СОФИЙСКИ УНИВЕРСИТЕТ "Св. Климент Охридски"
ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА
Катедра Информационни технологии

Д И П Л О М Н А Р А Б О Т А

на Владислав Светославов Илиев
специалност: софтуерни технологии
фак. №: М-21557

Тема: Приложение осъществяващо услугата изпращане / приемане на кратки съобщения (SMS) от персонален компютър свързан с GSM устройство

Дипломант:.....
/Владислав Илиев/

Ръководител:.....
/доц. Боян Бончев/

гр. София, 2007 г.

Съдържание

1. Увод

Вероятно всеки, който ползва GSM телефон, поне веднъж е получавал или изпращал кратко текстово съобщение. За разлика от гласовото повикване, то не е така натрапчиво – може да бъде прочетено и отговорено по-късно, в по-удобен момент. Съобщението бива прочитано от дисплея на телефона, а въвеждано чрез не толкова удобна за целта, но вършеща работа клавиатура. Това, на което ще обърнем внимание, е че и прочитането и писането (и изпращането) се извършват от *човек*. В настоящата дипломна работа е разработен софтуер за получаването и изпращането на съобщения *програмно*, от *компютър.* За целта е необходим единствено GSM модем или телефон свързан чрез сериен интерфейс към компютъра. Повечето съвременни телефони и всички модеми имат такава възможност. Така компютър свързан с GSM апарат и изпълняващ предложения софтуер *може* да се превърне в център за автоматизирана обработка на SMS съобщения. Разработеното приложение реализиращо услуга поема всички детайли на директната работа с модема и предоставя прост програмен интерфейс (три функции и една структура от данни) към клиентските приложения. Очаква се именно разработчиците на приложения, нуждаещи се от възможността за изпращане и получаване на кратки съобщения, да са потребители на предложената работа. Разработваните от тях приложения ще съдържат логиката, но ще бъдат напълно изолирани от детайлите на изпращането и получаването. В такъв смисъл предлаганата услуга може да се разглежда като транспортен слой. Допълнително предимство от използването на механизма отдалечено извикване на процедура (RPC), е че няколко клиентски приложения (дори работещи на различни компютри) могат да ползват една услуга (и съответно един модем).

1.1. Примери за клиентски приложения

- *отдалечен мониторинг и контрол, телеметрия;*

  Потребителска програма следи електрически параметри / сигнали / температури / налягания / др. При определени събития (напр. авария) или по зададено разписание изпраща кратък текст, съдържащ необходимите данни. Въз основа на прието съобщение включва / изключва машини (напр. помпи) – полезно при отдалечен и трудно достъпен обект.
- *охранителна дейност;*

  При нерегламентиран достъп потребителска програма изпраща алармено съобщение на определени номера.

- модерното *„гласуване чрез кратки съобщения“*

  Потребителска програма получава постъпващо кратко съобщение, отброява подадения глас и връща кратко съобщение, за да благодари на подателя за участието.

1.2. Цели и задачи на дипломната работа

Целта на дипломната работа е да се разработи софтуерно приложение (windows service), осъществяващо услугата изпращане / приемане на кратки съобщения (SMS).
Услугата трябва да предостави приложен програмен интерфейс (API) за логическото изпращане и получаване на съобщения, т.е. да задоволи потребността от проста SMS функционалност, капсулирайки детайлите на директната работа с GSM модем.
За постигането на тази цел е необходимо изпълнението на следните задачи:

- имплементиране на библиотека от класове за комуникация с модем и обработка на кратки съобщения в тяхното шестнадесетично представяне;
- изграждане на приложението като услуга – за да може да се зарежда заедно с операционната система, без намеса на потребител;
- дефиниране и предоставяне на интерфейс към SMS функционалността под формата на процедури за отдалечено извикване (RPC);
- изграждане на приложение за конфигуриране на услугата (Control Panel аплет), чрез което потребителят да задава параметрите на комуникацията с GSM модема, и параметрите на обвързването към процедурите за отдалечено извикване;
- изграждане на демонстрационно клиентско приложение, ползващо услугата. Предназначено е за онагледяване на възможностите и тестване на услугата. Може да служи като пример за разработчика, изграждащ друго клиентско приложение, базирано в някаква степен на разглежданата услуга.

Както бе споменато вече, потенциалният потребител на софтуерното приложение (услуга) от настоящата дипломна работа е разработчик на (клиентско) приложение, което да ползва предоставяната SMS функционалност и въпреки това да остава отделено (дори като двоичен код) от конкретиката в имплементацията й.

1.3. Обзор на съдържанието

В част 2. е направено сравнение между софтуерното решение от дипломната работа и съществуващи алтернативи с подобна функционалност. Целта й е да открои основните разлики, определящи подходящия избор в зависимост от конкретните изисквания.
В част 3. са дадени някои основни сведения за необходимия хардуер – GSM устройство.

Описани са основни разлики между модем и телефонен апарат и е даден пример за инсталирането, захранването и свързването на GSM модем към персонален компютър.

В част 4. са описани кратките (SMS) съобщения. Включва кратко неформално описание и интересни факти, следвано от техническа спецификация на PDU (Protocol Data Unit) формата – видове съобщения (SMS-SUBMIT и SMS-DELIVER), съставящи ги битови и октетни полета, кодиране на символите и конкатенация. Описана е програмната представа на SMS съобщенията.

В част 5. е дадена предварителна представа за използваните технологии, както и мотивация за употребата им.

В част 6. е разгледана имплементацията на класовете осигуряващи необходимите буфериран вход и изход за комуникация по сериен интерфейс с GSM устройство. Това са SerialStream – работещ със серийния порт, и вътрешният SerialStream::CyclicBuffer – служещ за буфериране на входа от модема.

В част 7. са въведени т.нар. AT команди – командният интерфейс на модемите. Показана е примерната им (ръчна) употреба чрез инструмента HyperTerminal и програмното им използване за инициализация на модема, за изпращане на съобщения и за прочитане на получени съобщения.

В част 8. е описана техниката отдалечено извикване на процедура (RPC). Разгледани са средствата за дефиниране на интерфейс, компилирането на клиентските и сървърното приложения, използващи RPC библиотеката и начина за предоставяне на сървърната функционалност за отдалечено извикване.

В част 9. се разглеждат приложенията реализиращи услуги (Windows services) – както от програмна гледна точка – реализация на услуги и контрол панел аплети за конфигурирането им, така и от администраторска гледна точка – инсталирането им и инструменти на операционната система за боравене с тях (sc.exe).

Част 10. е посветена на демонстрационно клиентско приложение извикващо отдалечените процедури, предоставяни от сървъра. Дадено е ръководство за потребителя. Разяснена е реализацията, която може да служи като работещ пример при разработката на други клиентски приложения използващи същия RPC интерфейс. Описани са основни моменти при тестването на услугата.

2. Предлагани от операторите услуги/приложения

Към момента два от трите мобилни оператора в България предоставят програмни приложения/услуги за изпращане на кратки съобщения (за третия авторът не разполага с информация). Общо и за услугата на Глобул (http://www.globul.bg) и за приложението на МТел (http://www.mtel.bg), е че изискват свързаност на клиентския компютър с Интернет. И при двете акцентът пада върху изпращането на кратки съобщения за сметка на получаването на такива. Така използвайки Интернет като преносна среда за съобщенията, които иначе би трябвало да се излъчат от GSM апарат към Центъра на услугата, за тези съобщения се намалява наполовина натоварването върху мобилната мрежа, както предполагаемо и себестойността им. Това бе ключовото им сходство, разликите в подхода им ще бъдат щрихирани по-долу.

2.1. bSMS на Глобул

Услугата на Глобул е TCP базирана. За изпращането на кратко съобщение се подава заявка по протокол HTTP/HTTPS (метод GET или POST) към сървър на оператора. Пример за използване на приложния програмен интерфейс за изпращане на кратко съобщение (SMS API) е следния:
http://192.168.0.1:55077/sms_api?uid=xxxx&pass=xxxx&from=9999&to=359898123456&msg=test&dlr=1
Сървърното приложение е зададено чрез комбинацията IP адрес, порт.
Потребителят се идентифицира с име (uid) и парола (pass).
Задават се и адресат (to) и съдържание на съобщението (msg).
Характерна особеност на услугата е, че съобщенията биват изпращани от кратък номер, който бива указан чрез параметъра from. Така при дадения пример, абонатът 359898123456 ще получи съобщение "test" от изпращач 9999.
Допълнителни възможности са искането за потвърждаване на доставянето на съобщението (dlr), задаване на валидност на съобщението и изпращането на двоични (binary) данни, вместо текст. Като цяло може да се констатира, че предложеното е услуга, на чиято база да се надгради приложение за краен потребител.

2.2. eSMS на МТел

Приложението eSMS е десктоп базирано, предназначено за краен потребител. То предоставя графичен потребителски интерфейс за изпращане на кратки съобщения – за

въвеждане на номер на получателя и текст на съобщението. Предлага споделяне на информацията за контактите и интеграция с MS Outlook и Outlook Express. За разлика от bSMS тук съобщенията имат за номер на изпращач абонаментния номер на потребителя (въведен и потвърден чрез PIN по време на инсталацията). За получателя е неразличимо дали краткото съобщение е написано и изпратено от GSM апарат или от eSMS приложението. За изпратеното съобщение съответно се таксува абоната изпращач. Накратко приложението замества неудобното въвеждане на текста на съобщението от GSM телефон с въвеждане от компютърна клавиатура и вместо съобщението да се излъчва от телефона към Центъра на услугата то се предава по Интернет.

2.3. Сравнение с предлаганата в дипломната работа услуга

За разлика от предлаганата в дипломната работа услуга, която се нуждае от допълнителен хардуерен модул – именно GSM модем, системите bSMS и eSMS не използват такова устройство. От своя страна те са зависими от наличието на Интернет връзка, докато за реализирането на системата, описвана в дипломната работа, такова изискване няма. Поради ползването на Интернет, а не на мобилната мрежа за пренос на краткото съобщение до Центъра на услугата, системите на операторите намаляват приблизително наполовина времето от момента на изпращане на съобщението до момента на получаването му. Въпреки всичко, времето за пренос на изпратеното съобщение от Центъра на услугата до получателя остава непроменено. Друга съществена разлика между предлаганите от операторите системи и настоящата е в начина им на получаване на кратки съобщения. При eSMS "отговорът се доставя директно до мобилния телефон", т.е. въобще не постъпва за компютърна обработка. При текущата реализация на bSMS се налага репликиране на MySQL база данни и някаква форма на активно чакане – проверка за нов запис през определен период от време.

Системата от дипломната работа е независима от оператора – винаги може да се замени SIM картата на един опреатор с тази на друг и това е напълно безотносително към работата на софтуера

Тук може би е мястото и за кратък сравнителен анализ между разработваната система и друга базирана изцяло на Интернет. Два са ключовите момента определящи предпочитанията:

- има ли Интернет достъп при компютъра приемащ и изпращащ *информация.*
- има ли (удобен) Интернет достъп при потребителите на тази информация.

Ако отговорите и на двата въпроса са положителни, то предлаганата в дипломната работа

SMS система може да бъде заменена от превъзхождащата я както по скорост, така и по обем на информационния поток система, ползуваща Интернет като преносна среда.
Но ако отговорът поне на един от въпросите е отрицателен може да се помисли за изграждане на система базирана на предлаганата софтуерна услуга. Например за отдалечен обект, до който прекарването/ползването на Интернет би било неприемливо скъпо, а има покритие от мобилен оператор. Или пък ако потребителите ползват стандартни GSM телефони, а не устройства, които са свързани с Интернет (такива вече има). В последния случай обаче ще се появи необходимостта на всяко от тези устройства да има подходящ софтуер за комуникация със сървъра.

За по-добрата представа на общата картина ще направим следната класификация на споменатите по-горе системи/приложения (вкл. от дипломната работа) въз основа на комуникационната връзка и по-точно на двата й края.

0. Нито при сървъра, нито при клиентите се ползва GSM устройство. Такива са системите базирани изцяло на Интернет **(Интернет-Интернет).**
1. В единия край се ползва GSM устройство (при получателите), а в другия не. Такива са и двете гореописани bSMS и eSMS. В другия край при тях се ползва Интернет **(Интернет-GSM)**.
2. И в двата края на връзката се ползват GSM устройства. Именно за изграждане на такива системи е предназначена разработваната в дипломната работа услуга **(GSM-GSM)**

3. GSM модеми

3.1. Модеми, модули, телефони, цени

Сравнително широко разпространени GSM модеми на българския пазар са

- GM29 на Sony-Ericsson и
- MC35iT на Siemens

Приблизителна цена на изброените GSM модеми към момента е 150 €
От посочените по-горе компании в специализиарани електронни магазини се продават и голямо разнообразие от GSM модули, представляващи платки за осигуряване на GSM възможностите, но предназначени за понататъшна доработка – напр. осигуряване на захранване, комуникационен интерфейс, поставянето им в кутия с подходяща степен на защита и т.н. Това позволява по-оптималното ползуване на възможностите и доближаване

до изискванията на клиентите (възможности / цена). Цената е около 100 €.

Друга алтернатива е използването на конвенционален GSM телефонен апарат с вграден интерфейс за връзка с компютър - най-често USB, инфрачервен или Bluetooth порт. Цената варира в зависимост от допълнителните възможности 100 € - 300 €.

3.2. GSM модем или GSM телефон

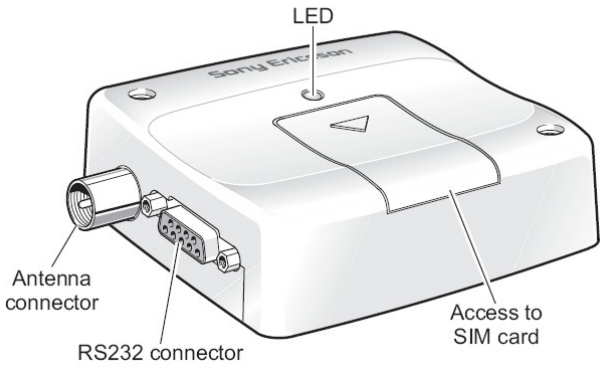
Предимствата на използването на GSM модем пред използването на телефон са няколко:

- GSM модемът има по-здрава конструкция (напр. няма дисплей), тъй като е предназначен за индустриална употребата, а не от потребител.
- използва външна антена, която увеличава чувствителността и позволява употребата му на места с по-слабо ниво на сигнала.
- при приемането и изпращането на множество съобщения мобилният телефон може да прегрее, докато модемът е проектиран именно за такъв режим на работа
- никой производител не препоръчва непрекъснатото зареждане на телефона, докато модемът е със захранване през цялото време.

От софтуерна гледна точка обаче разлика няма стига да предоставят стандартните AT команди. Ето защо модем, телефон, GSM апарат, GSM устройство ще се ползват в изложението взаимозаменяемо.

3.3. GM29 – инсталиране

Ние ще се спрем на един от готовите за употреба (от краен клиент) и цитирани по-горе GSM модеми - GM29 на Sony-Ericsson. Следва краткото му описание, което може да се намери в [1].



Фиг. 1: GM29 - цялостен изглед

3.3.1. Поставяне на антена

Фигура 1 показва цялостния изглед на модема и част от конекторите му.

Antenna connector – конектор, към който се свързва антена. Към него се свързва външна антена осигуряваща по-висока чувствителност към радио вълните.

RS232 connector – Sub-D конектор (9 pin) за свързване чрез сериен интерфейс към компютър.

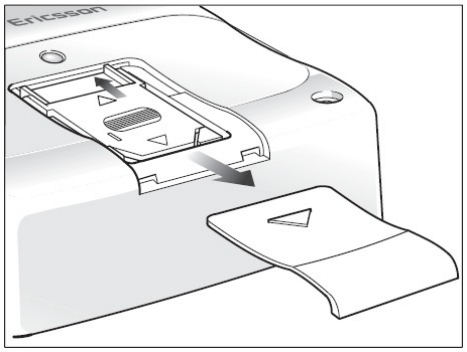
LED – светодиод указващ състоянието на модема. Периодичното му примигване означава

готовност на модема за работа, а бързото примигване е индикация за изпращано / приемано в момента съобщение. Непрекъснатото светене означава, че модемът е включен, но още не е част от мобилната мрежа на оператор.

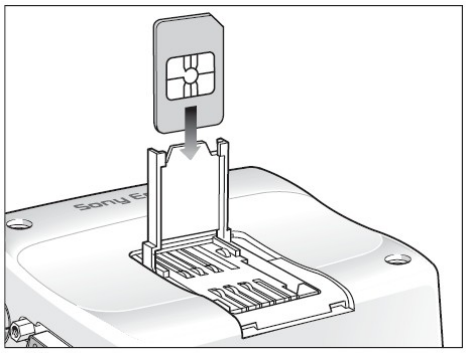
Access to SIM card – капаче даващо достъп до мястото за поставяне на SIM картата.

3.3.2. Поставяне на SIM карта

Поставянето на SIM картата става след издърпване на капачето – както е показано на следващите две фигури



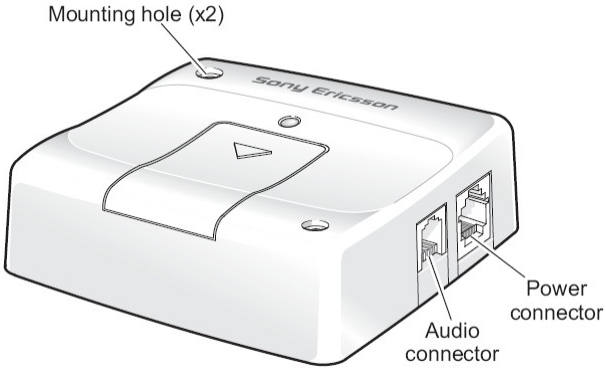
Фиг. 2: GM29 - отваряне на капачето



Фиг. 3: GM29 - поставяне на SIM карта

3.3.3. Захранване

На фигура 4 са показани конектора за звук (Audio connector) и конектора за електрическо захранване (Power connector) на модема. Тъй като ще се ползуват само SMS възможностите на модема, няма да се спираме на употребата на конектора за звук. Конекторът за захранване е тип RJ11, със 6 контактуващи пластини.



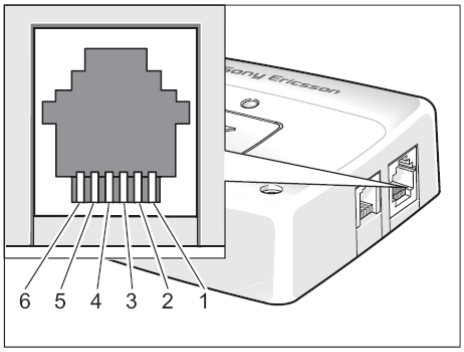
Фиг. 4: GM29 - конектори

Захранващото напрежение се подава между VCC (1) – положителен полюс и GND (2) – отрицателен полюс (фиг. 5).

Захранващото напрежение е постоянно и е в следните рамки 5 V DC – 32 V DC.

Сигналът HR_IN служи за изключване и нулиране на модема, а TO_IN за включване на модема.

Включването и изключването на модема може да се извършват и софтуерно.



1 VCC 3 HR_IN 5 n/c
2 n/c 4 TO_IN 6 GND

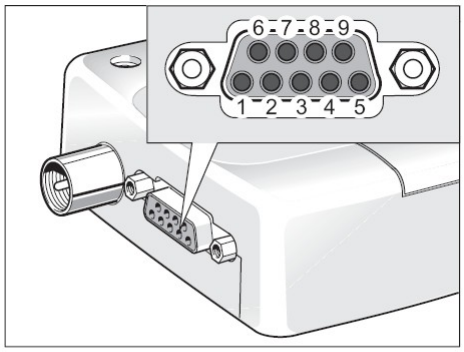
Фиг. 5: GM29 - конектор за захранване

3.3.4. Свързване към компютър – RS232

Връзката между компютъра и модема се осъществява чрез сериен интерфейс RS232. В терминологията на серийната комуникация модема трябва да се разглежда като *устройство за предаване на данни* (data circuit-terminating equipment (DCE)), а свързания компютър като *крайно устройство* (data terminating equipment (DTE))

Табл. 1: RS232 - Сигнали

pin		I/O	описание
1	DCD	O	Открита носеща честота (DCE)
2	RD	O	Приемани данни (DCE)
3	TD	I	Предавани данни (DTE)
4	DTR	I	Готовност за работа на DTE
5	GND	-	Земя
6	DSR	O	Готовност за работа на DCE
7	RTS	I	Заявка за предаване на данни (DTE)
8	CTS	O	Готовност за приемане на данни (DCE)
9	RI	O	Индикатор за позвъняване (DCE)



1 DCD 4 DTR 7 RTS
2 RD 5 GND 8 CTS
3 TD 6 DSR 9 RI

Фиг. 6: GM29 - сериен порт

4. Кратки съобщения – SMS

4.1. Кратко описание

Услугата за кратки съобщения (Short Message Service – SMS) позволява изпращането и получаването на съобщения, чиито информационен обем е ограничен до 140 октета. Тези 140 октета могат да представят 160 символа от подразбираната GSM азбука, 140 байта или 70 Unicode символа. Предвиден е и механизъм за конкатениране на няколко съобщения в едно по-дълго.

Първоначално предполаганата основна употребата на кратките съобщения е била за известяване на потребителя за определени събития напр. постъпване на гласова поща, за телеметрия и сходни, но в последствие силно се развива изпращането на текстови съобщения между самите абонати. За 2006 година приходите от кратки съобщения се оценяват на 80 милиарда долара при средна цена 11 цента на съобщение. Печалбата е почти 90% от цената [2]. Важно звено в реализирането на услугата е т.нар . Център на услугата за кратки съобщения (Short Message Service Center – SMSC). В зависимост от своята насоченост кратките съобщения се разделят на "мобилно произхождащи" (mobile originated) и "мобилно завършващи" (mobile terminated). Мобилно ориентирани са

съобщенията, чиято посока е от мобилния телефон към Центъра на услугата, а мобилно терминирани са тези, чиято посока е от Центъра на услугата към мобилния телефон. Центърът на услугата осигурява механизъм "запази и препрати" за съобщенията и в действителност при изпращането на съобщение от един абонат на друг се извършва следната последователност. От мобилният телефон се изпраща мобилно произхождащо съобщение (към Центъра на услугата). Центърът на услугата запазва съобщението и опитва изпращането му към адресирания мобилен телефон (мобилно терминирано). В случаите когато последния не е достъпен Центърът на услугата може да направи още няколко опита за доставяне на съхраненото съобщение. Именно на този механизъм се дължи и едно съществено предимство на кратките съобщения – те биват запазвани когато получателя не е достъпен и доставяни веднага когато стане отново достъпен.

Мобилно ориентираните и мобилно терминираните са групирани като съобщения "точка-до-точка" в спецификацията GSM 03.40. Освен тях има и трети вид – разпращане до всички в обхвата на клетката (Cell Broadcast) – описан е в спецификация GSM 03.41 и няма да се спираме на него.

4.2. PDU формат

PDU е съкращение на Protocol Data Unit (протоколна единица данни). Тези единици данни са кодирани и структурирани спрямо протокола за предаване на цифрова информация по т.нар. "въздушен интерфейс" (air interface). Протоколът е описан в техническата спецификация [3]. Въздушният интерфейс в случая е радио базирана връзка между мобилния телефон и Центъра на услугата и представя нива 1 и 2 от седемслойния OSI модел.

Когато се използва мобилен телефон, обикновено се въвежда текстово съобщение чрез клавиатурата на телефона, въвежда се телефонения номер, на който да се изпрати съобщението, натиска се "YES" бутона и обикновено се получава уведомлението “MESSAGE SENT” ("съобщението е изпратено"). Когато обаче се използва GSM модул тази процедура трябва да се извърши чрез AT команди от някакъв вид терминал, тъй като клавиатура не е налична. В някои случаи модулът дори не поддържа текстов режим за SMS съобщения и те трябва да се кодират направо във PDU вид.

От друга страна използването на PDU режима дава на потребителя много по-добър контрол върху изпращаната информация. Например може да се иска изпращането не на текст, а на сурови (двоични) данни и това се постига чрез PDU формата.

Има няколко вида PDU единици. Това са:

SMS-DELIVER, SMS-DELIVER-REPORT, SMS-SUBMIT, SMS-SUBMIT-REPORT, SMS-STATUS-REPORT, SMS-COMMAND.

От тях най-важни и с най-широка употреба са SMS-SUBMIT и SMS-DELIVER и са разгледани в настоящата дипломна работа.

SMS-SUBMIT се отнася към мобилно произхождащите съобщения и се изпраща от мобилен терминал към Центъра на услугата. Употребява се при изпращане на SMS съобщение.

SMS-DELIVER се отнася към мобилно завършващите съобщения и се получава чрез мобилен терминал от Центъра на услугата. Употребява се при получаване на SMS съобщение.

Общ вид на PDU пакета:

PDU = SCA + TPDU,

където SCA е Service Centre Address (номер на Центъра на услугата),

TPDU е Transport Protocol Data Unit (единица данни по транспортния протокол)

Например към момента SCA номерата за мобилните оператори са съответно:

Табл. 2: Номера на центровете на услуги (SCA)

MTel	+35988000301
Globul	+35989100000
Vivatel	+359878000333

Едно съобщение използващо центъра на услугата на МТел може да изглежда така:

07915389080003F111000C91538918325476000 0C41554747A0E4ACF416110945805B5CBF379F85CFE

В получер шрифт е изписан SCA, а останалото е TPDU.

PDU формата е шестнадесетично кодиран двоичен формат, което означава че 2 шестнадесетични цифри представят един байт от данните. Когато става въпрос за изпращаните данни байтът обикновено се нарича октет (байтове се съхраняват, октети се изпращат). TPDU се състои от заглавна част съдържаща управляваща информация и "полезен товар" съдържащ потребителските данни. Различните части ще бъдат разгледани подробно по-късно.

При дадения по-горе пример номера на Центъра на услугата (SCA) бе явно зададен в самото съобщение. В следващия пример

0011000C91538918325476000 0C41554747A0E4ACF416110945805B5CBF379F85CFE

SCA е неявно зададен (00).

Това означава, че за номер на Центъра на услугата ще се ползва този, който е записан в SIM картата (подразбиращ се SCA). С помощта на AT команди (AT+CSCA) подразбиращият се SCA може да бъде променен, въпреки че това почти никога не се налага, тъй като SIM картите на даден оператор идват вече със зададен номер на Центъра на услугата на съответния оператор.

4.2.1. Предаване и представяне на данни

Октетите на PDU се предават в последователност – първо се предава най-предния (началния) октет след това втория и т.н до последния. Битовете в един октет се предават от по-младшите към по-старшите.

Например при предаване на следната (част от) последователност от октети

03FFCAFE

за онагледяване може да подредим битовете в таблица

ОКТЕТ	**Двоично представяне**							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
03	0	0	0	0	0	0	1	1
FF	1	1	1	1	1	1	1	1
CA	1	1	0	0	1	0	1	0
FE	1	1	1	1	1	1	1	0

Фиг. 7: Двоично представяне

Имайки предвид че първият предаван бит е най-десния, редът на предаване на битове ще е следния:

11000000111111110101001101111111→

Това бе относно реда на предаване на битовете и байтовете. Следва преглед на представянето на числови и буквеноцифрови данни.

За параметрите в TPDU частта има четири вида представяния: целочислен, октет, полуоктет и буквеноцифен.

Когато битове от определен брой октети, пълни или частични, представят **цяло число** те трябва да бъдат интерпретирани по следния начин:

1) Между октетите: Октетите с най-малък номер (на октет) съдържат най-старшите битове.

2) В един октет: Битовете с най-голям номер (на бит) са най-старши (спрямо битовете от същия октет).

По-долу е даден пример на октетното и битовото представяне и последователност на предаване на целочислено поле.

Нека 2-та най-десни бита на октет номер 5, пълните октети номера 6 и 7 и трите най-леви бита на октет номер 8 представят цяло число както е показано на фиг.8.

	7	6	5	4	3	2	1	0
Октет №								
●	●	●	●	●	●	●	●	●
5							5_{b1}	5_{b0}
6	6_{b7}	6_{b6}	6_{b5}	6_{b4}	6_{b3}	6_{b2}	6_{b1}	6_{b0}
7	7_{b7}	7_{b6}	7_{b5}	7_{b4}	7_{b3}	7_{b2}	7_{b1}	7_{b0}
8	8_{b7}	8_{b6}	8_{b5}					
●	●	●	●	●	●	●	●	●

Фиг. 8: Целочислено представяне

$5_{b1}5_{b0}6_{b7}6_{b6}6_{b5}6_{b4}6_{b3}6_{b2}6_{b1}6_{b0}7_{b7}7_{b6}7_{b5}7_{b4}7_{b3}7_{b2}7_{b1}7_{b0}8_{b7}8_{b6}8_{b5}$

$5_{b0}5_{b1}\underline{5_{b2}}\underline{5_{b3}}\underline{5_{b4}}\underline{5_{b5}}\underline{5_{b6}}\underline{5_{b7}}6_{b0}6_{b1}6_{b2}6_{b3}6_{b4}6_{b5}6_{b6}6_{b7}7_{b0}7_{b1}7_{b2}7_{b3}7_{b4}7_{b5}7_{b6}7_{b7}\underline{8_{b0}}\underline{8_{b1}}\underline{8_{b2}}\underline{8_{b3}}\underline{8_{b4}}8_{b5}8_{b6}8_{b7}$

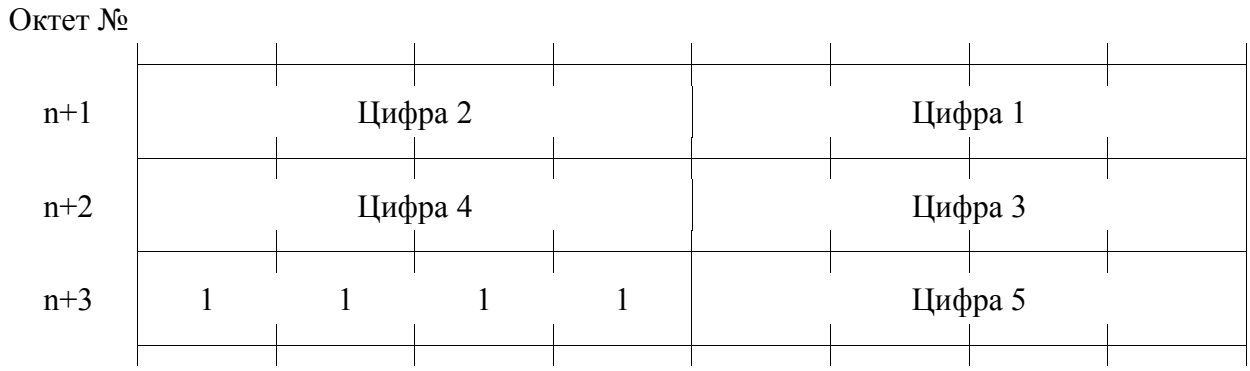
21-те бита от октети 5, 6, 7 и 8 в краткото съобщение дадено в таблицата, представят цяло число, показано на по-горния ред и са предавани в реда, даден в по-долния ред. Подчертаните битове *не* са от представянето на цялото число.

Поле, което е **октетно** представено винаги се състои от определен брой пълни октети. Всеки октет от полето представя една десетична цифра. Октетите с най-малък номер съдържат най-старшите десетични цифри.

Поле, което е **полуоктетно** представено се състои от определен брой пълни октети и възможно още половин октет. Всеки полуоктет от полето представя една десетична цифра. Октетите с най-малкия номер съдържат най-старшите десетични цифри. В един октет, полуоктетът съдържащ битове с номера 0 – 3 представя по-старшата десетична цифра. В един полуоктет, битовете с най-голям номер са най-старши.

В случая когато полуоктетно представено поле съдържа нечетен брой цифри, битовете 4 – 7 на последния октет са "запълващи битове" и трябва да са винаги "1111".

Пример:



Фиг. 9: Полуоктетно представяне

Поле, което използва **буквеноцифено** представяне се състои от определен брой 7-битови знаци представени във вида на подразбираната азбука дефинирана в GSM 03.38.

Всяко адресно поле се състои от следните подполета:

- поле с дължина на адреса в един октет
- поле с тип на адреса в един октет
- поле със стойността на адреса с променлива дължина

Посоченият по-горе номер на центъра на услугата **07915389080003F1** ще бъде използван за пример:

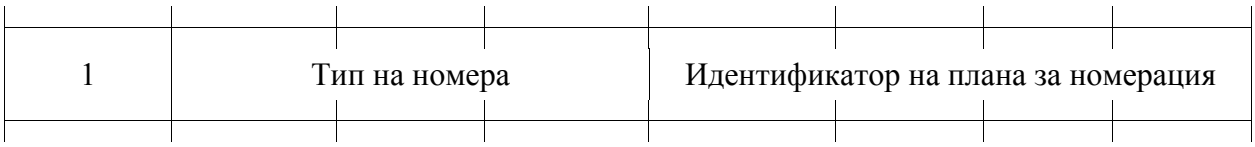
Октет	Двоично представяне								Значение
	7	6	5	4	3	2	1	0	
07	0	0	0	0	0	1	1	1	Дължина на адреса*
91	1	0	0	1	0	0	0	1	Тип на адреса
53	0	1	0	1	0	0	1	1	Цифрите 3 и 5
89	1	0	0	0	1	0	0	1	Цифрите 9 и 8
08	0	0	0	0	1	0	0	0	Цифрите 8 и 0
00	0	0	0	0	0	0	0	0	Цифрите 0 и 0
03	0	0	0	0	0	0	1	1	Цифрите 3 и 0
F1	1	1	1	1	0	0	0	1	Цифрата 1 и запълващи битове

Фиг. 10: Полуоктетно представяне

Полето с дължина на адреса задава броя на следващите октети.

**Забележка:* Когато става въпрос за SCA дължината е броя на следващите октети, а когато става въпрос за адрес на получателя на съобщението е броя на полуоктетите представляващи стойността на адреса [4].

Полето тип на адреса има следния формат



Фиг. 11: Тип на адрес - формат

Тип на номера:

Битове *6 5 4*

0 0 0 неизвестен

0 0 1 международен номер

0 1 0 национален номер

Има и други типове номера, на които няма да се спираме.

Идентификатор на плана за номерация (приложим за горните три типа номера):

Битове *3 2 1 0*

0 0 0 0 неизвестен

0 0 0 1 ISDN/телефонен план за номерация

1 0 0 0 национален план за номерация

Има и други планове за номерация , на които няма да се спираме.

4.2.2. Видове съобщения

Както бе споменато по-горе има няколко вида PDU съобщения: SMS-DELIVER, SMS-DELIVER-REPORT, SMS-SUBMIT, SMS-SUBMIT-REPORT, SMS-STATUS-REPORT, SMS-COMMAND. От тях най-важни за обща употреба са SMS-SUBMIT и SMS-DELIVER.

По-долу за тях ще бъдат разгледани съставящите ги полета – начина на подреждането им. Значението на всяко едно от полетата ще бъде разгледано в следващата подточка.

PDU видът **SMS-DELIVER** пренася кратко съобщение от Центъра на услугата до мобилния телефон.

Брой	Битове							
октети	7	6	5	4	3	2	1	0
1	TP-RP	TP-UDHI	TP-SRI	-	-	TP-MMS	TP-MTI	
2-12	TP-OA							
1	TP-PID							
1	TP-DCS							
7	TP-SCTS							
1	TP-UDL							
0-140	TP-UD							

Фиг. 12: SMS-DELIVER - карта

SMS-SUBMIT пренася кратко съобщение от мобилния телефон до Центъра на услугата.

Брой	Битове							
октети	7	6	5	4	3	2	1	0
1	TP-RP	TP-UDHI	TP-SRR	TP-VPF		TP-RD	TP-MTI	
1	TP-MR							
2-12	TP-DA							
1	TP-PID							
1	TP-DCS							
0,1 или 7	TP-VP							
1	TP-UDL							
0-140	TP-UD							

Фиг. 13: SMS-SUBMIT - карта

4.2.3. Полета

TP-MTI – Message Type Indicator

TP-MTI е 2-битово поле, позиционирано на битове 0 и 1 на първия октет на всички PDU съобщения. То може да заема следните стойности и значения:

Табл. 3: Message Type Indicator - стойности

		Тип на съобщението	
бит 1	бит 0	Посока SC → MS	Посока MS → SC
0	0	SMS-DELIVER	SMS-DELIVER-REPORT
1	0	SMS-STATUS-REPORT	SMS-COMMAND
0	1	SMS-SUBMIT-REPORT	SMS-SUBMIT
1	1	Запазено	

TP-MMS – More Messages to Send

TP-MMS е 1-битово поле, позиционирано на бит 2 на първия октет на SMS-DELIVER съобщения. То може да заема следните стойности:

бит 2: 0 – Още съобщения чакат при Центъра на услугата за този MS

1 – Няма повече съобщения чакащи в Центъра на услугата за този MS.

TP-RD – Reject Duplicates

TP-RD е 1-битово поле, позиционирано на бит 2 на първия октет на SMS-SUBMIT съобщения. Указва дали Центърът на услугата да приема "идентични" SMS съобщения произхождащи от същия мобилен апарат, когато оригиналното съобщение все още се пази в Центъра на услугата. Сравнението се прави само чрез полетата TP-MR и TP-DA.

бит 2: 0 – Инструктира Центъра на услугата да приема повтарящи съобщения

1 – Инструктира Центъра на услугата да отхвърля повтарящи съобщения

TP-VPF – Validity Period Format

TP-VPF е 2-битово поле, позиционирано на битове 3 и 4 на първия октет на SMS-SUBMIT съобщения. То описва формата на параметъра за период на валидност (TP-VP). Периодът на валидност указва колко време Центъра на услугата ще съхранява SMS при недосъпен адресат преди да отхвърли SMS съобщението.

Има четири различни формата на периода на валидност:

Табл. 4: Validity Period Format - стойности

бит 4	бит 3	Значение
0	0	Няма поле за период на валидност
0	1	Относителен формат на периода на валидност
1	0	Разширен формат на периода на валидност
1	1	Абсолютен формат на периода на валидност

Най-малко октети заемащ и може би заради това най-често използван е относителният формат и единствено той ще се ползва в настоящата дипломна работа.

TP-SRI – Status Report Indication

TP-SRI е 1-битово поле, позиционирано на бит 5 на първия октет на SMS-DELIVER съобщения. То може да заема следните стойности:

бит 5: 0 – Уведомление за статуса няма да бъде върнато към изпращащото мобилно ` устройство

1 – Уведомление за статуса ще бъде върнато към изпращащото мобилно устройство

Този бит съответства на TP-SRR в SMS-SUBMIT съобщение.

TP-SRR – Status Report Request

TP-SRR е 1-битово поле, позиционирано на бит 5 на първия октет на SMS-SUBMIT съобщения. То може да заема следните стойности:

бит 5: 0 – Не изисква уведомление за статуса на текущо изпращаното съобщение да бъде върнато към изпращащото мобилно устройство

1 – Изисква уведомление за статуса на текущо изпращаното съобщение да бъде върнато към изпращащото мобилно устройство

Уведомление за статуса е съобщение от типа SMS-STATUS-REPORT.

TP-UDHI – User Data Header Indicator

TP-UDHI е 1-битово поле, позиционирано на бит 6 на първия октет на всички PDU съобщения. Описва дали полето с потребителски данни TP-UD ще съдържа заглавна част. Едно от приложенията на заглавна част на потребителските данни е за реализиране на конкатенирани съобщения (разгледани по нататък).

TP-UDHI може да заема следните стойности:

бит 6: 0 – TP-UD не съдържа заглавна част

1 – TP-UD съдържа заглавна част

TP-RP – Reply Path

TP-RP е 1-битово поле, позиционирано на бит 7 на първия октет на SMS-DELIVER и SMS-SUBMIT съобщения. Указва дали се изисква отговор от получаващото мобилно устройство. Ако TP-RP е изискван, отговарящото мобилно устройство ще опита да използва същия Център на услугата, който е използван и от изпращащото устройство, за да осигури по-голяма вероятност за доставяне на отговора.

TP-MR – Message Reference

TP-MR е поле с дължина 1 октет позиционирано на втория октет от SMS-SUBMIT съобщения. То е целочислено представяне на референтен номер даден на SMS-SUBMIT съобщение и може да заема стойности от 0 до 255. Употребата му е свързана със следенето и различаването на изпратените, доставените и получените съобщения. Няма да бъде използвано за нуждите на дипломната работа и в рамките й ще заема стойност 00.

TP-OA – Originating Address

TP-OA е поле форматирано в съответствие с начините за предаване и представяне на данни. Намира се в SMS-DELIVER съобщения и указва адреса (номера) на изпращача на съобщението.

TP-DA – Destination Address

TP-DA е поле форматирано в съответствие с начините за предаване и представяне на данни. Намира се в SMS-SUBMIT съобщения и указва адреса (номера) на получателя на съобщението.

TP-PID – Protocol Identifier

TP-PID е поле с дължина един октет. Описва протокол от по-високо ниво, за който е предназначено или от който е пристигнало даденото кратко съобщение. Например TP-PID може да зададе краткото съобщение да бъде преобразувано като e-mail, телекс, телетекс и др., но разбира се Центърът на услугата трябва да позволява това. За нуждите на изпращането на съобщение между мобилни станции TP-PID е 00 и е единствената стойност ползвана в дипломната работа.

TP-DCS – Data Coding Scheme

TP-DCS е поле с дължина един октет. TP-DCS (схема за кодиране на данните) описва как са кодирани потребителските данни (TP-UD), какъв вид азбука/набор от символи са използвани, какъв клас (CLASS) е краткото съобщение, дали данните са компресирани, дали самото съобщението представлява индикация за чакащо съобщение (напр. гласово, факс, e-mail). От най-съществено значение е вида азбука, който ще бъде разгледан по-долу и донякъде класа на съобщението, който указва паметта по подразбиране, в която да се съхранява съобщението след пристигането си. Тези две характеристики на схемата за кодиране на данните са атрибути на служещата за интерфейс структура SMS, а останалите се ползват с техните стойности по подразбиране.

Съобщенията могат да са безкласови или с клас 0, 1, 2 или 3.

Табл. 5: Класове съобщения

Безкласови съобщения	Обикновено такива са съобщенията изпращани до мобилен телефон. Обикновено се съхраняват в паметта на телефона.
Съобщения клас 0	Тези съобщения не се съхраняват никъде, а се изпращат направо към дисплея на телефона. В модул, тъй като няма дисплей, те могат да се пренасочат към памет чрез AT команда (AT+CNMI)
Съобщения клас 1	Тези съобщения са специално насочени към мобилното устройство, ако то притежава памет, иначе се съхраняват в SIM картата.
Съобщения клас 2	Тези съобщения са специално насочени към SIM картата.
Съобщения клас 3	Тези съобщения нормално би трябвало да бъдат прехвърлени към терминално устройство ако това се изисква. Това се контролира от AT командата AT+CNMI.

TP-SCTS – Service Centre Time Stamp

Полето TP-SCTS (клеймо на Центъра на услугата) е полуоктетно представено и представя локалното време по следния начин:

Табл. 6: Service Centre Time Stamp - формат

	Година:	Месец:	Ден:	Час:	Минута:	Секунда:	Вр. зона:
Цифри: (полуоктети)	2	2	2	2	2	2	2

Времевата зона показва разликата изразена в четвъртини от часа, между локалното време и GMT. Ако бит 3 от седмия октет (вр. зона) на TP-SCTS е 0 то разликата е положителна, а ако е 1 разликата е отрицателна.
В интерфейсната структура SMS това поле е представено от атрибута SCTimeStamp.

TP-VP – Validity Period

Само относителният формат на периода на валидност ще бъде разгледан.
TP-VP (Relative format) е един октет в целочислено представяне, даващ дължината на периода на валидност считано от момента на получаването на SMS-SUBMIT от Центъра на услугата.
Представянето на времето е следното:

Табл. 7: Период на валидност - стойности

стойност на TP-VP	**Период на валидност**
0 до 143	(TP-VP + 1) x 5 минути (т.е 5 минутнти интервали до 12 часа)
144 до 167	12 часа + ((TP-VP – 143) x 30 минути)
168 до 196	(TP-VP – 166) x 1 ден
197 до 255	(TP-VP – 192) x 1 седмица

TP-UDL – User Data Length

- При кодиране на потребителските данни (TP-UD) чрез подразбираната GSM 7-битова азбука:

  Полето дължина на потребителските данни (TP-UDL) дава целочислено представяне на броя септети в последващото поле с потребителски данни. Ако се ползва механизъм за разширение, действителния брой символи в съобщението ще е по-малък от броя на септетите. Ако присъства заглавна част на потребителските данни (TP-UDH) тогава стойността на TP-UDL е сума от броя на септетите в TP-UDH (включително евентуално запълване) и броя на септетите в последващото TP-UD поле.
- При кодиране на TP-UD чрез 8-битова азбука:

  TP-UDL дава целочислено представяне на броя октети в последващото поле с потребителски данни. Ако TP-UDH присъства, тогава стойността на TP-UDL е сума от броя на октетите в TP-UDH и броя на октетите в последващото TP-UD поле.
- При кодиране на TP-UD чрез UCS2 азбука:

TP-UDL дава целочислено представяне на броя октети в последващото поле с потребителски данни. Ако TP-UDH присъства, тогава стойността на TP-UDL е сума от броя на октетите в TP-UDH и броя на октетите в последващото TP-UD поле.

Трите вида кодиране на символи ще бъде разгледано по-долу.

TP-UD – User Data

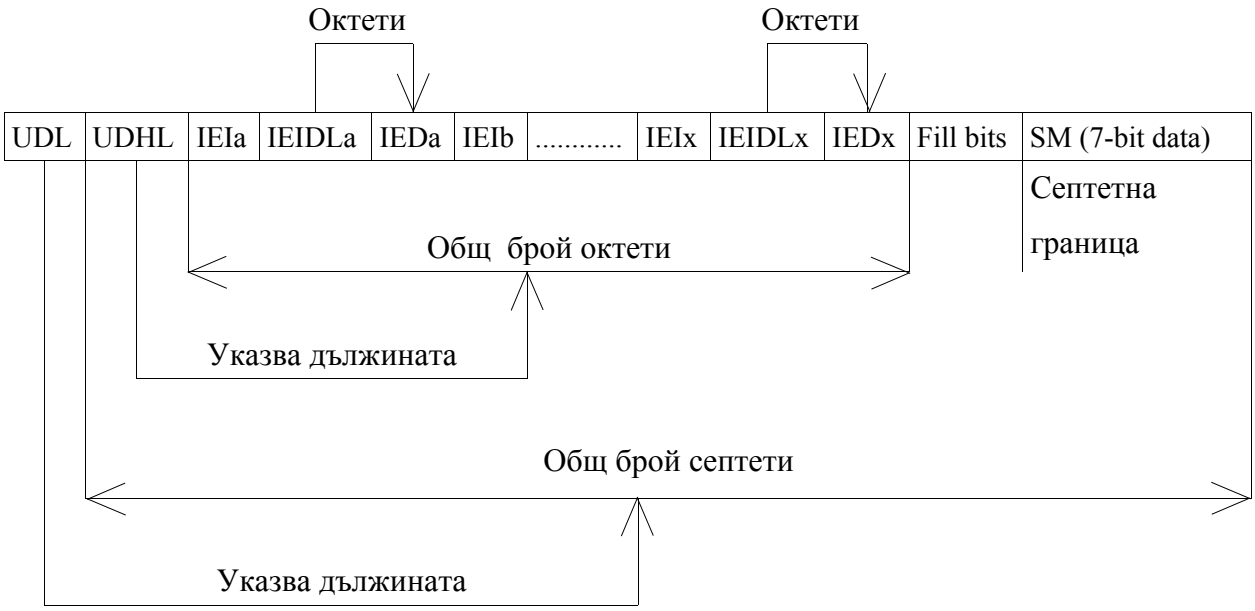
Максималната дължина на полето TP-UD (потребителски данни) е 140 октета. TP-UD може да съдържа единствено самото кратко съобщение или и заглавна част в допълнение, в зависимост от заданието на TP-UDHI.

Когато TP-UDHI има стойност 0, полето TP-UD съдържа само краткото съобщение, където потребителските данни могат да бъдат 7-битови (подразбираща се азбука), 8-битови или 16-битови (UCS2) данни.

Когато TP-UDHI има стойност 1 първите октети на полето TP-UD съдържат заглавна част. Заглавната част започва от първия октет на полето TP-UD и има следния общ вид:

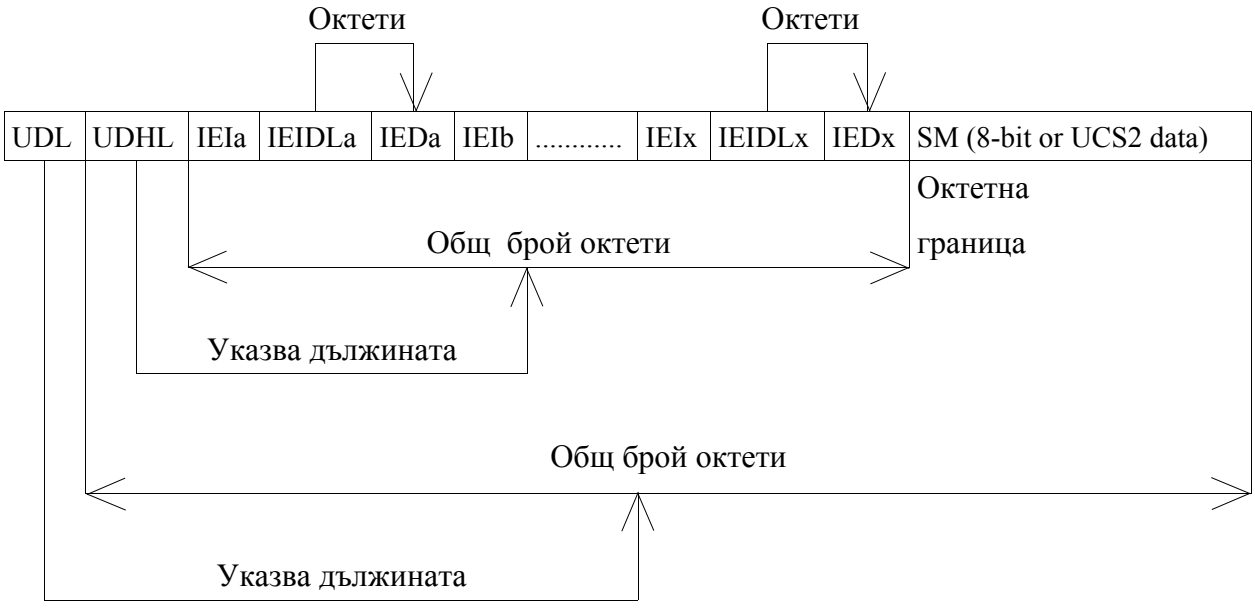
Поле	Означение	Дължина
Дължина на заглавната част на потребителските данни	UDHL	1 октет
Идентификатор-на-Информационен-Елемент "A"	IEIa	1 октет
Дължина на Информационен-Елемент "A"	IEIDLa	1 октет
Данни на Информационен-Елемент "A"	IEDa	0 до "n" октета
Идентификатор-на-Информационен-Елемент "B"	IEIb	1 октет
Дължина на Информационен-Елемент "B"	IEIDLb	1 октет
Данни на Информационен-Елемент "B"	IEDb	0 до "n" октета
Идентификатор-на-Информационен-Елемент "X"	IEIx	1 октет
Дължина на Информационен-Елемент "X"	IEIDLx	1 октет
Данни на Информационен-Елемент "X"	IEDx	0 до "n" октета

Фигура 14 показва разположението на TP-UDL и TP-UD при 7-битови данни. Fill bits са запълващи битове (1) до достигане на септетна граница. SM е краткото съобщение.



Фиг. 14: Разположение на TP-UDL и TP-UD при 7-битови данни

Фигура 15 показва разположението на TP-UDL и TP-UD при 8-битови или 16-битови (UCS2) данни. SM е краткото съобщение.



Фиг. 15: Разположение на TP-UDL и TP-UD при 8 и 16-битови данни

4.2.4. Конкатенация на кратки съобщения

От разгледаните по-горе заглавни части ще се спрем единствено на употребата на Идентификатор-на-Информационен-Елемент (IEI) със стойност 00. Значението му е (идентифицира) "конкатенирани кратки съобщения, 8-битов референтен номер". Използването му позволява кратките съобщения да бъдат слепени за да образуват едно по-дълго съобщение. В този случай данните на Информационния-Елемент се кодират както

следва:

Октет 1 Референтен номер на конкатенираното кратко съобщение.

Този октет трябва да съдържа брояч (по модул 256) указващ референтния номер за определено конкатенирано съобщение. Референтният номер трябва да е постоянен за всяко от кратките съобщения съставящи определеното конкатенирано съобщение.

Октет 2 Най-голям (последен) номер сред кратките съобщения съставящи конкатенираното съобщение.

Този октет трябва да съдържа стойност от 1 до 255 указваща общия брой на кратките съобщения в конкатенираното съобщение. Стойността трябва да е постоянна за всяко от кратките съобщения съставящи определеното конкатенирано съобщение.

Октет 3 Последователен номер на текущото кратко съобщение.

Този октет трябва да съдържа стойност от 1 до 255 указваща последователния номер на текущото кратко съобщение в конкатенираното съобщение. Стойността трябва да започва от 1 и да бъде увеличавана с единица за всяко изпращано кратко съобщение от съставящите определеното конкатенирано съобщение.

Например при дадено кратко съобщение с пореден номер 2 от 4 съобщения, конкатенирани в едно по-дълго ще имаме следното:

UDHL = 05

IEI = 00

IEIDL = 03

IED_1 = AC (примерна стойност)

IED_2 = 04

IED_3 = 02

Тъй като заглавната част (включително октета за дължината й) е общо 6 октета заемани от TP-UD (потребителските данни), то максималният брой символи пренасяни от едно от кратките съобщения съставящи конкатенирано, се намалява на:

- за 7-битовата азбука от 160 на 153 (160-7)
- за 8-битовата азбука от 140 на 134 (140-6)
- за 16-битовата азбука (UCS2) от 70 на 67 ((140-6)/2)

4.2.5. Три вида кодиране на символи

Както неколкократно по-горе бе споменато съществуват три вида кодиране на символите.

При GSM 7-битовата азбука се ползва кодиране на символите според дадената таблица, която с някои различия представлява първите 128 знака от ASCII таблицата. За представянето на един знак са достатъчни 7 бита. Последователността на предаването на битовете им става в съответствие с правилата за представяне и предаване на данните.

Bit Values				B6	0	0	0	0	1	1	1	1
				B5	0	0	1	1	0	0	1	1
				B4	0	1	0	1	0	1	0	1
B3	B2	B1	B0		0	1	2	3	4	5	6	7
0	0	0	0	0	@	Δ	SP	0	¡	P	¿	p
0	0	0	1	1	£	_	!	1	A	Q	a	q
0	0	1	0	2	$	Φ	"	2	B	R	b	r
0	0	1	1	3	¥	Γ	#	3	C	S	c	s
0	1	0	0	4	è	Λ	¤	4	D	T	d	t
0	1	0	1	5	é	Ω	%	5	E	U	e	u
0	1	1	0	6	ù	Π	&	6	F	V	f	v
0	1	1	1	7	ì	Ψ	'	7	G	W	g	w
1	0	0	0	8	ò	Σ	(	8	H	X	h	x
1	0	0	1	9	Ç	Θ	)	9	I	Y	i	y
1	0	1	0	A	LF	Ξ	*	:	J	Z	j	z
1	0	1	1	B	Ø	1)	+	;	K	Ä	k	ä
1	1	0	0	C	ø	Æ	,	<	L	Ö	l	ö
1	1	0	1	D	CR	æ	-	=	M	Ñ	m	ñ
1	1	1	0	E	Å	ß	.	>	N	Ü	n	ü
1	1	1	1	F	å	É	/	?	O	§	o	à

Табл. 8: Символи кодирани с GSM-7

Пример: Нека текстът на краткото съобщение да е TEST. Всяка буква се представя чрез 7 бита, които ще номерираме по следния начин:

T6 T5 . . . T0

E6 E5 . . . E0 и т.н.

Тогава пакетирането на битовете става по следния начин:

Номера на битове							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
E0	T6	T5	T4	T3	T2	T1	T0
S1	S0	E6	E5	E4	E3	E2	E1
T2	T1	T0	S6	S5	S4	S3	S2
0	0	0	0	T6	T5	T4	T3

Фиг. 16: Пакетиране на битове – пример А

Последният октет се запълва с 0 при положение, че не се използва докрай.

Използвайки дадената таблица за битовете на съответните знаци получаваме следното:

Номера на битове								Шестнадесетична стойност
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
1	1	0	1	0	1	0	0	D4
1	1	1	0	0	0	1	0	E2
1	0	0	1	0	1	0	0	94
0	0	0	0	1	0	1	0	0A

Фиг. 17: Пакетиране на битове – пример Б

Така "TEST" ще се кодира като D4E2940A.

Пример: При декодиране на съобщението

0011000C91538918325476000C415**54747A0E4ACF416110945805B5CBF379F85CFE**

се получава "This is a PDU message"

При 8-битовото кодиране се използва разширение на представената таблица до 256 и при него всеки знак има дължина точно 1 октет. Това позволява и по-директното съставяне и четене на шестнадесетичните PDU. Недостатъците са, че по този начин могат да се предават до 140 символа, а и не всички телефони поддържат това кодиране.

При 16-битовото кодиране се използва набора двуоктетен (двубайтов) Universal Character Set, (който е предшественик на UTF-16) и при него всеки знак има дължина точно 2 октета. Това позволява и директното съставяне и четене на шестнадесетичните PDU,. Освен това може да се ползва освен латиница и други символи напр. кирилица. Недостатъците са, че по този начин могат да се предават не повече от 70 символа. Повечето съвременни телефони поддържат това кодиране.

4.2.6. Структура SMS и обвиващ клас PDU

Структурата SMS е част от интерфейса за отдалечено извикване на процедура и е дефинирана във файла sms.idl

Притежава следните атрибути:

char SCA[32] – номер на центъра на услугата. Низ от C стил, терминиран с '\0'.

char Phone[32] – номер на получателя/изпращача на съобщението. Низ от C стил, терминиран с '\0'.

int ClassUse – дали съобщението има клас или не. 0 – безкласово съобщение, 1 – съобщението има клас зададен в Class

int Class – клас на съобщението (0, 1, 2 или 3). Валиден е само при ClassUse != 0.

int Alphabet – азбука (кодиране) на съобщението. 0 – подразбираната 7-битова азбука, 1 – 8-битова азбука, 2 – 16-битова (UCS2) азбука. Ако е различно от 0, 1 или 2 (напр. -1) въз

основа на съдържанието на Data ще се прецени дали да се ползва 7-битова или 16-битова азбука

wchar_t Data[1120] – текст на съобщението. Низ от C стил, терминиран с L'\0'. Типът е wchar_t за да може да побере както еднобайтови ASCII символи, така и двубайтови UNICODE символи. На един елемент от Data съответства точно един символ.

Ако текстът е по-дълъг от побиращия се в едно кратко съобщение се изпращат няколко съобщения формиращи едно конкатенирано. При получаване на съобщения – части от конкатенирано съобщение Data се попълва с текста на цялото конкатенирано съобщение. По този начин SMS представя една по-висока степен на абстракция – (евентуално) конкатенирано съобщение и освобождава клиентските приложения от нуждата сами да разчленяват по-дългите съобщения или да съчленяват отделните части.

Размерът на Data е достатъчен за конкатенирано съобщение съставено от до 7 кратки съобщения 7-битово кодирани, 8 съобщения 8-битово кодирани или 16 съобщения 16-битово кодирани.

int ValidityMinutes – дължина в минути на периода на валидност на съобщението спрямо момента на постъпването му в Центъра на услугата.

SMSTimeStamp SCTimeStamp – клеймо на Центъра на услугата. Отговаря на TP-SCTS. SMSTimeStamp е структура с полета Year, Month, Day, Hour, Minute, Second, Zone.

Преди да преминем към представянето на обвиващия клас PDU ще направим кратко уточняване на термините SMS / PDU.

SMS – логическите данни, структурата. Програмен интерфейс с клиентските приложения (Send, Receive)

PDU – сериализиран SMS в шестнадесетичен текстов вид. Интерфейс с модема.

Външният интерфейс на GSM класа борави с SMS, а вътрешния с PDU.

В някакъв смисъл PDU е техническата версия, а SMS – логическата.

Класът PDU има за цел от една страна да преобразува даден логически SMS в една или няколко PDU единици в шестнадесетичен вид, предначначени за директно подаване към GSM модема, а от друга страна – обратното – да преобразува една или няколко PDU единици, получени от GSM модема, в шестнадесетичен вид към логически SMS.

Следните методи на класа PDU се използват от класа GSM (разгледан в т.7.3) при изпращане на съобщение (вж и Приложение А – 12.1):

void Decompose(SMS sms) – "разбива" подадения sms на единици (units) за изпращане.

int Count() const – дава броя на единиците за изпращане получени при горната операция.

int UnitServiceCenterBytes() const - дава броя байтове за SCA в единица за изпращане с

номер number

int UnitDataOctets(int number) const - дава броя октети (байтове) в единица за изпращане с номер number

const char* Unit(int number) const - дава шестнадесетичното представяне на единица за изпращане с номер number

Следните методи на класа PDU се използват от класа GSM при получаване на съобщение:

bool Compose(const char* hexdata) - Съставя SMS въз основа на подадената единица в шестнадесетичен вид hexdata. Връща true ако с подадената PDU единица е готов (евентуално конкатениран) SMS, иначе връща false.

SMS GetSMS() const - Връща съставения SMS

Тук е мястото да кажем, че получаваното съобщение се представя от GSM модема под формата на една или няколко PDU единици, които разбира се са в шестнадесетичен вид.

5. Използвани технологии

5.1. Програмен език

Като програмен език се използва C++. Обектната му ориентираност е добра предпоставка за писането на изразителен код. Системната му насоченост позволява по-пряка работа при комуникацията с модема – извикват се функциите от програмния интерфейс на операционната система (Win32 API). Предимство е и компилирането на C++ програмите до изпълним код, което допринася за ефективността им по време на изпълнение. От своя страна компилираните програми се изпълняват "самостоятелно" от операционната система, т.е. не се нуждаят от допълнителен интерпретатор или рамка (framework).

5.2. Отдалечено извикване на процедури (RPC)

Технологията на отдалеченото извикване на процедури позволява създаването на разпределени клиент/сървърни програми. RPC библиотеките и функциите наставки (stubs) поемат грижата за повечето от детайлите, отнасящи се за мрежовите протоколи и комуникация. Това дава възможност на разработчика да се съсредоточи върху детайлите от приложната област, абстрахирайки се от мрежовите подробности.

От друга страна при предоставянето на функционалността на сървъра под формата на процедури за отдалечено извикване, а не чрез специално измислен за целта протокол се освобождава разработчика на клиентските приложения от нуждата да разучава новия протокол, защото вместо това се ползва стандартизиран и добре документиран механизъм

на отдалечено взаимодействие. Нещо повече, според [6] "технологията RPC може да бъде използвана за създаването на клиентски и сървърни приложения в хетерогенна мрежова среда включваща и операционни системи като Unix и Apple". В този смисъл RPC в сървърното приложение (услугата) осигурява един широк набор от евентуални клиентски приложения, включително и от операционни системи различни от тази приютяваща (host) услугата – Windows.

5.3. Приложения реализиращи услуги

Приложенията реализиращи услуги или т.нар. windows services са подходящи за продължително изпълнявани задачи и могат да бъдат стартирани заедно с операционната система. В повечето случаи те не взаимодействат с потребителя и не се нуждаят от неговата намеса. Тези характеристики на услугите ги правят идеални за случая, в който дадена (сървърна) функционалност трябва да бъде на разположение през цялото време, независимо от това дали в момента има или няма клиенти на тази функционалност.

5.4. MFC и sqlite3 за демонстрационното приложение

Microsoft Foundation Class (MFC) библиотеката е приложна рамка за програмиране за Microsoft Windows. Написана на C++ (избрания и за дипломната работа език), тя предоставя голяма част от кода необходим за боравенето с прозорци, менюта, диалогови кутии. Използвана е за графичния потребителски интерфейс на демонстрационното приложение.

За по-удобната работа с демонстрационното приложение (напр. наличието на телефонен указател) се ползва библиотеката sqlite3 (вж. [9]). Тя е изключително лека (един .dll файл) и предоставя цялата необходима SQL функционалност за работа с бази данни. Програмният й език е C, което осигурява безпроблемната й интеграция в C++ проект.

5.5. Среда и инструменти за разработка

Като интегрирана среда за разработка (IDE) е използвана Microsoft Visual C++ .NET 2003. Инструментите за разработка са: cl.exe – за компилиране на кода, rc.exe – за компилиране на ресурсите, mc.exe – за компилиране на съобщенията (предназначени за EventLog), link.exe – за обвързване на компилираните единици, uuidgen.exe и midl.exe – за дефиниране на RPC интерфейс.

6. Буфериран вход и изход

6.1. Класът SerialStream

Класът SerialStream служи за четене от и писане в сериен комуникационен порт. Концепцията му е писането да се извършва директно, а четенето да е буферирано. Това означава, че операциите по писане работят директно с файловия манипулатор представляващ отворения порт, докато при четенето нещата са по-индиректни – нишка очаква получаването на байт от порта и при получаването му веднага го записва в буфер и очаква нов. Така операциите за четене на SerialStream четат не директно от порта, а от вече записаните в буфера байтове. Вътрешен клас SerialStream::CyclicBuffer имплементира един такъв буфер. Друга особеност на предоставяните методи за четене, е че те се блокират при текуща работа на нишката с буфера. Така чак след като се прочетат всички налични байтове и се поставят в буфера, се позволява четене. По такъв начин се предотвратява прочитането на "половин отговор" от модема.

Следва кратко описание на по-възловите методи.
void Connect(const char* port_name);
void Connect(const wchar_t* port_name);
Методите Connect приемат единствен параметър низ от C стил, указващ на кой сериен комуникационен порт се намира устройството – в случая GSM модем или телефон. Отваря порта за четене и писане. Предефинирането на този метод – веднъж за аргумент const char* и след това за const wchar_t* позволява употребата му както в приложения компилирани с набор от символи MBCS (MultiByte Character Set) така и в приложения компилирани с UNICODE.

void Reconnect();
Използва се след като веднъж е извикан метод Connect(). Опитва да се свърже (да отвори) порт с име вече зададено от Connect().

void Disconnect();
Затваря отворения порт.

void Write(char byte);
void Write(unsigned char byte);

void Write(const char bytes[], int count = -1);
void Write(const unsigned char bytes[], int count = -1);
Фамилията от методи Write() пишат директно в отворения порт един байт или зададен брой байтове от масив. Ако параметърът count има стойност -1 масивът се разглежда като терминиран от 0.

static DWORD WINAPI ReadThread(LPVOID lpThis);
Нишка извършваща действителното четене от серийния порт и поставяне на прочетените байтове в буфера. Блокирана е по входна операция – прочитане на един байт. При завършването й байтът се поставя в буфера и се чака наново. Ако пристигането на следващ байт се забави повече от определеното от SetCompleteTime() време, операцията по четене се счита за завършена и се разрешава четене от буфера.

void SetCompleteTime(int millis);
Задава време, изразено в милисекунди, за изчакване след получаване (прочитане) на байт, за да се приеме операцията по четене за приключена. Операцията по четене се маркира като приключена, ако след прочитане на последният байт изтече даденото време без да постъпи нов байт.

int AvailableWord(char* buffer, int count, int* next_read, const char* delim, int millis);
Предоставя достъп до едноименния метод на SerialStream::CyclicBuffer

int NextWordIs(const char* word, const char* delim, int millis);
Предоставя достъп до едноименния метод на SerialStream::CyclicBuffer

6.2. Класът SerialStream::CyclicBuffer

Представя опашка от байтове имплементирана като цикличен буфер. Служи за съхраняване на записани байтове и прочитане (извличане) на тези байтове. Организацията е действително опашка – първите записани байтове биват прочитани първи. Следва едно сравнително подробно описание на реализацията.
Член променливи:

- int m_size – големина в байтове на цикличния буфер. Това означава че най-много m_size на брой байта могат да бъдат съхранени в опашката. При постъпване на още един повече той ще покрие (унищожи) най-рано влезлия байт.

- char* m_buffer – указател към масива от байтове представящи опашката.
  Паметта за опашката се заделя динамично от конструктора на обекта и се освобождава от деструктора.
- int m_begin – индекс на началото на опашката
- int m_end – индекс на края на опашката
- bool m_write_started – задава дали е започнала операция по запис (и е активна към момента)
- HANDLE m_write_complete – манипулатор на събитие означаващо, че операцията по запис в опашката е приключила. В началото на всяка операция за писане в цикличния буфер флагът на това събитие се сваля. Вдигането на флага на това събитие става чрез отделна функция (Complete())
- HANDLE m_read_cancel – манипулатор на събитие означаващо, че операцията по четене от опашката е отменена
- CRITICAL_SECTION m_rw_sec – критична секция за синхронизиране на действителното писане и четене от опашката, включително и работата с началото и края й.

Публични методи:

- CyclicBuffer(int size) – конструктор.
  Инициализира ресурсите на опашката – буфера, събитията и критичната секция
  Параметри:
  - int size – задава големината на конструирания буфер. Това е максималния брой байтове, които могат да бъдат записани, без да бъдат четени (извличани) преди да настъпи препълване.
- ~CyclicBuffer() – деструктор.
  Автоматично освобождава заетите от конструктора ресурси.
- int Read(char* buffer, int count, int millis) – прочита не повече от определения брой байтове, ограничено по време.
  Параметри:
  - char* buffer – указател към област от паметта, където ще бъдат копирани непрочетените досега байтове.
  - int count – максималният брой байтове, които да бъдат копирани
  - int millis – милисекунди на изчакване на операцията при положение, че няма операция по запис и никакви байтове в опашката. При текуща операция по запис се изчаква завършването й т.е. millis се игнорира.

Връщана стойност:

Действителният брой прочетени байтове или -1 когато не е дочакано завършването на операцията по запис, а е изтекло времето за чакане или четенето е отменено.

Забележка:

В указаната област от памет buffer се копират не повече от count на брой байта от началото на опашката. Ако наличните байтове в опашката са по-малко от count, то само те се копират. При следващо извикване на този метод четенето започва от мястото, където предишната операция е приключила. По този начин се осъществава извличането на байтове от опашката.

В случай, че не са налични никакви байтове в опашката се изчаква до завършване на операция по запис или до изтичане на времето за изчакване millis милисекунди. Прекъсване на операцията преди изтичане на зададеното време може да стане чрез метода CancelRead()

Табл. 9: Поведение на метода Read()

активна операция по запис	***брой налични байтове в опашката***	***Поведение***
не	0	Изчаква се завършване на евентуално започнала операция по запис или изтичане на определеното време или отмяна на четенето. След това копира евентуално новопостъпилите байтове.
не	> 0	Копира наличните байтове.
да	0	Изчаква до безкрайност завършването на операцията по запис или до отмяна на четенето. След това копира наличните байтове
да	> 0	

- void CancelRead() – прекъсва операцията по четене, когато последната изчаква настъпването на събитие "операцията по запис е приключена"

  Забележка:

  Ако бъде използван метода CancelRead() без да е активна операция по четене, то отменянето ще остане и следващия път, когато бъде извикана Read() ще бъде отменена.
- void Write(const char* buffer, int count) – записва в опашката count на брой байта, копирайки ги от паметта сочена от buffer.

Параметри:

- const char* buffer - указател към област от паметта, от където да бъдат копирани байтове и поставени в края на опашката.
- int count – брой байтове, които да бъдат въведени в опашката

Забележка:

При първото от серия последователни извиквания на този метод се влиза в критичната секция за синхронизиране на достъпа до опашката и се сваля флага на събитието "операцията по запис е приключена". Серията се завършва с извикване на метода WriteComplete().

- void WriteComplete() – обозначава завършването на операцията по запис.

  Забележка:

  Използва се за да разреши забраненото (от Write()) дотогава четене от опашката. Може да се разглежда като затваряща скоба на първото извикване на Write().
- void Clear() - изчиства опашката от всички въведени и непрочетени байтове.

  Забележка:

  Тъй като работи с началото и края на опашката, които са синхронизирани чрез критичната секция, този метод не трябва да се извиква при активна операция по запис освен ако извикването е от същата нишка извикала Write() (и още не извикала WriteComplete()).
- bool AvailableWord(char* word, int count, int* next_read, const char* delim, int millis);

  Параметри:

  - char* word – памет, в която да се постави прочетената "дума"
  - int count – големина на предоставения буфер buffer
  - int* next_read – указател към целочислена променлива, в която да се запише дължината, която заема в буфера наличната "дума", заедно с ограничителите си
  - const char* delim – низ от стил C съдържащ символите, които разделят "думата" от останалата част на буфера
  - int millis – времеизчакване изразено в милисекунди

  Връщана стойност:

  true ако в буфера има "дума" иначе false.

  Забележка:

  "Дума" се счита някаква последователност от символи в буфера евентуално предхождана и евентуално следвана от ограничители - символи зададени в delim.

  Ако next_read е NULL, "думата" се поставя в word и текущата позиция за четене се

премества след "думата" и евентуалните й ограничители.

Ако next_read не е NULL, в *next_read се записва дължината на "думата" заедно с ограничителите си, но в word не се поставя нищо, а позицията за четене остава същата.

- bool NextWordIs(const char* word, const char* delim, int millis);

  Параметри:

  - const char* word – зададената "дума" за сравнение
  - const char* delim – низ от стил C съдържащ символите които разделят "думата" от останалата част на буфера
  - int millis – времеизчакване изразено в милисекунди

  Връщана стойност:

  true ако в буфера има "дума" която съвпада с word иначе false.

  Забележка:

  "Дума" се счита някаква последователност от символи в буфера евентуално предхождана и евентуално следвана от ограничители - символи зададени в delim. Ако в буфера се намери такава дума то тя заедно с евентуалните си ограничители се прочита (премахва от буфера) и се връща true. В противен случай се връща false и позицията за четене от буфера остава непроменена.

7. AT команди

7.1. Инструмента HyperTerminal

HyperTerminal е помощно средство за комуникация вградено в операционната система Windows 2000 и по-високи версии. Стартира се чрез:
Start → All Programs → Accessories → Communications → HyperTerminal

7.1.1. Свързване чрез COM порт

В първоначалния диалог Connection Description се задава име на връзката, напр. "GSM" и се избира иконка.

В следващия диалог Connect To се избира срещуположния край на създаваната връзка. За целите на тестване на модема от елемента за избор "Connect using" се избира серийния COM порт, към който е закачен GSM модема, напр. COM6.

Забележка: Един от начините да се разбере към кой COM порт е свързан GSM модема е от Device Manager (Start → Control Panel → Administrative Tools → Computer Management

→ Device Manager → Ports).

При избиране на сериен порт следващия диалог *COM6* Properties позволява настройване на комуникационните параметри. Един широко разпространен и поддържан набор настройки е следния:

Bits per second: 9600; Data bits: 8; Parity: None; Stop Bits: 1.

Почти всички устройства предначначени за серийна комуникация (вкл. и GSM модемите) поддържат тези комуникационни параметри.

7.1.2. Примерни команди, ръчно въвеждане

След свързване с GSM модема вече можем да зададем няколко пробни команди. След написването на командата въвеждането й (изпълнението й) става чрез натискане на Enter.

Показани са само няколко команди даващи предварителна представа за работата с GSM модема чрез серийния интерфейс.

Пример:

AT

OK

Обяснение: Най-простата команда към модема е AT (от ATtention), на която ако в действителност от другата страна на връзката стои модем, отговорът ще е OK.

Пример:

AT+CPIN?

+CPIN: READY

Обяснение: С тази команда се проверява готовността на SIM картата. В посочения случай PIN кодът е въведен и картата е готова.

Пример:

ATD08811111111;

OK

Обяснение: С командата ATD се набира (гласово) непосредствено посочения номер. Обърнете внимание на точка и запетаята след командата.

Пример:

ATH

OK

Обяснение: Прекъсва текущото повикване. С тази команда може да бъде прекъснато изпълнението на горната.

Пример:

AT+CMGL=4

+CMGL: 1,1,,97

0791538987003043040DD....

Обяснение: Извежда списък на всички SMS съобщения.

Забележка: Възможно е при въвеждането на командите да виждате само техния отговор. Това се дължи на изключено "echo". За включването му се въвежда командата:

ATE1

Самото въвеждане може да не се вижда, но след натискане на бутон Enter би трябвало да видите отговор OK.

Подробно описание на AT командите може да се намери в [1] и [5].

7.2. Инициализация

За правилното взаимодействие с GSM модема е необходимо той да бъде подходящо инициализиран. Както и другите команди така и тези за инициализация са AT команди. В настоящата подточка ще разгледаме само настройките необходими за получаване и изпращане на съобщения (в PDU формат).

Изключване на ехото.

Необходимо е за текущата реализация, за да може след изпълнение на зададена команда, да бъде прочитан директно съответния й отговор (а не ехо на командата и чак след това отговора й). Управление на ехото се извършва чрез ATE командата, изключването му е чрез

ATE0

Въвеждане на PIN код.

Проверка за това дали е необходимо въвеждането на PIN код се извършва чрез командата

AT+CPIN?

Отговор READY означава, че не е необходимо въвеждането на PIN код.

Отговор SIM PIN означава, че е необходимо въвеждането на PIN код. Това става по следния начин (напр. PIN кодът е 1234):

AT+CPIN="1234"

Задаване на формат на SMS съобщенията.

Текущата реализация работи единствено с PDU формата (който за повечето мобилни устройства е подразбиращ се и често пъти единствено възможен). Явното ползване на PDU формата се задава чрез командата

AT+CMGF=0

Индикация за получено съобщение.

Инициализацията на начина за указване на получено съобщение е наложителна, тъй като повечето мобилни устройства (вкл. модеми) по подразбиране не пращат индикация за постъпило кратко съобщението по серийния интерфейс.

Прочитането на текущата настройка става чрез

AT+CNMI?

От петте параметъра (mode, mt, bm, ds, bfr), задавани от AT+CNMI за текущата реализация от значение са само първите два и допустимите им стойности се задават чрез командите

AT+CNMI=3,1,d1,d2,d3

AT+CNMI=2,1,d1,d2,d3

където d са съответните стойности прочетени чрез командата AT+CNMI?

Задаване на памет за получаване на съобщения.

Мобилните устройства позволяват задаването на три вида памети

mem1 – за четене и изтриване на съобщение

mem2 – за записване на съобщения и изпращането им

mem3 – за получавани съобщения

За да може да бъдат четени получаваните съобщения е необходимо mem1 и mem3 да са (сочат) една и съща памет, напр. "SM" – SIM катрата или "ME" – на телефона.

Това се постига чрез командата

AT+CPMS="XX","DD","XX"

където "XX" е паметта за получаване, четене и изтриване на съобщения (напр. "SM"), а "DD" е паметта за писане по подразбиране.

7.3. Клас капсулиращ работата чрез AT команди с GSM модема

Класът GSM капсулира AT комуникацията с GSM модема.

Четене на изхода от модема

След всяка команда трябва да бъде прочетен върнатия от модема отговор. Това се налага поради буферирането – ако не бъде прочетен и съответно премахнат от буфера върнатият отговор, то при следваща команда ще се получи разминаване тъй като ще бъде прочетен отговора на предишна комадна, а не на текущата. Накратко – буферът трябва да бъде освобождаван от върнатите отговори на команди веднага след тяхното изпълнение. При този начин на работа това, което ще постъпва допълнително в буфера са т.нар. неочаквани отговори. Те дават моментна индикация за получаването на кратко съобщение или за

входящо повикване.

За създаването на такава организация на работа е предвидена функцията ReadUnsolicited. Тя бива извиквана, когато е приключило последното изпълнение на команди. Задачата й е да следи за споменатите по-горе неочаквани отговори и на тяхна база да нотифицира за получени съобщения или за входящи повиквания. Когато отново е необходимо изпълнението на команди и четенето на техните отговори тази функция трябва да бъде спряна. Това се осъществава чрез вдигане на флага на събитие (SetEvent). Самата функция е блокирана до получаването на изход (неочаквани отговори) от модема или до активиране на споменатото събитие.

Съобщения предназначени за изпращане биват подреждани в опашка. Това се налага от сравнително бавната физическа обработка на изпращане на съобщение (около 5-8 сек) и възможността да постъпят няколко заявки за изпращане едновременно.

7.3.1. Изпращане на кратко съобщение (SMS)

Два са публичните методи на класа GSM за изпращане на SMS съобщение:

bool Send(SMS sms);

void Post(SMS sms);

Методът Send() извиква DoSend() и изчаква връщането му, което обаче може да отнеме неопределено много време. В зависимост от това дали DoSend() е изпратил успешно съобщението или е регистрирал грешка Send() връща true или false.

Методът Post() просто поставя подадения sms в опашка от съобщения m_post_sms. Нишката изпълняваща статичния метод SendPosted() в последствие извлича едно по едно съобщенията от опашката и извиква DoSend() за всяко от тях за действителното изпращане. От това следствията са две – първо Post() не се блокира от сравнително бавната физическа обработка на изпращане на съобщение и второ, тъй като не изчаква изпращането, не може да получи и съответно да върне успешно или неуспешно е то.

bool DoSend(SMS sms)

Това е методът за действително изпращане на SMS съобщението. Самото изпращане е сихнронизирано чрез критичната секция m_send_sec, за да се предотврати опита за едновременна работа с GSM модема от няколко нишки.

Алгоритъмът е следния:

Създава се обект от клас PDU.

Този обект изпълнява PDU::Decompose(SMS sms) за подаденото съобщение, с което се

подготвят PDU единиците за изпращане в шестнадесетичен вид. Те са повече от една в случаите, когато съобщението е с по-голяма дължина.
Изпраща се всяка от PDU единиците чрез командата
AT+CMGS=*len*
където *len* е дължината на TPDU в октети (т.е. на PDU единицата без да се броят октетите на SCA частта).
Изчаква се до получаването на символа '>' (по-голямо), с което GSM устройството указва готовността си да поеме съобщението, след което се предават и октетите на самата PDU единица (в шестнадесетичен вид).
От получения (след няколко секунди) отговор от GSM устройството се преценява дали съобщението и изпратено или има грешка (напр. грешен номер).
На диаграмата от Приложение А – 12.2 е дадена последователността на взаимодействия между обектите при изпращане на съобщение.

7.3.2. Получаване на кратко съобщение (SMS)

За получаване на съобщение се ползва публичния метод:
bool Receive(SMS* sms, int millis);
Новополученото съобщение се попълва в паметта указана от sms, а максималното времето за изчакване се задава в милисекунди чрез millis. При millis == -1 се изчаква до безкрайност.
Връщаната стойност е true ако е получено ново съобщение в зададения времеви интервал, а false ако е изтекло времето.
Receive() създава събитие (event) и го поставя в опашката m_new_sms_events, с което се регистрира за получаване на нотификация за новополучени съобщения. Блокира се очаквайки настъпването (вдигането на флага) на това събитие, а когато това събитие настъпи използва m_new_sms за да попълни sms.

Получаването на съобщение се указва чрез неочаквана индикация за ново съобщение. Методът за прочитане на неочакваните отговори е ReadUnsolicited(). След като бъде прочетен един такъв неочакван отговор, той бива обработен от ProcessUnsolicited().
ProcessUnsolicited() проверява за подниз "+CMTI:" което е начало на индикацията за получено ново съобщение, и след това и паметта, в която е записано и индекса на съобщението в тази памет. Чрез командата
AT+CMGR=*idx*

където idx е споменатия индекс, се прочита новопостъпилото съобщение.
Прочитаното съобщение е в PDU формат. Предава се като параметър на GSM::Compose, който от своя страна за всяко от получените, но още незавършени съобщения m_received извиква PDU::Compose. Идеята тук е следната – в списъка m_received се съхраняват всички конкатенирани съобщения, на които обаче им липсва още някое кратко съобщение за да бъдат изцяло завършени. За новопостъпилото съобщение се проверява дали е част от някое от съобщенията в m_received и ако е така то бива конкатенирано. Когато дадено съобщение от m_received вече съдържа всичките си части (което включва и случая на едно съобщение с една част) то се прехвърля в m_new_sms и се изтрива от списъка m_received. Тогава се вдига флага на всички събития от опашката m_new_sms_events, с което всички регистрирали се за получаване на нотификация за новополучено съобщение (и блокирани) извиквания на метода Receive() биват уведомявани (и разблокирани)
На диаграмата от Приложение А – 12.3 е дадено последователността на взаимодействия между обектите за получаване на съобщение.

8. Отдалечено извикване на процедура (RPC)

8.1. Дефиниране на интерфейс

Дефинираният интерфейс е формална спецификация на това как да комуникират помежду си клиентското и сървърното приложение [7]. Интерфейсът определя взаимното разпознаване на клиента и сървъра, отдалечените процедури, които клиентското приложение може да извиква, типовете данни на парметрите на процедурите и на връщаните стойности. Чрез него се специфицира и начина, по който данните се предават между клиента и сървъра.
Дефинирането на интерфейса може да се направи чрез езика за дефиниране на интерфейс на Microsoft - Microsoft® Interface Definition Language (MIDL). Този език съдържа дефиниции от стила на езика C, разширени с ключови думи, т.нар. атрибути, които описват начина на предаване на данните в мрежова среда.
Файлът за дефиниране на интерфейс (IDL) съдържа дефиниции на типове, атрибути и прототипи на функции, описващи начина на предаване на данните в мрежова среда.
Файлът за конфигурация на приложението съдържа атрибути, които конфигурират приложението за определена работна среда, без обаче да засягат мрежовите й характеристики.

8.2. Компилация

Генериране на уникален идентификатор (UUID)

Първата стъпка при дефинирането на интерфейс е генерирането на уникален идентификатор чрез инструмента uuidgen. Уникалният идентификатор позволява взаимната идентификация на клиентските и сървърните приложения. За генерирането на шаблонния файл за sms интерфейса бе изпълнена командата:

C:\> uuidgen -i -osms.idl

В последствие към sms.idl файла се добавят декларациите на структурата SMS и на отдалечените процедури Post(), Send() и Receive(), работещи със споменатата структура.

Генериране на "коренни" файлове (stubs) за клиентското и сървърното приложение

Използва се инструмента midl. За генерирането на sms.h, sms_c.c и sms_s.c бе изпълнена командата:

C:\> midl sms.idl

Компилиране на сървърното приложение – Visual C++ проект

Към Visual C++ проекта се добавят sms.h и sms_s.c.

Добавя се и директорията на sms_s.c.

В Project → Properties → Configuration Properties → Linker → Input → Additional Dependencies

се обявява зависимостта от rpcrt4.lib

В Project → Properties → Configuration Properties → C/C++ → Preprocessor → Processor Definitionsce добавя директивата _WIN32_WINNT=0x0500 (или по-висока версия)

Създава се и се добавя към проекта .cpp файл (напр. sms_sp.cpp), който да дефинира функциите декларирани в sms.h, включително и midl_user_allocate и midl_user_free.

Той разбира се трябва да включи sms.h чрез директивата #include "sms.h".

За сега е достатъчно да се спомене, че отдалечените процедури Post(), Send() и Receive() извикват едноименните статични функции на класа SMSService, предавайки им същите параметри.

Компилиране на клиентското приложение – Visual C++ проект

Към Visual C++ проекта се добавят sms.h и sms_c.c.

Добавя се и директорията на sms_c.c.

В Project → Properties → Configuration Properties → Linker → Input → Additional Dependencies
се обявява зависимостта от rpcrt4.lib
В Project → Properties → Configuration Properties → C/C++ → Preprocessor → Processor Definitions се добавя директивата _WIN32_WINNT=0x0500 (или по-висока версия)
Създава се и се добавя към проекта .cpp файл (напр. sms_cp.cpp), който да дефинира функциите midl_user_allocate и midl_user_free декларирани в sms.h.
Той разбира се трябва да включи sms.h чрез директивата #include "sms.h".
Забележка: Вероятно е клиентското приложение да е MFC проект използващ прекомпилирани заглавни файлове. Поради несъвместимостта на генерираните от midl файлове с директивата за използване на прекомпилирани заглавни файлове последната трябва да се изключи.

8.3. Регистриране на интерфейс

Отдалечените процедури, които могат да бъдат извикани от клиентските приложения са описани в интерфейса sms.idl и в последствие в генерирания sms.h. Това именно са Post(), Send() и Receive(). В генерираният (от midl) файл sms_s.c файл се съдържа код за използване на библиотеката имплементираща механизма за отдалечено извикване на процедури. Тук ние ще разгледаме начина на регистриране на предоставяния интерфейс, което дава възможност клиентските приложения да намират и използват отдалечените процедури.
Последователност на извикваните функциите
1. RpcServerUseProtseqEp()
Тази функция указва на RPC библиотеката да ползва зададеното протоколно съгласуване (protocol sequence) в комбинация със специфицираната крайна точка (end point) за получаване на извиквания на отдалечени процедури. Например протоколно съгласуване ncacn_ip_tcp и крайна точка 8099, означава че извикванията ще се очакват на порт 8099 чрез TCP/IP протокола.
2. RpcServerRegisterIf2()
Тази функция регистрира интерфейса чрез RPC библиотеката. Първият параметър на тази функция е манипулатор на спецификацията на интерфейса, който представлява генерирана от MIDL структура указваща регистрирания интерфейс.
3. RpcServerListen()
Тази функция сигнализира на RPC библиотеката да "слуша" за (очаква и према)

отдалечени извиквания на процедури.

Забележка: Посочената последователност може да бъде открита в SMSService::InitRPC(), в която обаче вместо RpcServerUseProtseqEp() се ползва RpcServerUseProtseqEpT() (с T в края). RpcServerUseProtseqEpT() е предефинирана да работи както с MBCS (char) така и с UNICODE (wchar_t).

9. Приложения реализиращи услуги (Services)

9.1. Клас капсулиращ услуга

Клас SimpleService

Класът SimpleService представя приложение предоставящо услуга (windows service) и капсулира функциите необходими за работата на услуга.

Използването му е индиректно – чрез наследяване от потребителски клас. Потребителският клас трябва да дефинира виртуалните методи OnStart(), OnStop() и незадължително OnShutdown().

Един единствен обект от наследен от SimpleService клас може има в една програма, т.е. само една услуга може да се реализира от програмата. Това не е съществено ограничение предвид факта, че почти всички услуги могат да работят в самостоятелен процес. При значителен брой услуги този преразход на ресурси може да се окаже важен и да се появи нуждата от споделянето на един процес от повече услуги. За тази цел са предвидени възможности в приложния интерфейс на операционната система, но не са вградени в класа SimpleService.

След като бъде дефиниран наследяващ SimpleService клас напр. SMSService и неговите виртуални методи, се създава единствения обект от SMSService и се извиква метода му Run() с параметри получени от входната точка – главната функция main().

Методът Run() е блокиращ и завършва чак когато на услугата бъде подадена командата stop.

Методи за инсталиране и деинсталиране на услуги.

Класът SimpleService предоставя статични методи за:

- инсталиране Install()
- деинсталиране UnInstall()
- проверка за инсталиране IsInstalled()

Методи за работа с параметри.

Класът SimpleService предоставя методи за:

- GetParameter() - прочитане на параметър от целочислен тип (DWORD)
- GetParameter() - прочитане на параметър от стригов тип (LPTSTR)
- SetParameter() - запис на параметър от целочислен тип
- SetParameter() - запис на параметър от стригов тип

Тези методи са предназначи за извикване след изпълнението на Run(), т.е. от виртуалните методи OnStart(), OnStop() и OnShutdown(). За хранилище на параметрите те ползват регистрите на операционната система – т.нар. windows registry. Името на ключа, под който се записват и четат данните, включва в себе си името на услугата, която бива предоставяна на приложението след извикването на Run().

Клас SMSService

В проекта smssvc класът SMSService наследявя SimpleService и определя поведението си чрез дефиниране на виртуалните методи OnStart(), OnStop() и OnShutdown().

В SMSService::OnStart() последователно се извикват методите

- InitParameters() - за инициализиране на параметрите на усулгата – настройки за GSM модема и настройки за отдалеченото извикване на процедура.
- InitGSM() - с вече инициализираните параметри за GSM модема, последния се пуска в действие.
- InitRPC() - с вече инициализираните параметри за отдалечено извикване на процедура се създава входна точка за получаване на заявки.

При неуспех на InitParameters() или InitRPC(), след съответния запис в т.нар Event Log (чрез метода LogEvent()), приложението прекратява изпълнението си.

Неуспех на InitGSM() се отбелязва в Event Log, но опитите за свързване с GSM модема продължават периодично. Това означава, че дори и в момента на стартиране на услугата GSM модема да не е бил захранен или свързан, то при отстраняване на проблема, услугата ще започне използването му без да се налага рестартирането й.

Класът SMSService притежава статичен обект GSM m_gsm и предоставя следният интерфейс до него

static int Send(SMS sms, int timeout) – служи за изпращане на кратко съобщение с ограничение по време. Този именно статичен метод бива извикван при изпълнението на отдалечената процедура int Send(SMS sms, int timeout) дефинирана в sms_sp.cpp.

static int Receive(SMS *sms, int timeout) – служи за получаване на кратко съобщение с ограничение по време. Този именно статичен метод бива извикван при изпълнението на отдалечената процедура int Receive(SMS sms, int timeout) дефинирана в sms_sp.cpp.

Класова диаграма е дадена в Приложение А – 12.1.

9.2. Конфигуриране на услугата

Под конфигуриране на услугата се разбира задаването на настройки касаещи работата й чрез специално за целта приложение от вида аплет за контролен панел – smssvc.cpl. По същество то е библиотека за динамично свързване експортираща функцията CPlApplet(). Класът ControlPanel структурира и капсулира извикванията на тази функция от операционната система. ControlPanel предоставя за предефиниране виртуални методи отговарящи на различни извиквания на CPlApplet() - OnDblclk(), OnExit(), OnGetCount(), OnInit(), OnInquire(), OnSelect() и OnStop().

В проекта smssvccpl класът SMSSvcPanel наследява ControlPanel и предефинира

OnDblclk() - за да покаже диалоговия прозорец за взаимодействие с потребител и

OnInquire() - за да отговори на запитването за иконка и име на аплета.

Показваният прозорец е от тип лист със свойства (клас Sheet наследяващ Property Sheet) и има три страници за настройки на отдалеченото извикване на процедура (клас PageRPC), за настройки на GSM модема (клас PageGSM) и за наблюдение на статуса на услугата smssvc и спирането, стартирането и рестартирането й.

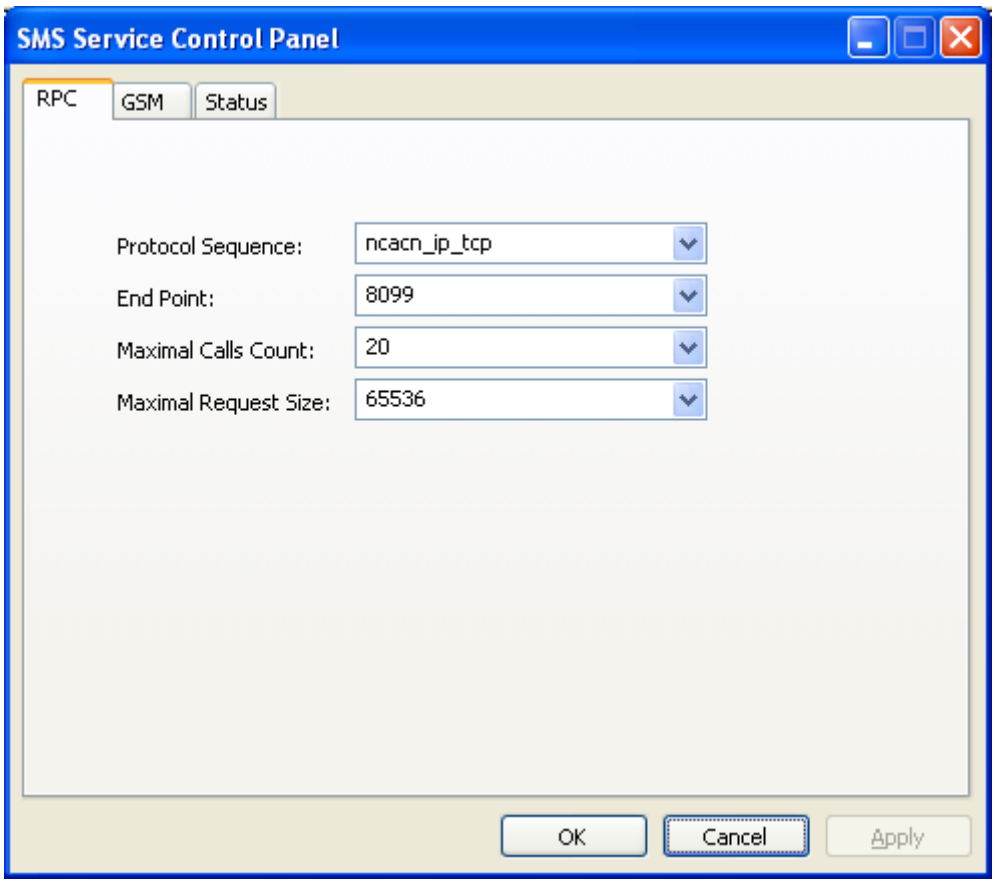
Параметри на услугата

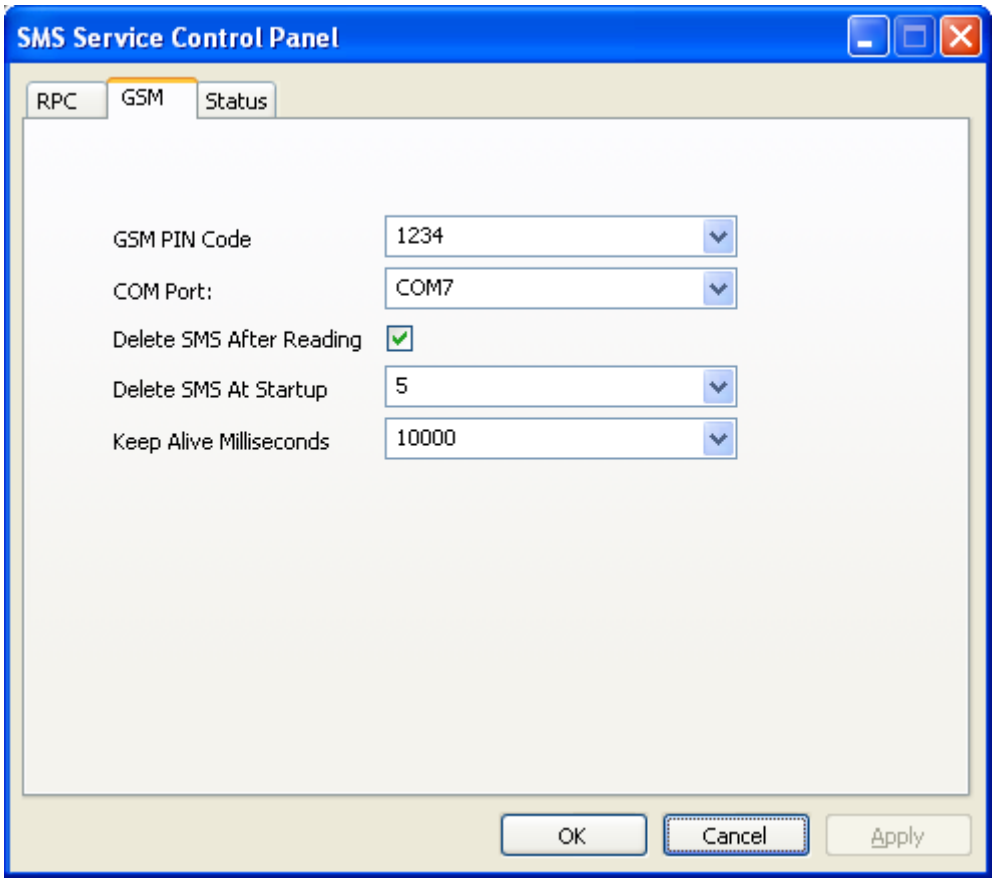
Параметрите могат да се редактират както чрез графичния потребителски интерфейс предоставян от контролния аплет описан по-горе, така и директно в windows registry чрез използването на инструмента regedit. Ключът под който се намират е

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\smssvc\Parameters [8]

В дадената по-долу таблица са групирани точните имената на параметрите, така както са записани в windows registry и силно наподобяващи съответните етикети в контролния аплет по местоположението им в съответните страници (със свойства).

параметър	*примерна стойност*	*обяснение*
в страница RPC на контролния аплет		
RPC_ProtocolSequence	ncacn_ip_tcp	протокол за комуникация при извикване на отдалечена процедура
RPC_Endpoint	8099	точка на достъп при извикване на отдалечена процедура
RPC_MaxSize	65536	максимална големина в байтове на една RPC заявка
RPC_MaxCalls	20	максимален брой едновременно обслужвани RPC заявки
в страница GSM на контролния аплет		
COM_Port	COM2	Сериен комуникационен порт
GSM_PINCode	0000	PIN код на SIM картата
GSM_DeleteNewSMS	1	Дали да изтрива новите съобщения веднага при пристигането им, за да не запълват паметта на телефона
GSM_DeleteAtStartup	5	Колко съобщения да изтрива при стартиране за да освободи място за нови
GSM_KeepAliveMilliseconds	10000	Време за проверка на връзката с модема





Фиг. 18: снимки на екрана - конфигуриране на RPC и GSM

9.3. Инсталиране

Инсталиране на услугата и контролния й аплет (в Control Panel) се извършва с подаването на подходящи параметри на smssvc.exe изпълнена от командния ред. Диаграма на разгръщане е дадена в Приложение А – 12.4.

Първо smssvc.exe трябва да бъде копирана в постоянната си директория – т.е. там където ще остане да се изпълнява под формата на услуга.

В същата или друга (постоянна) директория трябва да се копира контролния аплет smssvc.cpl (и евентуално smssvc.cpl.manifest).

От командния ред при текуща директория тази на smssvc.exe се извиква:

smssvc -i full-path-and-name-of-smssvc.cpl

където -i означава интсталиране,

а full-path-and-name-of-smssvc.cpl е пълното име на smssvc.cpl (заградено в двойни кавички ако съдържа интервали)

Вътрешно се извикват методите SimpleService::Install() и ControlPanel::Install(), за да инсталират съответно приложението реализащо услуга и контролния аплет.

За деинсталиране от командния ред се изпълнява

smssvc -d

където -d означава деинтсталиране.

ВНИМАНИЕ: ***Задайте коректен PIN код (чрез контролния аплет) преди стартиране на приложението! В противен случай рискувате SIM картата да бъде БЛОКИРАНА (при три неуспешни опита на приложението да въведе кода, т.е. при третото му стартиране).***

За ръчни инсталиране, деинсталиране, проверка и контрол на услугата може да се ползва командата sc разгледана по-долу.

9.4. Работа с командата sc

Командата за управление на услугите (SC.exe) е полезна за тестване и дебъгване на програми реализиращи услуги. SC имплементира извиквания към всички функции за управление на услугите. Задаването на параметрите става чрез командния ред. Една базова функционалност на командата SC е стартирането и спирането на услуга. Друга полезна функционалност е представяне на информация за състоянието и настройките на дадена услуга.

Командата SC е вградена за операционната система Windows XP.

Синтаксисът на командата SC е следния:

sc [ServerName] Command ServiceName [OptionName= OptionValue]

ServerName

Име на сървъра ако командата се изпълнява за отдалечен компютър. Параметърът се изпуска ако се работи с локалния компютър.

Command

Една от следните команди. (Не са изброени всички. За изчерпателен списък вж. MSDN)

Табл. 10: Командни опции за SC.EXE

Команда	Описание
Create	Регистрира (в Windows registry) приложение реализиращо услуга
Delete	Премахва (от Windows registry) приложение реализиращо услуга
Qc	Показва конфигурацията на услуга
Query	Показва статуса на услуга
Start	Стартира приложение реализиращо услуга
Stop	Изпраща STOP заявка към приложение реализиращо услуга

ServiceName

Име на услугата (така както е записано в регистрите на операционната с-ма Windows registry)

OptionName

Име на опционален параметър.

OptionValue

Стойност за параметъра с име OptionName.

Примерни извиквания на командата SC:

Регистрира (в Windows registry) приложение реализиращо услуга

sc create smssvc binPath= "C:\dev\sms\smssvc.exe" type= own start= auto DisplayName= "Short Messages Service"

Показва конфигурацията на услуга

sc qc smssvc

Показва статуса на услуга

sc query smssvc

Стартира приложение реализиращо услуга

sc start smssvc

Изпраща STOP заявка към приложение реализиращо услуга

sc stop smssvc

Премахва (от Windows registry) приложение реализиращо услуга

sc delete smssvc

9.5. Предимства на услугата пред компилиране на GSM частта към програма

Основното предимство на услугата е това, че може да се ползва едновременно от няколко програми. Услугата е единственото приложение работещо директно с GSM устройството, а предоставя интерфейс (под формата на процедури за отдалечено извикване), който може да се ползва от множество клиентски приложения едновременно. В противния случай, когато gsm частта е компилирана към всяко от приложенията ще се появи изискването всяко от тях да ползва отделен порт и съответно отделно устройство, което е нецелесъобразно или да пък да ползват един и същи порт и устройство, което ще доведе до конфликт.

Дори и когато само едно приложение ползва услугата отново има няколко предимства.

Едното е, че клиентското приложение може да работи на един компютър, а услугата да се предоставя от друг компютър, като двата разбира се трябва да са свързани чрез мрежа.

Например чрез RPC протокола ncacn_http, който се ползва съвместно с IIS (Internet Information Server) заявките могат да се предават чрез HTTP, т.е. дава се Интернет достъп до услугата.

Друго предимство е, че при промяна на реализацията на услугата, доколкото предоставяния интерфейс е постоянен, не е необходимо прекомпилиране на клиентските приложения. Това дава възможност за прозрачно подобряване на имплементацията напр. употребата на нов вид модем (с някои различия в AT командите), както дори и за използване на различна технология за изпращане и получаване на кратки съобщения – напр. по Интернет, вместо чрез GSM модем.

Предлаганата услуга е в изпълним, компилиран вид е не ангажира разработчика с подробности по съвместното компилиране с новосъздаван програмен код.

10. Демонстрационно клиентско приложение

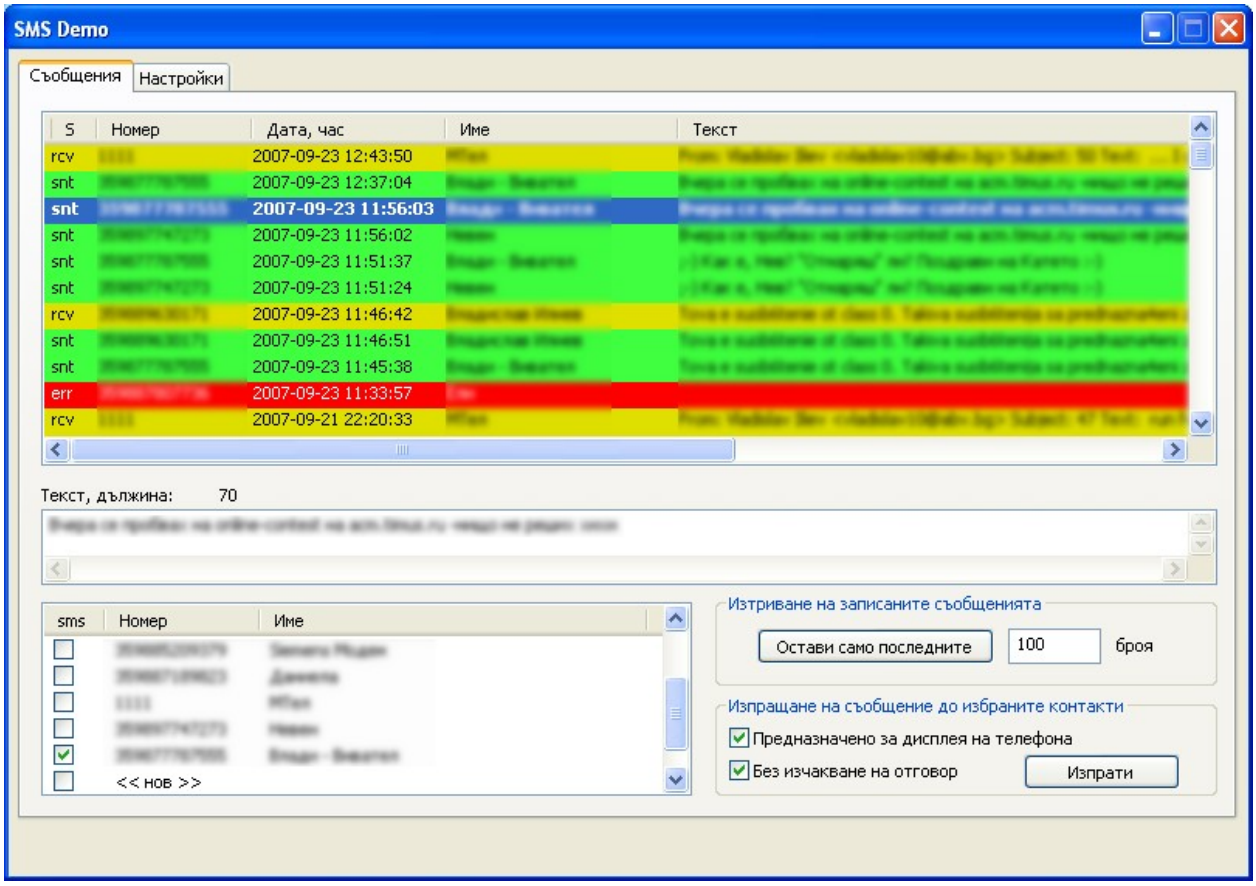
SMSDemo.exe е клиентско приложение с графичен потребителски интерфейс за изпращане и получаване на кратки съобщения. Самото изпращане и получаване на съобщения става чрез извикване на отдалечените процедури предоставяни от услугата. В този смисъл SMSDemo от една страна демонстрира възможностите на услугата по нагледен за потребителя начин, а от друга, кодът на приложението е един пример за

боравене с отдалечените процедури предоставяни от услугата, служещ на разработчика.

10.1. Инструкция за потребителя

Приложението SMSDemo има две страници – "Съобщения" и "Настройки".

10.1.1. Страница "Съобщения"



Фиг. 19: снимка на екрана - страница "Съобщения" на SMSDemo

В горната половина на страницата е таблицата с получените и изпратените съобщения. Всяко записано съобщение е представено от един ред от таблицата.

В колоната "S" с трибуквено съкращение е даден състоянито на съобщението, което определя и фоновия цвят на реда:

- rcv – жълт фон – получено съобщение
- snt – зелен фон – изпратено съобщение
- err – червен фон – грешка при изпращане на съобщение

В колоната "Номер" стои номерът, от който е получено или на който е изпратено съответното съобщение.

В колоната "Дата, час" стои времето на съобщението. В случай на получено съобщение това е времето на получаване на съобщението от Центъра на услугата, а при изпращано

съобщение това е системното време в момента на изпращане.

В колоната "Име" е името съответстващо на дадения номер.

В колоната "Текст" стои самия текст на съобщението.

Продълговатото поле е за въвеждане на текст за изпращане. Над него е дадена дължината на въведения текст.

В долния ляв квадрант е таблицата с въведените контакти – с колони "sms", "Име", "Номер".

Въвеждането на нов контакт става чрез двукратно щракване върху последния ред на таблицата - "<< нов >>". Появава се диалогов прозорец "Добавяне на нов контакт", в който се въвеждат името и номера за съответния контакт. Номерът трябва да е в международен формат, без знака +, напр. 359812345678. Не бива в номера да има други знаци освен цифри, както и не бива да започва с 0.

Редактирането или изтриването на вече въведен номер става чрез двукратно щракване върху съответния ред. Появяващият се диалогов прозорец "Редактиране / Изтриване на контакт" позволява изтриване на даденият контакт при натискане на бутон "Изтрий" (и потвърждение), както и запис на въведени корекции с бутона "Запиши".

Групата "Изтриване на записаните съобщения" дава възможност да се изтрият старите съобщения. Самото изтриване става чрез натискане на бутон "Остави само последните" (N броя).

Групата "Изпращане на съобщение до избраните контакти" дава възможност да се изпрати съобщение на отбелязаните (с отметка) контакти. При избрана опция "Предназначино за дисплея на телефона" се изпраща съобщение с клас 0. След прочитане такова съобщение автоматично ще бъде изтрито. Самият текст на съобщението за изпращане, разбира се, се задава в продълговатото поле за текст. За самото изпращане се натиска бутон "Изпрати". Върху начина на изпращане влияе опцията "Без изчакване на отговор". При избрана опция съобщението се предава на услугата smssvc, което става мигновено, и се счита за изпратено (т.е Post()). Когато опцията не е избрана се изчаква докато услугата smssvc действително изпрати съобщението (Send()) и получи отговор от мрежата и въз основа на него се преценява дали е изпратено успешно.

При щракване върху колоната "sms" се отбелязват всички контакти, а при повторно щракване отметките се изчистват.

При двукратно щракване върху съобщение (от таблицата със съобщения), като подготовка за отговор, в текстовото поле се зарежда текста на съобщението, а от контактите се отбелязва съответния номер. В случай че номерът е непознат (не е въведен) се появява

диалоговият прозорец "Добавяне на нов контакт", със зареден вече номер.

10.1.2. Страница "Настройки"

Чрез тази страница се задават параметрите за свързване с RPC сървъра.
Когато те не са зададени (например при първото пускане на програмата) при стартиране на приложението се появява предупреждение за грешка "The RPC protocol sequence is invalid".
Един комплект подразбирани стойности е следния

Табл. 11: Примерни настройки на SMSDemo

параметър	стойност
Protocol Sequence	ncacn_ip_tcp
End Point	8099
Network Address	127.0.0.1

За задаването им трябва да се натисне бутон "Bind" за записване на стойностите и използването им за свързване със сървъра. Стойностите се съхраняват в windows registry под ключа
HKEY_CURRENT_USER\Software\FMI_M21557\smsdemo\Settings

10.2. Описание на реализацията

10.2.1. Вградена база данни – sqlite3

За целите на структурираното съхранение на данните се ползва система за управление на база данни SQLite версия 3 (3.4.1) предоставяна от [9] . SQLite е малка *безплатна* C библиотека, която може да бъде вградена в продукт нуждаещ се от възможностите на самостоятелна база данни, без необходимост от администриране. Един единствен файл представлява една база от данни, което позволява лесното прехвърляне на данните от един компютър на друг и създава удобство при архивиране. В случая на приложението SMSDemo, базата данни е файла sms.db. В него се съхраняват релационните таблици за контактите (contact) и съобщенията (message).
Схема на таблицата Contact (контакти)

- _id INTEGER PRIMARY KEY AUTOINCREMENT -- първичен ключ
- _phone TEXT -- телефонен номер на контакта
- _name TEXT-- име на контакта

Схема на таблицата Message (съобщения)

- _id INTEGER PRIMARY KEY AUTOINCREMENT -- първичен ключ
- _type INTEGER -- вид на съобщението, вж. таблицата по-долу
- _datetime TEXT -- дата и час
- _phone TEXT -- телефонен номер, на/от който е изпратено/получено съобщението
- _text TEXT -- текст на съобщението

Употребата на sqlite библиотеката се извършва не чрез прякото използване на C API интерфейса й, а чрез обвивнето му в класовете SQLiteDB и SQLiteDB::Statement.

Следва описание на основните методи на тези класове.

За класа SQLiteDB:

void Open(const char* dbname);

void Open(const wchar_t* dbname);

Служат за отваряне на база данни, където dbname е името на файла с базата данни. Един SQLiteDB обект в даден момент може да е отворил най-много една база данни.

void Close();

Затваря отворената база данни.

Statment Prepare(const char* query);

Statment Prepare(const wchar_t* query);

Подготвят заявка за изпълнение от базата данни.

int GetLastInsertRowId();

Връща идентификационния номер на последния въведен в базата данни запис.

За класа SQLiteDB::Statment:

int Step();

Изпълнява подготвената заявка и/или преминава на следващия ред от SELECT заявка.

int Columns();

Дава броя на колоните във върнатия от изпълнението на (SELECT) заявка резултат.

void Text(int column, const char** text);

void Text(int column, const wchar_t** text);

int Integer(int column);

Извлича текст или цяло число от зададена колона. Използва се текущия ред от резултата на SELECT заявка. За премиваване към следващия ред се ползва Step();
Достъпът до готов за употреба обект SQLiteDB става чрез метода
SQLiteDB* CsmsdemoApp::GetDB();
който връща указател към обекта.

10.2.2. Клиентска част за RPC

Обвързване със сървърната част

За обвързване със сървърната RPC част служат методите:
bool CsmsdemoApp::Bind();
void PageSettings::OnBnClickedBind();
Методът Bind() зарежда необходимите параметри protocol_sequence, endpoint и network_address от windows регистрите. Въз основа на тези параметри съставя стринг, дефиниращ обвързването чрез функцията RpcStringBindingCompose. Съставеният стринг дефиниращ обвързването и манипулатора sms_IfHandle се подават на функцията RpcBindingFromStringBinding за да извърши действителното обвързване.
Методът OnBnClickedBind() е част от потребителския интерфейс и се извиква при натискане на бутон "Bind" от страницата "Настройки". Зададените от потребителя настройки Protocol Sequence, End Point и Network Address първо се записват в windows регистрите и след това се извиква Bind(). Тук трябва да се отбележи, че ако вече има валидно обвързване и някоя отдалечена процедура (например Receive(SMS*, int)) се изпълнява (или е блокирана), то дори и да се зададе ново обвързване (чрез натискане на бутон "Bind") то ще се отрази чак при следващото извикване на отдалечената процедура. Това за Receive() означава, че новото обвързване ще влезе в сила след получаване на съобщение (или след рестартиране на приложението SMSDemo ;)

Изпращане на кратко съобщение

bool CsmsdemoApp::PostSMS(SMS sms);
bool CsmsdemoApp::SendSMS(SMS sms);
И двата метода са обвивки на извикване на отдалечена процедура съответно Post() и Send(), като улавят евентуални изключения хвърлени от механизма за извикване на отдалечени процедури и предупреждават потребителя за това.

Получаване на кратко съобщение

bool CsmsdemoApp::ReceiveSMS(SMS* sms, int timeout);

Този метод също е обвивка на извикване на отдалечената процедура за получаване на съобщение. При получено съобщение връща true, а при грешка (включително изключение от механизма за извикване на отдалечени процедури) или изтекло време за изчакване връща false.

10.3. Тестване на услугата

Функционалността предоставяна от услугата, както и самата RPC връзка между клиентско приложение и услугата са тествани чрез приложението с графичен потребителски интерфейс smsdemo.

Извършени са следните видове тестове:

- изпращане на съобщение до 160 латински символа – базова функционалност;
- изпращане на съобщение с над 160 латински символа – конкатениране;
- изпращане на съобщение до 70 символа кирилица – кодиране, базова функционалност;
- изпращане на съобщение с над 70 символа кирилица – кодиране, конкатениране;
- изпращане на съобщение със зададен клас (0) – класови съобщения;
- получаване на съобщение до 160 латински символа – базова функционалност ;
- получаване на съобщение с над 160 латински символа – конкатениране;
- получаване на съобщение до 70 символа кирилица – кодиране, базова функционалност;
- получаване на съобщение с над 70 символа кирилица – кодиране, конкатениране;

Освен това са проведени и следните тестове за "издръжливост", при които системата показва способност да се "самовъзстанови" (без потребителска намеса):

- прекъсване на RPC връзката и последващото й възстановяване;
- прекъсване на серийната връзка между модема и компютъра (изваждане на кабела) и последващо възстановяване;
- прекъсване на захранването на модема и последващото му възстановяване.

Тестваните модеми са:

- Siemens MC35i
- Sony-Ericsson GM29
- Sony-Ericsson K600i (телефон)

11. Използвана литература

[1] **Книга:** Sony Ericsson (2003), GM29 Integrator's Manual

[2] **Статия:** Short Message Service

http://en.wikipedia.org/wiki/Short_message_service

[3] **Книга:** European Telecommunications Standards Institute (2001),

ETSI TS 100 901 V7.5.0 - Technical realization of the Short Message Service (SMS)

[4] **Статия:** Sony Ericcson (2003), Application Note Construction of SMS PDU's

[5] **Книга:** Siemens (2003), MC35i AT Command Set

[6] **Статия:** MSDN, Platform SDK, Remote Procedure Call

[7] **Статия:** Remote Procedure Call

http://msdn2.microsoft.com/en-us/library/aa378651.aspx

[8] **Статия:** Nigel Thompson (1995), Creating a Simple Win32 Service in C++

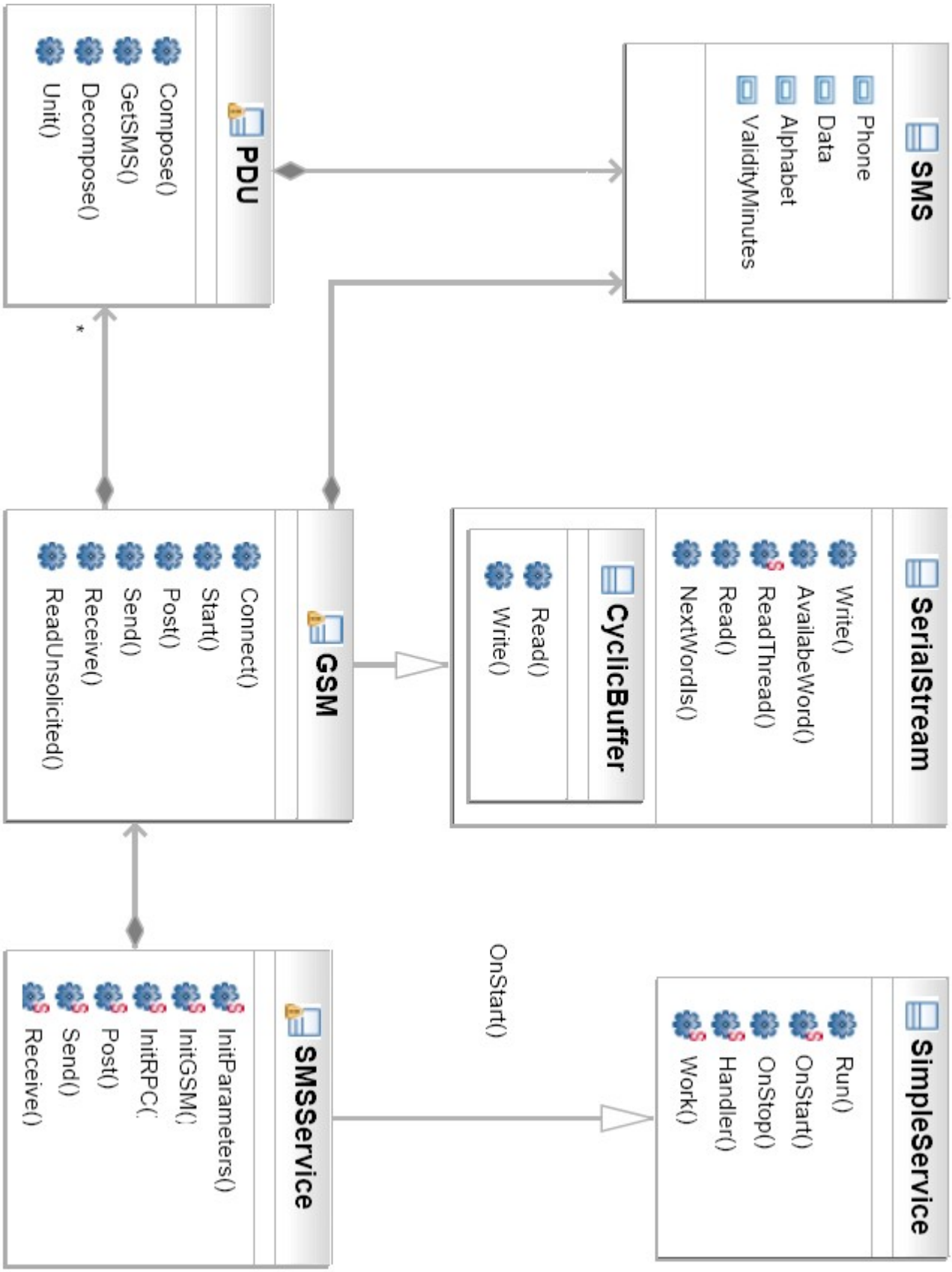
http://msdn2.microsoft.com/en-us/library/ms810429.aspx

[9] **Статия:** SQLite Database

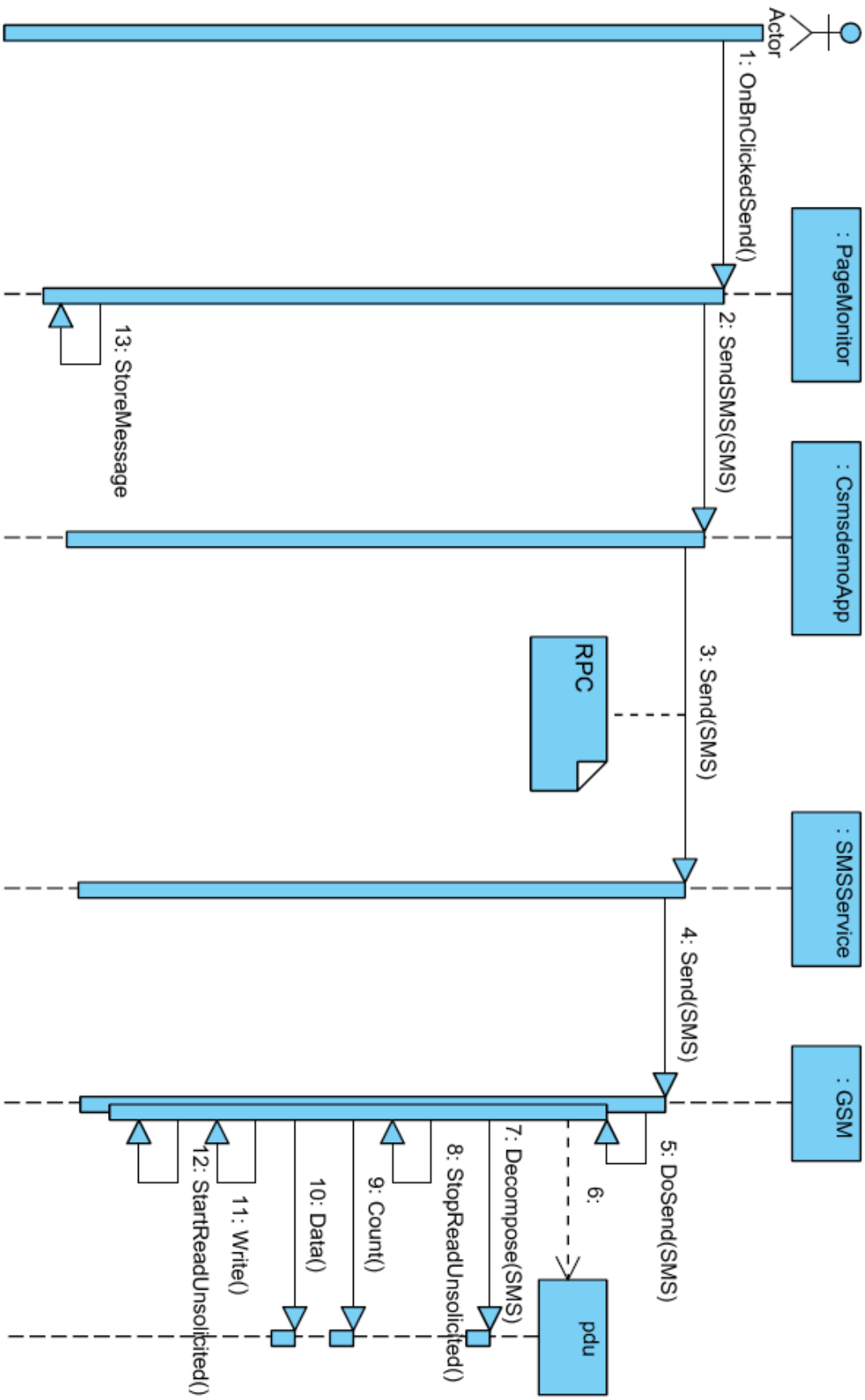
http://www.sqlite.org

12. ПРИЛОЖЕНИЕ А - UML диаграми

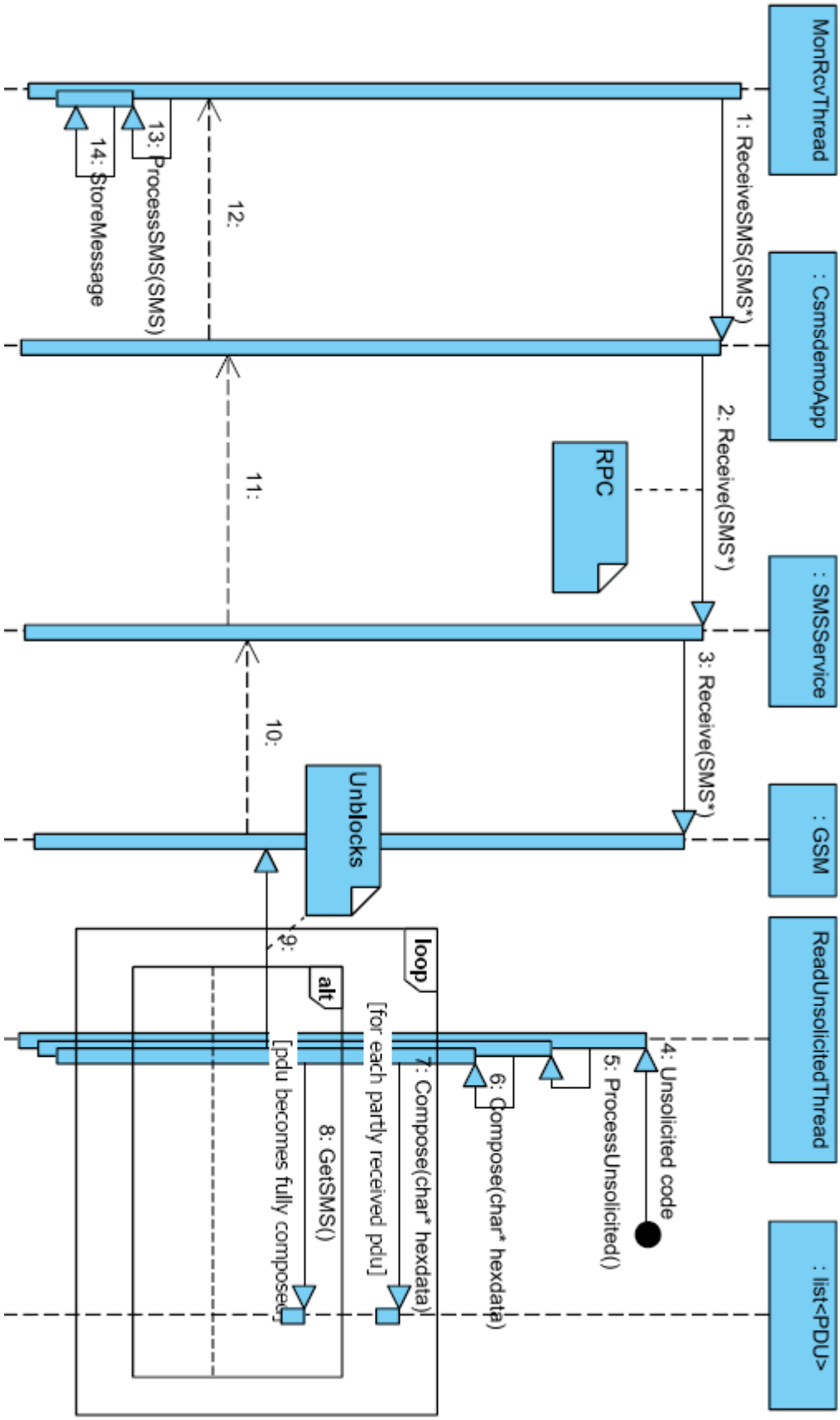
12.1. Клас диаграма – основни класове на услугата



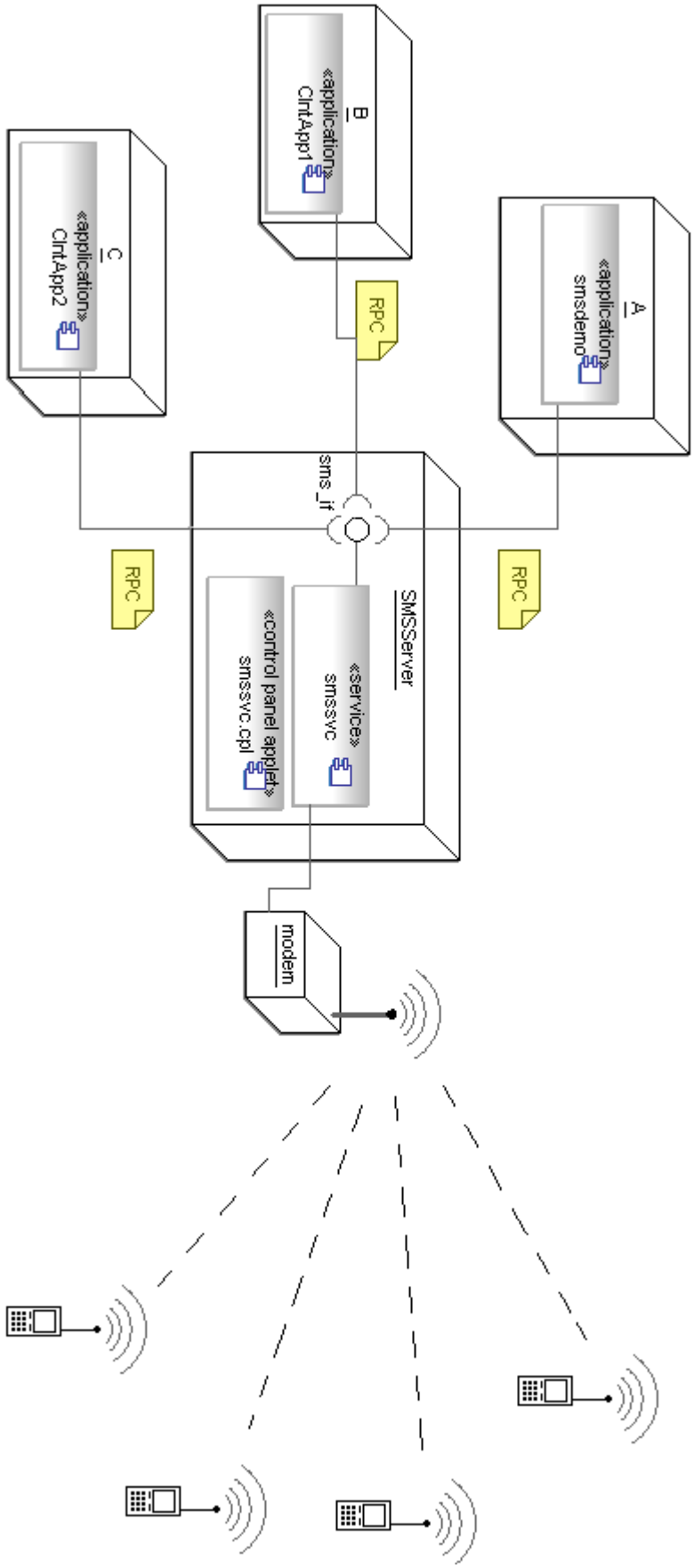
12.2. Диаграма на последователност – изпращане на съобщение



12.3. Диаграма на последователност – получаване на съобщение



12.4. Диаграма на разгръщане



13. ПРИЛОЖЕНИЕ Б – Индекс на фигурите и таблиците