устройства, даващи им възможност да

разширят кръга от услуги, които клиентите могат да получат от една такава система.

GPS ще си остане най-добрият избор в търсенето на технология за позициониране, поради факта, че ползването й е безплатно, а пазарът прелива от разнообразен хардуер, работещ с нея. Комбинацията от GPS и позициониране чрез клетъчната мрежа би могла да се окаже печеливша, но проблемите при имплементирането й трябва добре да се обмислят. Само по себе си позиционирането чрез клетъчната мрежа не би могло да свърши работа в една система за КУПС.

С пускането в действие на клетъчните мрежи от второ поколение идват и нови технологии като GPRS, позволяващи високоскоростен пакетен обмен на данни и използване на протоколи от високо ниво като TCP/IP и UDP. Това дава възможност на доставчиците на системи за КУПС да внедрят нови интерактивни възможности в тях, като по този начин да ги направят още по атрактивни за потребилите. SMS съобщенията запазват своите качества, като метод за предаване на данни в такива системи и би могло успешно да бъдат ползвани или като резервен канал на GPRS или паралелно с него, ако например клиентът не се нуждае от допълнителните възможности, които GPRS предоставя.

Интернет ресурси

GPS World Magazine
http://www.gpsworld.com

Списание покриващо широк кръг теми засягащи GPS технологията. Тук може да се прочете за последните новости в развитието на навигацията, позиционирането, контрола върху транспорта и т.н

GLONASS Center
www.glonass-center.ru

Страницата на центъра за управление на системата GLONASS.

Galileo Satellite System
http://europa.eu.int/comm/dgs/energy_transport/galileo/index_en.htm

Страницата на евопейската навигационна система Galileo.

Цели и задачи

В предходната глава бяха описани, ключовите елементи на компютърните системите за управление на превозни средства, както и нуждата от развитието им в крак с новите изисквания към тях и новите технологии достъпни на пазара. Имайки предвид тази информация, сега трябва ясно да се дефинират стъпките, през които ще премине процесът на разработка на един важен елемент от тези системи, а именно методът за комуникация с навигационните устройства, вградени в превозните средства, който е тема на настоящата дипломна работа.

В тази глава се описват основните цели и задачи, които стоят при създаването на метода за комуникация. В началото се описват изискванията (задачите), които един такъв метод трябва да изпълнява, изискванията към производителността на използвания хардуер, изисквания към самия метод, изискванията към средата за изпълнение на програмата и др. След като задачите са ясно дефинирани, ще направя кратко описание на целите, които те поставят, за да се разработи продукт покриващ тези изисквания.

Задачи

Методът за комуникация M2M в среда на мобилен оператор, трябва да представлява софтуерно приложение, което да дава възможност за комуникация с вградени в автомобилите навигационни устройства по GPRS канал, използвайки TCP/IP протокол. Той е отговорен за валидирането на получените данни и заявки, запазването им в специално проектирана за целта база данни и поддържането на надеждна връзка между централния сървър и навигационното устройство. Също така, приложението трябва да осигури лесен за внедряване програмен интерфейс, даващ възможност на други софтуерни приложения, които са част от системата, да комуникират с устройствата. Следва по-конкретна формулировка, по категории, на изискванията към метода.

Изисквания към сложността на използване

Приложението трябва да осигурява достатъчно лесен начин за конфигурация и контрол на следните параметри:

- **TCP/IP настройки** – това са настройки за порт и IP адрес или домейн името на сървъра, на който приложението ще се изпълнява. Приложението трябва да е конфигурирано да „слуша” на даден порт и IP адрес на този сървър, за да може да обработва заявките за връзка и данните от навигационните устройства.

- **настройки за връзка с базата данни** – това са данни като потребителско име, парола, име и порт на сървър и др., необходими на приложението, за да осъществи връзка с база данни, където получената информация ще се съхранява.

- **възможност за ръчно стартиране, спиране и рестартиране** – на потребителя (администратора) трябва да бъде дадена възможност за лесно стартиране, спиране или рестартиране на приложението с цел подръжка, установяване източника на грешки и т.н. При тези операции приложението трябва успешно и сигурно да стартира или спира всички активни TCP сесии с устройствата и връзката с базата данни.

- **избор на протокол за комуникация с конкретния тип устройство** – приложението трябва да е максимално гъвкаво с цел да може да комуникира с различен тип устройства, ползващи различен протокол на

съобщенията. За целта приложението трябва да може да бъде конфигурирано, кой протокол за комуникация да използва за даден порт и IP адрес.

- **поддържане на дневник с информация за събития, настъпили по време на изпълнение** – приложението трябва да поддържа така наречения "log" файл, където да записва информация за настъпили грешки по време на изпълнение, както и информация за своето състояние по време на настъпването на грешката.

Изисквания към надеждността

Приложението, имплементиращо метода за комуникация с навигационните устройства, трябва да работи стабилно възможно най-дълго време без да е необходима човешка намеса, която да осигурява нормалната му работа. В случая под нормална работа се разбира по време на изпълнението си приложението да поддържа активна и стабилна връзката между сървъра и всички устройства, да обработва данните идващи от тях, да ги записва в база данни, да обработва заявки на други софтуерни компоненти желаещи да комуникират с устройствата, без това да предизвиква изключителни ситуации или тотален срив на приложението и операционната система. Въпреки това, ако настъпи:

- **критична грешка (загуба на връзка с базата данни, грешка в TCP стека и т.н)** – приложението трябва да запише датата и часа на настъпването на това събитие в „log" файла и ако е възможно данни характеризиращи състоянието му в момента на настъпване на събитието, след което да започне последователни опити за рестартиране на нормалната си работа, докато това се случи.

- **комуникационна грешка (загуба на връзка с устройството, получаване на грешни данни) –** приложението трябва незабавно да предприеме действия по затваряне на TCP сесията с устройството, разчистване на системите ресурси заети за поддържането на сесията, запис в базата на данни на датата и часа на настъпване на събитието и ако е възможно причината за настъпването му.

Изисквания към сигурността

Приложението трябва да изпълнява следните изисквания, за да подсигури факта, че към системата са свързани, само навигационни устройства, които имат право на това:

- **криптиране на данните –** данните, които се обменят между сървъра и устройството, трябва да бъдат, криптирани чрез използване на криптографски алгоритъм, осигуряващ добро ниво на защита и не изискващ много изчислителни ресурси за изпълнението си.

- **защита от атаки на повторение** – приложението трябва да включва защита от така наречените атаки на повторение (repeat attacks), представляващи „подслушване" на пакетите, които се обменят между две точки по TCP/IP и после „наводняване" на сървъра, чрез циклично препращане на „подслушаните" пакети, без техния смисъл да бъде разкодиран.

- **валидиране на уникалния номер на устройството** – преди да започне да обработва на данните, които устройството изпраща, приложението трябва извлече уникалния му номер, който то включва във всяко съобщение и да провери дали този номер е записан като валиден в базата данни. Ако номерът е валиден, TCP сесията си остава активна и приложението започва да приема данните на това устройство.

- **отбелязване, че дадено устройство е вече свързано -** за да предоврати опитите да се установят много TCP сесии използващи един и същ номер на устройство, приложението трябва да отбележи в базата данни, че дадено устройство вече е свързано и да проверява това поле всеки път, когато се получи заявка за връзка с този номер.

Изисквания към производителността

Приложението трябва да изпълнява следните изисквания по отношение на своята производителност, като изпълнението им не трябва да нарушава нормалната му работа и да заема колко се може по-малко памет и процесорно време:

- **брой активни TCP сесии** – приложението трябва да е в състояние да работи с най-малко 700 едновременно отворени TCP сесии и най-много 1000 такива. Тези ограничения, в броя на сесиите, произхождат от факта, че при изискваната минимална хардуерна конфигурация, по-голям брой сесии би означавал натоварване на машината до недопустимо рисково ниво. Всяка една сесия всъщност означава едно устройство свързано към системата. При надвишаване на максимално допустимия брой устройства, трябва да се пусне нова машина, на която да се изпълнява още едно копие на приложението. По този начин капацитетът на системата се увеличава с още 1000 устройства.

- **брой заявки от други приложения** – най-малко 100 и най-много 150 заявки идващи от други компоненти на системата, жеалещи да комуникират с дадено устройство през интерфейса, който приложението предоставя, трябва да могат да бъдат изпълнени за максимално допустимото за това време. Когато броят на едновременните заявки надвиши 150, приложението трябва да отхвърля всяка следващата заявка докато не дойде в състояние, в което да може да я изпълни.

- **максимално време за обслужване на заявка** – всяка една заявка за комуникация с дадено устройство през интерфейса на приложението трябва да бъде обслужена за максимум 60 сек., което е и максимално допустимия период за чакане след, който се приема, че е настъпила някаква грешка и резултат не може да бъде получен в следствие на нея. След изтичане на максимално допустимото време, се връща отговор, че изпълнението не е възможно, а TCP сесията на устройството, ако тя е все още активна, се прекъсва. Под обслужване на заявка се има предвид, приемането ѝ, изпращане на съответната команда към устройството, приемане на отговора от устройството и връщане на резултат към заявителя.

Изисквания към хардуера и софтуера

Приложението трябва да може да работи нормално при следната минимална хардуерна конфигурация на машината и инсталиран допълнителен софтуер:

- **хардуерни изисквания**

  - ✓ AMD Athlon XP 3700+
  - ✓ 1024 MB RAM
  - ✓ 20GB HDD
  - ✓ 100 Mb LAN Card

- **софтуерни изисквания**

  - ✓ Windows XP Server
  - ✓ IIS 5.0 или следващата версия
  - ✓ .NET Framework 1.1 или следващата версия

Изисквания към програмните интерфейси

Приложението трябва да имплементира няколко програмни интерфейса, за да може да бъде контролирано от физически отдалечена машина, намираща се в локалната мрежа на сървъра или извън нея и за да може системата да функционира като цяло. Тези интерфейси трябва да са изпълнени чрез използване на широко разпространени и стандартни технологии, за да може приложението да работи съвместно с компоненти писани на различни програмни езици и работещи под други операционни системи. Интерфейсите са:

- **интерфейс за дистанционен контрол** – дава възможност на други софтурени приложения да изпълняват команди необходими за извършването на дистанционен контрол и администриране на приложението.

- **интерфейс даващ информация за статуса –** предлага команди връщащи информация за текущото състояние на приложението в момента (брой активни сесии, дали приложението слуша нормално за идващи заявки от устройствата и т.н)

- **интерфейс за обмен на данни с конкретно устройство** – дава възможност за изпращане на команди към дадено устройство, което в момента е свързано към системата.

Изисквания към съхранението на данните

Данните за всички потребители на системата, автомобили, устройства, както и данните получавани от тях трябва да се съхраняват в специално проектирана за целта база данни, която трябва да покрива следните изисквания:

- **брой клиенти към базата** – базата данни трябва безпроблемно да работи с над 1000 едновремнно активни клиента, свързани към нея и да обработва данните от тях за време, което да не нарушава нормалната работа на приложението.

- **обем на данните –** базата трябва да поддържа таблици с обем достатъчен за съхраняването на данните на всяко едно устройство за максимум 30 дни., което е прието като максмималния срок, за който някой потребител може да иска справка за своите автомобили.

Цели

След като изискванията към приложението имплементиращо метода за комуникация тип M2M между навигационното устройство и приложен сървър са дефинирани, тук ще бъдат изброени целите, които тези изисквания пораждат, за да бъде създадено подобно приложение и покрити горе изброените изисквания.

Целите са групирани в относителни категории, като за всяка категория е направена кратка обосновка показваща кои от изискванията към приложението тя покрива. От тук нататък приложението, тема на дипломната работа, ще се нарича "мениджър на устройства" (МУ), тъй като основната му задача е да контролира и управлява навигационните устройства вградени в автомобилите.

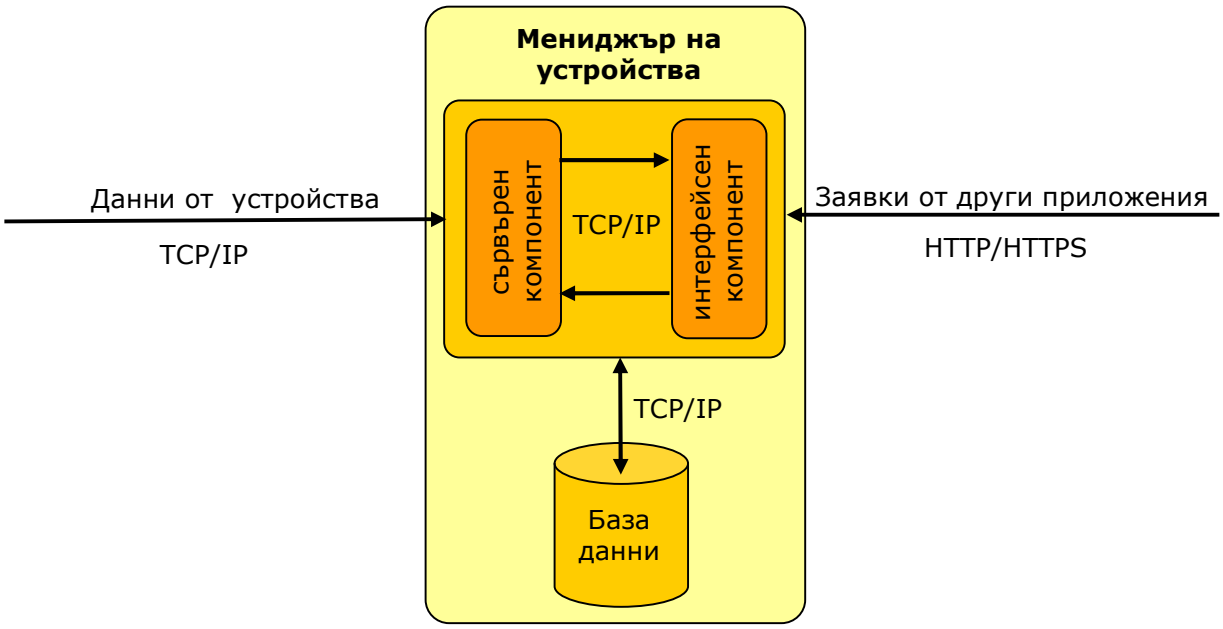
Създаване на "мениджър на устройства" като приложение от няколко компонента

За да бъде поддръжката на мениджърът на устройства по-лесна, а създаването му по-бързо и ефективно, най-добре е да се възприеме подход разделящ го на няколко софтуерни компонента, изпълняващи различни части от неговите задачи. В тази точка те само ще бъдат дефинирани, а по-нататък ще бъде дадена повече информация за тяхната роля в приложението.

- **сървърен компонент** – това е компонентът отговорен за приемането на заявките идващи от навигационните устройства, поддържане на комуникация с тях, обработката на данните им, записването им в база данни и изпращането на команди към тях.

- **интерфейсен компонент** – това е компонентът грижещ се за осъществяването на връзка между сървърния компонент и други софтуерни приложения, които са част от системата.

- **база данни** – цялата информация необходима за работа на системата, като цяло се съхранява тук.



Фиг. 3 Логическа схема на компонентите на МУ

Създаване на сървърния компонент като приложение от тип windows услуга

Създаването на сървърния компонент като приложение тип windows услуга, предлага някой много полезни възможности, които за нашия случай биха били особено полезни. Най-важното от тях е, че тези приложения не изискват какъвто и да било потребител да бъде регистриран и влязъл в системата в момента, за да могат да работят. Те се зареждат преди всички останали приложения и могат да вършат работата си в „сянка” без нужда от потребителска намеса. По този начин, ако се случи приложният сървър, на който мениджърът на устройства се изпълнява, да се рестартира, няма да е необходимо някой да отиде, да влезе в системата и ръчно да стартира наново приложението, защото Windows ще го стартира автоматично.

От гледна точка на производителността на приложението, нишките, които то създава, ще се изпълняват в така наречения kernel mode на Windows и съответно на процесора, което ще им даде най-висок приоритет за изпълнение, а операционната система ще работи с тях използвайки интерфейси от ниско ниво, които са в пъти по-бързи от тези от високо

ниво. По този начин мениджърът на устройства ще покрие изцяло изискването да заема възможно най-малко памет и процесорно време.

В Windows приложенията от тип услуги се управляват от така наречения **Service Control Manager (SCM)**, който предоставя вече готов интерфейс и модел за управление на всички услуги работещи в системата, което освобождава разработчика от понякога доста тежката задача по създаването на графичен интерфейс за работа с дадено приложение. SCM също така елиминира нуждата от създаване на собствен модел за справяне и възстановяване от необработени изключителни ситуации, които могат да настъпят по време на изпълнение на програмата, защото може да бъде настроен така, че при възникване на такава изключителна ситуация сам, без човешка намеса, да рестартира приложението. Подобна възможност е изключително подходяща за нашите цели.

.NET средата предоставя вече готови и доста удобни методи за програмно управление на услугите, т.е при нужда лесно може да се създаде графичен интерфейс за работа с услугата. Необходимо е само да се създаде бизнес логиката на програмата, да се регистрира като услуга в системата, а от там целият контрол и настройка на приложението може да става през SCM. Също така SCM поддържа собствен, леснодостъпен и подробен "log" файл, в който записва всички събития, не само грешките, заедно с часа и датата на настъпването им, отнасящи се до изпълнението на приложенията-услуги, които се намират под негов контрол. По този начин покриваме изискването за подържане на "log" файл от нашето приложение, без да е необходимо да създаваме собствен такъв.

Създаване на интерфейсния компонент като приложение от тип web услуга

Създаването на интерфейсния компонент като приложение тип web услуга, носи следните предимства:

- **стандартни технологии** – web услугите (web service) са приложения, които притежават някаква функционалност достъпна чрез стандартни технологии и протоколи за комуникация, а именно XML, SOAP, HTTP, WSDL. Чрез използването на този подход, се дава възможност на компоненти написани на различни езици, да работят с мениджъра на устройства, като тези компоненти могат да се намират на физически отдалечени машини, а комуникацията да минава през Интернет.

• **лесна имплементация** – с използването на Visual Studio .NET и ASP .NET, създаването на web услуги става изключително лесно, като Visual Studio върши по-голямата част от необходимата работа.

Създаване на отделни класове, имплеменнтиращи функционалността на компонентите

Тъй като мениджърът на устройства ще бъде разработван в обектно-ориентирана среда за програмиране, функционалността на сървърния компонент ще бъде разделена на класове. Интерфейсния компонент само ще извиква методи от тези класове без да има свой такива, предлагащи някаква различна функционалност. Класовете са следните:

• **клас сървър** – имплементира методи за "слушане" за TCP заявки от устройства, валидиране на заявки, създаване на обект за всяко устройство след успешно валидиране, поддържане на списък с активните в момента устройства, поддържане на TCP сесията активна, образуване на текстови низове представляващи команди към устройството.

• **клас за "превод" на данните** – методите от този клас "превеждат" (parse) данните идващи от всяко устройство, като по този начин имплементират специфичния протокол на съобщенията за конкретния тип устройство, необходим за работа с него. Всички останали класове ползват методи на този клас, за да интерпретират данните от устройствата, когато това е необходимо.

• **клас на устройството** – след успешно валидиране на заявката за връзка, за всяко едно устройство се създава обект от този клас, съдържащ собствена нишка, която от тук нататък се занимава с приемане и изпращане на данни от/към устройството.

• **клас за работа с базата данни** – имплементира методи за четене и запис на данни в базата.

Създаване на модел за комуникация между компонентите на МУ

Тъй като сървърният и интерфейсният компонент на практика представляват две различни приложения работещи в свой собствени процеси (в случая AppDomain-и), те нямат директен достъп до своите

методи и затова е необходимо да се създаде модел за комуникация между двата компонента.

В .NET средата най-удобният за тази цел подход е използването на така наречената .NET Remoting технология, която прави създаването на подобен модел много лесно.

Създаване на MySQL база данни

Изборът на MySQL, като база за съхранение на данните е удачен за нашите цели, тъй като той покрива напълно изискванията за производителност и обем на таблиците, а от чисто икономическа гледна точка предимството му е, че е безплатен при некомерсиално използване.

Обобщение

Дизайнът на мениджъра на устройства предвижда, той да е високооптимизирано приложение, което да поддържа едновременна комуникация по TCP/IP протокол с много устройства и да съхранява техните данни в специално проектирана за целта MySQL база данни.

Приложението е проектирано на модулен принцип, като целта е да се разбие функционалността на колко се може по-малки софтуерни модули, като по този начин се намалява вероятността от допускане на принципни грешки, подобрява се възможността за поддръжка на приложението и идентифициране на грешки в него.

Интернет ресурси

WebServices.org
http://www.webservices.org/

Портал към различни сайтове, статии, пазарни проучвания и други материали свързани с web услугите и използваните в тях технологии.
Introduction to Windows Service Applications

http://msdn2.microsoft.com/en-us/library/d56de412

Статия даваща начални сведения и препратки към много други източници отнасящи се към програмирането на приложения windows-услуги с Visual Studio .NET.

M2M Blog
http://www.m2mblog.com/

Сайт поддържан от едни от най-изтъкантите експерти в областта на M2M услугите, съдържащ новини, анализи и връзки към източници по темата.

Структурно описание

Следед като вече целите и задачите на проекта са ясно дефинирани, време е да преминем към описанието на процеса на проектиране на система с подобни параметри. Целта на тази глава е да даде ясна представа за дизайна на приложението и затова как функционират и комуникират различните му компоненти. Стриктното изпълнение на изискванията към приложението поставят различни проблеми пред разработчика тъй като подходът към решението им е специфичен за всеки един програмен език и среда за разработка. Там където са били налични повече от едно решения, съм разгледал всяко едно тях, като изтъквам неговите предимства и недостатъци и накрая обосновавам решението си да избера някое от тях.

Освен информация за структурата и функционалността на приложението, съм се постарал да дам синтезирано описание на различните технологии, като например web услугите и .NET Remoting, използвани при изграждането му и чието познаване е абослютно необходимо, за да се добие представа как мениджърът на устройства функционира.

Избор на среда за разработка

Приложението "Мениджър на устройства", предмет на дипломната работа, е написан изцяло на езика C#, в средата на Microsoft Visual Studio .NET 2003.

Visual Studio .NET е един съвремен и изключително мощен инструмент за създаване на Windows приложения, web услуги, web приложения и т.н възползвайки се от възможностите, които революционната технология .NET Framework предлага. Средата предоставя изключително удобен интерфейс и предлага безброй готови за вграждане в приложенията компоненти. Многото шаблони за различните типове приложения, които могат да се създават, дават възможност за летящ старт на всеки един проект, като са необходими само няколко предварителни настройки, след което Visual Studio .NET свършва повечето от тежката работа по конфигурирането на проекта и не само това.

За целите на разглеждания проект, като например работа с TCP/IP, VS .NET е може би най-мощната среда в момента на пазара, тъй като включва готови класове за работа с него, както на ниско и така и на високо ниво. Класовете от високо ниво изискват само няколко реда код, за да осигурят работа на приложението с TCP/IP, като не изискват никакви дълбоки познания за протокола. Приложения, като нашето, организиращи сложна комуникация по TCP/IP могат да се възползват от многото мощни и бързи функции за четене, изпращане и слушане за данни, работа с DNS, HTTP, XML и т.н, които VS .NET предлага.

Една от целите поставени в "Глава II" е МУ да се реализира като приложение от тип windows услуга. Създаването на подобни приложения с използване например на C++ и Visual Studio 6.0 доскоро беше привилегия само на изключително опитни програмисти и въпреки това пак отнемаше много време. С Visual Studio .NET създаването на windows услуга е по-лесно от всякога, тъй като той извършва по-голямата и обърквaща част от работата и прави писането й подобно на писането на най-обикновено приложение. Така разработчикът може да се възползва от всички привилегии на приложенията тип windows-услуги, без да се налага да става специалист по концепциите и програмирането им.

Писането на web услуги, с които трябва да бъде реализиран интерфейсния компонент на мениджърът на устройства, също не прави изключение, когато става въпрос за простота на създаването. Програмирането на web услуги е свързано с детайлното познаването на много

технологии и протоколи, като XML, SOAP (Simple Object Access Protocol), WSDL (Web Service Description Language), UDDI (Universal Description, Discovery and Intergation), ASP .NET и т.н, но Visual Studio .NET отново сваля това тегло от разработчика, като върши по-тежката работа, като например сериализирането и десериализирането на данните в SOAP протокол, интерпретиране на XML документи и т.н.

Според изискванията към мениджъра на устройства, той трябва да бъде разделен логически на два отделни софтуерни компонента, които на практика са две отделни приложения. В такъв случай пред разработчика стои задачата да изгради програмна връзка между тези два компонента, а използването за тази цел на технологии като COM, DCOM, CORBA и т.н е сложен проблем. .NET Framework и Visual Studio .NET дават решение и на този проблем, като представят технологията .NET Remoting, която прави комуникацията между обекти намиращи се в различни процеси тривиална задача.

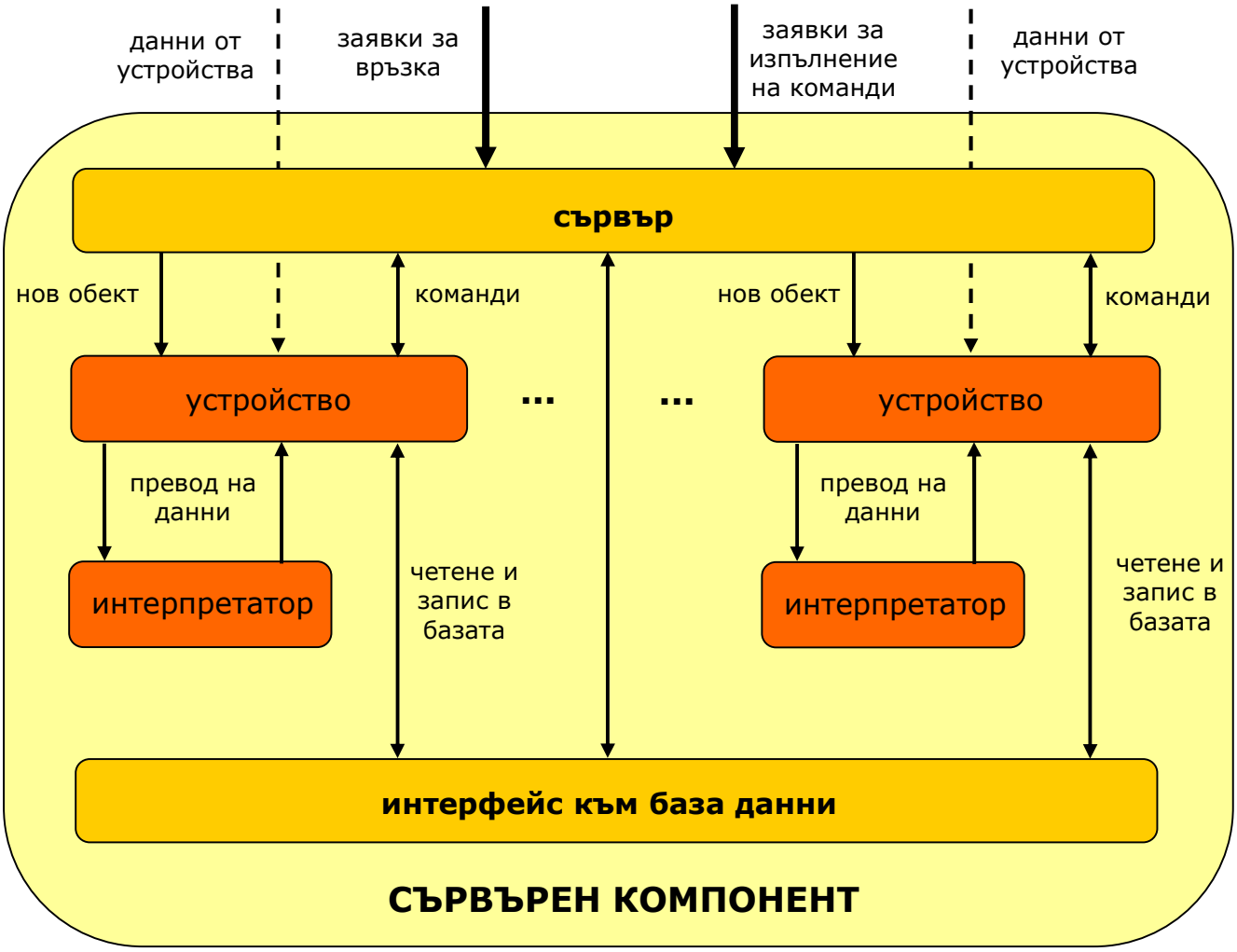
След като бяха изтъкнати всички предимства на Visual Studio .NET, имащи отношение, към настоящия проект, трябва да се кажат няколко думи и за тези на избрания при разработката език за програмиране – C#. C# е сравнително млад, 100% обектно ориентиран език, бързо набиращ популярност не само заради лекотата си на използване, но и заради гъвкавостта и мощността, характерни за Java и C++, които предлага. Работейки с .NET Framework и Visual Studio .NET мисля, че няма по-подходящ избор на език за програмиране от C#.

Проектиране на сървърния компонент

Сървърният компонент (СК) е основният компонент в мениджърът на устройства, грижещ се обмяната и записа на данни идващи от навигационните устройства. Възприемайки модулния принцип на проектиране, този компонент ще бъде разделен на по-малки модули (класове), имплементиращи част от неговата функционалност.

На фиг. 4 са показани отделните модули на сървърният компонент и как те си взаимодействат. Сървърният компонент ще представлява всъщност една DLL библиотека, която съдържа в себе си класове (обекти), изпълняващи различни задачи. Това ще направи дизайна й невероятно гъвкав, защото в последствие тя може да бъде вграждана в различни приложения, без да са необходими някакви специални промени по нея, което би усложнило доста този процес. В случая приложението, което „обвива" сървърния компонент, е

приложение тип windows-услуга, което се изпълнява в kernel mode на процесора, тоест с доста по-голям приоритет от останалите.



Фиг. 4 Връзки между отделните модули на сървърния компонент

Тъй като системата трябва да поддържа много видове устройства, работещи с различни протоколи на съобщенията, имплентирането на нови такива също трябва да бъде направено гъвкаво. Затова, цялата работа по интерпретиране (превеждане) на получените съобщения в смислена информация ще бъде отделена в обект наречен "интерпретатор". Когато други модули от сървърния компонент е необходимо да интерпретират получено съобщение, те ползват методи на този обект, като това кой точно метод да извикат зависи от конкретната ситуация, в която модулът трябва да знае каква информация да очаква. По този начин, при изграждане на сървърен компонент за нов тип устройство, ще трябва само да се напише нов

клас "интерпретатор" имплентиращ специфичния протокол на съобщенията за това устройство.

Нека сега разгледаме всеки един модул по отделно, като дадем по-подробна информация за неговия дизайн, функции в компонента и как той ги изпълнява.

Обект "сървър"

Това ще е основният модул в сървърния компонент и единственият, който ще е достъпен за други приложения. За да може да бъде създаден този обект, той ще изисква следната информация, която да бъде поддадена чрез параметри към конструктора му:

- ✓ низ за връзка с база данни
- ✓ IP адрес и порт

Низът за връзка с базата данни, представлява текстов низ, съдържащ информация като потребителско име, парола, IP адрес и порт, протокол за комуникация и т.н, необходима за да се осъществи връзка със сървъра, на който се намира базата данни. Повече информация за базата данни и интерфейса за достъп до нея има в точки "Обект интерфейс към база данни" и "Проектиране на база данни".

На зададения IP адрес и порт сървърният обект непрестанно ще "слуша" за чакащи TCP заявки за връзка. Когато подобна заявка пристигне, той ще отваря TCP сесия със заявителя, като очаква да получи специфично съобщение, наречено "welcome string". Това съобщение може да е различно за отделните типове устройства, които работят към системата, но там винаги трябва да се съдържа уникалния номер на устройството, който сървърния обект извлича, използвайки метод от обекта "интерпретатор". След като извлече уникалния номер на устройството, обектът сървър използва методи от обекта "интерфейс към база данни", за да провери дали подобен номер е записан като валиден в базата данни и дали устройство с такъв номер вече не е свързано към системата. Ако проверката премине успешно, се създава нов специален обект "устройство", който от тук нататък ще се грижи за управлението на комуникацията по TCP/IP с това устройство, а в базата данни се отбелязва, че то е свързано към системата, за да се предотврати осъществяването на повторна връзка с този номер. Ако проверката не издържи, отворената TCP сесия се затваря и всички ресурси заети от нея се освобождават.

Както беше споменато по-горе, ако валидирането на заявката за връзка премине успешно, се създава нов специален обект, който да се грижи за по-натъшната комуникация с устройството. От тук нататък обратната връзка с него се организира чрез поддържане на специален тип списък на такива обекти, чиито елементи представляват двойка ключ-данни. В случая като ключ се използва уникалният номер на устройството, който по-късно може да се използва за търсене. В полето за данни на списъка се записва указател към обекта, отговарящ за конкретното устройство. Когато сървърният обект поиска да изпрати данни към дадено устройство, той проверява дали уникалния му номер се съдържа в списъка и ако е така, взима указателя към обекта записан срещу този ключ. Повече за това как се изпращат данни към устройството и как се поддържа този списък е описано в точка "Обект устройство".

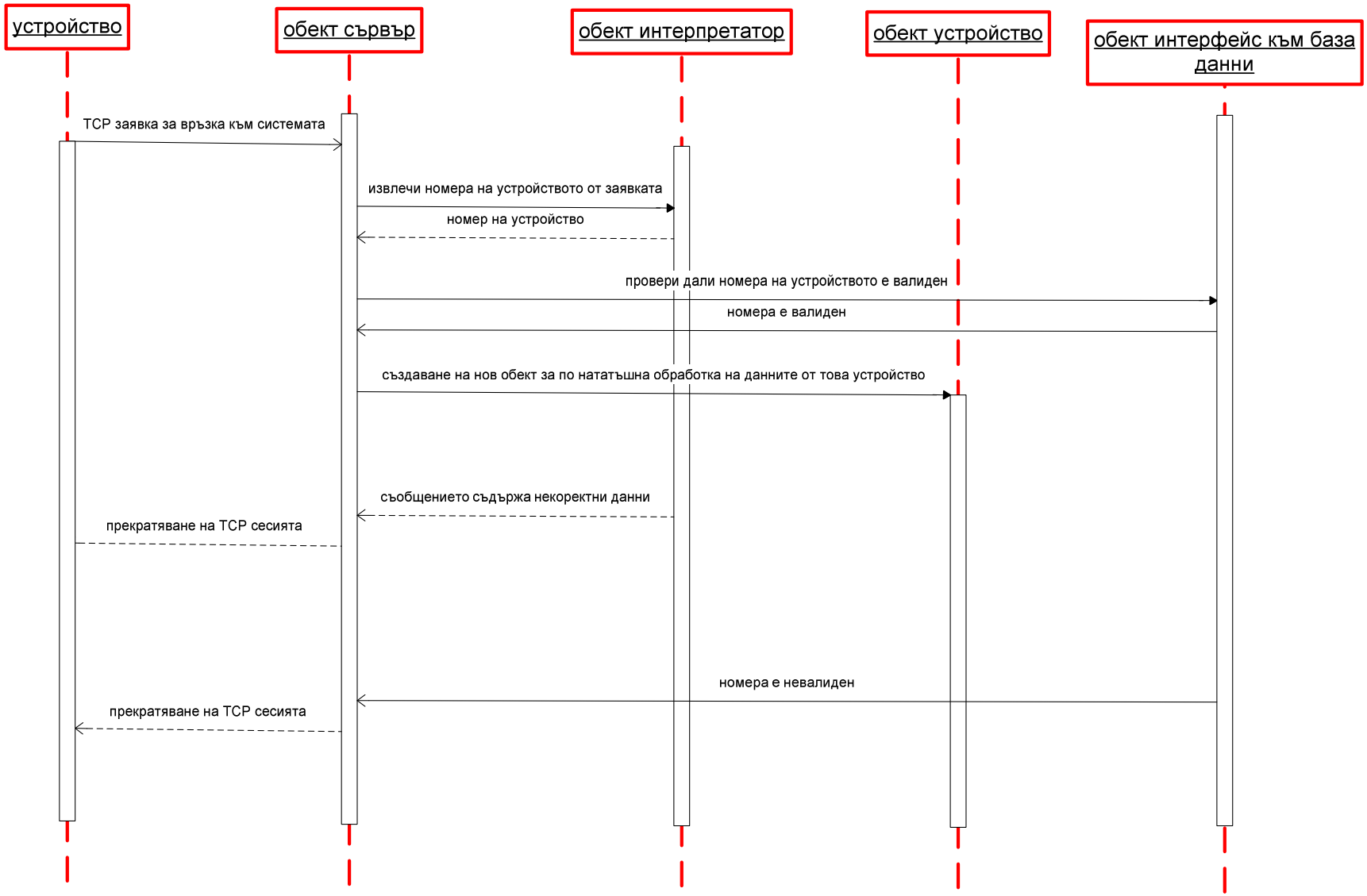
Има още едно, макар и доста по-сложно, решение на този проблем. При него обаче създаването на отделен обект за всяко устройство отпада. В .NET средата с TCP/IP се работи чрез така наречените сокети (sockets), които могат да се разглеждат като едни виртуални TCP портове, през които минава цялата комуникация за дадена сесия. Идеята на това решение е отново да се поддържат списъци с елементи от тип ключ-данни, но данните да не са указатели към специални обекти, а указатели към сокет обекти, които обслужват дадена TCP сесия. Данните, които пристигат на тези сокети ще се буферират в специален буфер, който операционната система поддържа за всеки сокет, и ще останат там докато не се прочетат и буферът не се изчисти. Ако буфера достигне максималния си размер преди да бъдат прочетени данните, всички следващи данни, които пристигат за него ще бъдат автоматично игнорирани от операционната система. Списъкът от сокети, ще бъде разделен на части от по 20, като всяка отделна част ще се обслужва от отделна нишка, която ще обхожда последователно всеки един сокет, ще прочита данните, които се пазят в неговия буфер и ще ги записва в базата данни. По този начин, за да обслужим 1000 устройства ще са нужни 50 нишки вместо 1000, както е при първия вариант.

От гледна точка на производителността и икономията на системни ресурси, вторият вариант изглежда по-удачен. Недостатъците му проличават в момента, в който трябва да се изгради модел за ефективно управление на нишките, които облужват устройствата и когато се наложи да се изпращат данни към тях, последното от които е много важно условие. Списъкът със сокети се управлява динамично, тоест когато например дадена TCP сесия се разпадне, чрез методи предоставени от средата за разработка, сокетът и ключът за тази сесия се премахват от списъка, а елементите след него се преместват с една стъпка нагоре. Тогава често ще се стига до ситуация, в която ще има например 10 обслужващи нишки, но 5 от тях ще обслужват 20 сокета, а останалите 2 сокета, тоест не всички ще са максимално ефективни.

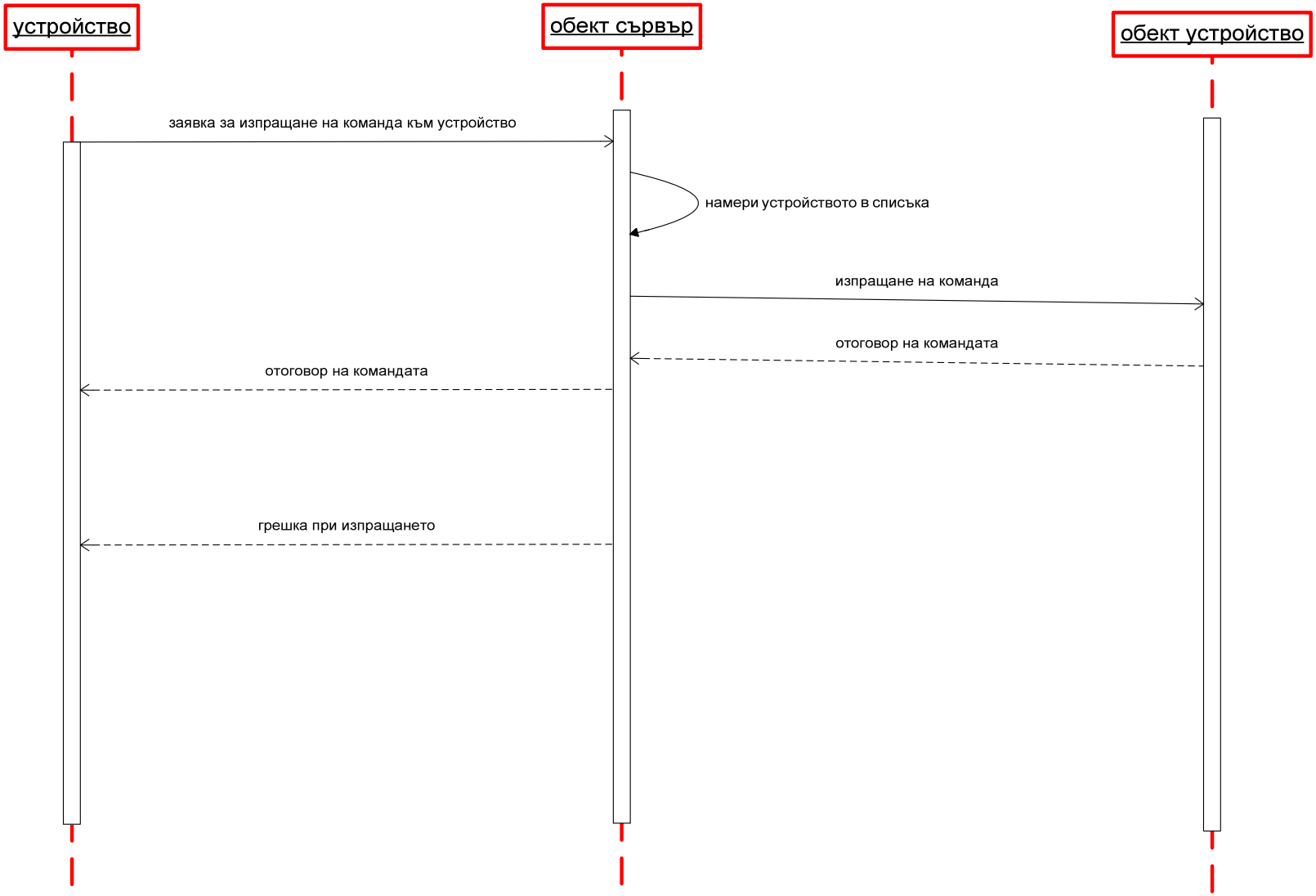
В този случай, за да се подобри ефективността на работа, трябва да се създаде сложен оптимизационен модел, при който главната нишка на сървърния обект да следи колко активни сокета има в момента, колко активни нишки и при нужда да унищожава и създава нови нишки за обслужване, което никак не е проста задача и води допълнителни затруднения със себе си.

Освен заради гореспоменатите прични, изборът на решение пада върху първия вариант и защото при наличие на отделен обект за всяко устройство, можем да вградим методите за комуникация и други данни за него, които трябва да се поддържат за всяко устройство, директно в обекта, което го прави автономен и следва принципите на обектно ориентираното програмиране. При един достатъчно мощен процесор, каквито са повечето в момента, поддържането на 1000 нишки едновременно не би било проблем, така че този недостатък може да се счита за преодолим.

Сървърният обект ще включва и методи чрез, които обвиващите го приложения ще могат да го стартират, спират и рестартират. Стартирането ще включва, осъществяване на връзка с базата данни и стартирането на основната нишка “слушаща” за заявки за връзка, а спирането последователно унищожаване на всички обекти-устройства и на главната нишка.



диаг. 1 Работа на сървърният обект при приемане на заявки за връзка от устройства



диаг. 2 Работа на сървърният обект при обработка на заявки за изпращане на команда

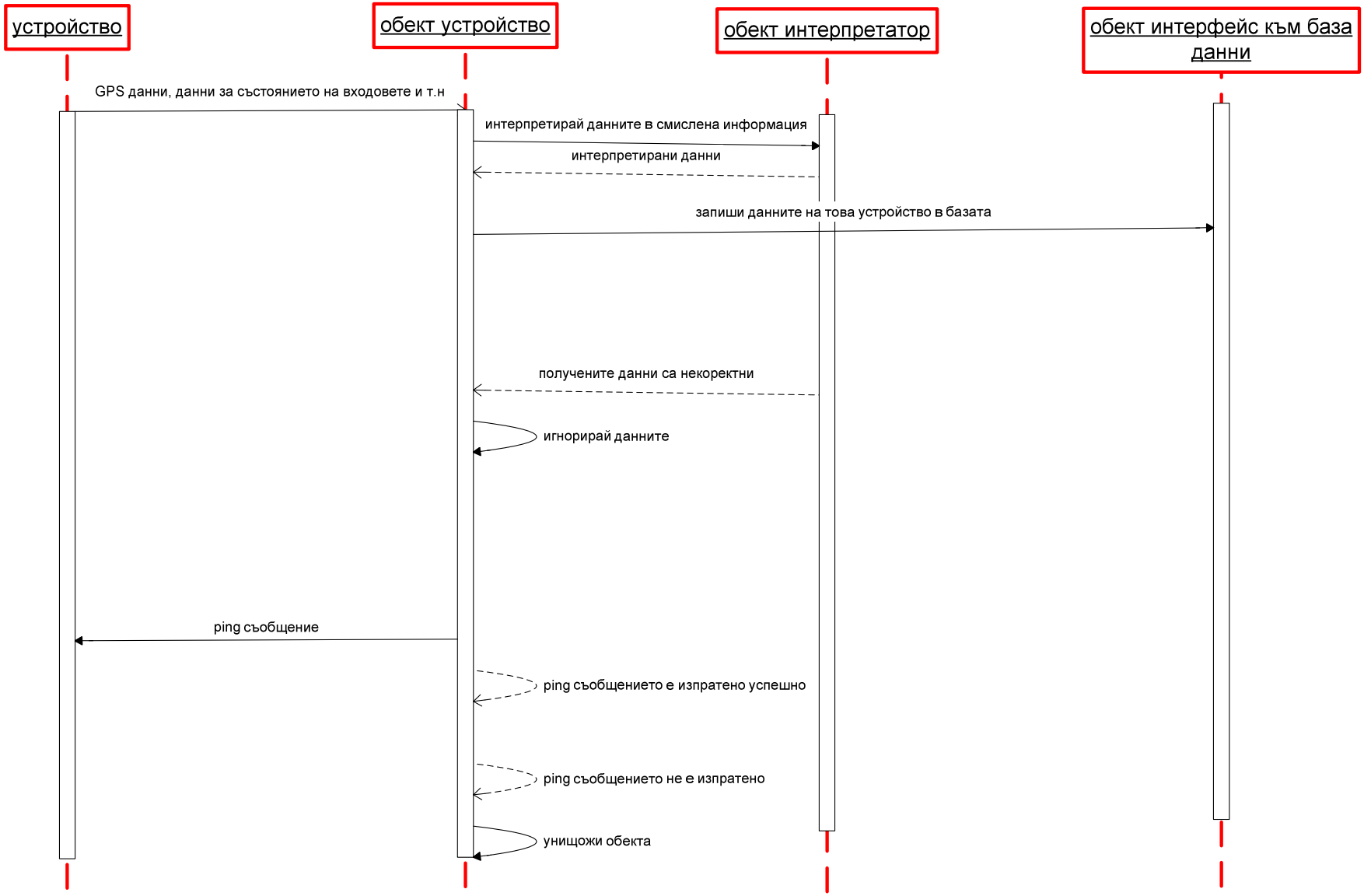
Обект "устройство"

След като дадена заявка за връзка премине валидацията, сървърният обект създава за всяка успешна заявка по един такъв обект, който се грижи за поддръжката на комуникацията между устройството и сървърния компонент.

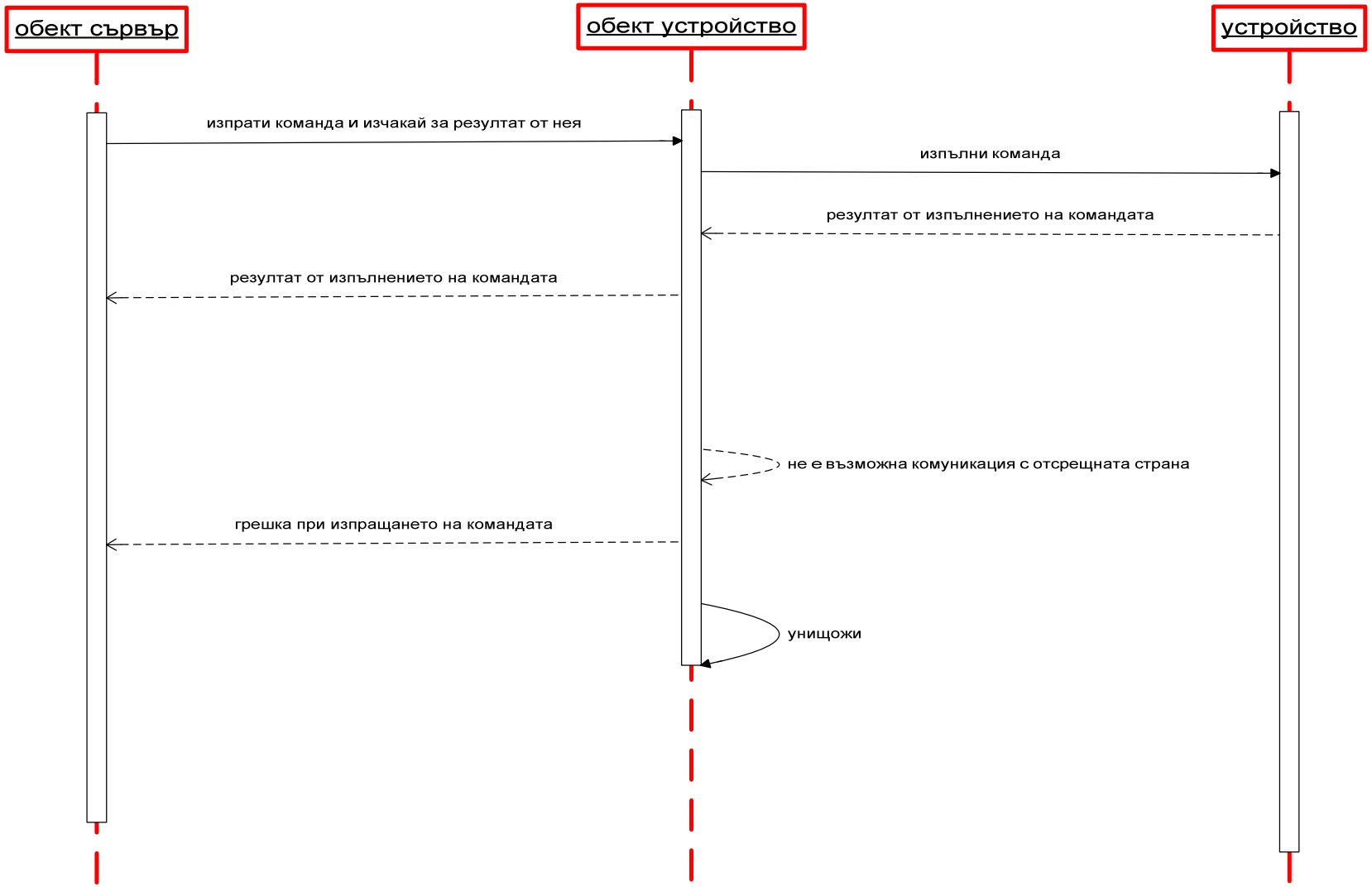
За бъде създаден такъв обект е необходимо да се подаде следната информация към конструктура му:

- ✓ указател към TCP сокет през, който ще минава комуникацията
- ✓ указател към интерфейс за връзка с база данни
- ✓ низ съдържащ уникалния номер на устройството, което обектът обслужва
- ✓ низ съдържащ номера на устройството в базата данни
- ✓ число задаващо големината на буфера за четене и приемане на данни

След като бъде създаден обектът "устройство" започва да изпълнява своя собствена нишка, която периодично проверява дали има данни пристигнали на сокета, с който обекта е инициализиран. Ако има нови данни те се прочитат, интерпретират се и ако всичко с тях е наред се записват на съответното място в базата данни. Ако данните са неточни и не могат да се интерпретират правилно, то те се игнорират и не се записват в базата. Обектът "устройство" включва в себе си метод за изпращане на команди през сокета, с който е инициализиран. Когато сървърния обект желае да изпрати команда към дадено устройство, той взима указател към съотвения обект "устройство" от списъка си, формира текстовия низ представляващ командата и го подава като параметър към метода за изпращане. Ако получи отговор, методът за изпращане на команди го препраща директно на сървърния обект, като задължението да го интерпретира се пада на последния. Ако настъпи някаква грешка при комуникацията с устройството, като например разпадане на връзката или липса на отговор от острещната страна, обекта "устройство" генерира съобщение, с което уведомява сървърния обект, че е настъпила грешка, той от своя страна го изважда от списъка си с активни обекти, след което обекта "устройство" се самоунищожава. Съществува вариант, при който методите формиращи низа представляващ дадена команда, да бъдат вградени в обекта "устройство", но тъй като командите са едни и същи за всички тях е по-удачно да се изолират в сървърния обект, като по този начин запазват компактността на обекта "устройство" и намаляват обема памет, който целия сървърен компонент може да заема, защото може да имаме 1000 обекта "устройство", но само един сървърен обект за даден модел устройство.



диаг. 3 Работа на обекта „устройство” при обработка на данните на дадено устройство



диаг. 4 Работа на обекта „устройство” при обработка на заявка за изпращане на команда към устройство

Обект "интерпретатор"

Както е показано на фиг. 4 всеки обект "устройство" създава своя инстанция на обект наречен "интерпретатор". Този обект съдържа методи, чрез които данните получавани от устройствата могат да се интерпретират или "превеждат" в смислена информация, която после да се обработи по съответния начин.

На практика в обекта "интерпретатор" е заложен протокола на съобщенията, който сървърният компонент и устройството ползват, за да комуникират. Този протокол е специфичен за всеки тип (модел) устройство и затова, когато към системата се добавя поддръжка за нов тип навигационни устройства, се ползва вече изградения модел на сървърен компонент, като се пренаписва само обекта "интерпретатор".

Системата **Follow Me on the Web,** работи с навигационни устройства модел MPS U1, разработка на фирма Мултипроцесорни Системи ООД. Комуникацията с тези устройства се извършва чрез директен обмен по TCP/IP протокол на ASCII текстови съобщения, съдържащи различни данни. Затова обектът "интерпретатор" написан за U1 сървърния компонент, съдържа по един метод за превод на всяко възможно съобщение, което може да се обмени между сървъра и устройството. Обектите ползващи методи от обекта "интерпретатор", трябва да знаят във всеки момент какво съобщение да очакват, за да могат да извикат подходящия метод и да интерпретират съобщението правилно. Ако някой метод, интерпретиращ дадено съобщение, върне грешка при "превода", т.е данните, които са му подадени не изпълняват изискванията на възприетия протокол, то това съобщение се игнорира и данните от него не се обработват по-нататък.

Така описания подход има два неприятни недостатъка. Един от тях е фактът, че въпреки това, че при създаването на нов сървърен компонент е необходимо да се пренапише само обекта "интерпретатор", това не винаги е толкова просто колкото изглежда. Също така, след създаването на нов сървърен компонент със специфичен обект "интерпретатор", задължително се налага прекомпилация на приложението и повторно „разгъване" върху сървърната машина, а ако инфраструктурата на приложението не е проектирана правилно, това може да доведе до временно спиране на цялата система.

Най-добрия подход към тези проблеми би бил използването на XML схеми за дефиниране на протокола на съобщенията между сървъра и устройствата. С използването на такива схеми е възможно дори с прост текстов редактор да се опишат някакви данни, спазвайки изискванията на

така наречените дефиниции на XML схеми (XML schema definition). Тогава, описвайки протокола с една или няколко схеми, можем да посочим на сървърния компонент, че за този модел устройство трябва да ползва тези схеми, за да интерпретира данните, които получава. За целта, обаче, устройството трябва да изпраща не прости ASCII текстови съобщения, а XML документи генерирани вътре в него по същата тази схема. Тогава след като получи XML документ сървърът го интерпретира според схемата, ползвайки мощните инструменти, които Visual Studio .NET предлага за тази цел и след това ги обработва по съответния начин. Следователно, при този подход, за да добавим поддръжка към системата на нов протокол на съобщенията, е неоходимо само да сменим файла съдържащ XML схемата с дефиницията му и нищо повече. Без нужда от прекомпилация, без нужда от писане на нови интерпретатори.

От казаното до тук определено изглежда, че вторият подход е много по-удобен. В случая с U1, проблемът се състои в това, че вграждането в него на необдходимата XML и HTTP поддръжка, би затруднила сериозно изчислителните му възможности, което се оказва голям проблем и налага използването на прости ASCII текстови съобщения за комуникация. При модели навигационни устройства MPS U2 и MPS U3, чиято изчислителна мощ е доста по-голяма, използването на XML за дефиниране на протокола е предвидено и предстои скорошното му вграждане в системата **Follow Me on the Web**.

Обект "интерфейс към база данни"

Методите от този обект дават възможност за четене и запис на данни от/в базата данни чрез използване на стандартните SQL заявки. Това са например, методи за вмъкване на данните от NMEA съобщенията, валидация на номер на устройство, запис дали устройството е свързано към системата или не и т.н.

Две особености на проектирането заслужават повече внимание. Според целите и задачите на приложението, базата данни, която е ключов елемент от него, трябва да бъде създадена чрез използване на MySQL като сървър за база данни. На пръв поглед това изглежда много подходящо, тъй като качествата на MySQL са безспорно доказани, а ползването му в зависимост от избрания лиценз е или безплатно или доста евтино. Visual Studio .NET обаче не предлага вградена поддръжка за работа с MySQL бази данни, което налага ползването на драйвери за целта от трети производител. За щастие такива драйвери за MySQL под .NET не липсват и за целите на проекта бяха избрани тези на украинската фирма CoreLab Ltd. – MySQL Direct .NET. Този

продукт е изграден по модела на драйверите на Microsoft за разбота с MS SQL сървър, съвместим е с ADO .NET технологията и предлага всичко необходимо.

В показаната на фиг. 4 структура на сървърния компонент всички негови модули ползват една и съща инстанция на обекта "интерфейс към база данни". Всъщност те ползват създадения от този обект така наречен пул с връзки към базата данни (connection pool), като големината на пула (броят отворени връзки) се задава като параметър в низа за връзка с база данни на сървърния обект. Един пул с връзки към базата данни изисква от MySQL сървъра да създаде само една нишка за обслужването му (а не примерно по една нишка за всяка връзка), което драстично намалява натоварването му. Когато даден метод поиска връзка с базата, MySQL драйвера вади една от пула, ако има такава, методът си свършва работата и я затваря, но на практика тя не се затваря, а просто се връща в пула, след което може да се ползва повторно. Подобен подход увеличава доста производителността на сървърния компонент и намалява натоварването върху базата, тъй като не е небходимо при всяка заявка да се прави изцяло нова връзка към базата, което е времеотнемаща и натоварваща операция.

Приложението windows услуга

Според изискванията към мениджърът на устройства, сървърният му компонент трябва да бъде изграден като приложение от тип windows услуга. Както се видя от изложеното по-горе, сървърният обект е изграден като DLL библиотека с цел да се подобри преносимостта на приложението и възможността за поддръжка. За да се удовлетвори изсикването за изграждане на windows услуга, ще създадем приложение от такъв тип, като Visual Studio .NET предлага готов шаблон за подобен проект.

Стандартните приложения windows-услуги задължително имплентират два метода в себе си Start и Stop, които се изпълняват при настъпване на съответното събитие, като нашето няма да прави изключение. При стартиране или спирането на услугата, всъщност ще се стартира или спира работата на сървърния компонент с тази особеност, че този път той ще работи в привилегирован режим на процесора и системата (kernel mode).

Друга важна задача на windows услугата е да регистрира сървърния компонент, като дистанционно достъпен обект (remotable object) в Remoting системата на Windows, като част от осъществяването на връзка между него и интерфейсния компонент. Повече информация за връзката между двата компонента има в следващата точка съдържаща описание на интерфейсния компонент.

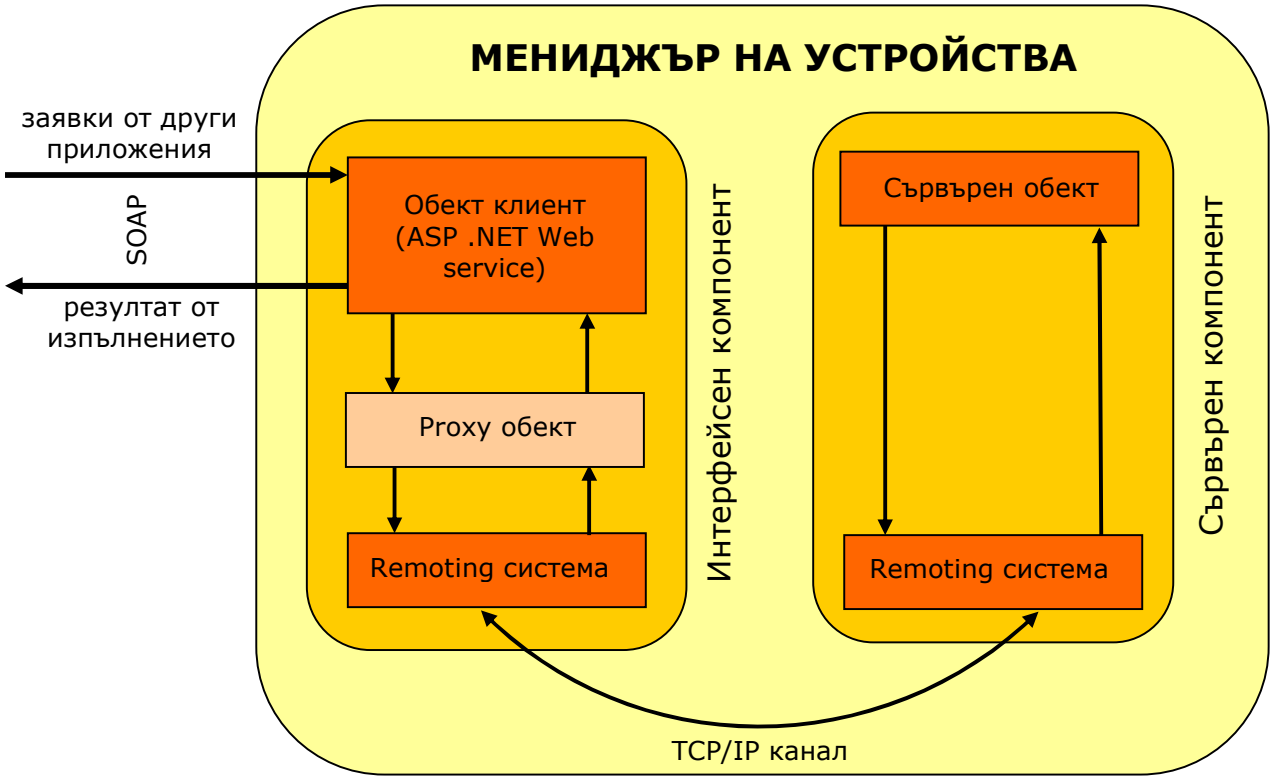
Проектиране на интерфейсния компонент

За разлика от сървърният компонент, интерфейсният няма да бъде проектиран като самостоятелна DLL библиотека, която в последствие да може да бъде имплементирана от други приложения. При интерфейсния компонент не се цели преносимост, а предоставяне на удобен и широко достъпен интерфейс за други приложения, които желаят да обменят данни с някое от устройствата.

Задачата на сървърния компонент най-общо е само да слуша за данни от устройствата и да ги записва, но за да има реална полза за потребителя от една такава система, трябва да има и компонент, който да предлага възможност за извършване на полезна дейсност с тези данни. В системата **Follow Me on the Web** този компонент се нарича "мениджър на потребителите" (user manager) и той се грижи за визуалното представяне на събраните данни и при нужда за осъществяване на връзка между потребителя и устройството вградено в автомобила му. В случая точно за осъществяването на тази връзка е неоходимо мениджърът на устройства да предостави интерфейс към сървърния си компонент, тъй като той е отговорен за комуникацията с навигационните устройства. Web услугите предоставят уникална възможност за реализирането на подобен интерфейс, който да не бъде тясно свързан с някой език и среда за програмиране, а да ползва стандартни технологии.

Този интерфейс представлява всъщност съвкупност от методи, които други приложения могат дистанционно да извикват. Методите на интерфейсният компонент представляват една своеобразна "обвивка" на тези на сървърният компонент, тъй като често те само извикват даден сървърен метод и връщат резултата от неговото изпълнение без да правят допълнителна обработка над него.

Дефиницията на този интерфейс се вгражда в достъпен отвън WSDL файл, който е специфичен XML документ, където се съдържа цялата информация (като име на метода, параметри към него, тип на връщаната стойност) необходима на клиента на web услугата, за да изивика съответния метод. След като клиентът добие представа за интерфейса, който web услугата предлага, прочитайки нейния WSDL файл, той използва XML базираният SOAP протокол, за да формира SOAP заявка за изпълнение към web услугата. Когато такава заявка се получи, ASP .NET сървърът автоматично я интерпретира и извиква посочения в нея метод, след което резултатът от изпълнението му автоматично се кодира обратно в SOAP съобщение и се връща на заявителя.



Фиг. 5 Свързване на компонентите на МУ с използване на .NET Remoting технология

Фактът, че .NET средата автоматично сериализира и десериализира съобщенията в SOAP протокол е много важен, тъй като това е един от най-тънките моменти при създаването на web услуги, а в случая разработчикът е освободен от това бреме и може да се концентрира върху по-важните задачи на проекта. За осъществяване на връзка между компоненти, намиращи се на различни машини, на пръв поглед в случая може да се използва и технология като DCOM (Distributed Component Object Model). Тогава интерфейсният компонент ще бъде изграден като DCOM сървър, имплементиращ някаква функционалност, а компонентите които желаят да комуникират с мениджъра на устройства, като DCOM клиенти. Дори и да пренебрегнем факта, че DCOM е доста по-труден за имплементиране от web услугите, използването му е в разрез с едно от основните изисквания към интерфейсният компонент, а именно осигуряване на платформена независимост на компонентите, които го ползват. DCOM е технология на Microsoft, налагаща техния модел на проектиране на приложения и използване само на продукти и езици съвместими с Windows, докато web услугите, както многократно беше казано, използват стандартни технологии и могат да се имплеметират от приложения работещи на всякакви платформи. Затова е по-удачно подхода към създаването на интерфейсния компонент да бъде чрез използването на web услуги.

Извикването на методите на интерфейсния компонент на мениджъра на устройства става по стандартния за web услугите начин. Той предлага съвкупност от методи, които всъщност обвиват тези от обекта "сървър", като не позволяват те да бъдат извиквани директно от други приложения от съображения за сигурност. Интерфейсният компонент се свързва със сървърния, използвайки технологията .NET Remoting, както е показано на фиг. 5. Както беше казано и преди, използването на .NET Remoting се налага, тъй като двата компонента - сървърния и интерфейсния - на практика са две различни приложения, намиращи се в отделни AppDomain-и, според терминологията на .NET Framework-а. В такъв случай директното извикване на методите на единия компонент от страна на другия е невъзможно най-малкото, защото операционната система не би го позволила.

При използване на .NET Remoting, двете комуникиращи страни се разделят на страна клиент и страна сървър. Клиентът е приложение, което извиква методи на отдалечен обект (remote object) сървър. В случая обектът клиент е интерфейсният компонент, по-специално ASP .NET web услугата, а обектът сървър е сървърният компонент на мениджъра на устройства. Преди обаче да бъде осъществена каквато и да е комуникация, сървърният обект трябва да определи, т.е да се направи съответната конфигурация, по какъв канал ще си комуникира с клиента, на кой адрес и порт, времето му на живот и какъв тип ще бъде. Тази информация е необходима, за да може по-късно обектът клиент да адресира правилно заявките си за извикване на методи от обекта сървър. Конфигурацията може да бъде зададена, както програмно, чрез съответните класове на .NET Framework, така и чрез специален XML файл, където тя да се опише.

Съществуват два възможни канала за комуникация между обекта клиент и обекта сървър – по TCP/IP и по HTTP. Тоест, съобщенията и данните, които двете страни си обменят, автоматично се сериализират и десериализират в един от двата протокола и се предават вътрешно за машината, използвайки вградения в операционната система стек за съответния протокол. Изборът на един от двата зависи от конкретните изисквания на приложението. В случая аз избрах TCP/IP. На сървърния обект трябва да бъде фиксиран TCP адрес и порт, на който по-късно клиентът ще може да го достигне. Адресът представлява сбор от името на асемблито, на приложението плюс името на класа и именованото пространство (namespace), в което той се намира. В .NET Remoting сървърните обекти могат да бъдат два вида – клиентски активирани (client activated objects - CAO) и сървърно активирани (server activated object - SAO). Сървърните обекти, които ще бъдат достъпни за отдалечения клиент, се регистрират в remoting системата, като такива, но реално се създават при първата клиентска заявка за достъп до тях ако са от втория вид или когато клиента извика оператора New или Activator обекта, ако са от първия вид. Освен това, сървърните обекти могат да бъдат от тип

Singleton или *SingleCall*. Обектите от първия тип винаги имат една инстанция, която обслужва заявките на клиентите, докато тези от втория тип имат по една своя инстанция за всяка клиентска заявка. Друга важна опция, която трябва да се зададе при конфигурацията на сървърния обект, е времето му за живот. Това време почва да се отчита в момента на активирането (клиентски или сървърно) на обекта, като след изтичането му всички указатели към него се унищожават, той се отрегистрира от remoting системата и вече може да бъде събран от системата за събиране на боклука в .NET. В случая, сървърния обект на мениджъра ма устройства е сървърно активиран обект от тип Singleton, времето му на живот е безкрайно, а конфигурацията му е зададена програмно в конструктора на приложението windows услуга.

След като сървърният обект е конфигуриран правилно, той вече може да бъде достигнат от обекти клиенти. За целта клиентският обект, желаещ да ползва методите на сървърния, трябва да зададе абсолютно точно адреса дефиниран в конфигурацията на сървърния обект, иначе remoting системата няма да може да го идентифицира и комуникацията няма да се осъществи. В интерфейсния компонент тази конфигурация се инициализира при настъпването на събитието OnStart() на web услугата. Така, при получаване на първата заявка към услугата, връзката между обекта клиент и обекта сървър ще се осъществи и след това обработката на всяка следваща заявка ще става по вече изградения канал, без да е необходимо той да се създава и да се унищожава всеки път. Ако всичко по настройките е правилно, в remoting системата се регистрират обект клиент и обект сървър, като в първия се създава така наречения proxy клас, който е локално огледално копие на сървърния обект. Така методите на сървърния обект могат да се викат от клиентския все едно са част него, но на практика там има само указатели към методите на сървърния обект, управлявани от remoting системата. Това е така, защото сървърния обект на мениджъра на устройства е конфигуриран да бъде използван като обект, към чийто методи трябва да се обръщаш по адрес (Marshal By Reference object), а не по стойност. Когато обръщението е по стойност, сървърният обект се изкопирва изцяло в клиентския и заявките към него се обслужват от локалното копие. При заявка за изпълнение по адрес към някой от методите на proxy класа, .NET remoting системата приема тази заявка, пренасочва я по предварително избрания канал към сървърния обект, той я изпълнява и връща резултата от изпълнението към клиента по същия този канал, т.е заявката се обслужва от инстанция на обекта в сървърната страна.

Проектиране на базата данни

Базата данни, част от приложението мениджър на устройства, е мястото където се съхранява цялата информация, необходима за функционирането на системата за КУПС като цяло, не само на мениджъра на устройства. Тази информация включва следните категории данни:

- ✓ данни изпратени от устройствата (GPS координати, настъпили събития, състояние на входовете и изходите) – съхранявани съответно в таблици ***GPSRecords*** и ***PinData***

- ✓ данни за самите устройства (уникален номер на устройството, GSM номер, GSM пин код и т.н) - съхранявани в таблица ***GPSDevices***
- ✓ данни за автомобила (марка, модел, регистрационен номер и т.н) – съхранявани в таблица ***GPSDevices***

- ✓ данни за клиентите и потребителите (име, парола, фирма, ниво на достъп и т.н) - съхранявани съответно в таблици ***Custommer*** и ***Users***.

- ✓ данни с описание на

    - каналите за обмен на информация през GSM мрежата, които устройството използва – таблица ***Channels***
    - точките за достъп на GPRS средата (APN - Access Point Name) – таблица ***APN***
    - разрешените GSM номера за рапорт - таблица ***GSMNumbers***
    - нивата на достъп на всеки потребител - таблица ***Access Level***
    - вида на сигнала, който даден вход на устройството измерва - таблица ***PinNames***

Повече информация за таблиците, данните в тях и как са организирани тези данни може да се прочете в глава 4, в точката с описание на имплементацията на базата данни.

Изборът на алгоритъм за съхраняване и обработка на данните е жизнено важен при създаването на база данни. MySQL поддържа два от най-използваните алгоритъма – InnoDB и MyISAM. Като всяко нещо, те си имат предимства и недостатъци и изборът на един от тях зависи от конкретното предназначение на базата данни и изискванията към нея. Също така те позволяват да бъдат прецизно конфигурирани, с цел да паснат още повече на нуждите на системите, където са използват. В случая InnoDB беше избран,

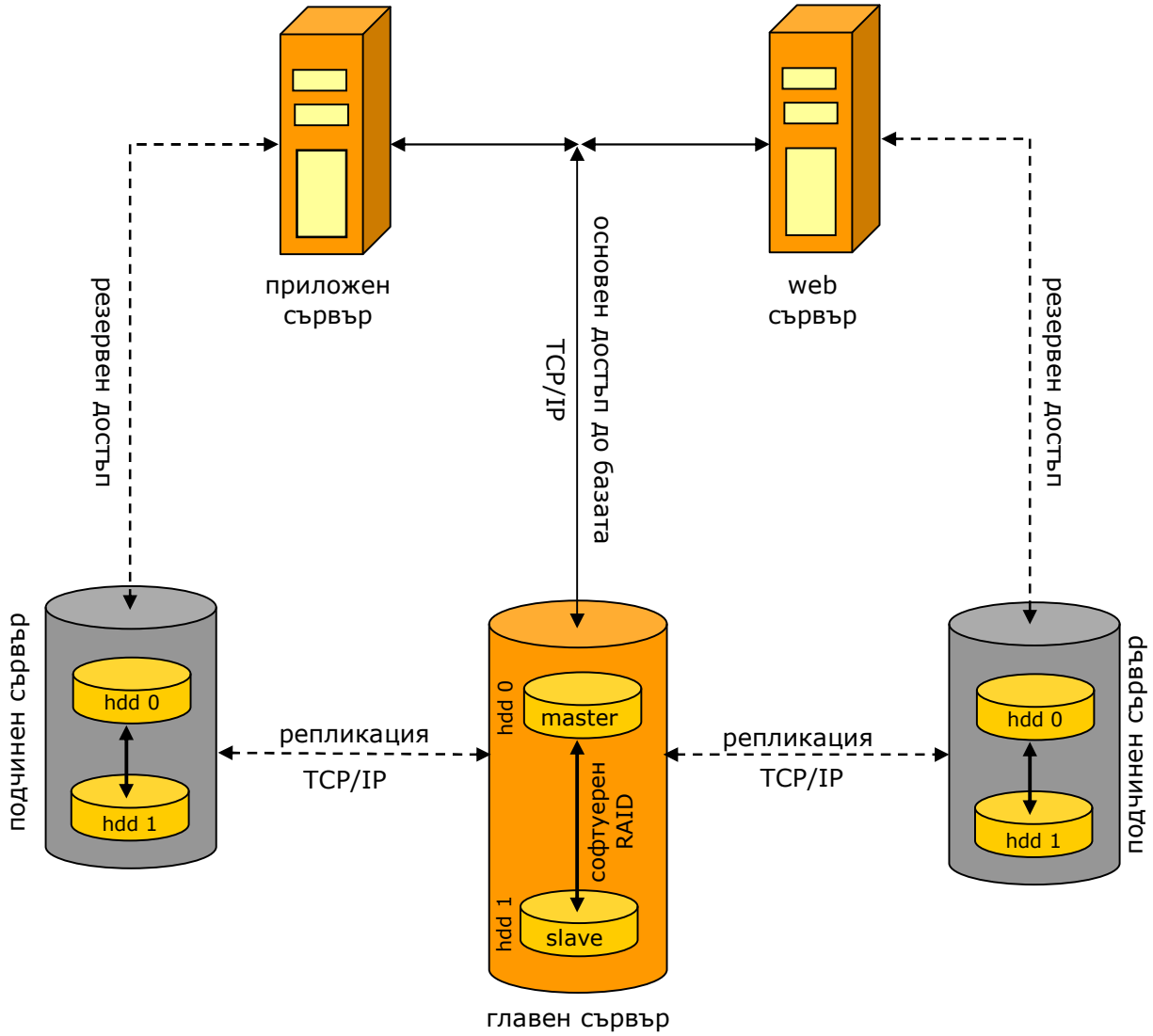
като основен алгоритъм за съхранение на информация в базата данни на мениджърът на устройства, главно заради неговата висока производителност и същевременно малкото процесорно време и мощ, което изисква. Това е може би най-бързият алгоритъм за съхранение и обработка в релационните бази данни в момента, способен да поема натоварвния от порядъка на 800 заявки за вмъкване/обновяване (insert/ update) в секунда. Вероятността мениджърът на устройства да се натовари до такава степен е много голяма, тъй като той трябва да е способен да управлява до 1000 устройства едновременно и базата данни не трябва да прави изключение в това отношение.

Също така, базата данни трябва да е в състояние да съхранява данните на всички 1000 устройства за период от максимум 30 дни. При средна големина на едно съобщение, съдържащо GPS данните от 80b и при средна честота на рапорт на позицията 15 сек., това прави около 13 GB данни генерирани от 1000 устройства и събирани от само един от приложните сървъри, които трябва да могат да се съхраняват в базата. На практика InnoDB няма огрничение, за максималния размер на таблиците, докато MyISAM алгоритъмът има такова от 4GB на таблица. Следователно, InnoDB покрива всички изисквания на нашето приложение и е заслужилият победител при избора на алгоритъм за съхранение и обработка.

За да се съхраняват данните максимум 30 дни в базата, трябва също така да се проектира метод, с който тези данни да се изчистват от там след изтичане на този срок. Тъй като сървърът с базата данни е физически отделна машина е по-сложно този метод да се проектира като част от сървърния или интерфейсния компонент. Затова този метод е реализиран като отделно приложение от тип windows услуга, което се изпълнява директно на сървърната машина. Приложението е настроено на всеки 1 час да сканира таблицата със записите за позицията на даден автомобил и да премахва всеки запис, чиято дата на вмъкване е по-стара от 30 дни спрямо сегашната. Фактът, че се изпълнява като windows услуга, му позволява да заема минимум системни ресурси и по-този начин да не пречи на нормалната работа на базата данни.

Друг важен проблем, който трябва да бъде решен при създаването на база данни, е как да се запази нормалната работа на системата и данните в базата ако цялата машината или част от нея (например твърд диск) спре да функционира по някаква причина. В мениджъра на устройства този проблем е решен чрез използването на репликация (replication) на базата данни и софтуерен RAID между дисковете. Репликацията на база данни е възможност за поддържане на резервно копие на данните в главната машина в реално време, чрез динамичното им копиране в друга машина, която се явява огледално копие на главната. За целта едната машина се обявява за главна и

всички приложения работят само с нея, а другата, съдържаща резервното копие, за подчинена, като броят на подчинените машини може да бъде повече от един. Записите, които се добавят към таблиците в главната база данни ,автоматично се добавят и към таблиците на подчинените машини.



Фиг. 6 Реализация на репликацията в базата данни на МУ

Ако се случи главната машина да откаже и да спре работа, то някоя от подчинените машини веднага става главна и поема работата, без да спира нормалното функциониране на системата. След като повредената машина бъде поправена, тя отново може да бъде пусната в действие (но като подчинена машина тъй като за главна вече е обявена тази, която е поела работата в момента, в който тя се е повредила) като, след включването ѝ към системата MySQL, автоматично ще добави в нейните таблици записите, които

за попълнени през времето, в което тя е била извън строя. Използването на репликация позволява подчинените машини да се намират дори на физически отдалечено място от главната машина, като по този начин, се предовратяват възможните сривове на системата, предизвикани например от спиране на електрическото захранване в цялата сграда.

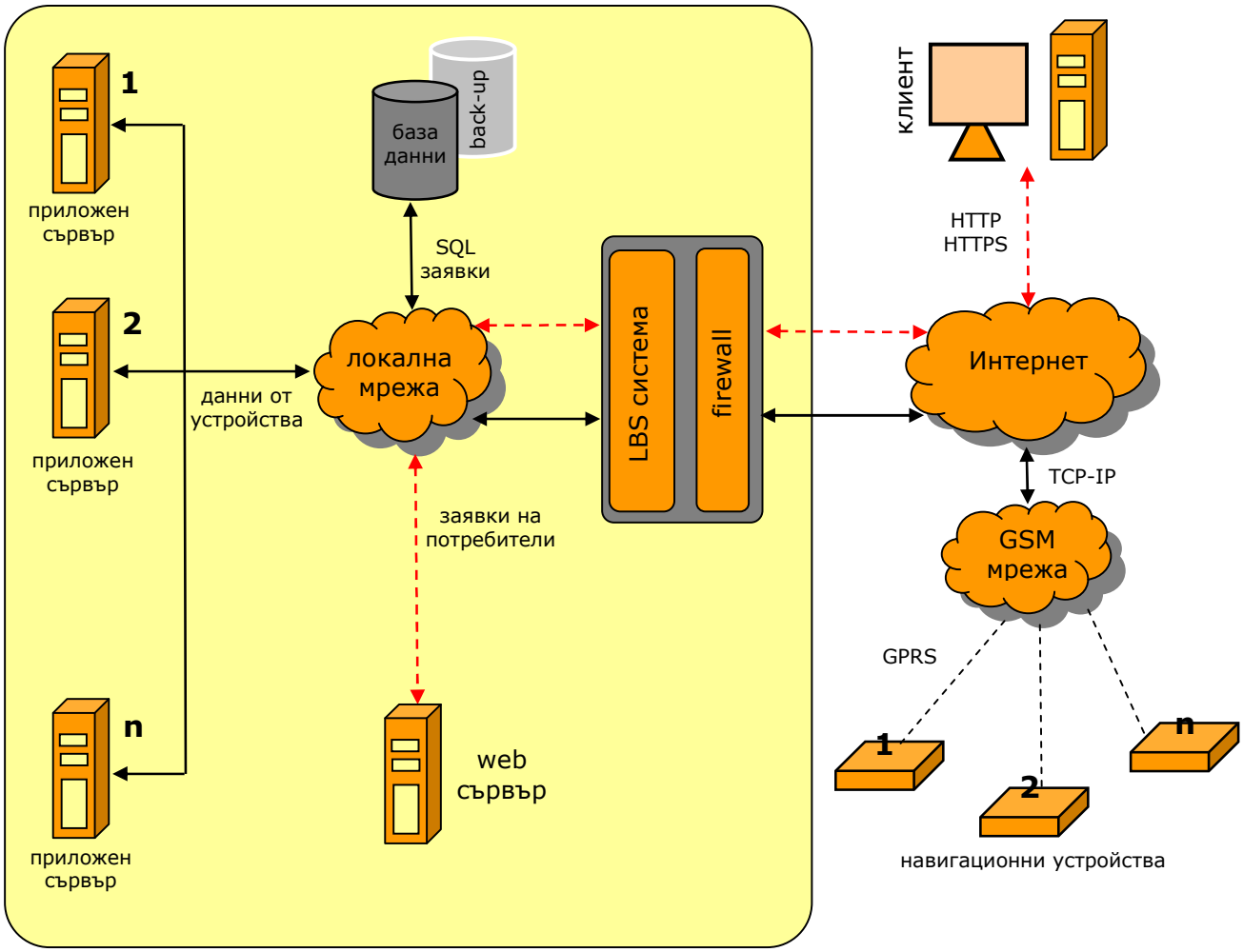
Като двойна защита от опасността от загуба на данни при повреда на машината, на която работи базата към нея е добавена поддръжка и за така наречения софтуерен RAID. Той се реализира, като към самата машина се добавя един подчинен твърд диск, на който Windows XP Server софтуерно поддържа огледално копие на главния. Тоест, когато някакви данни се добавят към главния диск, те автоматично се добавят на същото място и на подчинения. Ако главният диск спре да работи, функциите му веднага се поемат от подчинения без да се предизвиква срив на системата, тъй като информацията на подчинения е идентична с тази на главния.

Проектиране на инфраструктурата

Последният етап от проектирането на приложението "мениджър на устройства" е проектирането на необходимата му инфраструктурата. Тъй като мениджъра на устройства е не просто приложение, изпълняващо се на дадена машина, а подкомпонент на една по-голяма система, то неговото нормално функциониране зависи и от други нейни елементи, които трябва да се организират и свържат по съответен начин. Под инфраструктура на системата се има предвид точно това – как са свързани, по какъв начин комуникират, по какви протоколи и какви данни обменят различните компоненти на мениджъра на устройства, като приложни сървъри, база данни, клиенти, навигационни устройства и т.н.

На фиг. 7 е показано как компонентите на цялата система **Follow Me on the Web** са свързани по между си. Навигационните устройства изпращат своите данни по GPRS канала на GSM мрежата, които след това се пакетират в TCP/IP протокол и през Интернет достигат до някой от приложните сървъри на системата. Преди да достигнат до самия приложен сървър, данните преминават през два етапа. Първо трябва да минат през проверката на firewall-а на локалната мрежа. Ако не успеят, заявката им за връзка или данните им се игнорират и TCP сесията с тях се затваря, а ако успеят достигат до така наречената система за балансиране на товара (LBS) на мениджъра на устройства. Това всъщност е програма, използваща специален алгоритъм за разпределение на заявките на потребителите, когато има

няколко машини за обслужването им, с цел всяка една машина да бъде оптимално натоварена. В случая заявките на потребителите са заявките за връзка и данните на навигационните устройствата, а машините, които ги обслужват са приложните сървъри, на които се изпълнява мениджъра на устройства.



Фиг. 7 Инфраструктурна организация на мениджъра на устройства

Тъй като всеки един приложен сървър има капацитет от максимум 1000 едновременни TCP сесии, когато броят на устройствата обслужвани от системата стане по-голям от общия капацитет на всички приложни сървъри работещи в момента, се налага добавянето на нова машина към системата, която да работи като приложен сървър. В такъв случай може да настъпи ситуация, в която всички или повечето от заявките се обслужват от една от машините, докато останалите през това време бездействат и създават

предпоставки за претоварване или отказ на системата. Затова се налага използването на система за балансиране на товара. Съществуват няколко доста сложни и мощни алгоритъма за балансиране на товара, използвани в някой от най-големите Интернет сайтове и системи в света, които разпределят натоварването в зависимост от моментната заетост на всяка една машина в системата. За нашите цели обаче не ни е необходимо толкова прецизно балансиране и затова е използван прост алгоритъм наречен Round Robin. Чрез него всяка следваща заявка за връзка от дадено устройство се пренасочва към следващия поред приложен сървър в системата, без да се отчита индивидуалната натовареност на машините или на системата като цяло. Например, гледайки фиг. 7, ако една заявка за връзка се обслужи от приложен сървър N1, то следващата ще бъде пренасочена за обслужване към приложен сървър N2, последващата към приложен сървър N3 и т.н Ако даден сървър, към който се пренасочва текущата заявка, е достигнал капацитета си, то тази заявка се препраща последователно към следващия поред, докато се достигне сървър, който може да я обработи. Ако такъв не се намери, то заявката се отхвърля след като и последния поред сървър заяви, че не може да я обслужи. Това наглед просто балансиране, дава много добри резултати и задоволява изцяло нуждите на системата в сегашния й вид.

Както се вижда от фиг.7, всички машини от системата са свързани чрез локална мрежа и комуникират по TCP/IP протокол по между си. Web сървърът, отбелязан там, е машината, която обслужва заявките от клиентските интернет браузъри. Тези заявки идват по стандартния HTTP протокол, но ако се обменя секретна информация, като например потребителско име и парола, се използва криптирания му вариант – HTTPS. Това е и машината, която комуникира с МУ през интерфейсния му компонент и генерира заявки за изпращане на команди към някое устройство. Тъй като web сървърът и приложението, което се изпълнява на него, са извън темата на настоящата дипломна работа, няма да се впускам в детайлно описание на това как той работи. Въпреки това, описвайки инфраструктурата на системата, трябва да се спомене и неговото място в нея, в противен случай схемата на фиг. 7 не би имала смисъл.

Базата данни също комуникира по TCP/IP протокол с всички останали компоненти, като изпълнява стандартни SQL заявки изпратени от някой от тях. Както подробно описах в предходната точка, тя се изпълнява на отделна машина ползвайки MySQL, като сървър за база данни и включваща две нива на подсигуряване от загуба на информация и следователно срив на системата.

Обобщение

Представеното приложение, наречено мениджър на устройства, е комплексна система за автоматизиране на комуникацията между централна компютърна система и на практика неограничен брой навигационни устройства, които използват мрежата на мобилен оператор, като преносна среда за данни. Комуникацията се осъществява през TCP/IP протокол, което позволява вграждането на нови възможности в навигационните устройства и предоставяне на нови услуги на потребителите. По този начин навигационните устройства се издигат от нивото на обикновенни GPS приемници и стават сложни микро компютри способни да поставят под контрол целия автомобил и да позволят на потребителя да го упражнява, от която и да е точка в света, използвайки обикновен web браузър. Използването на GPRS и TCP/IP за пренос на данни е свързано с някой проблеми при тяхната имплементация, а някой биха добавили, че икономически използването им е неизгодно и че SMS съобщенията вършат идеална работа като канал за пренос данни. Истината, както винаги е някъде по средата. GPRS предлага много по-високи скорости при преноса на данни, постоянна свързаност с отдалечената страна, TCP/IP протокол за комуникация и т.н , но цената му на някой места все още е относително висока, а ако се използва като единствен канал за комуникация с навигационното устройството и се случи той да откаже, в един момент може да се стане така, че устройството да се окаже недостижимо по дистанционен път. Точно тук SMS съобщението може да реши проблема, като например се прати съобщение с команда за рестартиране устройството, след което да се възстанови отново комуникацията през GPRS. Ситуации като горната, които могат да възникнат има най-различни и логичният извод е че комбинацията от GPRS, като основен канал за пренос на данни и SMS като резервен е най-удачна. В първата фаза на развитие на приложението "мениджър на устройства" липсва поддръжка на SMS комуникация с навигационното устройство, но ползите от нея са безспорни и за това тя е предвидена за вграждане във втората фаза на проекта.

От гледна точка на дизайна и програмната реализация, мениджъра на устройства е изключително гъвкав, създаден чрез стандартни широко използвани технологии, предлага лесен и бърз начин за интегриране на нови модели навигационни устройства към системата, в която работи и може лесно да бъде интегриран във вече съществуващи такива системи. Липсата на XML поддръжка при интерпретирането на протокола на съобщенията, използван от конкретен модел устройство, може да се изтъкне като недостатък в това отношение, но тя е предвидена за вграждане в следващия етап от развитието на приложението. Във втората фаза е предвидено да бъдат взети и някои по-

сериозни мерки към сигурността на приложението при комуникация с устройството и особено при връзката между двата му основни компонента чрез .NET Remoting и web услуги, защото тези взети досега са само основните такива, които трябва да бъдат вградени.

Интернет ресурси

Microsoft .NET Remoting: A Technical Overview
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/hawkremoting.asp

Статия описваща .NET Remoting технологията в детайли и даваща много препратки към други източници по темата.

Replication in MySQL
http://dev.mysql.com/doc/mysql/en/replication.html

Дава начални сведения за репликацията, като технология и подробни инструкции как да се конфигурира тя под MySQL.

Visual Studio .NET
http://msdn.microsoft.com/vstudio/

Страницата на Visual Studio .NET. Портал към необятно количество ресурси за VS .NET, C#, .NET Framework и т.н

XML.com
http://www.xml.com/

Портал поддържан от световно известното издателство O'Reilly с много "горещи" новини засягащи XML и препратки към най-различни ресурси по темата.

Intel® Load Balancing System
www.intel.com/design/network/solutions/lbs/

Съдържа описание на система за балансиране на товара, разработен от Intel, както и добре представена начална информация за тези системи, като цяло.

Описание на имплементацията

Дизайнът на приложението и изискванията към него вече са изчистени. Сега знаем как ще работи то, от какви компоненти се състои, как те си комуникират, какви технологии и протоколи изпозлват и т.н Време е да направим следващата стъпка, а именно програмната реализация на вече проектираната система. Тя крие своите особености, които е важно да бъдат добре разбрани, ако човек желае да изцяло да познава описваната система.

В тази глава съм систематизирал информация като кратко описание, параметри, тип на връщаната стойност, възможни изключителни ситуации и т.н за всички класове, свойства или методи на съответния компонент от приложението. Пълният изходен код заедно с проектите на Visual Studio .NET, за всеки един компонент може да бъде намерен на съпътстващото дипломната работа CD. Това са приложенията, представляващи DLL библиотека (имплентираща сървърния компонент), приложение windows услуга (обвиващо сървърния компонент), приложение web услуга (имплементиращо интерфейсния компонент) и описание на таблиците на базата данни. Тези компоненти се отнасят и са сглобени в едно приложение съгласно дизайна описан в предходната глава.

Сървърен компонент

DLL библиотека

Клас Server

Public конструктори	
Server	Създава нова инстанция на сървърния клас

Public свойства	
DeviceCount	Връща броя на устройствата свързани в момента към сървъра
OnLine	Връща информация за статуса на сървъра.

Public методи	
Start()	Стартира чакането и обработката на заявки за връзка.
Stop()	Спира чакането и обработката на заявки за връзка.
Restart()	Спира и след това автоматично стартира чакането и обработката на заявки за връзка.
DeviceOnLine()	Връща масив с уникалните номера на устройствата свързани в момента към сървъра
ReadGPRSTimer()	Изпраща команда за прочитане на интервала, на който дадено устройство праща рапорт за позицията по GPRS.

Protected методи	
AcceptConnection()	Стартира чакането и обработката на заявки за връзка.
Listener()	Чака и обработва заявки за връзка
ChangePingTimer()	Променя ping таймера на всеки обект устройство

Protected методи	
SetPingTimer()	Задава интервала, на който всеки обект устройство трябва да прави ping към отсрещната страна според текущата времева зона на заетост
MsgID()	Генерира случайно число, представляващо номер на съобщение
OnSocketError()	Обработва събитието SocketError възникнало в някой от обектите устройства

Server
-ipAddress : IPAddress -port : int -db : DDB -p : Parse -devCount : int +devList : SortedList -t : Thread -tPing : Thread -notBusyZoneS : int -notBusyZoneE : int -medBusyZoneS : int -notBusy : int -medBusy : int -veryBusy : int
+Server(in ip : string, in port : int, in dbConnData : string) +Start() +Stop() +Restart() -Listener() -AcceptConnection(in sock : Socket) +DeviceOnLine() : string[] -SetPingTimer() -ChangePingTimer(in interval : int) +ReadGPRSTimer(in devIDs : string[]) : SortedList -MsgID() : string -OnSocketError(in sender : object, in e : SocketErrEventArgs) +DeviceCount() : int +OnLine() : bool

Фиг. 8 UML диаграма на клас - *Server*

U1.Server конструктор

```
public Server
(
    string ip,
    int port,
    string dbConnData

) : MarshalByReference
```

параметри	
ip	IP адрес на който сървърът трябва да слуша за заявки
port	TCP порт на който сървърът трябва да слуша за заявки
dbConnData	низ съдържащ информация необходима за осъществяване на сървър с база данни

изключителни ситуации

няма

описание

Създава нова инстанция на сървърния обект, като инициализира променливите за IP адрес, порт и низ за връзка с база данни със стойностите, които му се поддават като параметри. Всички методи от този обект ползват именно тези стойности и те не могат да се променят в процеса на работа. При създаването на сървърния обект се инициализират редица други обекти, като обект "интерпретатор", обект "интерфейс към база данни" и т.н, необходими за работата на сървъра. Класа наследява *MarshalByReference* класа, което позволява по-късно методите му да бъдат извиквани по адрес, а не по стойност от обекти клиенти.

U1.Server.DeviceCount свойство

```
public int DeviceCount
```

параметри

няма

изключителни ситуации

няма

връщана стойност

Цяло число представляващо броя на устройствата свързани в момента към сървъра.

описание

Броя на устройствата свързани в момента към сървъра представлява броя на елементите в списъка с такива, който сървърният обект поддържа. Това свойство е само за четене.

U1.Server.OnLine свойство

```
public bool OnLine
```

параметри

няма

изключителни ситуации

няма

връщана стойност

true ако сървъра е активен иначе **false**.

описание

Свойството проверява дали статусът на нишката, слушаща и обработваща заявки, е активен. Ако е така значи сървърът работи нормално. Иначе работата му е нарушена поради някаква причина и нишката е спряна.

U1.Server.Start() метод

```
public void Start()
```

параметри

няма

изключителни ситуации

няма

връщана стойност

няма

описание

Стартира работата на сървърния компонент. Под стартиране се има предвид активиране на нишката, слушаща и обработваща заявки за връзка на IP адреса и порт зададен при инициализирането на сървърния обект, отваряне на връзката с базата данни и стартиране на нишката, извършваща периодичен "ping" на всички устройства. За да се стартира работата на "Мениджъра на устройства", задължително трябва да се изпълни този метод, иначе обектът ще остане просто инициализиран, но няма да върши никаква работа.

U1.Server.Stop() метод

```
public void Stop()
```

параметри

няма

изключителни ситуации

няма

връщана стойност

няма

описание

Спира работата на сървърния компонент. Спирането се състои в прекратяване на нишката, слушаща и обработваща заявки, последователно обхождане на всички елементи от списъка с активни обекти "устройства" и тяхното унищожаване и затварянето на връзката с базата данни. След извикването на този метод, работата на сървъра спира и никакви заявки не се приемат, нито се обработват данни идващи от устройствата. За да се възобнови работата на сървъра и на сървъният компонент, трябва отново да се извика метода Start().

U1.Server.Restart() метод

```
public void Restart()
```

параметри

няма

изключителни ситуации

няма

връщана стойност

няма

описание

Рестартира работата на сървърния компонент чрез последователно извикване първо на метода Stop() после на метода Start().

U1.Server.DeviceOnLine() метод

```
public string[] DeviceOnLine()
```

параметри

няма

изключителни ситуации

няма

връщана стойност

Масив тип текстов низ, елементите на който представляват уникалните номера на устройствата свързани в момента към сървъра.

описание

Методът преправя вътрешния списък от устройства свързани в момента, който сървърния обект поддържа и копира в свой масив от тип текстов низ само елементите, представляващи уникални номера на устройства и накрая връща този масив като резултат от изпълнението.

U1.Server.ReadGPRSTimer() метод

```
public string ReadGPRSTimer
(
   string devID,
   bool waitForReplay
)
```

параметри	
devID	уникалният номер на устройството, към което трябва да се изпрати командата
waitForReply	true, ако извикващия иска метода да чака за отговор иначе false

изключителни ситуации

няма

връщана стойност

Интервалът от време, в секунди, на който устройството е програмирано да рапортува позицията си. Null ако не се получил отговор или отговора е бил некорктен.

описание

Методът генерира текстов низ, представляващ команда за прочитане на таймера (интервала на рапорт) на устройството по протокола, използван от навигационните устройства типи U1. След като генерира командата, методът претърсва списъка от устройства, свързани в момента, намира устройството със съответния номер, взима указател към обекта му и използва метода от него за изпращане на команда към устройството. Ако е програмиран да чака за отговор на командата, методът ще чака, ако не е ще се върне веднага след изпращането й. Ако отговор се получи, методът го конвертира в секунди и го връща като отговор на заявителя.

U1.Server.AcceptConnection() метод

```
private void AcceptConnection
(
   Socket sock
)
```

параметри	
sock	сокет, който е назначен за комуникация с отсрещната страна при приемане на заявката за връзка

изключителни ситуации	
SocketException	комуникационна грешка при работа със сокета. Ситуацията не се обработва, а се прехвърля на по-високо ниво в стека заедно с кода за грешката
ArgumentException	устройство с дадения номер вече е свързано към системата
ObjectDisposedException	сокетът е унищожен по някаква причина преди да стигне до метода

връщана стойност

няма

описание

Когато сървърният обект получи заявка за връзка, той й назначава сокет, през който да се комуникира със страната желаеща връзка и вика метода AcceptConnectio(), за да валидира заявката. Методът извлича уникалния номер на устройството, желаещо връзка, проверява го в базата данни дали е валиден и ако всичко е наред създава нов обект “устройство” за тази заявка, като го инициализира със сокета, който е първоначално назначен и накрая го добавя към списъка с устройства свързани в момента към сървъра. Ако настъпи грешка при една от горните стъпки, обработката на заявката се прекратява и сокета, който й е назначен се унищожава.

U1.Server.Listener() метод

```
private void Listener()
```

параметри

няма

изключителни ситуации

SocketException	комуникационна грешка при работа със сокета.
ThreadAbortException	спиране на нишката

връщана стойност

няма

описание

Това е методът изпълняван в нишката чакаща и обработваща заявки за връзка към сървъра. Когато се получи заявка, се вика метода AcceptConnection() и продължава да слуша за нови заявки.

U1.Server.ChangePingTimer() метод

```
private void ChangePingTimer
(
    int interval
)
```

параметри	
interval	Интервала, в секунди, на който всеки един обект трябва да прави ping към отсрещната страна

изключителни ситуации

няма

връщана стойност

няма

описание

Методът обхожда последователно от началото до края елементите от списъка с устройства, свързани в момента към сървъра и използва метода Change() на обектите им, за да смени таймера (интервала от време), на който всеки един обект трябва да прави ping отсрещната страна

U1.Server.SetPingTimer() метод

```
private void SetPingTimer()
```

параметри

няма

изключителни ситуации

няма

връщана стойност

няма

описание

Този метод се използва от специална нишка, която периодично, на зададен период от време, проверява текущото време и сменя ping таймера на обектите устройства в зависимост от това, в коя зона на заетост е деня в момента. Зоните на заетост, могат да се параметризират, и представляват интервали от време през деня, в които GSM мрежата е различно натоварена.

U1.Server.MsgID() метод

```
private string MsgID()
```

параметри

няма

изключителни ситуации

няма

връщана стойност

Текстов низ съдържащ генерираното число в шестнайсетичен вид

описание

Генерира случайно число в интервала 8001-65535, което се използва като номер на съобщението, което се изпраща към устройството. След като получи съобщението успешно, устройството връща този номер, за да удостовери, че всичко е наред. Ако номерът не се получи обратно или се получи грешен номер, съобщението се счита за неуспешно изпратено и се изпраща отново. Използването на номера на съобщенията се налага от протокола използван при комуникация с навигационни устройства тип U1.

U1.Server.OnSocketError() метод

```
private void OnSocketError
(
    object sender,
    U1Device.SocketErrEventArgs e
)
```

параметри	
sender	обект генерирал събитието
e	данни, които обектът генерирал събитието праща заедно с него

изключителни ситуации

няма

връщана стойност

няма

описание

Обработва събития свързани с грешки в обектите устройства. Когато в даден обект настъпи грешка, от която той не може да се възстанови, той се самоунищожава и генерира събитие уведомяващо сървърния обект за това. След получаване на съобщение за настъпило такова събитие сървърния обект изважда обекта, изпратил съобщението, от списъка си с устройства свързани в момента към него.

Клас U1Device

Public конструктори	
U1Device	Създава нова инстанция на обект устройство

Public свойства	
BytesRead	Връща количеството данни, в байтове, изпратени от устройството и прочетени от обекта
BytesSend	Връща количеството данни, в байтове, изпратени от обекта към устройството
OnLine	Показва дали връзката с устройството е активна

Public методи	
Command()	Изпраща низ, представляващ команда, към устройството

Protected методи	
Read()	Прочита данните пристигнали на сокета
Send()	Изпраща данни чрез сокета
Worker()	Чака и обработва данни, идващи от устройството
Ping()	Изпраща ping съобщение към устройството на даден интервал от време

U1Device
#sock : Socket #devID : string #devDBID : string +timer : Timer #GPSdata : string[] #DevConf : string[] #bytesRead : byte[] #bytesSend : byte[] -numOfBytesRead : int -numOfBytesSend : int #db : DDB #p : Parse #t : Thread -disposed : bool = false
+U1Device(in sock : Socket, in devID : string, in db : DDB, in buffSize : int, in devDBID : string) -Worker() -Read() : string -Send(in str : string) : int +Command(in comm : string, in waitForReply : bool) : string -Ping(in state : object) *+SocketErr() : SocketErrEventHandler* #OnSocketErr(in e : SocketErrEventArgs) +SocketError(in devID : string, in e : SocketException) +BytesRead() : int +BytesSend() : int +OnLine() : bool +Dispose() #Dispose(in disposing : bool) -~U1Device()

Фиг. 9 UML диаграма на клас – U1Device

U1.U1Device конструктор

```
public U1Device
(
   Socket sock,
   string devID,
   DDB db,
   int buffSize,
   string devDBID
)
```

параметри	
sock	сокет, на който устройството е свързано
devID	уникалният номер на устройството
db	обект интерфейс към база данни
buffSize	размер на буфера на сокета, за четене и изпращане на данни
devDBID	номер на устройството в базата данни

изключителни ситуации

няма

описание

Създава нова инстанция на обекта устройство, като го инициализира с необходимите данни, за да се осъществи комуникация по TCP/IP между обекта и устройството. След като обектът бъде създаден, комуникацията с устройството, което той обслужва, става само през него.

U1. U1Device.BytesRead свойство

```
public int BytesRead
```

параметри

няма

изключителни ситуации

няма

връщана стойност

Цяло число, представляващо количеството данни в байтове, които са изпратени от устройството и прочетени от обекта

описание

Представлява брояч, който се инкрементира при всяко четене на данни от сокета със съответната стойност равна на количеството прочетени байтове.

U1. U1Device.BytesSend свойство

```
public int BytesSend
```

параметри

няма

изключителни ситуации

няма

връщана стойност

Цяло число, представляващо количеството данни в байтове, които са изпратени от обекта към устройството

описание

Представлява брояч, който се инкрементира при всяко изпращане на данни през сокета със съответната стойност равна на количеството изпратени байтове.

U1. U1Device.OnLine свойство

```
public bool OnLine
```

параметри

няма

изключителни ситуации

няма

връщана стойност

true ако нишката на обекта е активна и обработва данни от устройството, иначе **false**

описание

Проверява дали нишката на съответния обект устройство е активна, което означава, че връзката между обекта и устройството не се разпаднала.

U1. U1Device.Command() метод

```
public string Command
(
   String comm,
   bool waitForReply
)
```

параметри	
comm	текстов низ, представляващ команда към устройството
waitForReply	true ако трябва да се чака за отговор на командата, иначе false

изключителни ситуации

няма

връщана стойност

Текстов низ, представляващ отговора на командата получен от устройството, null ако отговор на командата не се очаква или отговорът е грешен.

описание

Изпраща подадения му текстов низ към устройството през сокета на съответния обект. Ако се изисква от него чака отговор на командата и ако го получи, го връща към изпращача, както го е получил без допълнителна обработка. Ако получаване на отговор не се изисква от него или отговор не се получи за определено време, връща null към изпращача.

U1. U1Device.Read() метод

```
private string Read()
```

параметри

няма

изключителни ситуации

`SocketException`	комуникационна грешка при работа със сокета.
`ObjectDisposedException`	сокетът е унищожен по някаква причина преди извикването на метода

връщана стойност

Текстов низ, представляващ данните прочетени от сокета

описание

Прочита байтовете, ако има такива, пристигнали на сокета, конвертира ги в текстов низ и ги връща на заявителя. Ако на сокета няма пристигнали данни, методът не се връща веднага, а блокира изпълнението, докато пристигнат данни.

U1. U1Device.Send() метод

```
private int Send
(
  string str
)
```

параметри	
str	текстов низ, който да се изпрати към устройството

изключителни ситуации	
SocketException	комуникационна грешка при работа със сокета.
ObjectDisposedException	сокетът е унищожен по някаква причина преди извикването на метода

връщана стойност

Цяло число, представляващо количеството данни изпратени през сокета; -1 ако настъпи грешка и нищо не е изпратено.

описание

Изпраща подадения му текстов низ през сокета на съответния обект устройство, като преди да го изпрати, го конвертира в байтове.

U1.U1Device.Worker () метод

```
private void Worker()
```

параметри

няма

изключителни ситуации

SocketException	комуникационна грешка при работа със сокета.
ObjectDisposedException	сокета е унищожен по някаква причина по време на изпълнението на нишката

връщана стойност

няма

описание

Това е метода, който всяка главна нишка обект устройство. Метода проверява на даден интервал от време дали има пристигнали данни на сокета на обекта и ако има ги прочита, интерпретира и записва в базата данни. Ако получените данни не могат да се интерпретират правилно, те се игнорират и не се записват в базата.

U1.U1Device.Ping () метод

```
private void Ping()
```

параметри

няма

изключителни ситуации

няма

връщана стойност

няма

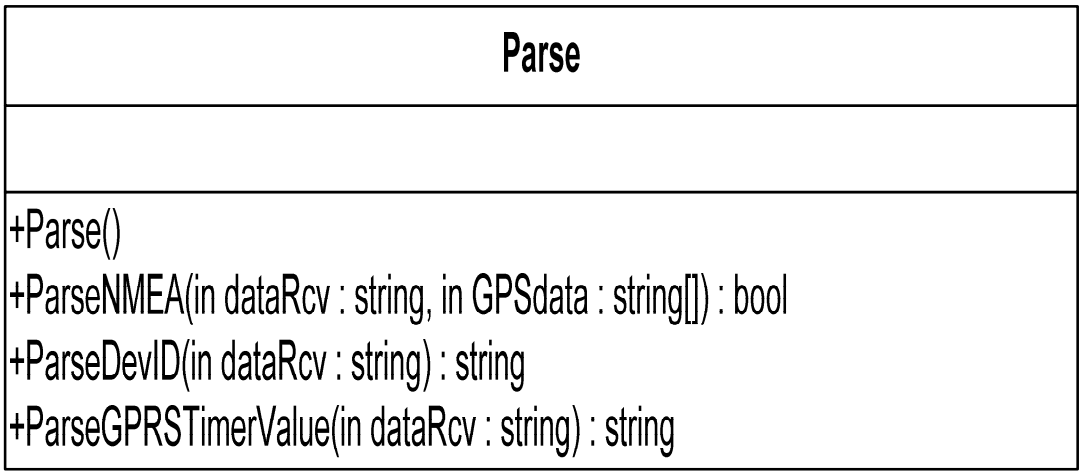
описание

Изпраща ping съобщение, на зададен интервал, към устройството през сокета на обекта, за да поддържа TCP сесията между двете страни активна.

Клас Parse

Public конструктори	
Parse	Създава нова инстанция на обект интерпретатор

Public методи	
ParseDevID	Извлича уникалния номер на устройство от дадено съобщение
ParseNMEA	Извлича данните за географски координати, скорост, време и т.н от NMEA съобщенията
ParseGPRSTimerValue	Конвертира в секудни стойността получена, като отговор на командата за четене на таймера за рапорт по GPRS



Фиг. 10 UML диаграма на клас – Parse

U1. Parse конструктор

```
public Parse()
```

параметри

няма

изключителни ситуации

няма

връщана стойност

няма

описание

Създава нова инстанция на обект интерпретатор.

U1. Parse.ParseDevID() метод

```
public string ParseDevID
(
   String dataRcv
)
```

параметри	
dataRcv	текстов низ, представляващ съобщението получено от дадено устройство

изключителни ситуации

няма

връщана стойност

Текстов низ съдържащ уникалния номер на устройство извлечен от подаденото съобщение. null ако съдържанието на съобщението съдържа грешни данни.

описание

Преглежда целия текстов низ, който му се подава и проверява дали е с необходимата дължина, ако е продължава с обработката в съответствие с U1 протокола на съобщенията, извлича уникалния номер на устройство и го връща като резултат от работата. Ако съобщението не е с необходимата дължина това значи, че не може да се разчита, че уникалния номер на устройство вътре е валиден или че данните в него са коректни.

U1. Parse.ParseNMEA() метод

```
public bool ParseNMEA
(
   string dataRcv,
   string[] GPSdata
)
```

параметри	
dataRcv	текстов низ, представляващ съобщението получено от дадено устройство
GPSdata	масив от тип текстов низ, чийто елементи съдържат съответната извлечена стойност за географска ширина и дължина, скорост, време и т.н

изключителни ситуации

няма

връщана стойност

true ако всички стойности са извлечени коректно иначе false

описание

Преглежда целия текстов низ, който му се подава, проверява дали е с необходимата дължина, ако е продължава с обработката, извлича стойностите от NMEA съобщението за географска ширина и дължина, скорост, време и т.н и записва всяка една на съответното място в масива `GPSdata[]`. За да ползват данните извлечени от съобщението, викащите методите трябва да си подготвят абсолютно същия масив `GPSdata[]`, който ParseNMEA да запълни.

U1. Parse.ParseGPRSTimerValue() метод

```
public string ParseGPRSTimerValue
(
   string dataRcv
)
```

параметри	
dataRcv	текстов низ, представляващ съобщението получено от дадено устройство

изключителни ситуации	
ArgumentOutOfRangeException	полученото съобщение не е с необходимата дължина

връщана стойност

текстов низ съдържащ извлечената от съобщението стойност на таймера за рапорт в секунди. null ако настъпи грешка при обработката.

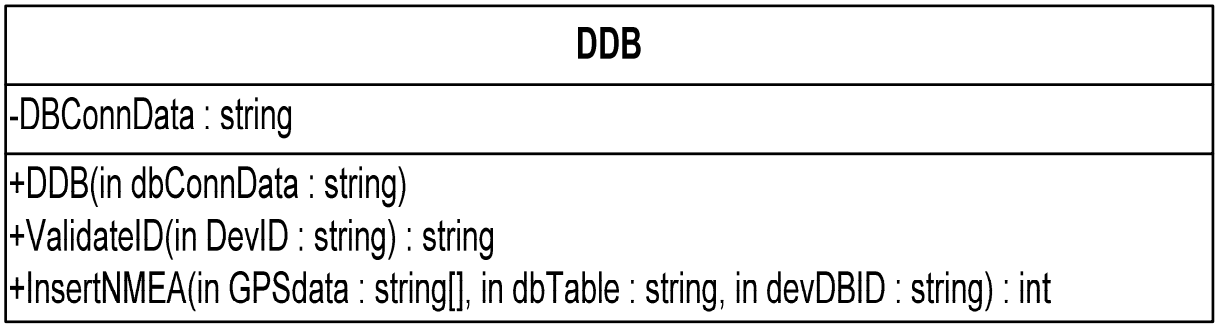
описание

Използва се за обработка на съобщения изпратени в отговор на команда за прочитане на стойността на таймера на устройството за рапорт по GPRS. Отговора на тази команда е специфично съобщение определено от U1 протокола и в него стойността на таймера не е дадена директно в секунди. Метода преглежда дали подаденото съобщение е с необходимата дължина и ако е продължава обработката, извлича стойността на таймера, конвертира я в секунди и я връща, като резултат от изпълнението. Ако съобщението не е с необходимата дължина, то данните в него се считат за некоректни и то се игнорира.

Клас DDB

Public конструктори	
DDB	Създава нова инстанция на обект интерфейс към база данни

Public методи	
InsertNMEA	Записва стойностите излечени от NMEA съобщение
ValidateID	Проверя дали даден номер на устройство съществува в базата данни и дали това устройство в маркирано като активно



Фиг. 11 UML диаграма на клас – DDB

U1. DDB конструктор

```
public DDB
(
   string dbConnData
)
```

параметри	
dbConnData	текстов низ, съдържащ информация, като потребителско име, парола и т.н необходима за осъществяване на връзка с база данни

изключителни ситуации

няма

връщана стойност

няма

описание

Създава нова инстанция на обект интерфейс към база данни, без да осъществява връзка към сървера. Пула от връзки се създава при първото извикване на някой от методите на обекта, които използват текстовия низ за връзка даден в конструктора, когато те извикат метода Open() на драйвера за MySQL под Visual Studio .NET. При следващото извикване на Open(), от някой друг метод използващ абсолютно същия низ за връзка, нов връзка не се създава, а се дава такава от пула, ако е налична.

U1. DDB.InsertNMEA() метод

```
public int InsertNMEA
(
   string[] GPSdata,
   string dbTable,
   string devDBID
)
```

параметри	
GPSdata	масив съдържащ извлечените от NMEA съобщението стойности за географска ширина и дължина, скорост, време и т.н
dbTable	името на таблицата от базата данни, където трябва да се запишат данните от NMEA съобщението
devDBID	номер на устройството в базата данни

изключителни ситуации

няма

връщана стойност

Броя колони от таблицата, в които са записани данни

описание

Записва данните извлечени от дадено NMEA съобщение и подадени му в масив, в посочената таблица от базата. Метода не връща информация ако настъпи грешка при записването, а само броя на колоните, в които в крайна сметка е записано нещо. За да следи за грешки извикващия трябва да знае в колко точно колони трябва да се запишат данни и ако върнатата от метода стойност не отговаря на очакваното значи е настъпила грешка.

U1. DDB.ValidateID() метод

```
public string ValidateID
(
   string DevID
)
```

параметри	
DevID	номера на устройство, който трябва да се провери

изключителни ситуации

няма

връщана стойност

номера на устройството в базата данни ако неговия уникален номер е валиден и то е записано, като активно, иначе null.

описание

Проверява дали подадения номер на устройство съществува в базата данни и дали устройството е записано, като активно в нея.

Приложение windows услуга

Приложението windows услуга, част от сървърният компонент, е абсолютно стандартно и е създадено чрез шаблона, който Visual Studio .NET предлага за изграждане на тези приложения. То е една своеобразна обвивка на DLL бибилиотеката на сървърния компонент, която я изпълнява в привилигирован режим на операционната система. Поради тази причина няма да го описвам на ниво класове и методи, както DLL библиотеката от предходната глава, а ще се спра само на местата, където има специфични промени и ще се постарая да ги изясня. Останалите класове на приложението са генерирани от Visual Studio .NET и са абсолютно стандартни за тях. Пълният код и проект на приложението може да бъде намерен на съпътстващото дипломната работа CD.

Както стана ясно сървърният и интерфейсният компонент на "мениджъра на устройства", са две отделни приложения, намиращи се в различни AppDomain среди и комуникиращи по между си използвайки .NET Remoting технологията. За целта в Remoting системата двата компонента се регистрират съответно като обект сървър и обект клиент. Регистрирането на сървърният компонент като такъв става в конструктора на приложението windows услуга по следния начин:

```
1\ tcpChnl = new TcpChannel(30301);

2\ ChannelServices.RegisterChannel(tcpChnl);

3\ U1srv = new Server("10.0.0.89",30300, dbConnData);

4\ ObjRef srvRef = RemotingServices.Marshal(U1srv, "U1srvURI");
```

Тъй като двата компонента ще използват TCP канал за комуникация по между си, на първия ред създаваме обект от клас `TcpChannel` и го инициализираме с 30301 порт, а на втория ред използваме метода `ChannelServices.RegisterChannel`, за да го регистрираме в Remoting системата. Така вече имаме изграден канал, по който двата "отдалечени" обекта (remote objects) ще могат да комуникират. По-интересни са последните два реда от кода. Споменахме, че освен всичко останало, за нашето приложение е жизнено важно сървърния компонент да предлага, като достъпен отвън точно тази инстанция на сървърният си обект, която обслужва устройствата в момента, защото само от нея и от никоя друга може да бъде получен достъп до тях. Използването на достъп по адрес до методите на сървърният обект няма да бъде достатъчно в случая, защото ако използваме стандартния механизъм за регистриране на обект достъпен по адрес, то при регистрацията му трябва да посочим точен адрес, на който този обект може да бъде

намерен. Тогава при активиране на този обект, се създава негова нова инстанция, връща се указател към нея и клиента създава при себе си така наречение proxy клас, чрез който после извиква методите на сървърният обект по адрес, а не по стойност. Очевидно подобен вариант нас не ни устройва, защото новата инстанция на обекта няма да е тази, която реално обслужва устройствата и на практика няма да имаме достъп до тях.

Проблемът е решен от разработчиците на .NET Framework, като създават класа ObjRef, който се използва, за да се предаде указател към даден обект сървър на негов клиент. Обектите от този клас са проектирани да съдържат в себе си цялата необходима информация (типа на сървърният обект, неговото точно местонахождение и комуникационна информация), за да бъде достигнат той от своите клиенти. Тоест адреса на сървърният обект няма да бъде фиксиран, а ще бъде предаден динамично на обекта клиент, чрез указател към точно определена негова инстанция. Именно този подход е използван и тук. На третия ред от кода по горе, се създава обект от клас сървър, като се инициализира с необходимата му информация. Тъй като обекта наследява класа *MarshalByReference*, то можем да използваме функцията `RemotingServices.Marshal`, за да го конверитаме автоматично в обект от тип *ObjRef* и да му назначим URI (Uniform Resource Indentificator), чрез който, както ще видим по-късно, точно тази негова инстанция, която създадохме ще бъде достъпна за обекти клиенти.

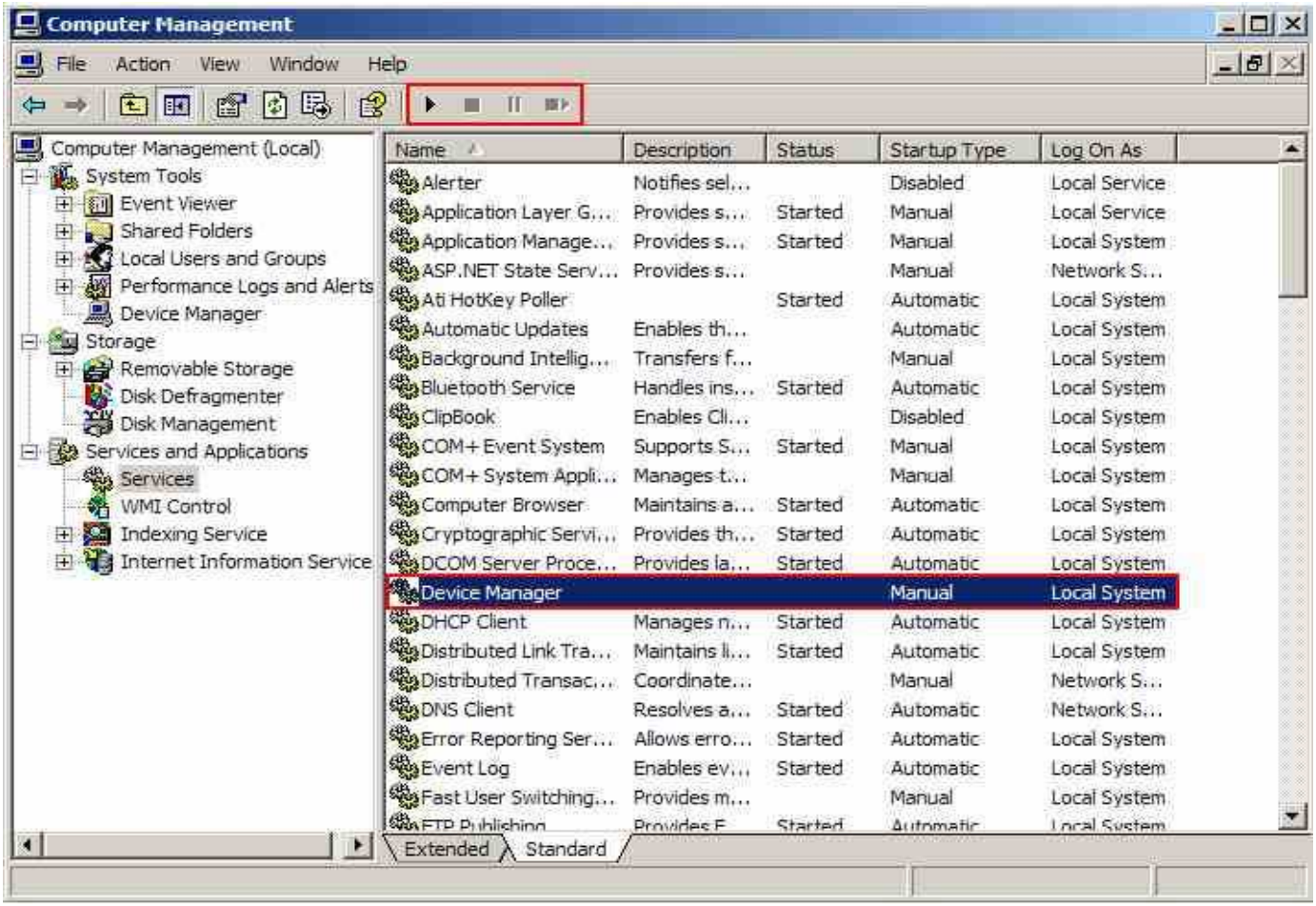
Други места, където приложението е модифицирано са стандартните за windows услугите събития OnStart() и OnStop(). Те настъпват съответно при стартиране и спиране работата на услугата или програмно от някое друго приложение или от SCM чрез стандартния интерфейс, който предлага за контрол на услуги. В случая "мениджъра на устройства" няма свой потребителски интерфейс и ползва този на SCM. Двете събития са предефинирани, така че при стартиране на услугата тя ще извиква метода Start() на сървърния компонент, а при спиране метода му Stop(). След като всички модификации по приложението са направени остава само то да бъде регистрирано като услуга в системата, с цел да може да бъде контролирано от SCM. Има два начина за регистрация на windows услуга, които Visual Studio .NET предлага. Първият е чрез създаване на специална инсталационна програма на приложението, която ще извърши автоматично всички действия необходими за регистрация, а вторият е да направим това ръчно чрез използване на програмата `installutil`, която е част от пакета Visual Studio .NET. Категорично първият вариант е по-удобен, но тъй като "мениджъра на устройства" няма да бъде широко комерсиален продукт сметнах, че еднократното извършване на едно просто действие като извикване на една програма от командния ред не би било толкова затормозяващо. Ето как става това:

- Стартирайте командия ред на Windows и отидете в директорията, в която се намира компилираният проект на приложението windows услуга.

- Извикайте програмата `installutil` и дайте като параметър името на компилирания изпълним файл на приложението windows услуга (в случая `installutil DMWinService.exe`)

  Заб. Уверете се, че пътя до програмата `installutil` е зададен при вас или стартирайте командния ред на пакета на Visual Studio .NET, който автоматично ще го добави, но не за постоянно.

Ако горните стъпки минат без проблем приложението windows услуга трябва да е вече регистрирано и достъпно чрез конзолата на SCM, както е показано на фиг. 12. За да стартирате или спрете услугата ползвайте бутоните Start, Stop или Restart на конзолата.



Фиг. 12 Мениджъра на устройства достъпен през SCM конзолата

Интерфейсен компонент

Интерфейсният компонент на "мениджъра на устройства" е имплементиран чрез стандартна web услуга, създадена с използване на шаблона, който Visual Studio .NET предоставя за подобни приложения. По същите причини както при приложението windows услуга и web услугата няма да бъде разглеждана на ниво клас и методи, а само ще се спра на местата, където тя е модифицирана, за да изпълнява изискванията към нея.

Основната задача на интерфейсния компонент е да прави връзка между външни приложения и сървърният компонент на МУ, предоставяйки им програмен интерфейс към него и както вече добре знаем това става чрез използване на .NET Remoting технологията. След като видяхме в предходната точка, как се конфигурира обекта сървър в приложението windows услуга нека видим, как става конфигурацията и в обекта клиент какъвто е web услугата. Най-доброто място за инициализиране на конфигурацията е при първото извикване на метод от web услугата или иначе казано при настъпване на събитието ApplicationStart(). Ето какво става, когато то настъпи:

```
1\ string cPath = Server.MapPath("remote_client.config");

2\ System.Runtime.Remoting.RemotingConfiguration.Configure(cPath);
```

Споменахме, че конфигурирането на обектите клиент и сървър в .NET Remoting може да става или програмно в самото приложение или чрез описване на параметрите на конфигурацията в XML файл със специален синтаксис. Тук това е направено по втория начин. В първия ред от кода по-горе в променливата `cPath` се записва пътя до файл на име `remote_client.config`, който представлява и конфигурационния файл. Нека погледнем какво има в него:

```
<configuration>
      <system.runtime.remoting>
            <application>

                  <client>

                        <wellknown url="tcp://localhost:30301/U1srvURI"
                        type="U1.Server, DeviceManager" />

                  </client>

            </application>
      </system.runtime.remoting>
</configuration>
```

Накратко казано в първите три полета от конфигурацията се указва, че това е конфигурационен файл, за приложение клиент в системата на .NET Remoting. Специфичната част е описанието на това, къде и как обекта клиент да намери обекта сървър, което става в полето `<client>`. В полето `<wellknown>` се задава адреса на сървърно активиран обект (server activated object), с който клиента иска да се свърже и ползва, както и неговият тип и асембли в което се намира. В случая е посочено, че сървърният обект е от тип *U1.Server*, част от асембли на име *DeviceManager* и може да бъде намерен на локалната машина на порт 30301. Както видяхме при конфигурацията на сървърният обект, *U1srvURI* е адреса на обект от тип *ObjRef*, който динамично ще предаде указател към конкретна негова инстанция. От тук нататък всички методи въпреки, че създават обекти "сървъри" със запазената дума *new*, ще получават не указател към нова инстанция на този обект, а към тази, към която току-що се свързахме. След като вече сме изградили файл с съдържащ конфигурацията на обекта клиент, остава само да кажем на приложението от къде може да я прочете. Това става, както е показано на втория ред от кода по-горе, чрез извикване на метода `RemotingConfiguration.Configure`, който използва преди това инициализираната променлива с пътя до файла - `cPath`.

Както споменахме в структурното описание на приложението, за да се възползват външни такива от интерфейса, който web услугата им предлага те просто трябва да добият предства за него прочитайки WSDL файла й и на базата на тази информация да изпращат съответните SOAP заявки за изпълнение на някой от методите. ASP .NET средата поема цялата работа по приемане, обработване и изпълнение на SOAP заявките към web услугата. От тук нататък за разработчика програмирането се свежда до обикновенното писане на методи и класове, като единственото, което се изисква от него е поставянето на атрибута *[WebMethod]* пред тези методи, които желае да предостави за изпълнение на външни приложения. За да бъде web услугата достъпна обаче се налага извършването на някой конфигурации на локалната машина. Ето какви са те:

- Уверете се, като погледнете в секцията *Control Panel->Add/Remove Programs*, че имате инсталиран *Internet Information Server* на вашата машина. Ако имате прескочете следващите подстъпки ако не направете следното:
    - ✓ От *Add Remove Programs* менюто изберете *Add/Remove Windows Components.*
    - ✓ От появилия се списък сложете отметка на опцията *Internet Information Services (IIS).*
    - ✓ Изберете OK, сложете ако ви се поиска, инсталационния диск на Windows и изчакайте проключването на инсталацията.

- Изкопирайте папката с проекта на приложението web услуга в директорията C:\Inetpub\wwwroot. Така web услугата вече ще бъде регистрирана в IIS.

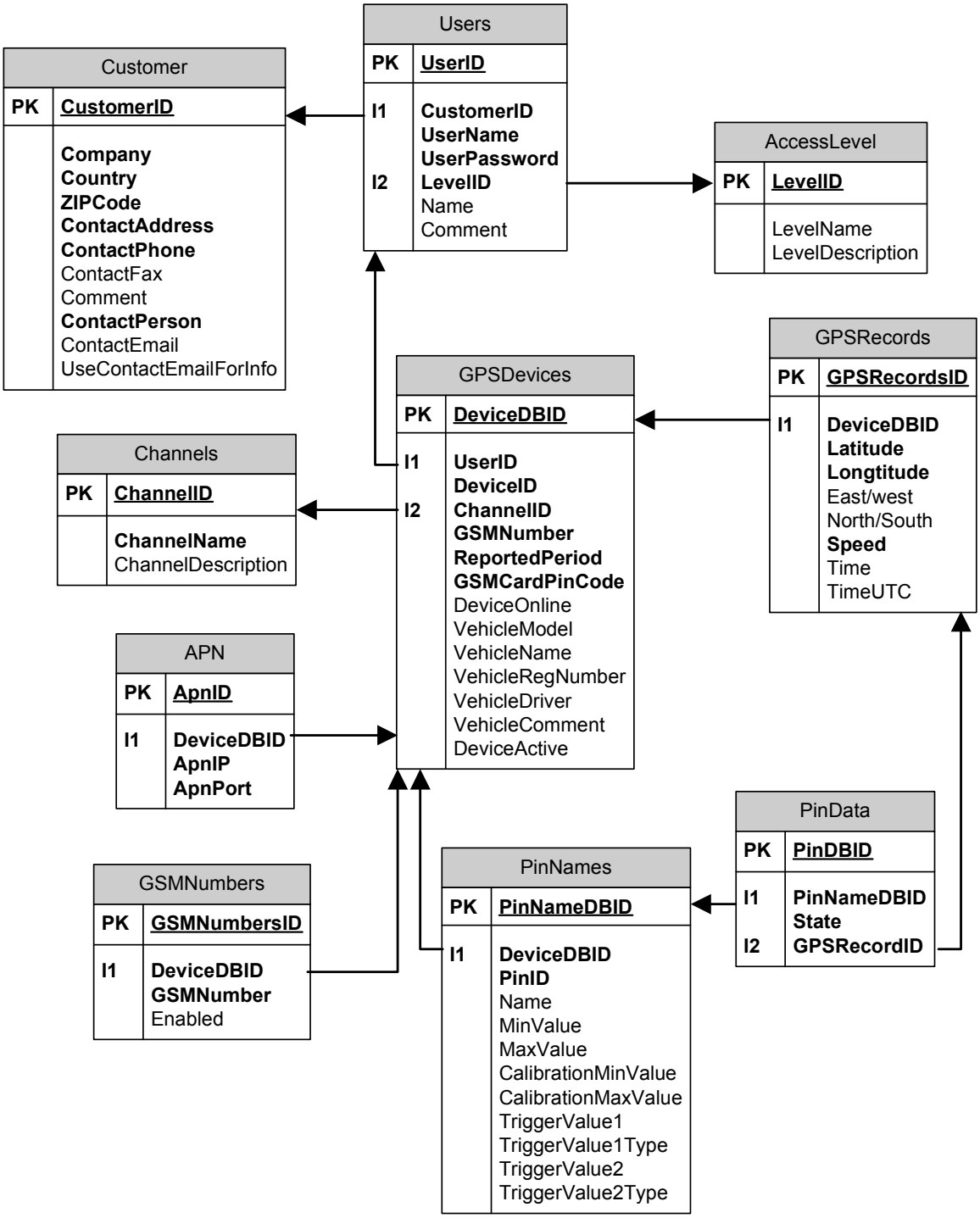
- Регистрайте ASP .NET, като услуга достъпна за всички приложения в IIS. За целта направете следното:

  - ✓ стартирайте командния ред на пакета Visual Studio .NET
  - ✓ напишете следното – *aspnet_regiis /i*
  - ✓ изчакайте приключването на инсталацията

- рестартирайте IIS сървъра, като стартирате SCM конзолата, изберете с десен бутон IIS от списъка в ляво и след това *All Tasks->Restart IIS*

Сега машината трябва да е напълно способна да изпълнява приложението web услуга.

База данни



Фиг. 13 Схема на базата данни

Custommers

Колони				
име на колона	**тип данни**	**not null**	**auto inc.**	**описание**
CustommerID	INTEGER	✓	✓	идентификационен номер в базата данни на клиента
Company	VARCHAR(200)			име на фирмата
Country	VARCHAR(100)			име на държавата
State	VARCHAR(100)			щат/регион
ZIP Code	VARCHAR(100)			пощенски код
ContactAddress	VARCHAR(200)			адрес за контакти
ContactPhone	VARCHAR(100)			телефон за контакти
ContactFax	VARCHAR(100)			факс за контакти
ContactPerson	VARCHAR(200)			човек за контакти
ContactEmail	VARCHAR(100)			имейл за контакти
UseContactEmailForInfo	TINYINT(1)			използвай имейла за контакти за изпращане на допълниелна информация. 0 – да, 1- не
Comment	TEXT			коментар за клиента

Ключове	
главен	**чужди**
CustommerID	няма

Описание

Съдържа цялата необходима информация за даден клиент, която после може да се използва за неговото идентифициране и за осъществяване на контакти.

Access Level

Колони				
име на колона	**тип данни**	**not null**	**auto inc.**	**описание**
LevelID	INTEGER	✓	✓	идентификационен номер в базата данни на нивото на достъп
LevelName	VARCHAR(50)			име на нивото на достъп
Level Description	TEXT			описание на нивото на достъп

Ключове	
главен	**чужди**
LevelID	**Users**_UserID

Описание

Дефинира нивата на достъп на потребителите. Нивата на достъп са необходими, тъй като даден клиент може да регистрира много потребители (например имаме фирма клиент, която дава възможност на различни служители да имат достъп до системата, като ги регистрира като отделни потребители), но да даде различно ниво на достъп за всеки един. Различните нива на достъп предлагат различни възможности на съответния потребител, като например потребител от едно ниво може да преконфигурира утройства, докато потребител от по-ниско ниво може само да ги вижда на карта. Всяко ниво има свой идентификатор, име и описание, което дава възможност за дефиниране на най-различни нива според изисванията на клиента.

Users

Колони				
име на колона	**тип данни**	**not null**	**auto inc.**	**описание**
UserID	INTEGER	✓	✓	идентификационен номер в базата данни на потребителя
CustommerID	INTEGER			идентификационен номер в базата данни на клиента, към който принадлежи потребителя
UserName	VARCHAR(50)			име на потребителя
UserPassword	TEXT			парола на потребителя
LevelID	INTEGER			идентификационен номер в базата данни на нивото на достъп за този потребител
Name	VARCHAR(200)			истинското име на потребителя
Comment	VARCHAR(100)			коментар за потребителя

Ключове	
главен	**чужди**
UserID	**AccessLevel**_LevelID
	Customers_CustommerID

Описание

Описва данните за потребител спадащ към конкретен клиент. Потребителското име и парола за този потребител, които той трябва да дава всеки път, когато иска достъп до системата се съхраняват тук. Също така тук се задава и нивото на достъп на този потребител.

GPS Devices

Колони				
име на колона	**тип данни**	**not null**	**auto inc.**	**описание**
DeviceDBID	INTEGER	✓	✓	идентификационен номер в базата данни на устройството
UserID	INTEGER	✓		идентификационен номер в базата данни на клиента на потребителя, на който принадлежи устройството
DeviceID	VARCHAR(8)	✓		уникален номер на устройството
ChnannelID	INTEGER			идентификационен номер в базата данни на канал, по който устройството рапортува
GSM Number	VARCHAR(50)	✓		GSM номер на устройството
ReportPeriod	INTEGER	✓		интервал, на който устройството в момента рапортува позицията си
GSMCardPinCode	VARCHAR(8)	✓		PIN код на SIM картата на устройството
DeviceOnLine	TINYINT(1)			показва дали устройството в момента е свързано към системата 0 – да, 1- не
DeviceActive	TINYINT(1)			показва дали е резрешено устройството да се свързва към сървъра; 0 – да, 1- не
VehicleModel	VARCHAR(100)			модел на автомобила, в който е инсталирано устройството
VehicleName	VARCHAR(100)			марка на автомобила
VehicleRegNumber	VARCHAR(100)			регистрационен номер на автомобила
VehicleDriver	VARCHAR(100)			шофьор на автомобила
VehicleCommet	TEXT			коментар за автомобила

Ключове	
главен	**чужди**
DeviceDBID	**Users**_UserID
	Channels_ChannelID

Описание

Съдържа цялата информация описваща дадено навигационно устройство, както и някой данни за неговата конфигурация.

GPS Records

Колони				
име на колона	**тип данни**	**not null**	**auto inc.**	**описание**
GPSRecordID	INTEGER	✓	✓	идентификационен номер в базата данни на поредния запис
DeviceDBID	INTEGER	✓		идентификационен номер в базата данни на устройството
Latitude	FLOAT	✓		географска ширина
Longitude	FLOAT	✓		географска дължина
East/West	TINYINT(1)			посока на движение 0 – East, 1- West
North/South	TINYINT(1)			посока на движение 0 – North, 1- South
Speed	INTEGER	✓		скорост във възли
Time	DATETIME			текущо време при вмъкването на записа от локалната машина
TimeUTC	DATETIME			време на изпращане на съобщението според GPS приемника

Ключове	
главен	**чужди**
GPSRecordID	**GPSDevices**_DeviceDBID

Описание

Тук се записват данните извлечени от NMEA съобщенията, които навигационните устройства изпращат на сървъра, за да рапортуват географската си позиция. По-късно на базата на тези данни позицията на устройство се визуализира на цифрова карта.

Pin Names

Колони				
име на колона	**тип данни**	**not null**	**auto inc.**	**описание**
PinNameDBID	INTEGER	✓	✓	идентификационен номер в базата данни на името на входа на устройството
DeviceDBID	INTEGER	✓		идентификационен номер в базата данни на устройството
PinID	INTEGER	✓		идентификационен номер на входа
Name	VARCHAR(100)			име на входа
MinValue	INTEGER			минимална стойност на входния сигнал за този вход
MaxValue	INTEGER			максимална стойност на входния сигнал за този вход
CalibrationMinValue	FLOAT			минимална калибрираща стойност за входния сигнал
CalibrationMaxValue	FLOAT			максимална калибрираща стойност за входния сигнал
TriggerValue1	INTEGER			1-ва стойност на сработване на входа
TriggerValue1Type	INTEGER			тип на 1-та стойност на сработване
TriggerValue2	INTEGER			2-ра стойност на сработване на входа
TriggerValue2Type	INTEGER			тип на 2-та стойност на сработване

Ключове	
главен	**чужди**
PinNameDBID	**GPSDevices**_DeviceDBID

Описание

Дадено навигационно устройство притежава някаква бройка аналогови входове, които могат да бъдат монтирани да измерват различни аналогови величини в автомобила, като например отворена/затворена врата, ниво на резервоара, налягане на маслото и т.н. Тъй като за всеки автомобил тези величини могат да бъдат различни, в тази таблица се описва какво точно измерва даден вход на устройството, за да може потребителят да знае информация за какъв сигнал вижда.

Pin Data

Колони				
име на колона	**тип данни**	**not null**	**auto inc.**	**описание**
PinDBID	INTEGER	✓	✓	идентификационен номер в базата данни на входа
PinNameDBID	INTEGER	✓		идентификационен номер в базата данни на името на входа
State	INTEGER	✓		състояние на входа
GPSRecordID	INTEGER	✓		идентификационен номер в базата данни на съобщението, с което е пристигнала информация за този вход

Ключове	
главен	**чужди**
PinDBID	**GPSRecords**_GPSRecordID
	PinNames_PinNameDBID

Описание

Тук се записват данните за състоянието на даден вход на навигационното устройство.

GSM Numbers

Колони				
име на колона	**тип данни**	**not null**	**auto inc.**	**описание**
GSMNumbersID	INTEGER	✓	✓	идентификационен номер в базата данни на GSM номера
DeviceBID	INTEGER	✓		идентификационен номер в базата данни на името на устройството
GSMNumber	VARCHAR(100)			GSM номер
Enabled	INTEGER			GSM номера е разрешен за рапорт 0 – да, 1 - не

Ключове	
главен	**чужди**
GSMNumbersID	**GPSDevices**_DeviceDBID

Описание

Всяко едно навигационно устройство има списък от няколко GSM номера, на които може да изпраща рапорт за позицията си или други данни по SMS. Списъка се въвежда с цел да се предотврати рапортуването на нелегитимни GSM номера. Тази таблица описва GSM номерата за всяко навигационно устройство, които са разрешени за рапорт по SMS

Channels

Колони				
име на колона	**тип данни**	**not null**	**auto inc.**	**описание**
ChannelID	INTEGER	✓	✓	идентификационен номер в базата данни на канала за данни
ChannelName	VARCHAR(100)			име на канала за данни
ChannelDescription	TEXT			описание на канала за данни

Ключове	
главен	**чужди**
ChannelID	няма

Описание

Таблицата описва различните канали, по които устройството може да изпраща данни и да бъде достигнато отвън. В GSM мрежата възможните канали са SMS, GPRS и CSD.

RouteData

Колони				
име на колона	**тип данни**	**not null**	**auto inc.**	**описание**
RouteDBID	INTEGER	✓	✓	идентификационен номер в базата данни на записаният маршрут
UserID	INTEGER	✓		идентификационен номер на потребителят, към който принадлежи дадения маршрут
RouteName	VARCHAR(100)			името, което потребителят е дал на този машрут
RouteData	TEXT			координати и вид на региона (елипса, линия, полигон) описващи дадения маршрут

Ключове	
главен	**чужди**
RouteDBID	UserID

Описание

Данните, които описват всеки машрут който потребителят зададе чрез аплета и желае да се проследи дали неговото превозно средство го спазва, се записват тук. Всеки потребител може да дефинира много маршрути, които представляват определен регион от картата, задаван чрез различни геометрични фигури. Тези фигури се описват от набор от точки дефинирани чрез двойка координати x,y.

Използвана литература

1. **Ahmed El-Rabbany**
   Introduction to GPS

   Artech House, **2002**
   ISBN 1-58053-183-0

2. **Л. Добош, Я. Духа, С. Мархевски, В. Визер**
   Мобилни Радиомрежи

   Джиев Трейд, 2005
   ISBN 954-9332-14-4

3. **H.M. Deitel, P.J. Deitel, B. Du Walt, L.K. Trees**
   Web Services: A Technical Introduction

   Prentice Hall, **2003**
   ISBN – 0130461350

4. **Regis J. "Bud" Bates**
   GPRS: General Packet Radio Service

   McGraw Hill, **2003**
   ISBN – 0071381880

5. **Johan Hjelm**
   Creating Location Services for the Wireless Web

   Wiley, 2002
   ISBN – 0471402613

6. **Zhong-Ren Peng, Ming-Hsiang Tsou**
internet GIS

Wiley, 2003
ISBN - 0471359238

7. **Майкъл Х. Ернандес**
Проектиране на Бази от Данни

СофтПрес, 2004
ISBN – 9546853011

8. **Том Арчър**
C# Поглед отвътре

СофтПрес, 2001
ISBN – 9546851620

9. **Джефри Рихтер**
Microsoft .NET Framework

СофтПрес, 2002
ISBN – 954685221X

Използвани съкращения

1. **CDMA** - *Code Division Multiple Access*
2. **CSD** – *Circuit Switched Data*
3. **GPRS** – *General Packet Radio Service*
4. **GSM** - *Global System for Mobile Communications*
5. **SMS** – *Short Message Service*
6. **SOAP** - *Simple Object Access Protocol*
7. **TCP/IP** – *Transmision Control Protocol / Internet Protocol*
8. **TDMA** - *Time Division Multiple Access*
9. **UDP -** *Universal Datagram Protocol*
10. **UMTS** – *Universal Mobile Telecomunications System*
11. **WCDMA** - *Wideband CDMA*
12. **WSDL** – *Web Service Description Language*
13. **КУПС** – *контрол и управление на превозни средства*
14. **МУ** – *мениджър на устройства*