СУ “Св. Климент Охридски”

Факултет по Математика и Информатика

Катедра “Информационни технологии”

Дипломна работа

Тема:
“Непостоянно свързани умни клиенти”

Дипломант : Ваня Валентинова Арсова

Специалност: Софтуерни технологии

Факултетен номер: M21501

Научен ръководител: доц. д-р Силвия Илиева

София, 2007

Съдържание

Съдържание 2
Списък на фигурите 6
Списък на таблиците 6
Списък на използваните съкращения 8
Увод 10
Цел 10
Полза от дипломната работа 11
Структура на дипломната работа 12
1. Обзор на наличните технологии 15
1.1. Windows приложения и уеб клиенти 15
1.2. Умни Клиенти 16
1.3. Сравнение между уеб клиентите и умните клиенти 19
1.4. Изводи 21
2. Умни клиенти. Кеширане на данните 22
2.1. Типове данни 22
2.1.1. Данни, предназначени за четене 22
2.1.2. Краткотрайните данни 23
2.1.3. Обобщение на основните характеристики 24
2.2. Кеширане на данни. Кешираща инфраструктура 24
2.3. Cashing Application Block – готов кеширащ приложен блок 27
2.4. Зареждане на данните 30
2.4.1. Предварително кеширане 30
2.4.2. Противодействащо кеширане 31
2.4.3. Сравнение между предварителното и противодействащо кеширане 32
2.5. Съгласуване на данни 32
2.5.1. Песимистично съгласуване 33
2.5.2. Оптимистично съгласуване 33
2.5.3. Съгласуване на данните с помощта на ADO.NET 34
2.6. Изводи 34
3. Непостоянно свързани умни клиенти 36
3.1. Непостоянно свързани проектантски стратегии 37
3.1.1. Подход базиран на данни 38

3.1.2. Подход ориентиран към услуги ... 42
3.1.3. Дизайн на непостоянно свързани умни клиенти при използване на подхода, ориентиран към услуги ... 44
3.1.3.1. Асинхронна комуникация ... 44
3.1.3.2. Минимизиране на сложните мрежови взаимодействия ... 45
3.1.3.3. Добавяне на възможност за кеширане на данните ... 46
3.1.3.4. Управление на връзката ... 47
3.1.3.5. Проектиране на съхрани и препрати механизъм ... 48
3.1.3.6. Управление на конфликтите при данните и бизнес правилата ... 49
3.1.3.7. Взаимодействие с CRUD подобни уеб услуги ... 53
3.1.3.8. Използване на подход базиран на задачи ... 55
3.1.3.9. Управление на зависимостите ... 56
3.2. Изводи ... 58
4. Microsoft и умните клиенти - CAB, Smart Client Software Factory, ClickOnce ... 60
4.1. CAB(Composite UI Application Block) ... 60
4.2. Smart Client Software Factory ... 63
4.3. Процес на разработване ... 64
4.4. ClickOnce ... 70
4.5. Обобщение ... 74
5. Примерно приложение ... 75
5.1. Бизнес сценарии, поддържани от Task Viewer ... 75
5.2. Архитектура на приложението ... 80
5.3. Проектиране на уеб услугите, използвани от примерното приложение ... 81
5.3.1. Изглед на разработката ... 81
5.3.2. Контекстен Изглед ... 82
5.3.2.1. Слой на услугата ... 83
5.3.2.2. Бизнес слой ... 84
5.3.2.3. Слой за достъп до ресурси ... 84
5.3.3. Логически изглед ... 85
5.4. База данни ... 86
5.5. Task Viewer ... 86
5.5.1. Изглед на потребителският интерфейс ... 87
5.5.2. Логически изглед ... 92
5.5.2.1. Шаблон Модел Изглед Представител ... 93
5.5.2.2. Агент на услуги ... 94
5.5.2.3. Асинхронни комуникация с уеб услуги ... 95

5.5.2.4. Навигация между изгледи ... 96
5.5.2.5. Кеширане, съгласуване, справяне с конфликти и управление на връзката 97
5.5.3. Изглед на имплементацията ... 97
5.5.4. Изглед на потока на управление ... 100
5.6. Изглед на разгръщане ... 101
5.7. Обобщение ... 102
6. Тестване ... 103
7. Алтернативни разработки и бъдещи насоки ... 109
7.1. Алтернативни разработки ... 109
7.1.1. iAnywhere технологии ... 110
7.1.2. iAnywhere решения ... 111
7.1.2.1. Синхронизация на ниво база данни ... 111
7.1.2.2. Подход, базиран на съхрани и препрати механизъм ... 112
7.1.2.3. Комбинация между синхронизация с база данни и подхода съхрани и препрати 113
7.1.2.4. Обмен на данни чрез репликация на файлове ... 113
7.1.2.5. Офлайн уеб приложения ... 114
7.1.2.6. Офлайн уеб приложения със синхронизация на ниво база данни ... 115
7.2. Компании, използващи умни клиенти ... 116
7.3. Бъдещи насоки ... 118
7.4. Обобщение ... 120
8. Заключение ... 122
9. Използвана литература ... 124
10. Приложение ... 127
Приложение А: Клас диаграми на аутентификационната услуга ... 127
Бизнес слой: Business Logic и Business Entity проекти ... 127
Слой за достъп до ресурси: Data Access ... 128
Приложение Б: Клас диаграми на услугата за данни ... 129
Слой на услугата: Data Contracts проект ... 129
Слой на услугата: Service Contracts проект ... 130
Слой на услугата: Service Implementation проект ... 131
Бизнес слой: Business Logic и Business Entity проекти ... 132
Слой за достъп до ресурси: Data Access ... 133
Приложение В: Task Viewer диаграми ... 134
Изгледи и представители на Infrastructure.Module ... 134

Изгледи и представители на Task Management модула..... 134

Списък на фигурите

Таблица 1: Сравнение доколко двете архитектури удовлетворяват бизнес изискванията 19
Таблица 2: Основни характеристики на типовете данни използвани при умните клиенти 24
Таблица 3: Сравнение между предварителното и противодействащото кеширане 32
Таблица 4: Сравнение между SQL Compact и SQL Express Edition 41
Таблица 5: Детайлно описание на случаите на употреба 77
Таблица 6: Task Viewer тест план 103
Таблица 7: Обща постановка при синхронизирането на ниво база данни 111
Таблица 8: Обща постановка при подхода, базиран на съхрани и препрати механизъм 112
Таблица 9: Обща постановка при метод, комбиниращ синхронизация на ниво база данни и подход съхрани и изпрати 113
Таблица 10: Обща постановка при обмен на данни чрез репликация на файлове 114
Таблица 11: Обща постановка при офлайн уеб приложения 115
Таблица 12: Офлайн уеб приложения със синхронизация на ниво база данни 116

Списък на таблиците

Фигура 1: Сравнение между ориентирания към услуги и базирания на данни подход 38
Фигура 2: Дърво на решенията каква Microsoft база данни да се използва при различни сценарии. 40
Фигура 3: Task Viewer - случаи на употреба 76
Фигура 4: Task Viewer – архитектура на приложението 81
Фигура 5: Структура на проектите на Task Viewer Web Service имплементацията, изградена с използването на WCF пакети 82
Фигура 6: Слоеве в контекста на Task Viewer услугите 83
Фигура 7: Клас диаграма за AuthorizationService слоя на услугата 85
Фигура 8: Task viewer - схема на базата данни 86
Фигура 9: Аутентификационна форма 87
Фигура 10: Скелет и потребителска структура на приложението 88
Фигура 11: Task Management Module 89
Фигура 12: Task Management и Administration module 90
Фигура 13: Форма за избор на проектите, които ще са достъпни в офлайн режим 91
Фигура 14: Форма за разрешаване на възникнали проблеми 91
Фигура 15: Форма за създаване на нов проект 92
Фигура 16: Форма за създаване на нов потребител 92
Фигура 17: Логически изглед на шаблона Модел Изглед Представител 93
Фигура 18: Изгледи и представители на Administration Module 94
Фигура 19: Разделяне на агента на услуги от прокси интерфейса 94

Фигура 20: Task Management Service Agent 95
Фигура 21: Асинхронни уеб заявки със поддържане на time-out..... 96
Фигура 22: Навигация между изгледите Project Navigation и TaskReview 96
Фигура 23: TaskViewer - структура на проектите..... 98
Фигура 24: Разделяне на интерфейсите от имплементацията на модулите 99
Фигура 25: Диаграма на последователност при използването на агент на услуги..... 101
Фигура 26: Изглед на разгръщане за примерното приложение 102
Фигура 27: Клас диаграма на бизнес слоя на аутентификационната услуга 127
Фигура 28: Клас диаграма на класовете фабрики използвани в аутентификационната услуга 128
Фигура 29: Клас диаграма на класовете Repository използвани в аутентификационната услуга 128
Фигура 30: Клас диаграма на Data Contract проекта, част от услугата за данни 130
Фигура 31: Клас диаграма на Service Contracts проекта на услугата за данни 130
Фигура 32: Клас диаграма на Service Implementation проекта на услугата за данни 131
Фигура 33: Клас диаграма на бизнес слоя на услугата за данни..... 132
Фигура 34: Клас диаграма на класовете фабрики на услугата за данни..... 133
Фигура 35: Клас диаграма на класовете Repository на услугата за данни..... 133
Фигура 36: Изгледи и представители на Infrastructure.Module..... 134
Фигура 37:Изгледи и представители на Task Management проекта..... 134

Списък на използваните съкращения

ACID	-	Atomic, consistent, isolate, durable
CAB	-	Composite UI Application Block
CDMA	-	Code division multiple access
CDPD	-	Cellular Digital Packet Data
COM	-	Component Object Model
CRUD	-	Create, Read, Update, Delete или създай, прочети, обнови, изтрий
DCOM	-	Distributed Component Object Model
Dll	-	Dynamic-Link Library
DTO	-	Data Transfer Object
GUID	-	Globally Unique Identifier
GPRS	-	General Packet Radio Service
GSM	-	General Service Model
HTML	-	Hypertext Markup Language
JMS	-	Java Message Service
J2EE	-	Java 2 Platform, Enterprise Edition
LAN	-	Local Area Network
MVC	-	Model View Controller
MVP	-	Model View Presenter
MSI	-	Microsoft Windows Installer
RDA	-	Remote Data Access
RDBMS	-	Relational Database Management System
SDK	-	Software Development Kit
SCSF	-	Smart Client Software Factory
UI	-	User Interface
URI	-	Uniform Resource Identifier
URL	-	Uniform Resource Identifier
WCF	-	Window Communication Foundation
WiFi	-	Wireless Fidelity
WLAN	-	Wireless Local Area Network

WMI	-	Windows Management Instrumentation
WOW	-	Windows on Windows
WPF	-	Window Presentation Foundation
WSDL	-	Web Services Description Language
WWLAN	-	Wireless Wide Area Network
XML	-	Extensible Markup Language

Увод

В последното десетилетие се наблюдава нарастване на обвързаността между бизнеса и новите технологии. Организациите се стремят да се разрастват, осигурявайки на своите служители гъвкав начин на работа, базиран на висока мобилност, и неограничен от времето и мястото достъп до информация. Корпорациите се нуждаят от сложен софтуер, предоставящ голямо разнообразие от възможности, покриващ изцяло областта им на действие, но не са готови да чакат години за неговото разработване. Нуждата е сега и веднага.

Мощни изчислителни и комуникационни устройства спомагат за осъществяването на необходимата мобилност. Разработчиците на софтуер гъвкаво трябва да се ориентират към нарасналите изисквания, необходимо е да се адресира нуждата за производство на комплексен софтуер за кратки срокове. Затова голямо значение придобиват базираните на модули приложения и готовите софтуерни фабрики. При софтуерните приложения се създават концептуално нови типове приложения, отговорни за задоволяването на мобилните корпоративните потребности. Едни от тях са smart clients(умни клиенти) приложенията, характеризиращи се с богат Windows интерфейс, оползотворяващи мощността на процесорите и гъвкавостта на уеб пространството.

Цел

Дипломната работа си поставя теоритично-практически цели в областта на разработването на модулни, непостоянно свързани умни клиенти с помощта на Microsoft Software Factories. В дипломната работа се разглеждат случаите, в които е удачно да се избере умен клиент пред алтернативното уеб решение. Разглеждат се специфичните проблеми, свързани с разработката на умни клиенти и възможни начини за тяхното решение. Различните нужди на компаниите налагат разработването на различни видове умни клиенти. Настоящият документ се спира подробно на непостоянно свързаните умни клиенти. Разглежда имплементирането на умен клиент с използването на CAB (Composite UI Application Block) и SCSF(Smart Client Software Factory). Чрез

използването на приложни блокове и софтуерни фабрики се скъсява времето за разработка и излизането на пазара на даден софтуерен продукт. CAB предоставя възможност за модулна разработка на приложения, базирана на добри практики (MSDN, Smart Client – Composite UI Application Block, 2005). Предимствата при използването на този подход са възможност продукта да излезе на пазара възможно най-скоро (най-важните части се разработват първи) и възможност за адаптирането му за различни клиенти. Един от важните въпроси при адаптирането е начинът за разгръщане на приложението, затова ще се разгледа ClickOnce технологията на Microsoft и възможностите, които тя предоставя. Обръща се внимание как SCSF помага на архитектите и разработчиците при разработването на умни клиенти. Имплементирано е демонстрационно приложение, илюстриращо част от проблемите и един начин за тяхното решение на базата на .NET Framework 2.0, CAB (Composite UI Application Block), SCSF (Smart Client Software Factory) и ClickOnce технологията.

Задачи, произтичащи от целта:

- Да се анализират проблемите, свързани с разработването на непостоянно свързан умен клиент
- Необходимо е умните клиентите да дават възможност за лесно и бързо добавяне на нова функционалност, базирана на модули. Да се изследва може ли Composite UI Application Block даде решение на този проблем.
- Един умент клиент може да се състои от няколко модула. Различните потребители имат достъп само до определена част от модулите. Умният клиент трябва да бъде проектиран така, че да разпознава потребителя и да зарежда само тези модули, за които клиентът има права. Да се анализира как това изискване може да бъде решено на базата на Microsoft технологиите.
- Да се проектира и разработи примерен офлайн умен клиент, като се приложи теорията и изследванията от предишните задачи.

Полза от дипломната работа

Дипломната работа е предназначена предимно за софтуерни архитекти и разработчици. Събраната и систематизирана информация позволява дипломната

работа да се използва като ръководство за разработване на умни клиенти. Табличното представяне на информацията може да служи като справочник на решения, за набора от проблеми, свързани със специфичната област на разработка.

Структура на дипломната работа

- *Обзор на наличните технологии* – първа глава е въвеждаща и има за цел да запознае читателите с основната концепция на умните клиентите. Тя обосновава нуждата от нов тип приложения, чрез разглеждане на характеристиките на десктоп и уеб приложенията и процеса им на развитие. В нея на кратко се описват различните типове умни клиенти. Прави се сравнение между тънките и умните клиентите. Сравнението е предназначено за софтуерните архитекти, които трябва да вземат решение на каква архитектура да се спрат.
- *Умни клиенти. Кеширане на данните* – При използването на умни клиенти въпросът за кеширане на данни стои на първо място. Кеширането е обширен въпрос, затова му е отделена цяла глава. Тя разглежда видовете данни, които могат да бъдат кеширани от гледна точка на предназначението им. Обръща се внимание на нуждата да бъде изградена кешираща структура и какви части трябва да я изграждат. Като вариант на вече готов кеширащ блок, който може да се използва, е разгледан Microsoft Cashing Application Block. Главата разглежда две стратегии за кеширане на данните (противодействаща и предварителна). Обръща внимание на двата начина за съгласуване на данни: песимистичен и оптимистичен. За улеснение на съгласуването Microsoft предоставя обекта Dataset. В главата са разгледани неговите възможности и как той може да помогне в процеса на обновяване на данните.
- *Непостоянно свързани умни клиенти* – трета глава се занимава в детайли с централната тема на дипломната работа – непостоянно свързаните умни клиенти. За разлика от предната глава, тук акцентите са архитектурата на самото приложение и как тя може да помогне за разрешаване на конфликтите, които са резултат от кеширането на данни,

когато приложението е било в офлайн режим. Разглеждат се базираният на данни и ориентираният към услуги подход. Обръща се внимание на версиите на Microsoft SQL Server и коя от тях може да се използва за обработка на сървъра и коя на клиента. Прави се сравнение между Microsoft SQL Server Express Edition и Microsoft SQL Server Compact Edition. Базата данни дава на готово голяма част от необходимата функционалност. При подхода ориентиран към услуги тази функционалност трябва да бъде заимплементирана ръчно от разработчиците. Главата разглежда въпросите, с които те се сблъскват при имплементирането на подхода ориентиран към услуги.

- *Microsoft и умните клиенти - CAB, Smart Client Software Factory, ClickOnce* - необходимо е умните клиенти да са модулни приложения с лесно разгръщане. Главата има за цел да запознае читателите с готовите блокове и фабрики предоставени от Microsoft и той да прецени доколко те са удобни за разработване на модулни приложения. Разглеждат се градивните компоненти на Composite Application Block-a(CAB), за да се покаже как могат да се имплементират модулни приложения. Представен е и Smart Client Software Factory(SCSF), който автоматизира част от задачите и служи за спазване на ръководни принципи при разработка на софтуер. В главата се разглежда как CAB и SCSF могат да бъдат използвани при проектирането на умни клиенти. В последната част се обръща внимание на различните варианти на разгръщане, които предоставя ClickOnce на CAB и SCSF базираните приложения.
- *Примерно приложение* – главата показва архитектурата на примерно умно приложение изградено на базата на ориентираният към услуги подход. Дефинира са случаите на употреба за приложението на тяхна база е решено да бъдат изградени два модула. Главата разглежда архитектурата на разработените за целта уеб услуги, базата данни и архитектура на самото умно приложение. Всяка архитектура е показано чрез набор от изгледи: изглед на имплементацията, контекстен и логически.
- *Тестване* – главата преглежда накратко как трябва да бъде тествано умно приложение. Тя се явява продължение на предната глава. В нея е описан

премерен тест план, който обхваща функционалността на примерното приложението.

- *Алтернативни разработки и бъдещи насоки* - В тази глава се разглеждат алтернативни начини за разработване на умни клиент приложения, които не са свързани с Microsoft технологиите. Разглеждат се технологиите и решенията, които iAnywhere предлага за пазара на умни клиенти. Главата описва случаи, поради които някои компании са решили да използват умни клиент приложения. Тя завършва с насоки как може да се разшири за в бъдеще дипломната работа, като се обобщават пропуснати части и се споменават нови технологии, невключени в настоящата разработка.
- *Заключение* – обобщава дипломната работа и изводите направени в нея.
- *Използвана литература* – съдържа информация за използваната литература в дипломната работа.
- *Приложение* – Разделено е на 3 части. Приложение А и Приложение Б съдържат клас диаграмите за уеб услугите разработени в примерното приложение. Приложение В съдържа диаграми на изгледите и предствителите за Task Viewer приложението.

1. Обзор на наличните технологии

За да се разбере нуждата от нов тип приложения като умните клиенти, е необходимо да се разгледат принципите, стоящи в основата на богатите и тънките приложни модели, да се разгледат предимства и недостатъците, асоциирани с всеки един от тях. Първата част е кратък исторически увод, въвеждащ в причините, довели до възникването на тези приложения. Втората част разглежда типовете умни клиенти, а последната се заема със сравнение на приложимостта на умните клиентите и уеб приложенията.

1.1. Windows приложения и уеб клиенти

В средата на 90-те броят на „богатите" клиенти, разработени за Microsoft Windows платформата, започва драматично да расте. Тези приложения са проектирани да се възползват от локалните хардуерни ресурси и функциите на клиентската операционна система. Много от тези приложения са монолитни, самостоятелни и без възможност за комуникация с останалите приложения или компютри в локалната мрежата. Появата на DCOM технологията е повратен момент в тяхното развитие, тя позволи на тези приложения да работят в разпредена среда. За съжаление клиентските и сървърните приложения са тясно свързани, а с прилагането на DCOM, те увеличават размера и комплексността си. Последствията са проблеми с поддържането и разгръщането им. Десктоп приложенията работят със споделени асемблита, а това води със себе си проблема, получил известност с наименованието dll ад.

Интернет ерата не променидесктоп реалността. Малките промени бяха свързани само с добавянето на функционалност, проверяваща дали е излязла нова версия или актуализация.

Но появата на Интернет промени софтуерната реалност, на сцената се появи нов тип приложение. Началото бе поставено от прости HTML страници, представящи определена информация. За кратко време те придобиват динамика, благодарение на скриптовите езици и стават достойни конкуренти на десктоп приложенията. Недостатъците на „богатите" клиенти не са им присъщи, а едно от най-големите им предимства е, че се разгръщат и обновяват централизирано,

намалявайки драстично разходите за поддръжка. Настъпи момент, в който много клиент – сървър приложения, се пренаписваха, за да се възползват от новата технология. Потребителите загубиха възможност да използват техники, с които бяха свикнали, като „drag and drop" функционалността, „undo, redo", чувствителната към контекста помощ и т.н. Времето за отговор на приложенията се увеличи драстично, а това ярко се отрази на бизнес приложенията с голям вход на данни. За да се извърши проста операция при тези приложения, стана нужно да се навигира между множество от страници. Като цяло сложните приложения спечелиха много от гледна точка на спестени пари за поддръжка, но и загубиха много от гледна точка на ефикасност и удобство, загубиха свободата си да работят в офлайн режим. Потърпевши от това са мобилните потребители, нямащи връзка с Интернет или локалната мрежа. На тях им се наложи да въвеждат информацията, събирана през деня повторно при завръщането им в офиса. Ефикасността на работа на десктоп потребителите, имащи връзка с Интернет и локалната мрежа, също не е напълно гарантираната. При ниска честота на канала(low bandwidth) или висока латентност, производителността им драстично намалява. Не на последно място стои проблема с трудната интеграция на уеб базираните приложения със специализирани периферии като бар код четци и касови апарати.

Сега живеем в информационно базирана икономика. За да се преуспее в този високо конкурентен пазар, потребителят трябва да събере и обобщи информация от много източници. Потребителите изискват от приложенията сами да събират и интегрират информацията, да допринасят в ежедневната им работа при планирането и анализирането на данните, при визуализирането и сондирането в резултатите. Тънките клиенти не могат да предоставят необходимите нива на функционалност, производителност, гъвкавост и интеграция. Нужни са бързи и отзивчиви приложения, работещи с разнообразни устройства. Ако добавим и увеличаващата се мобилност на работната сила, става ясно, че на софтуерния пазар има нужда от нов тип приложения.

1.2. Умни Клиенти

Smart client или умен клиент е термин, който Microsoft използва, за да опише този нов тип приложения. Според Дейвид Хил „Терминът умен клиент е

създаден, за да отличи разликите между типичните клиенти с богат интерфейс от вчерашния ден и новото поколение клиентски приложения (Hill, 2004). Тези приложения са комбинация от предимствата на десктоп и уеб приложенията. Те се възползват от богатите интерфейсни компоненти и техниките на десктоп базираните приложения като ги комбинират с централизиран начин за разгръщане и актуализиране. Умните клиенти предлагат по-голяма гъвкавост от традиционните настолни приложения, тъй като се възползват от съвременните техники и технологии: уеб услуги за комуникация, .NET Framework за решение на dll конфликтите. Подходът за изграждането на приложенията като монолитни единици, е заменен от модулен принцип на изграждане, даващ независимост на модулите, гарнирана с единен интерфейс за работа.

Умните клиенти варират в зависимост от изискванията отправени към тях. Microsoft дели умните клиентите в три категории, в зависимост от целевата платформа (Microsoft Patterns & Practices, 2004, стр. 16-19):

- Windows умни клиенти
- Офис умни клиенти
- Мобилни умни клиенти

Платформата влияе върху избора на инструменти и компоненти за разработка. Различните типове приложения се сблъскват със специфични проблеми, но запознаването с тях не е в целите на дипломната работа. В тази секция ще бъдат споменати накратко, за да се запознаят читателите със специфичните варианти.

В реални ситуации приложението може да бъде разработено и в трите вариации, ако това се изисква от нуждите на потребителя.

Windows умни клиенти – тези приложения представляват еволюция на традиционния Windows клиент с богат потребителски интерфейс. За разработването им може да се използва всеки инструмент, използван за разработката на Windows приложения. Серията Microsoft Visual Studio 2005 може да бъде свалена от сайта на Microsoft (Microsoft, Microsoft Visual Studio 2005). За разработка може да се използва и .NET Framework 2.0 SDK, който също е достъпен от сайта на Microsoft (Microsoft, .NET Framework 2.0 Software Development Kit (SDK), 2006). За ускорено разработване на такъв тип приложения могат да се използват Microsoft Enterprise Library (Microsoft, Enterprise Library 3.0 – April 2007, 2007), Composite Application Block (CAB)

(Microsoft, Smart Client – Composite UI Application Block, 2005), Microsoft Smart Client Factory (Microsoft, Web Service Software Factory, 2007). Целевите хардуерни платформи са настолни, лаптоп или таблет компютри. Пример за такъв тип клиент е Microsoft Outlook.

Офис умни клиенти – Офис smart client решенията интегрират различни източници на данни с възможностите, които предоставят Word, Excel или InfoPath. Тези приложения превръщат статичен контейнер за документи в динамична среда, която може да извлича и предоставя на клиента информация, в зависимост от контекста на документа. Тази информация може да бъде под формата на отчети, диаграми, справочник на задачи и т.н. Този тип приложения имат потенциала да станат неразделна част от управленския информационен цикъл в една организацията. За разработката им се използва Microsoft Visual Studio Tools за Microsoft Office (MSDN, Visual Studio Tools for Office), базираните на XML Smart Tags и Smart documents. Целевата платформа, за която са предназначени, е Microsoft офис пакета

Мобилни умни клиенти - Това са приложения, които работят на мобилни устройства с малки размери, умни телефони, джобни компютри и т.н. Тези приложения използват .NET Compact Framework (Microsoft, .NET Compact Framework 2.0 Service Pack 1 Redistributable, 2006), подмножество на .NET Framework (Microsoft, Microsoft .NET Framework Version 2.0, 2006), който е включен в Windows Mobile 6 (Microsoft, Windows Mobile 6, 2007). Благодарение на него, те могат да се възползват от богатия Windows потребителски интерфейс, но по начин оптимизиран за работа с устройства с малки размери. Мобилните умни клиенти се използват, за да осигурят мобилен достъп до основна информация и услуги, или да събират и обобщават данни, когато потребителя е мобилен. За разработването им се използва Microsoft Visual Studio. То съдържа инструменти и симулатори, които позволяват разработването и тестването на приложения с богат потребителски интерфейс за мобилни устройства. .NET Compact Framework осигурява контроли, middleware и комуникационна инфраструктура за свалянето и изпълнението на умните приложения. Те включват Windows Mobile 6.0 Smartphone SDK, Windows Mobile 6.0 Emulator Images for Smartphone, които могат да бъда свалени от сайта на Microsoft (Microsoft, Windows Mobile 6 Professional and Standard Software Development Kits Refresh, 2007), и Mobile Client Software Factory, за което може

да се намери информация в онлайн библиотеката MSDN (MSDN, Mobile Client Software Factory – July 2006, 2006).

1.3. Сравнение между уеб клиентите и умните клиенти

При избора на архитектура трябва да се оценят набор от фактори. Внимателно трябва да се определят текущите и бъдещи бизнес и приложни нужди, за да се реши дали подходът умен клиент е подходящ за дадено приложение. Ако се избере неподходяща архитектура, тя може да не изпълни поставените изисквания и да не удовлетвори очакванията на потребителя и на бизнеса като цяло. Смяната на архитектурата или опит да се интегрират нови технологии на по-късен етап може да бъде много скъпо начинание.

Таблица 1 изброява различни бизнес изисквания и дава поглед доколко подходящи за реализиране на тези изисквания са двата типа клиенти. Таблицата не претендира за изчерпателност, но е добър справочник за най-често срещаните такива.

Таблица 1: Сравнение доколко двете архитектури удовлетворяват бизнес изискванията

Бизнес изискване	**Тънък клиент**	**Умен клиент**
Приложението трябва да е достъпно вън от организацията и да има разнообразна публика	Подходяща архитектура	Ограничено приложение
Богат потребителски интерфейс, drag&drop функционалност, redo / undo и т.н.	Ограничено, не трябва да се избира, ако се изисква типична десктоп функционалност.	Подходяща архитектура
Интранет приложение, интегриращо се и координиращо действията си с други приложения в организацията	Неподходящо решение. Характерни са трудности с интегрирането, ограничени възможности за интеграция.	Подходящо решение. Предлага гъвкави начини за интегриране.
Висока производителност базирана на отзивчив потребителски интерфейс	Неподходящо решение. При бавна връзка или тежки приложения потребителският интерфейс изчезва за определен период от време. Липсва възможност за извършване на други дейности с приложението докато се чака, ако не се отвори нова негова страница в отделен прозорец или таб.	Подходящо решение. Ако се изисква комуникация с уеб услуга, която е времеемка, потребителят не е лишен от възможност да извършва други дейности с приложението. Не се налага отваряне на нова инстанция

		на приложението.
Работа в офлайн режим	Неподходящо, не може да бъде реализирано.	Подходяща архитектура
Работа със специфичен хардуер	Неподходящо	Подходящо
Местоположение на бизнес логиката	Централизирана на уеб сървъра и други поддържащи сървъри	На клиента и на сървъра.
Актуалност на данните	Най-често се работи спрямо основната база данни.	За оптимизиране на връзката се кешира голяма част от информацията, което може да доведе до остаряването и. Това е проблем, с който трябва да се справят различните имплементации
Инсталиране, разпространение, обновяване и поддръжка	Лесно. Приложението се инсталира на централизиран сървър, браузър базиран достъп, доставя се само тази част от съдържанието, която е поискана от потребителя.	Лесно, базирано на централизиран сървър и ClickOnce технология. Може да се имплементира автоматично разпознаване на наличието на нова версия, автоматично актуализиране. Актуализират се само необходимите компоненти, не се сваля цялото приложение.
Преносимост	Да, но по-комплексните тънки клиенти могат да са браузър зависими.	Не са ограничени до десктоп и лаптоп компютри , могат да бъдат разработени и за устройства с малки размер (умни телефони и т.н.)
Използваемост на локалните хардуерни ресурси	Чрез COM компоненти, налични са проблеми при използване на специализирани хардуерни устройства.	Безпроблемно
Възможност за взаимодействие с други локални приложения	Не	Да
Поддръжка на многонишкова обработка	Не	Да
Възможност за оптимизиране	Липсва	Могат да

на работата при приложения с ниска широчина на канала		оптимизират мрежовия поток, чрез групиране на заявките към сървъра, за да използват оптимално ниската свързаност.

1.4.Изводи

Умните клиенти са нов тип архитектура - хибрид между положителните черти на десктоп и уеб приложенията. Те идват като естествена част от еволюционния процес на разработване на бизнес системи. В зависимост от бизнес целите могат да бъдат разработени за десктоп, офис пакет приложения или устройства с малки екрани.

Умните клиенти се налагат над уеб приложенията, в случаите когато трябва да има възможност за офлайн обработка, оптимизиране на канала за връзка и интегриране със специализиран софтуер и хардуер, но не са добро решение, когато целевите потребители са разнородна публика вън от организацията.

2. Умни клиенти. Кеширане на данните

Първостепенно значение за умните клиентите играят данните, тяхното съхранение и синхронизацията им между клиента и сървъра. Условно данните могат да бъдат разделени на данни за четене и краткосрочни данни. Основен подход за подобряване на производителността при умните клиенти е кеширането на данните Запазването на валидността, консистентността и защитеността на данните, е основен проблем, когато се използва кеширане. Затова по време на проектиране на умен клиент, този проблем трябва да се вземе под внимание.

Ако приложението предоставя възможност за промяна на данните локално, клиентските промени трябва да бъдат синхронизирани със сървъра в определен момент. Трябва да се прецени как системата ще се справя с възникналите конфликтите при синхронизацията между клиента и сървъра и как ще следят настъпилите промени, за да бъдат изпратени на сървъра.

Тази глава разглежда детайлно типовете данни, кеширането и нуждата от използване на кешираща инфраструктура, техники за зареждане на кеша, подходи за кеширане, подходи за синхронизация и следене промените в данните.

2.1. Типове данни

От гледна точка на предназначение им, условно може да разделим данните на два типа данни: само за четене и краткосрочни данни. Природата на данните е от значение, когато разработваме умен клиент, защото от типа данни зависи начина, по който ще реализираме кеширането и синхронизацията им.

2.1.1. Данни, предназначени за четене

Данните предназначени за четене са данни, реферирани от други данни. Чрез съхраняване и използване на реферирана информация на клиента, се намалява количеството информация, което циркулира между клиента и сървъра (изпращат се само ключове, не цялата информация). По този начин се

подобрява производителността на приложението и се прави възможна работата в несвързан режим. Наличието на реферираните данни дава възможност за ранната им проверка и по този начин увеличава използваемостта на приложението. Върху тези данни не се извършват операции на добавяне, обновяване и изтриване. Те не се променят на клиента, но те могат да бъдат променени на сървъра от администратор или контролно лице. Затова трябва да се определи стратегия за обновяването им. Има два подхода за обновяване: чрез започване на подмяната от странна на сървъра или клиента сам да извлече данните от сървъра. Тригер за настъпването на тези действия може да е определено действие на клиента или изтичането на предварително зададен период от време. Задачата по обновяване на данните е облекчена, защото не е необходимо да се следят промени настъпили на клиента.

2.1.2. Краткотрайните данни

Краткотрайните данни могат да бъдат променяни, както на клиента, така и на сървъра. Обикновено краткотрайните данни се променят директно или индиректно от потребителска входна информация или манипулации със самите данни. Независимо от мястото, където е настъпила промяната, тези данни трябва да бъдат синхронизирани между клиента и сървъра. Този тип данни се използват за добавяне на нова информация, промяна или изтриване на съществуваща информация.

Едно от предизвикателствата при работата с тях е, че те могат да бъдат променяни паралелно от много клиенти. Когато данните имат висок коефициент на непостоянност, се повишава вероятността те да са в конфликт една с друга.

Ако краткотрайните данни се променят от клиентското приложение, то трябва да следи тези промени. Не трябва да се приема, че краткотрайните данни са потвърдени, преди да е настъпила синхронизация със сървъра и да е приключил процесът по разрешаване на конфликтите Когато се разчита на непотвърдени данни или те се използват за други локални промени, трябва внимателно да се отчете как ще бъде гарантирана консистентността на данните, ако синхронизацията не успее.

2.1.3. Обобщение на основните характеристики

Таблицата по-долу обобщава основните характеристики на двата типа данни

Таблица 2: Основни характеристики на типовете данни използвани при умните клиенти

	Променят се на клиента	**Променят се на сървъра**	**Необходимо е следене на промените на клиента**	**Употреба**
Данни, предназначени само за четене	☒	☑	☒	Реферират се от други данни
Краткотрайни данни	☑	☑	☑	Добавяне, променяне, изтриване на информация

2.2. Кеширане на данни. Кешираща инфраструктура

Задължително условие при разработването на непостоянно свързани умни клиенти е кеширането. То осигурява необходимите данни на клиента и има потенциала да ускорява производителността в приложенията. Не всички данни обаче могат да бъдат кеширани, затова внимателно трябва да се разгледат различните сценарии и да се избере, както подходящ подход за кеширане, така и стратегия кои данни да бъдат кеширани. Необходимо е да се вземе под внимание как данните ще бъдат управлявани и какъв е контекстът, в който могат да бъдат използвани.

Кеширането не трябва да е хаотично. Затова е необходимо изграждане на инфраструктура, осигуряваща консистентност на кешираните данни, общ подход за кеширане и обновяване на кеша, както и начин за скриване на тази функционалност от модулите на по-високо ниво.

Има два основни механизма за кеширане. Кеширащата инфраструктура трябва да имплементира поне един от тях:

- Кеширане за кратък период – данните се кешират в операционната памет. От гледна точка на производителността, това е много добро решение, но при рестартиране на приложението данните трябва наново да се изтеглят от сървъра. Този механизъм реализиран

самостоятелно, е неприложим при умните клиенти, защото възпрепятства работата в несвързан режим.

- Дълготрайно кеширане – данните се кешират в перманентна среда като изолирани хранилища или локалната файлова система. Този тип кеширане позволява работа с приложението, когато достъп до сървъра е преустановен.
- Може да се избере комбинация от двата типа кеширане, за да се оптимизира производителността на приложението.

Към кеширането трябва да се подходи внимателно и от гледна точка на сигурността на данните. Независимо от избрания механизъм за кеширане, трябва да е подсигурено, че клиентът има достъп само до тази информация, която му е разрешено да използва. Кеширането на чувствителна информация изисква внимателно боравене с нея, за да се осигури сигурността и. Това налага криптиране на данните при преноса им към клиента и криптиране на хранилището за данни, разположени на клиента.

При проектирането на кешираща инфраструктурата трябва да се осигури механизъм за извличане на свежи данни, независимо от състоянието им в кеша. По този начин се гарантира, че заявките при изпълнението си не използват данни загубили давност.

Желателно е към данните да бъдат асоциирани метаданни. Метаданните дават възможност за интелигентно управление на информацията. Консуматор на метаданните се явява кеширащата инфраструктурата. Метаданните могат да бъдат използвани, за да се специфицират ограничения или да се определи поведение свързано с употребата и управлението на данни. Например:

- Времеви ограничения – тези ограничения специфицират периодът, в който кешираните данни могат да бъдат използвани. Проверката за валидността на данните, се реализира на базата на времеви маркер или номер на версия. По този начин информацията може да бъде идентифицирана като нуждаеща се от обновяване. Когато данните загубят валидност или давност, те могат да бъдат изчиствани от кеша или обновени автоматично. В някои случаи е подходящо да се позволи на потребителя да използва остаряла референтна информация, а актуализирането и да се извърши по време на синхронизацията.

- Географски ограничения – определена информация може да бъде уместна само за определен регион. Например наличие на различни ценови листи в различните географски региони. Кеш инфраструктурата може да бъде използвана да извлича и съхранява информацията на база местоположението на клиента.
- Изисквания към сигурността – Данните могат да бъдат криптирани, за да се гарантира, че само потребителят, за който са предназначени, има достъп до тях. В този случай данните се доставят вече криптирани, а потребителя трябва да предостави акредитация, за да се позволи дешифрирането в кеширащата инфраструктура.
- Бизнес правила – метаданните могат да дефинират бизнес правила, които определят как кешираните данни трябва да бъдат използвани. Например, кеширащата инфраструктурата може да взима под внимание ролята на потребителя, за да определи до какви данни трябва да му се даде достъп.

С помощта на метаданните, кеширащата инфраструктура може успешно да се справи с проблеми в данните, тъй че приложението да не се занимава с тази задача. Метаданните могат да се изпращат в самите данни, или може да се използва out-of-band механизъм. Точният механизъм, който се използва за транспортиране на метаданните към клиента, зависи от това как приложението комуникира с мрежови услуги. Когато се използват уеб услуги, използването на soap хедъра, за да се предадат метаданните, е добро решение.

Кеширащата инфраструктура трябва да отчита типа на данните Реферираната информация не трябва да бъде синхронизирана със сървъра. Тя само трябва да бъде обновявана през определени интервали от време. Краткотрайната информация се променя и на сървъра, и на клиента. Всяка промяна направена на клиента, трябва да бъде синхронизирана със сървъра по някое време. Подходящо решение за боравенето с тези два типа данни е да се създадат два кеша с различна употреба.

Приложенията трябва да бъдат проектирани така, че да различават успешно синхронизираната на сървъра информация от тази, която подлежи на изменение. Чрез използването на подходяща кеширащa инфраструктура, може да се следи подлежащата на изменение и потвърждение информация. По този

начин се улеснява откриването и разрешаването на конфликти в данните. Действията на потребителя също могат да бъдат ограничени, така че той да няма възможност да взема важни решения на базата на подлежаща на изменение краткотрайна информация.

2.3.Cashing Application Block – готов кеширащ приложен блок

Разработчиците могат да разработят собствена кеширащa инфраструктура или да използват вече готова такава. Microsoft предоставя Cashing Application Block – кеширащ приложен блок. Той е част от Microsoft Enterprise Library (Microsoft, Enterprise Library 3.0 – April 2007, 2007). Производителността му е оптимизирана и е нишково обезопасен. Той може да се надгражда, чрез дефиниране на допълнителни политики за загуба на валидност и допълнителни хранилища за данни.

Подразбраната опция при използване на приложния блок е нулево хранилище (null backing store). При него данните се съхраняват в операционната памет, т.е. не се предоставя възможност за съхранение на данните между рестартиранията на приложението. Това хранилище е подходящо, в ситуации, в които е необходимо обновяване на кеша от оригиналния източник при всяко ново стартиране на приложението (Microsoft Corporation, 2006).

Друга опция, която може да се използва, е изолирано хранилище. То е дълготрайно хранилище и е препоръчително да се използва в следните ситуации:

- Необходимо е дълготрайно хранилище и броят на потребителите е малък
- Когато натоварването от използването на база данни е недопустимо
- Няма възможност за използване на база данни

Изолираното хранилище е подходящо за умни клиенти и сървърни приложения, при които всяка приложна област има свой собствен кеш. Ако се използва вариант на изолирано хранилище, което зависи от текущия потребител, сървърното приложение трябва да имперсонира потребителя, правещ заявка към приложението (Microsoft Corporation, 2006).

За хранилище може да се използва и база от данни. Блокът работи с Microsoft SQL Server, но може да се използва и с други бази. Ако блокът се използва с други бази, трябва да се напише допълнителен код. Този вариант е подходящ за умни клиенти и за сървърни приложения, където всяка приложна област има свой собствен кеш и има възможност на клиента за работа с база данни.

Приложният блок не поддържа опция да криптира данни, кеширани в операционната памет, но поддържа симетрично криптиране на данните за останалите типове хранилища. Ако злонамерен потребител открие начин да компрометира системата и да достъпи паметта, използвана от процеса на приложението, той ще има достъп до информацията, кеширана в операционната памет. Ако това е реална заплаха за приложението, то чувствителна информация като номера на кредитни карти, пароли и т.н., не трябва да се съхранява в кеша.

Приложният блок за кеширане предоставя четири политики за загуба на давност.

- Абсолютна - записа загубва давност в точно определено време.
- Отместваща се - записа загубва давност след специфично време което е изтекло след последния достъп до записа. Подразбраното време е 2 минути.
- Разширен формат- детайлно описва, кога записа губи давност. Например, неделя в 10.00 Разширените формати са изброени в ExtendedFormat.cs файла.
- Файлова зависимост. –записа загубва давност при настъпило изменение в специфичен файл.

Първите три са времево базирани и се използват за опресняване на непостоянни кеш записи.

Загубата на давност в блока е реализирана чрез процес от две части. Първата част се нарича маркиране, втората е известна като чистене. Процесът е разделен на отделни задачи, за да се предотвратят конфликтите, които могат да възникнат, ако приложението използва кеширани записи, докато BackgroundScheduler обекта се опитва да ги накара да загубят давност.

По време на маркирането, BackgroundScheduler прави копие на кеш таблицата и проверява всеки кеширан запис, за да види дали може да загуби

давност. Докато прави това той заключва записа. Ако записът е подходящ, BackgroundScheduler обектът го маркира като такъв в кеш обекта.

По време на чистенето, BackgroundScheduler отново проверява всеки маркиран кеш обект дали е бил достъпван след маркирането. Ако е бил използван, остава в кеша, в противен случай загубва давност и се премахва от кеша. В повечето случаи разработчиците ще искат да отреагират, когато данните загубват давност. Това може да се реализира като се прихваща Windows Management Instrumentation (WMI) съобщение или чрез използване на предефиниран вариант на Add метода, позволяващ им да специфицират, че приложението ще получи обратно извикване.

Caching Application Block поддържа процес на прочистване (scavenging), който също се реализира от BackgroundScheduler обекта. За разлика от процеса по загуба на давност, процесът по прочистване извършва маркиране и изчистване при едно минаване. Всеки път при добавяне на нов обект, се проверява дали количеството обекти в кеша не е достигнало пределния лимит. За да се специфицира лимита, се използва Configuration Console. Чрез този инструмент може да се зададе броя на обектите, които ще бъдат премахнати от кеша след като започне прочистването. Когато обект бъде добавен към кеша, може да му бъде зададена една от следните настройки: Low, Normal, High, или Not Removable. BackgroundScheduler обекта определя кои обекти трябва да бъдат изчистени като прави а първостепенно сортиране базирано на приоритета и второстепенно сортиране, базирано на последния достъп до обекта. Например, един обект с Low приоритет, който току що е бил използван, ще бъде премахнат преди обект със High приоритет, който не е използван 3 години. Подразбраната стойност, която се присвоява, е Normal. NotRemoveable приоритета се използва, когато е необходимо обекта да остане в кеша докато не загуби давност, а не принудително в процеса на прочистване. Кешът не трябва да бъде използван като единствено място, на което данните съществуват. Кешът трябва да се използва за подобряване на производителността, не трябва да се използва за дълготрайно хранилище.

Кеширащият приложен блок е проектиран, за да бъде използван от многообразни приложения и да бъде кеш с общо предназначение. Точките на разширение позволяват приложният блок да бъде адаптиран към изискванията на отделните приложения. Към приложния блок могат да се добавят нови

характеристики, чрез модифициране на програмния код(инсталацията му съдържа програмния му кода и библиотечни файлове)

2.4.Зареждане на данните

Данните, които се използват в умните клиенти, трябва да са налични в кеша. Съществуват следните подходи за тяхното зареждане:

- Предварително(Proactive) зареждане – при този метод се извличат всички необходими състояния от източника на данни и се кешират за целия цикъл на живот на приложението или процеса.
- Противодействащо(Reactive) зареждане – този метод извлича данните, при поискване и ги кешира за бъдеща употреба

2.4.1. Предварително кеширане

Данните се зареждат предварително, обикновено при стартиране на приложението или процеса. Ако не се използва внимателно, този подход може да доведе до бавно стартиране на системата. При имплементиране е желателно да се заредят възможно най-много състояния, за да не се получи ситуация, в която са заредени само част от необходимите състояния и да се правят чести заявки за останалите на по- късен етап. Тъй като зареждането на големи обеми от данни е бавен процес, трябва да се използва асинхронен програмен модел, т.е. в нишка на заден фон.

Предварителното кеширане се препоръчва в следните ситуации:

- Използват се статични или семантични състояния на данните, които имат точно определен период на актуализиране. Ако методът се използва с данни с неизвестен период на обновяване, състоянието им може да е загубило валидност преди да са били използвани.
- Използват се състояния с известен цикъл на живот.
- Използват се състояния с предварително известен размер. Когато предварителното кеширане се използва с неизвестни по размер данни, те могат да заемат повече ресурси от наличните и системата да изпадне в неустойчиво състояние.
- При проблем с ресурси (например бавен достъп до базата данни, проблемна мрежа или ненадеждни уеб услуги) е препоръчително

предварително да се кешират всички състояния и да се използват възможно най-дълго време.

Предимство на предварителното кеширане е, че гарантира наличността на данните в кеша. На теория не трябва да се проверява дали състоянието е налично. Въпреки това, проверката трябва да се изпълни, тъй като кеша може да е почистен. Производителността на приложението се подобрява, защото кеш операциите се оптимизират, когато информацията се зарежда предварително. Времето за отговор на приложението намалява, тъй като цялата информация е вече кеширана.

Недостатък на предварителното кеширане е, че то не води до най-оптимизираната система. Голяма част от състоянията се кешират, независимо дали са необходими или не и заемат значителна част от ресурсите. Предварителното кеширане може да доведе до усложнена имплементация в сравнение с традиционни техники за кеширане т.е. всеки обект да се извлича синхронно в добре познат програмен поток. Използването на предварителното кеширане предполага работа с няколко нишки. Появяват се трудности при синхронизация на работните нишки с главната нишка на приложението. Задължително е да се следи състоянието на нишките и да се обработват изключенията на базата на асинхронния програмен модел.

2.4.2. Противодействащо кеширане

Предимство на противодействащото кеширане е, че се съхраняват само необходимите данни, а това води до рационално използване на системните ресурси. Недостатък на подхода е намалената производителност при първо поискване на всяко парче информация, защото то се зарежда от източника, а не от кеша. Друг недостатък е условната логика, която трябва да се добави, за да се проверява дали тези данни не присъстват вече в кеша.

Противодействащото кеширане се препоръчва в ситуации, имащи следните особености:

- Трябва да се извлекат много състояния, а наличните ресурси не са достатъчни, за да се кешират всички състояния.

- Използват се надеждни ресурси с малко време за отговор, като бързи бази данни, надеждни мрежи и уеб услуги, които нямат да засегнат стабилността и производителността на приложението.
- Необходимо е да се кешира информация, която няма да е налична при стартиране на приложението.

2.4.3. Сравнение между предварителното и противодействащо кеширане

Таблица 3 обобщава разликите между двата вида кеширане. Удачно е използването на двата вида кеширане в дадено приложение за различни цели. Предварително могат да се кешират данните само за четене, тъй като те се използват често и по-голямата част от състоянията им ще бъдат използвани. Противодействащото кеширане е по-подходящо за краткосрочни данни, защото данните могат да бъдат променени в промеждутъка между момента на кеширане и използване

Таблица 3: Сравнение между предварителното и противодействащото кеширане

	Противодействащо кеширане	**Предварително кеширане**
Време за зареждане на приложението	В повечето случаи не го афектира	Може да доведе до бавно стартиране на системата, задължително е използване на асинхронен модел на работа
Използване на системните ресурси	Рационално	Заемат се повече ресурси, отколкото са необходими
Време за първи достъп до данните, които са необходими	Едно и също за всички данни, при първо поискване се прави заявка към източника, за да се зареди кеша	Бързо за предварително кеширани данни, за останалите времето е същото, както при противодействащото кеширане
Тип данни, за които е подходящо	Краткосрочни	Реферирани

2.5. Съгласуване на данни

Както бе споменато по-рано, един от основните проблеми при използването на умните клиенти, е синхронизацията. Данните, налични на сървъра, могат да бъдат променени преди клиентите да са успели да се

синхронизират с него. В този контекст особено внимание придобиват механизмите, осигуряващи съгласуване и реализиращи подходящо разрешаване на конфликти. Целта им е резултатната информация да е консистентна и коректна. Подходите за съгласуване на данни са два: песимистичен и оптимистичен.

2.5.1. Песимистично съгласуване

При песимистичното съгласуване клиентът заключва данните докато не завърши със собствените си промени. В този случай, ако друг клиент се опита да модифицира данните, опитът пропада или се блокира, докато собственикът на заключването не освободи данните.

Песимистичното съгласуване може да бъде много проблемно, защото един потребител или клиент може да държи значителен период от време заключени данни. Заключването може да ограничи използването на важни ресурси, като редове в базата данни и файлове, силно афектирайки работоспособността на приложението. Този тип съгласуване не може да се използва, когато клиентът работи в офлайн режим, защото в този случай няма възможност да се извърши заключване на данните.

В редки случаи песимистичното съгласуване е подходящо. Например ако е необходим пълен контрол върху измененията на важни ресурси.

2.5.2. Оптимистично съгласуване

При оптимистичното съгласуване данните не се заключват. Заявката, която се изпраща към сървъра, съдържа оригиналните и променените данни. Оригиналните данни се сравняват с информацията в базата. Ако се различават не се извършва обновяване, а се получава песимистична грешка и заявката се отказва. Ако оригиналните данни съвпадат с наличните в базата, се извършва актуализиране. За да се оптимизира този процес може да се използва времеви маркер или брояч на обновявания. Вместо да се изпраща оригиналната информация се изпраща и проверява само времевия маркер (брояча).

Оптимистичното съгласуване предоставя добър механизъм за обновяване на основните данни. То е подходящо, когато обновяванията са по-редки, отколкото операцията четене. В тези ситуации рискът от оптимистична грешка е

допустим. Оптимистичното съгласуване не е подходящо в случаите, когато данните се менят често, защото оптимистичните обновявания ще се провалят по-често. В повечето случаи, включително, тези в които клиентите работят в несвързан режим, оптимистичното съгласуване е правилният подход, защото позволява множество клиенти да работят с данните едновременно, без необходимост от заключване.

2.5.3. Съгласуване на данните с помощта на ADO.NET

Dataset обектите, които се предоставят от ADO.NET, са с практическо приложение при умните клиенти. Те предоставят функционалност за работа в несвързан режим и могат да следят локалните промени в данните. Във всеки ред от таблиците на Dataset обекта, заедно с текущото състояние на данните, се пази и оригиналната им версия, улесняваща идентифицирането на изменения. Това е от полза при синхронизиране на данните със сървъра и разрешаване на конфликти. Допълнително улеснение при dataset обектите е това, че те предоставят методи за обединяване на данните от различни ресурси.

Производителността е основен проблем при преноса на данни. Обикновено Dataset обектите съдържат голямо количество данни, което трябва да бъде пренесено през мрежата. За тези случаи ADO.NET предоставя GetChanges метода, който връща само променените данни По този начин може да се пренесе не цялата информация, а тази, която е променена. Този метод намира приложение при умни клиент приложения, които трябва да работят в несвързан режим. Когато приложението стане достъпно в онлайн режим, този метод може да се използва, за да се определи коя информация се е променила и е необходимо да бъде синхронизирана.

2.6.Изводи

Ако кеширането бъде използвано без централизиран подход, измененията в него и опити за оптимизиране, биха били много трудоемки и с множество проблеми. Затова е необходимо изграждането на кешираща инфраструктура, която да изолира проблемите на кеширането от по-горните слоеве, или да се използва вече готова такава. Тя трябва да поддържа техники за

съхранение на данните, стратегии за загуба на давност и стратегии за изчистване на кеша.

Данните могат да бъдат сравнително постоянни във времето (данни използвани само за четене) или краткотрайни. Когато се кешира информацията при създаването на кеша, трябва да се отчетат особеностите в поведението им. Подходящо за краткотрайните данни е противодействащо кеширане за кратък период, а за данните само за четене, предварително дълготрайно кеширане.

Краткотрайните данни се променят често и могат да бъдат променени на клиента и на сървъра. Когато се променят на клиента, подходящо е оптимистичното кеширане, особено в случаите, когато се работи офлайн. Промяната на сървъра е уместно да се компенсира чрез стратегията за краткосрочно кеширане. Периодът на краткосрочното кеширане трябва да отчита вероятното време за настъпване на изменения в данните на сървъра.

3. Непостоянно свързани умни клиенти

„Едно приложение е непостоянно свързано,
ако има случаи в които не може да взаимодейства със
услугите или данните в мрежата в разумно време."
(Microsoft Patterns & Practices, 2004, стр. 51).

Всекидневният ни живот става все по-тясно обвързан с Интернет или интранет мрежи. Започнахме да разчитаме на тази връзка и на нейната наличност 100% от времето. За съжаление, има редица проблеми, свързани с предоставянето на тези услуги. Това са: голяма латентност, проблеми с широчината на канала, недостъпност на сървърите поради месечни профилактики и т.н. В тези случаи възниква конфликт между потребителските интереси и постоянно свързаните приложения. За да се разреши този проблем, приложенията трябва да бъдат оборудвани с възможност да работят в не свързан режим.

Разработването на непостоянно свързани приложения, е подходящо в много разпространени ситуации. Голяма част от офлайн сценариите, включват потребители, категорично разкачени от мрежата и работещи без връзка. Например:

- Застрахователен агент трябва да създаде нова застрахователна полица, когато е извън офиса. Изисква се той да въведе цялата подходяща информация, да калкулира вноските и детайлите, но без да има възможност да се свърже със системите в офиса.
- На пътуващ продавач му се налага да въведе голяма поръчка докато е при клиент и няма достъп до вътрешна за компанията мрежа. Налага му се да се консултира с ценовата листа или с информационния каталог, за да въведе цялата информация за поръчката и да даде оценка за доставката и намаленията.

Други офлайн сценарии включват връзка, която прекъсва или е с ниско качество, например:

- Свързаността между центровете за поддръжка по света и главния офис може да не е на необходимото ниво, за да позволи онлайн

употреба през цялото време. Затова приложението трябва да предлага офлайн възможности.

Ако се даде възможност на потребителите да използват дадено приложение, когато са офлайн, ефикасността и работоспособността му се увеличават. В това се корени едно от първоначалните предимства на умните клиенти пред уеб базираните приложения. Те могат да позволят на потребителя да продължи да работи, независимо от наличието на връзка. Промените, направени върху данните, се съхраняват на клиента, а при преминаване в онлайн режим се синхронизират със сървъра във фонов процес. Приложението може да бъде в офлайн режим дълъг период от време - дни, дори седмици. За да се даде на едно приложение пълният набор от възможности за работа в офлайн режим, трябва да му се осигури инфраструктура, предоставяща на клиентите възможност да работят, когато нямат връзка с мрежовите ресурси. Тя задължително трябва да включва кеширане на данни, тъй че цялата необходима информация да е на клиента, и да съхранява детайли от работата на потребителя.

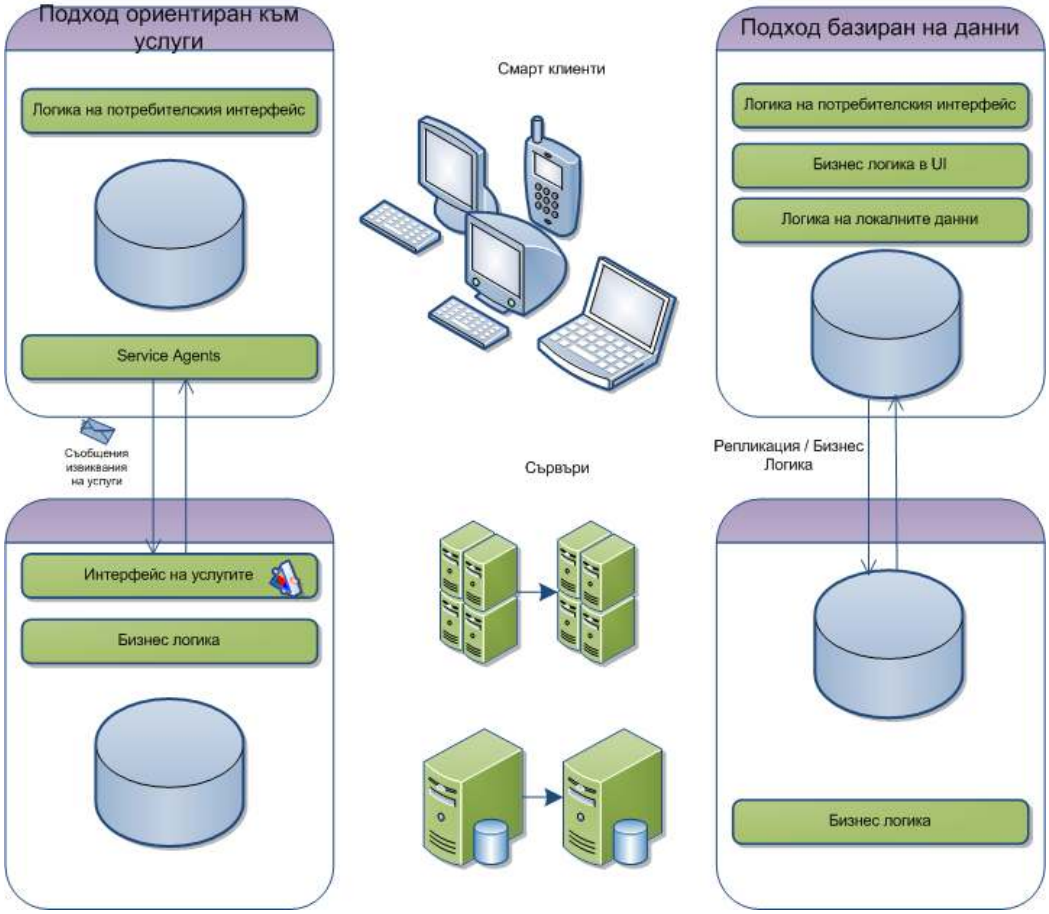
Точните характеристики, които са необходими на едно приложение, за да поддържа непостоянно свързани операции, зависят от свързаността, операционната среда и функционалността, която потребителят очаква, когато е онлайн или офлайн. Всички умни клиенти трябва да осигуряват някакъв тип офлайн функционалност на потребителя, дори ако тя е крайно ограничена. При проектирането и изграждането на офлайн приложения, стремежът е да се избегне генерирането на съобщения за грешка на клиента, поради това, че сървърът не е достъпен.

Тази глава разглежда въпросите, които софтуерните архитекти си задават, и отговорите, които търсят по време на проектирането на приложения с офлайн възможности. Прави се обзор на различните стратегии за дизайн на офлайн приложения и се обсъждат в детайли проектантските решения.

3.1.Непостоянно свързани проектантски стратегии

Има два основни подхода за проектиране на непостоянно свързани умни клиенти: базиран на данни (data centric) и ориентиран към услуги (service-oriented).

Приложенията използващи подхода базиран на данни използват релационна база данни (RDBMS), инсталирана локално на клиента. Благодарение на вградените и възможности, те предават локално променените данни обратно на сървъра при процеса на синхронизация. Базата данни се грижи за откриване и разрешаване на конфликтите. Приложенията използващи подхода ориентиран към услуги, съхраняват информацията в съобщения и ги подреждат в опашки, докато клиентът е офлайн. След като връзката се възобнови, съобщенията в опашката се изпращат на сървъра за обработка.



Фигура 1: Сравнение между ориентирания към услуги и базирания на данни подход

3.1.1. Подход базиран на данни

При този подход, сървърът публикува данните, а клиентът се абонира за тези, които са му необходими. Ако клиентът работи в офлайн режим, той променя данните в локалното си хранилище за данни. Когато клиентът премине

отново онлайн, хранилището за данни разпространява промените от клиента обратно към сървъра, а промените направени на сървъра се разпространяват към клиента. Конфликтите възникнали по време на сливането се разрешават на базата на *правила за разрешаване на конфликти,* специфицирани на клиента или сървъра, в съответствие с общите условия, дефинирани от бизнес анализатора.

Процесът на сливане на промените между клиента и сървъра е известен като сливаща репликация (merge replication). Промените могат да настъпят автономно на клиента или на сървъра. Затова когато сливането се извърши, клиентите и сървъра, подписани за това събитие, използват данните, които се съдържат при публикуващия данни, а не ги обменят чрез ACID (Atomic, consistent, isolate, durable) транзакции.

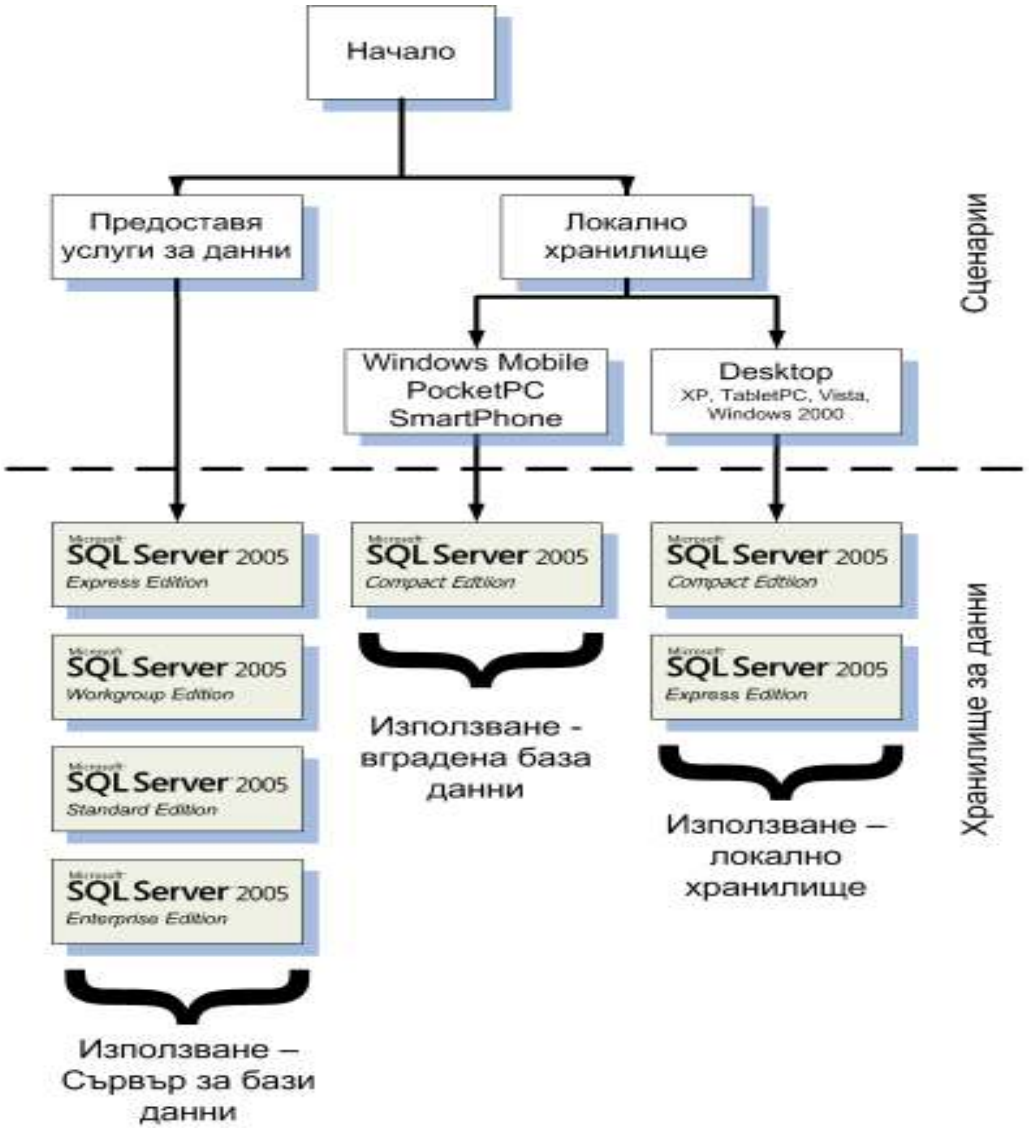
Главното предимство на този подход е, че целият код, следящ промените, се съдържа в релационната база данни. Обикновено това е код за откриване на грешките на ниво колона и ред в базата данни, валидация на данните и ограниченията. Приложението не трябва да добавя собствен код за проследяване на промени, за откриване и разрешаване на конфликти. То трябва да знае сливащо-репликационната схема, за да оптимизира работата си при възникване на конфликт в данните или обновяванията.

В модела базиран на данни, базата данни извършва синхронизацията. За това не трябва да се имплементира собствен механизъм за синхронизация на данни. Потребителят дефинира кои таблици изискват синхронизиране на данните и базата данни позволява на инфраструктурата си да следи промените, да открива и разрешава проблеми. Може да се разшири инфраструктурата на базата данни, за да поддържа специфично разрешаване на конфликти или анулиране чрез допълнително разработени resolvers, които използват COM обекти или T-SQL съхранени процедури.

Доскоро нуждата от локална база данни на клиента означаваше, че този подход не е подходящ за определени ситуации. Например при наличие на изискване приложението да се изпълнява на умни телефони, при необходимост механизмът за разпространение да е облекчен, при изискване приложението да се инсталира на потребители, които не са администратори. Тези ограничения се измениха с излизането на Microsoft SQL Compact, който работи върху умни телефони. Той може да се инсталира чрез ClickOnce технологията от

потребители, без администраторски права. Затова този вариант вече не би трябвало да се отхвърля в тези случаи, а да се разгледа наравно с ориентирания към услуги подход.

Microsoft предлагат голяма вариация от SQL версии. За да се избере коректната, трябва да се разгледат различните варианти. За реализиране на базата данни на сървъра е подходящо да се използва Express, Workgroup, Standard или Enterprise Edition. За локално хранилище на умният клиент е подходящо да се използва Compact или Express Edition. На фигура 2 е показано дърво обобщаващо решенията за избора на база данни (Lasker, 2006).



Фигура 2: Дърво на решенията каква Microsoft база данни да се използва при различни сценарии.

Microsoft препоръчват за локално хранилище SQL Server Compact Edition. Ако приложението изисква поддръжка на услуги за данни, тогава трябва

да се избере SQL Server Express Edition. Ласкър (Lasker, 2006) систематизира разликите между двете версии. Те са показани в таблица 4. Таблицата може да послужи на софтуерните архитекти като указание коя база да бъде избрана при определена имплементация.

Таблица 4: Сравнение между SQL Compact и SQL Express Edition

Характеристики	**SQL Server Compact Edition**	**SQL Server Express Edition**
Разгръщане/Инсталационни характеристики		
Размер на инсталацията	1.7mb размер на пакета 1.8mb размер върху диска	53.8mb размер на пакета ~197mb размер върху диска
ClickOnce разгръщане	☑	☑
Може да се инсталира отделно, вградено в приложението	☑	☒
Инсталация без администраторски права	☑	☒
Работи на Windows Mobile платформи	☑	☒
Централизирана инсталация чрез MSI	☑	☒
Работи в процеса на приложението	☑	☒
64-bit поддръжка	☒ Версия 3.1 ☑ Вградена 64 в следваща версия	☑ Windows on Windows (WOW)
Работи като услуга	☒ в процеса на приложението	☑ N + 1
Характеристики на файла за данни		
Формат на файла	Единичен файл	Множество файлове
Файлово хранилище за данни върху мрежов ресурс	☑	☒
Поддръжка за различни файлови разширения	☑	☒
Поддържан размер на базата данни	4GB	4GB
XML хранилище	☑ съхранява се като текст	☑
Програмируемост		
T-SQL Общи характеристики на заявките	☑	☑
Procedural T-SQL Select Case, If, характеристики	☒	☑
Remote Data Access (RDA)	☑	☒
ADO.NET Sync Framework	☑ Поддръжка с Visual Studio Orcas	☒ Планирана поддръжка с бъдеща версия
Подписване за сливаща репликация	☑	☑
Прости транзакции	☑	☑
Разпределени транзакции	☒	☑
Вграден XML, XQuery/QPath	☒	☑
Съхранени процедури, изгледи, тригери	☒	☑
Базирана на роля сигурност	☒	☑

Брой паралелни връзки	256	Неограничен

Недостатък на подхода базиран на данни е тясната свързаност между сървърната и клиентската база данни. При промяна на сървърната схема, директно ще се наруши работата на клиента. Това прави трудно управлението на промени в схемите на базите от данни.

При голям брой клиенти, е необходимо да се оптимизира разпространението на набори от данни (datasets). Сливащата репликация поддържа динамично филтриране, което позволява на администратора да дефинира офлайн набор от данни, който да бъде разпространен. За да се намали количеството данни, които се изпращат между клиента и сървъра, и за да се намали честотата на конфликтите, трябва да се използва филтриращи механизми осигурени от базата данни.

Ползата от използването на локална база данни е, че позволява локално управление на данните. Тя може да се използва за разпространение на данните обратно на сървъра и да помогне в извършването на синхронизационния процес.

Този подход е удачен в следните случаи:

- Инсталирането на база данни на клиента не е проблем
- Приложението може да функционира в двуслойна среда
- Ако е подходящо тясното свързване между клиента и сървъра чрез схема дефинициите на данните и комуникационния протокол
- Нужно е вградено (вече изградено) проследяване на промените в данните и готов синхронизационен процес
- Разчита се на база данни за разрешаване на конфликтите и за минимизиране на необходимия код за разрешаване на конфликти

3.1.2. Подход ориентиран към услуги

При този подход клиентите си взаимодействат с множество услуги. Предимство на тази стратегия е, че не е необходима релационна база на клиента. Затова подходът може да бъде приложен успешно към голям брой клиентски устройства, включително тези с малки процесорни възможности като мобилните телефони.

Подходът ориентиран към услуги е в частност подходящ, когато приложението трябва да работи в интернет или екстранет среда. Благодарение на слабата свързаност, могат да се използват различни информационни схеми на клиента и на сървъра и данните да се трансформират на клиента. Клиентът и сървърът нямат нужда да знаят за съществуването на другия. Те могат да се обновяват независимо. Най–големият недостатък на този подход е, че трябва да се напише повече инфраструктурен код, за да се облекчи съхраняването и изпращането на съобщения, както и код за детекция на това дали приложението е офлайн или онлайн. Това може да даде по-голяма гъвкавост при дизайна, но често значи повече работа при създаването на офлайн клиенти.

Подходът ориентиран към услуги е по-подходящ при умни клиенти, които трябва да работят с различни услуги. Защото потокът от съобщения е обвит и транспортният слой може да се мени без да афектира съдържанието на съобщенията. Например съобщение, генерирано първоначално за уеб услуга, лесно може да бъде изпратено на услуга консумираща Message Queuing съобщения.

Подходът ориентиран към услуги трябва да се използва в следните случаи:

- При изграждане на несвързани клиент и сървър, с различни версии и инсталации
- При нужда от повече контрол и гъвкавост при разрешаването на конфликтни данни
- Има възможност за изграждане на приложението с архитектура ориентирана към услуги
- Налична е специфична бизнес функционалност (специфични бизнес правила и обработка)
- Нужда от контрол върху схемата на данните, съхранени на клиента и гъвкавост, когато схемата на клиентските данни се различава от тази на сървъра.
- Приложението общува с различни услуги.
- Необходима е специфична схема на сигурност.
- Приложението работи в Интернет или Екстранет среда

3.1.3. Дизайн на непостоянно свързани умни клиенти при използване на подхода, ориентиран към услуги

Основните аспекти при дизайна на непостоянно свързани умни клиенти, при използването на подхода ориентиран към услуги, са

- Асинхронната комуникация
- Минимизиране на комплексните мрежови комуникации
- Възможност за кеширане
- Управление на свързаността
- Дизайн на съхрани и изпрати механизъм
- Управление на конфликтите при данните и бизнес правилата
- Взаимодействие със създай, прочети, обнови, изтрий (CRUD) подобни уеб услуги
- Използване на подход, базиран на задачи
- Управление на зависимостите

Тук ще разгледаме в детайли всеки един от тях и как той влия върху разработка на умни клиенти.

3.1.3.1. Асинхронна комуникация

Умният клиент не трябва да ограничава работата на потребителя. Приложението трябва да е отзивчиво във всеки един момент от време. Някои заявки към сървъра обаче изискват известно време, за да се изпълнят. В такива случаи потребителският интерфейс не трябва да замръзва, а трябва да продължи да отговаря на командите на потребителя. Асинхронната комуникация дава решение на този проблем и това трябва да се отчете при проектирането на нови умни клиенти. Когато липсва връзка с необходимата уеб услуга, асинхронната комуникация позволява заявките да се съхраняват в опашка и да се изпълнят на по-късен етап. Аналог на асинхронната комуникация е синхронна комуникация, която работи във фонов режим. Но това не е добър вариант за непостоянно свързани клиенти и трябва да се избягва. Асинхронната комуникация трябва да бъде предпочетена пред синхронната. За да работят като умни клиенти, готовите приложения трябва да бъдат препроектирани така, че синхронната комуникация, която използват, да се замени с асинхронен модел на

комуникация. Когато при препроектирането е необходимо да се направят минимални промени, може да се използва подобие на синхронната комуникация върху асинхронна структура, известна като sync –on-async модел.

Не е необходимо да се използва синхронен модел за онлайн режим и асинхронен за офлайн. Асинхронното поведение е подходящо и за двата режима. Когато приложението е в онлайн режим, асинхронните заявките се изпълняват почти в реално време.

3.1.3.2. Минимизиране на сложните мрежови взаимодействия

Независимо от режима, в който работят, приложенията не могат да преценят дали дадена заявката ще успее или не. Затова частично свързаните умни клиенти трябва да намалят до минимум сложните взаимодействия с услуги, намиращи се в мрежата.

За да се позволи офлайн работа на приложението, трябва да се направят определени допускания за успеха на заявката и за измененията, които могат да настъпят в данните. Следенето на тези допускания между заявките е сложно. За да се облекчи този товар, умният клиент трябва така да се проектира, че да работи, доколкото е възможно, с прости мрежови заявки.

Обикновено, заявките, които не връщат данни (fire and forget requests), не са проблем за непостоянно свързаните клиенти. Приложението може да съхрани заявката и да я изпрати след свързване. Когато приложението е офлайн, то не знае дали заявката е успяла, затова то трябва да приеме, че тя е успяла. Допускането може да повлияе на последващата обработка.

Ако заявката връща данни, които са необходими на приложението, за да продължи работата си, то трябва да използва пробни или фиктивни стойности без данни. В тази ситуация, приложението трябва да е проектирано да следи пробните данни и да ги потвърждава. Потребителският интерфейс трябва да е проектиран така, че да осведомява потребителя за данни, които са пробни и висящи.

В ситуации, в които потребителят извършва набор от дискретни задачи, докато е офлайн, приложението трябва да позволи на всяка единица работа да успее или да се провали самостоятелно(когато потребителят въвежда поръчки,

трябва да се подсигури, че успехът на отделните поръчки е независим един от друг).

Относително лесно е да се осигури, че няма зависимости между единиците работа, когато приложението трябва да направи само една заявка за единица работа. Това позволява приложението да следи висящите заявки и да ги обработва, когато се върне онлайн.

3.1.3.3. Добавяне на възможност за кеширане на данните

Приложението трябва да подсигури данните, тъй че ако мине в офлайн режим, потребителят да продължи да работи. В някои случаи кеширането се извършва за оптимизация на производителността, но тук трябва да се осигури кеширане, за да са достъпни данните в офлайн режим. Когато приложението е офлайн, може да не се трият остарелите данни, вместо това те могат да се използват, за да се позволи на потребителя да продължи работа. В други случаи, за нуждите на приложението е необходимо данните да се изтрият от кеша, за да се предотврати използването им и да се предотвратят проблеми, които биха настъпили по-късно. В тези случаи, приложението може да забрани използването на тази функционалност, докато не се придобият нови данни чрез процеса на синхронизация. Освежаването на информацията в кеша може да настъпи по много начини, в зависимост от поведението и функционалността на приложението. В някои случаи това се извършва автоматично, когато тя загуби валидност, периодично на базата на някакво разписание, когато се извършва синхронизационна операция или когато сървъра промени данните и информира клиента. Други приложения могат да предлагат възможности ръчно да се определи кои данни да се кешират, позволявайки на потребителя да преглежда или работи в офлайн режим.

Реферираната информация се променя рядко, а приложението работи с значително количество такава информация. Тези данни лесно могат да бъдат кеширани на клиента, но понякога те подлежат на промяна. Затова трябва да бъде предварително имплементиран механизъм за отразяване на тези изменения.

Имате две опции това да бъде направено: данните да се изтеглят(pull) или лансират(push). При лансиране на данни сървърът предварително информира

клиента и му изпраща изменените данни. При подхода базиран на данни, това става чрез репликация на обновените данни в клиентското хранилище. При подхода ориентиран към услуги, това може да бъде реализирано чрез съобщение, съдържащо обновените данни. В този случай клиентът трябва да имплементира крайна точка, към която сървърът да се свързва.

При изтеглянето на данни клиентът изисква от сървъра обновената информация. Клиентът може да разбере за наличието на нова информация като проверява данните на сървъра през определен период или като проверява метаданните, асоциирани с първоначалните данни. Ако данните са налични предварително, но не са все още валидни, клиентът може предварително да ги издърпа от сървъра и да ги използва, когато те станат валидни.

3.1.3.4. Управление на връзката

Трябва внимателно да се оцени средата и свързаността, при които клиентът ще работи. Някои приложения трябва да работят продължителен период от време в офлайн режим. Други очакват свързаност през цялото време, но е необходимо се справят с временна несвързаност. Някои приложения трябва да осигуряват само определен набор от функции, когато са в офлайн режим, а други почти пълната си функционалност. Някои отчасти свързани сценарии включват пълна несвързаност към мрежата и работа без мрежова връзка. Понякога приложенията са офлайн без да бъдат изрично разединени от мрежата. Приложението може да обработва един или няколко от тези сценарии. Връзката може да бъде управлявана ръчно или автоматично.

Ръчен мениджмънт на връзката – приложението може да позволява на потребителя да избере дали да работи онлайн. Когато потребителят избере да работи офлайн, приложението трябва да съхрани цялата информация, която ще е нужна на потребителя. В този случай потребителят взаимодейства с приложението, знаейки, че е офлайн. Приложението не се опитва да извършва мрежови заявки докато изрично не му бъде казано да се върне в онлайн режим и да извърши операция по синхронизация.

Към приложенията може да бъде добавена поддръжка за уведомление от страна на потребителите за типа на връзката. По този начин приложението може

да добави оптимизация, свързана с качеството на връзката (batch request например).

Автоматично управление на връзката - приложението може да бъде проектирано така, че да се адаптира в зависимост от връзката

- Прекъсваща връзка – в някои случаи трябва да се забрани използването на част от функционалността, в други трябва да се предоставя пълната функционалност.
- С менящо се качество – приложението може да различи бавна връзка и голяма латентност. Ако качество на връзката се намали, то може да започне да кешира данни по-агресивно.
- Несигурен достъп до услугите - ако услугите са недостъпни, приложението може да приеме поведението си в офлайн режим. Ако работи с няколко услуги и една от тях стане не достъпна, може да приеме, че всичките са офлайн.

3.1.3.5. Проектиране на съхрани и препрати механизъм

При имплементирането на архитектура ориентирана към услуги, трябва да се осигури съхрани и препрати механизъм. Той се използва за създаване, съхраняване и евентуално препращане на заявките към местопредназначението им. Най-тривиалната имплементация е опашката от съобщения. Новите съобщения се слагат в опашката за съобщения и се изпращат към адреса на получателя. Тук ще разгледаме това решение. При офлайн режим, опашката трябва да има начин да съхрани съобщението за по-късно обновяване на сървъра, например чрез записването им във файл. Освен това проектирането трябва да осигури функционалност, че съобщенията са успешно предадени на техния получател. Трябва да се вземат предвид следните сценарии.

- Липса на потвърждение, че съобщението е било изпратено, както трябва.
- Липса на връзка между клиента и сървъра
- Липса на потвърждение от сървъра – в този случай, трябва да се изпрати независимо потвърждение, за да се информира клиентът, че информацията е пристигнала.

Механизмът за съхранение и препращане трябва да поддържа допълнителна функционалност, като шифроване на съобщенията, определяне на приоритети, заключване и синхронизиране.

Изграждането и проектирането на надеждни архитектури, базирани на съобщения, е сложна задача и изисква значителен опит и експертност. Поради тази причина, е препоръчително да се използват комерсиални услуги като Microsoft Message Queuing, но тя изисква инсталирането на допълнителен софтуер на клиента, затова не е подходящ вариант за всички случаи. Друг вариант е използването на Smart Client Offline Block.

3.1.3.6. Управление на конфликтите при данните и бизнес правилата

В определен момент от време, промените направени на клиента, трябва да бъдат синхронизирани и съвместени с тези на сървъра. Това повишава възможността за конфликти или други проблеми, които приложението, потребителите или администраторите, трябва да разрешат.

Конфликтите при бизнес правилата са резултат не от конфликт между две парчета данни, а защото правилата са били нарушени някъде и данните трябва да бъдат поправени спрямо правилата. Например приложението работи дълго време в офлайн режим, въведена поръчка може да съдържа кеширан продукт, който междувременно е бил премахнат от сървъра. Когато данните за поръчката се изпратят към сървъра, той ще разбере, че продукта е бил премахнат. Ако продукта е бил заменен, системата трябва да предостави възможност той да бъде заменен в поръчката с новия наличен. Това не е конфликт в данните тъй като данните не конфликтират с нищо, но все пак не е коректен случай и поръчката трябва да бъде поправена. Въпреки, че изключенията при данните и бизнес правилата са различни, те могат да бъдат разрешение чрез използването на едни и същи подходи и инфраструктури. Такива подходи са разделяне и заключване на данни, проследяване на непотвърдени данни, управление на загубилата давност информация. Те са разгледани подробно по-долу.

Деленето на данни може да бъде използвано в ситуации, в които различните индивиди имат контрол над отделни секции от данните. По такъв

начин то позволява на потребителите да правят условни промени без да се страхуват от получаването на конфликт. То може да се комбинира с песимистично заключване.

Има два типа заключване - песимистично и оптимистично. Когато системата слага изключителни заключвания, за да осигури, че само една страна работи с определени данни по това време, имаме песимистичното заключване. Например преди да отиде на път, продавач може да достъпи базата и логически да заключи потребителските сметки в дадена географска зона. По този начин никой няма да може да промени данните преди той да се върне. При синхронизацията на данни няма да има никакви конфликти.

Главният проблем с песимистичното заключване е, че множество страни могат да имат нужда от тази информация по едно и също време и ще се наложи да чакат за данните, когато те станат достъпни. Това прави песимистичното заключване добро при запазване на целостта на данните, защото няма възможност за конфликти, но лошо от гледна точка на паралелността на работа.

В реалността, песимистичното заключване е подходящо в малко случаи при непостоянно свързаните приложения. В системите за мениджмънт на документи, потребителите могат да заключват файлове за дълги периоди от време, докато работят върху тях. Когато сложността расте, песимистичното заключване става непрактично.

Повечето непостоянно свързани умни клиенти използват оптимистично заключване, което позволява на множество страни да достъпват и работят с информацията паралелно, с допускането, че промените направени върху данните от различните хора няма да доведат до конфликт. Оптимистичното заключване позволява висок паралелен достъп до данните, на цената на намалена цялостност на данните (интегритет). Конфликтите са често срещани и е необходима стратегия за справяне с тях. В повечето офлайн сценарии, трябва да се използва оптимистичното заключване.

Докато потребителите работят офлайн, всички данни, които променят, не са потвърдени като промяна на сървъра. Чак след като данните бъдат слети с тези на сървъра и няма конфликти, данните могат да се приемат като потвърдени. Важно е да се следи непотвърдената информация. Когато данните се потвърдят, могат да бъдат маркирани като такива и да бъдат използвани уместно. Ако е необходимо да се използват непотвърдени данни в

потребителският интерфейс, те трябва да бъдат маркирани като такива. По принцип приложението не би трябвало да използва тези данни в повече от една задача, ако е непотвърдена, за да не засегне други дейности, които разчитат на потвърдена информация. Използването на потвърдена информация не гарантира, че няма да се появи конфликт, защото тя може в последствие да е била променена от някой друг.

Дори, ако данните не са се променили, те могат да спрат да са коректни, защото вече не са текущи. Тези данни са известни като остарели. При проектиране, трябва да се определи как да се борави с остарялата информация и как да се предпазим от това умният клиента да я използва. Това е особено важно за непостоянно свързаните клиенти, тъй като тя може да е текуща, когато клиента за пръв път е излязъл офлайн, но може да остарее преди клиента да се върне отново онлайн. В допълнение, данните, които са текущи на клиента, могат да остареят докато достигнат до сървъра.

Ако заявка към сървъра влезе в опашката и е готова за изпращане при първо появяване онлайн, шансовете тя да се натъкне на конфликт или изключение се увеличават правопропорционално с времето прекарано в опашката.

Има множество техники за управление на остарялата информация. Може да се използват метаданни, за да се опише валидността на данните и да се укаже кога данните ще загубят валидност. Това може да предотврати изпращане на остаряла информация към клиента. На сървъра може да се проверяват всички данни от клиента и да се определи дали те са остарели преди да се позволи сливането им с данните на сървъра. Ако данните са остарели, сървърът може да се погрижи, клиентът да обнови референтните си данни.

Рискът от остаряла информация е значително по-голям при отчасти свързаните умни клиенти, отколкото при постоянно свързаните. Поради тази причина, умните клиент приложения трябва да извършват допълнителни валидационни стъпки. Чрез добавяне на допълнителна валидация, може да осигури по-голяма толерантност на услугите към остарялата информация.

Понякога остарелите съобщения са неизбежни. Чрез моделирани бизнес правила трябва да бъде заявено как да се обработва остарялата информация,. В някои случаи, остарялата информация е приемлива. Например, да предположим, че имаме каталожен номер на продукт и той се промени. Продуктът все още

съществува, само трябва да се намери новият му каталожен номер. Ако обаче се извършва парична транзакция между две сметки и едната от тях е затворена, транзакцията не можете да се извърши. Тук давността има значение.

Добро правило е бизнес обектите да се справят с остарялата информация. Бизнес обектите могат да валидират, че данните са текущи, ако са остарели, да не правят нищо или да намерят еквивалентната текуща информация, и да върнат информацията на клиента, за да бъде обновена, или да използват бизнес правило, за да автоматизират уместен отговор.

Няма еднозначен отговор кое е най-доброто място за управление на остарелите данни. В някои случаи бизнес правилата диктуват, че сървърът е най-доброто място да се управлява остарялата информация, ако клиентът не може да разрешава конфликтите. Ако сървърът няма достатъчно информация автоматично да се справи със ситуацията, може да изиска от клиента почистване на данните преди да се извърши синхронизация със сървъра. В друг случай може да се окаже, че използването на остарялата информация не е проблем.

За да се изяснят изискванията за изглаждане на конфликти в дадена организация, трябва да се разберете начинът, по който организацията функционира. В някои случаи е малко вероятно да се получат конфликти, тъй като различни хора се грижат за различни части от данните. В други случаи, конфликтите ще се случват по-често и трябва да се осигури механизъм, който да ги разрешава.

Независимо какви предпазни средства се използват, вероятността клиентът да предостави данни, които водят до нарушение на бизнес правилата или конфликт на данни, е голяма. Когато възникне конфликт, услугата трябва да осигури възможно най-много детайли за конфликта, за да може потребител или администратор да изглади проблема. В някои случаи конфликтът не е сериозен и може да бъде овладян автоматично от клиента или сървъра.

Изглаждането на конфликти може да бъде сложен и зависим от сценария на употреба проблем. Всяко приложение си има своите особености, но трите основни опции за изглаждане на конфликти са:

- Автоматично изглаждане на сървъра
- Специфично изглаждане на клиента
- Изглаждане от трета страна

Автоматично изглаждане на сървъра

В някои случаи, приложението може да е проектирано така, че сървърът да използва бизнес правила и да автоматизира процеса за обработка на конфликти, без да афектира клиента. Справянето с конфликтите на сървъра е добро за използваемостта и спестява усилия на потребителя. Клиентът винаги трябва да се държи информиран за направените съгласувания. Например да се връща отчет на съгласуванията, обясняващ конфликтите и как са били разрешени. Това помага на клиентското приложение да държи данните си консистентни и да информира потребителя за резултата от разрешаването на конфликтите.

Специфично изглаждане на клиента

Клиентът знае повече за контекста на оригиналната заявка. В някои случаи единствено той има информация за разрешаването на конфликта. В трети случай потребител или администратор трябва да определи къде конфликтът трябва да бъде разрешен. За да се извърши ефективно разрешаване на конфликта при клиента, услугата трябва да му изпрати достатъчно информация, с която да му позволи да направи интелигентни решения за това как конфликтът трябва да бъде разрешен.

Изглаждане от трета страна

В някои случаи, може да е нужно трета страна да разреши конфликтите. Например, при разрешаване на важни конфликти е нужна намесата на администратор. В тези случаи, клиентът трябва да бъде информиран, че решението виси. Клиентът може да продължи да използва експериментални стойности, но често трябва да изчака докато конфликтите бъдат решени. Клиентът трябва да бъде информиран, когато конфликтът се реши. Алтернативно, клиентът може да прави запитвания периодично, за да определи статуса на висящите данни, и когато получи потвърдени данни, да продължи.

3.1.3.7. Взаимодействие с CRUD подобни уеб услуги

Много уеб услуги се създават със създай, прочети, обнови, изтрий интерфейси (CRUD). Тук се разглеждат аспекти от разработването на умни клиенти, използващи CRUD уеб услуги.

Създаването на записи може да бъде относително проста задача при използването на CRUD уеб услугите. Най-важното нещо е уникално да се идентифицира всеки запис, който се създава. В повечето ситуации, това може да се направи чрез използването на уникален идентификатор за първичен ключ на записа. Тогава, дори два на пръв поглед идентични записа, създадени на различни сървъри, ще бъдат разглеждани като различни, когато настъпи момент на сливане или репликация. В други случаи, два на пръв поглед идентични записа, не трябва да се третират като уникални. В такива случаи, когато се появят конфликти в данните , се генерират изключения.

Има няколко различни метода, които могат да се използват, за да се създадат уникални идентификатори на офлайн клиент.

- Записът да се изпрати като data transfer object (DTO) без уникално ID и да се позволи на сървъра да присвои такова.
- Да се използва глобален уникален идентификатор (GUID), който клиентът ще присвои като System.Guid
- Да се присвои временно ID на клиента и след това да се замени с генерирано от сървъра
- Да се присвоява блок от уникални ID-та на всеки клиент
- Да се използва името на потребителя или ID-то му, за да се сложи префикс на всички ID-та и да се инкрементират на клиента. Така те ще бъдат глобално уникални по подразбиране.

При четене на данни не възникват конфликти в данните. При непостоянно свързаните клиенти могат да настъпят проблеми, дължащи се на кеширането Преди клиентът да мине офлайн, той кешира всяка информация, която му е необходима. Кешираните данни могат да остареят преди клиента да се върне онлайн, а това води до некоректни данни на клиента и проблеми, когато се извършва синхронизиране със сървъра.

Обновяването на данните е операцията, която най-често води до конфликти. Множество потребители могат да се опитват да обновят данните едновременно, а това води до конфликт при настъпването на сливаща репликация. Могат да се използват няколко метода за минимизиране на настъпването на такива конфликти.

Изтриването е недвусмислена операция, защото даден запис може да бъде изтрит само веднъж. Опит да се изтрие един и същи запис повече от

веднъж не афектира сървъра. Това, което трябва да се има предвид, е следното: може да се маркират данните като пробно изтрити от клиента и след това в опашката да се сложат заявките за изтриване. Това означава, че ако сървърът не може да изтрие записа по някаква причина, изтриването може да бъде отказано на клиента.

Както и при създаването на записи, трябва се използва уникален идентификатор, за да е сигурно, че се изтрива коректният запис от сървъра.

3.1.3.8. Използване на подход базиран на задачи

Базираният на задачи подход, използва единица работа и я капсулира като потребителска задача. Обектът Задача трябва да се погрижи за необходимото състояние и за потребителския интерфейс на взаимодействията, които са нужни на потребителя, за да завърши задачата. Този подход е практически удобен при изграждането на умни клиенти с офлайн възможности, защото позволява капсуловане на детайлите на офлайн поведението на едно място. Това позволява на потребителския интерфейс да се съсредоточи върху UI- въпросите, вместо върху логическата обработка. Един обект Задача обвива функционалността, която потребителят асоциира с независима единица работа. Гранулярността и детайлите на задачата зависят от точния сценарий. Ето няколко примера за задачи:

- Вкарване на информация за поръчка
- Промени в клиентските детайли
- Създаване и изпращане на задача

За всяка от тези задачи, се инстанциира обект Задача и се използва, за да се направлява потребителя през процеса, за да съхранява цялата необходима информация, за да взаимодейства с потребителския интерфейс, и да взаимодейства с необходимите услуги.

Когато приложението е в офлайн режим, то трябва да съхранява заявките в опашка и да прави локални промени, използвайки пробни и непотвърдени стойности. По време на синхронизацията, приложението трябва да направи действителните заявки към сървъра и да промени локалните състояния, за да потвърди успеха на заявките. Чрез капсуловане на детайлите на процеса в един обект Задача, който слага заявката в опашката и проследява пробните и

потвърдени промени на състоянието, може да се облекчи разработката на приложението и да се позволи всички задачи да бъдат обработени по стандартен начин. Обектът Задача може да осигури детайлна информация за състоянието на задачата чрез различни свойства и събития, включващи:

1. Висящ статус – индикатор, че задачата има висяща синхронизация
2. Потвърден статус – индикатор че задачата е синхронизирана и потвърдена като успешна.
3. Конфликтен статус – индикатор, че се е получила грешка при синхронизация. Други свойства могат да дадат детайли за конфликта или грешката.
4. Завършена – индикатор за процентното завършване или флаг дали задачата е приключила.
5. Достъпност на задачата – някои задачи, няма да бъдат достъпни, когато приложението е офлайн или онлайн, или ако задачата е част от потока на потребителските действия. Тогава може да не е достъпна докато не се извърши определена задача преди това. Това свойство може да бъде свързано с флаг за меню опция или бутон от лентата за инструменти, за да предотврати извършването на неподходяща задача от потребителя.

   Друго предимство на подхода базиран на задачи е, че фокусира приложението върху задачата, което може да доведе до по-интуитивно приложение.

3.1.3.9. Управление на зависимостите

Ако потребителска задача включва повече от една заявка към сървъра, към задачата трябва да се подходи много внимателно. Трябва да се осигури, че потребителят може да извърши цялата задача, докато е офлайн. Предизвикателството на сървърните заявки е, че те често са зависими една от друга. Зависимостите могат да са резултат от комплексна бизнес логика, което допълнително усложнява обработката на афектираните свързани задачи. Зависимостите могат да бъдат обратни и еднопосочни

Еднопосочни – ако първата заявка успее, а втората пропадне, може да се наложи да се отмени първата чрез компенсираща транзакция, това може да добави комплексност в приложението.

Обратни зависимости – ако приложението оперира офлайн и предаде една заявка към сървъра като част от задача към множество услуги, то трябва да приеме, че заявката е успешна, за да могат да се сложат в опашката и последващите заявки и да не се блокира потребителя от извършване на задачата. В този случай, всички следващи заявки, зависят от успеха на първата. Ако първата заявка се провали по време на синхронизацията, приложението знае, че всички последващи свързани заявки, трябва да се провалят или да се игнорират.

Справяне със зависимостите на сървъра

За да се намали сложността, асоциирана със зависимостите между сървърните заявки, уеб услугите трябва да осигуряват една сървърна заявка за потребителска задачата. Това позволява потребителя да изпълни задача, която ще бъде обработена по време на синхронизационната фаза като атомарна заявка към уеб услугата. Една атомарна сървърна заявка елиминира нуждата да се проследяват зависимостите, свързани със сървърните заявки, което може значително да усложни клиент или сървър имплементацията на приложението. Вместо да се дефинират следните отделни заявки: BookCar(), BookHotel(), BookAirlineTickets(), те може да се комбинират в една стъпка BookVacation(Car car, Hotel hotel, Tickets airlineTickets). Комбинирането на тези стъпки по този начин дефинира атомарна заявка. В този случай уеб услугата ще е отговорна за извършване на необходимата комбинация между елементите.

Справяне със зависимостите при клиента

Зависимостите между сървърните заявки могат да се следят и на клиента. Този подход предлага значителна гъвкавост и позволява на клиента да контролира комуникацията между произволен брой услуги. От друга страна е трудно да бъде имплементиран и тестван. Подходът базиран на задачи е добър начин за следене на свързаните заявки на клиента, и осигурява начин за капсуловане на цялата необходима бизнес логика и обработването на грешките на едно място, което облекчава разработката и тестването.

Например, обектът задача, използван, за да се резервира ваканция, знае че трябва да извърши три заявки към сървъра. Той имплементира необходимата бизнес логика така, че да може да контролира заявките към сървъра по подходящ начин, ако настъпи състояние на грешка. Ако наемането на кола се провали, може да продължи с останалите заявки. Ако заявката за билет се

провали, ще трябва да откаже хотела и резервацията на кола, както и да създаде компенсираща транзакционна заявка за всяка услуга.

Използване на организационен междинен софтуер (Middleware)

Понякога зависимостите и съответстващите им бизнес правила в приложението са достатъчно сложни и изискват някаква форма на организационен междинен софтуер като Microsoft BizTalk Server, който координира взаимодействията между множество уеб услуги и клиентски приложения. Организационния междинен софтуер се намира в средния слой и осигурява фасада за уеб услугите при взаимодействието им с умния клиент. Фасадната уеб услуга представлява подходящ интерфейс, позволяващ атомарни уеб услуги за задача. Когато се получи заявка, организационната услуга обработва заявката като инициира координирани извиквания към необходимите уеб услуги. Ако е необходимо, тя агрегира резултатите преди да ги върне на клиента. Този подход предоставя удобен начин за отчитане на взаимодействията между множество уеб услуги. Biztalk осигурява различни услуги като трансформация на данни и механизъм за обработване на бизнес правила, който може значително да помогне при комуникациите между разпределени уеб услуги и остарели системи, както и в комплексни бизнес сценарии. В допълнение, този подход осигурява необходимата достъпност, гарантира надеждност, помага за осигуряване на консистентност между отделните услуги.

3.2.Изводи

В главата се разгледаха основните подходи при разработване на офлайн умни клиенти, единият е базиран на данни, а вторият е ориентиран към услуги. Подходът базиран на данни използва релационна база данни на сървъра и клиента. Поддържането на офлайн функционалността, синхронизирането, заключването, изглаждането на проблеми се извършва основно от базата данни. Разработчикът трябва да се съсредоточи върху дефиниране на бизнес правилата. При подхода ориентиран към услуги обаче, разработчикът трябва да се погрижи за много повече аспекти. Освен клиентските аспекти, трябва да се отчетат и сървърните. Трябва да се избере асинхронната комуникация пред синхронната, трябва се минимизират сложните мрежови взаимодействия, задължително

трябва да се използва кеширане, добре е да се отчита връзката и приложението да приема различно поведение в зависимост от нея. За комуникацията трябва да се използва съхрани и препрати механизъм. Може да се използва подход базиран на задачи за управление на дейностите, реализиращи една задача, особено ако те включват комуникация с различни уеб услуги за изпълнение на задачата.

Като цяло, подходът базиран на данни, дава бързо като имплементация решение, но подходът базиран на услуги, дава много по-гъвкаво решение на цената на допълнителен труд от страна на разработчиците. Базираният на данни подход тясно свързва клиента със сървъра, промяна в схемата на единият води до промяна в схемата на другият, а когато трябва да се работи с външни уеб услуги е неприложим и трябва да се изгради още един, но вече базиран на съобщения. Кой подход от двата ще се избере и кои от аспектите на имплементация ще се адресират, зависи от софтуерните архитекти и разработчици.

4. Microsoft и умните клиенти - CAB, Smart Client Software Factory, ClickOnce

Умен клиент както бе споменато по-рано е термин въведен от Microsoft. Освен създаването на този термин, те разработиха доста солидни приложни блокове, фабрики и технология, за да поддържат такъв тип приложения. Тази глава ще разгледа технологиите въведени от Microsoft за разработка на умни клиенти. Главата прави бърз преглед на градивните части на CAB и приложението му, продължава със SCSF и надстройките на CAB, които тази фабрика предлага. Разгледани са два подхода за проектиране на умни клиенти, т.е. как бизнес обхвата може да бъде трансформиран в градивни обекти на CAB и SCSF, единият е базиран на диаграмата на случаите на употреба, а вторият на бизнес обектите. Разбира се за умните клиенти важен аспект е разгръщането, затова последната част на главата е посветена на технологията за разгръщане, която те могат да използват.

4.1.CAB(Composite UI Application Block)

Composite UI Application Block е готов блок за изграждане на софтуерни приложения разработен, от екипа на Microsoft Patterns & Practices. Този блок осигурява следната функционалност: динамично зареждане на независими, но комуникиращи си модули в общ скелет (shell); поддържа посредник на събития (Event broker) за слабо свързани комуникации между модулите и частите на тези модули; готова за използване имплементация на шаблона команда; базови класове имплементиращи шаблона Модел Изглед Контролер; фреймуърк за добавяне на инфраструктурни услуги вариращи от услуги за идентификация и авторизация до локализатор на модули и услуга за зареждане на модули.

Тук накратко ще се опишат градивните блокове на CAB приложение:

Скелетно приложение (Shell application) – основно приложение, което е отговорно за инициализиране на цялата клиентска част. То е отговорно за динамично зареждане на модулите, за зареждане на базовите услуги на умното приложение, в допълнение то стартира и главната форма на скелета.

Скелет (Shell) – това е главната форма на приложението. Тя осигурява потребителския интерфейс, общ за всички динамично заредени модули. Скелетът винаги хоства един работен обект (WorkItem), който е входна точка за всяка друга част на приложението като услуги, модули и работни обекти, които са създадени и регистрирани от тези модули.

Множество, логически свързани работни обекти могат да бъдат обединение в една единица за разгръщане. Модулът е пример за единица за разгръщане. Конфигурацията на CAB базирани умни клиенти работи на ниво модул. Затова, откриването на точната гранулярност за обединение на работните обекти в модули е критично.

Услугата за зареждане на модули (ModuleLoader) е основополагаща услуга, осигурена от CAB. Тя е отговорна за зареждането на модулите, специфицирани в профилния каталог (profile catalog). Тя обхожда конфигурацията на профилния каталог и се стреми динамично да зареди асемблитата, специфицирани в каталожния елемент, след това се опитва да намери ModuleInit класа в асемблито. Всеки модул съдържа ModuleInit клас, който е отговорен за зареждането на услугите и работните обекти в модула

Профилен каталог – профилният каталог е конфигурация, специфицираща кои модули и услуги трябва да бъдат заредени в приложението. По подразбиране, каталогът е прост XML файл, който е разположен в директорията на приложението. XML файлът специфицира кои модули ще бъдат заредени. Чрез написване и регистриране на собствен IModuleEnumerator клас, може да се промени това поведение, тъй че каталогът да се взима от друго място, например чрез уеб услуга.

Работният обект (WorkItem), погледнато технически, е просто контейнер за управление на обектите, необходими за извършването на определена задача. Такива обекти са обекти на състоянието, изгледи, техните представители (presenter) и контролни обекти, команди за зареждане на определени действия. На по-високо ниво на абстракция може да се каже, че работният обект е клас, капсуловащ бизнес логиката, и е посветен на случаят на употреба. Може да бъде разглеждан като контролер на случая на употреба, който знае всичките отделни аспекти на случая на употреба. Като такъв, работният обект съхранява състоянието за всички части, които са включени в случая на употреба, като необходимите за показване изгледи, или подслучаи. Той играе ролята на входна

точка за случаят на употреба (или някой от под случаите му), за който е отговорен.

Работно място (Workspace) – работното място е контрол, който е основно отговорен за съхранение и представяне на елементите на потребителския интерфейс, създадени от работния обект. Обикновено работните места се добавят в скелета или към други разширяеми изгледи в йерархията от работни обекти. Те играят ролята на контейнери за частите на потребителския интерфейс, които трябва да бъдат попълнени динамично с елементи на потребителския интерфейс, осигурени от работния обект или неговите подчинени обекти. Всеки потребителски контрол може да бъде направен работно място чрез имплементирането на IWorkspace интерфейса.

UIExtensionSites – това са специални крепежни места за разширяване на фиксирани части на скелета, като менюта, лента за инструменти и статус лента. За разлика от работните места, тяхната цел е да се използват за части от потребителския интерфейс, които не трябва напълно да са заменени от работните обекти. Те трябва само да бъдат разширени като например да се добави ново подменю към меню.

Публикуването на събития се използва от издателите на събития за слабо свързани комуникация. Всеки издател имплементира събитие, маркирано с [EventPublication] атрибут. Събитията са уникални и се идентифицират уникално чрез URI, което представлява уникален низ. Само абонатите подписалите се за събитие с такова URI ще бъдат известени от CAB. Абонатите трябва да имплементират метод със същата сигнатура, както сигнатурата на съобщението използващо [EventPublication] и тези методи трябва да бъдат маркирани с [EventSubscription] атрибут.

Подписването за събитие е противоположно на публикуването му. Всеки клас, който иска да се подпише за определено събитие с дадено URI, трябва да имплементира обработчик на събития, който да отговаря на сигнатурата на метода, дефинирана при публикуването на събитието. Маркира се с [EventSubscription] атрибут и с URI-то на интересуващото го събитие. CAB уведомява абонатите всеки път, когато даден издател публикува съответстващо събитие.

Строителят на обекти (ObjectBuilder) е основополагащ елемент от CAB, който действа като фабрика за създаване на обекти, които изискват определена

стратегия за построяване при създаването си или се нуждаят от свойства, като автоматично инстанциране и инициализация на зависещи обекти. Като такъв, строителят на обекти има комбинирана роля на фабрика, фреймуърк за инжектиране на зависимости (dependency injection) и фреймуърк за строителни стратегии.

Инжектиране на зависимости (dependency injection) – инжектирането на зависимости е шаблон, позволяващ на фабриките автоматично да създават и инициализират свойства или членове на обекти със зависими обекти. Строителят на обекти предоставя тази функционалност.

CAB се използва за създаването на приложения с комплексен потребителски интерфейс, работещи на Windows платформа. Той предоставя архитектура и имплементация, които помагат при изграждането на приложения, благодарение на общи шаблони между бизнес приложенията. Предимствата от използването на приложния блок са изграждането на сложни, независими, но комуникиращи си модули; разделяне на задълженията между програмистите, разработващи модулите, и тези, разработващи базовата структура на приложението (Shell); предоставя архитектурна рамка за произвеждането на последователен висококачествен потребителски интерфейс; увеличава продуктивността и намалява като цяло времето за разработка.

4.2. Smart Client Software Factory

Въпреки че CAB осигурява надеждна инфраструктура, работата с този приложен блок, е сама по себе си предизвикателство. Тя изисква изпълняването на много еднотипни и често срещани задачи ръчно, като създаване на класове, наследяващи базовия клас за работен обект, за да се създаде контролен обект за случай на употреба, или ръчно създаване на класове реализиращи Контролер Изглед Модел шаблона. Smart Client Software Factory е разширение на Visual Studio 2005, което осигурява автоматизиране на често срещаните задачи и осигурява голям набор детайлна документация ((Patterns & Practices: Web Service Software Factory - Home, 2007), (MSDN, Smart Client Software Factory, 2007), за създаване на съставни умни клиенти, използващи CAB, включващи „как да ...” теми и набор от имплементации като Global Bank имплементацията.

Автоматизирането на задачите е направено с цел да се употребят общите директиви и ръководни принципи при разработването на софтуерни приложения. SCSF осигурява ръководни принципи за разработването на CAB – базирани умни клиенти и автоматизира създаването на нови модули и контролери за случаи на потреба в добавка към публикуването и подписването за събития. Основните термини, които въвежда SCSF са:

WorkItemController е клас въведен от SCSF, който съдържа основна инициализираща логика за работния обект. Вместо директно да се наследява класа WorkItem, когато се създават работни обекти със SCSF, се наследява WorkItemController, за да се използва добавената от този клас логика за инициализация.

ControlledWorkItem –това е основен клас за инициализиране на нови работни обекти, базирани на WorkItemController. Той изпълнява добавените инициализационни входни точки, осигурени от WorkItemController класа.

ModuleController се използва за капсуловане на специален работен обект в даден модул. Този обект изпълнява ролята на основен работен обект за модула. Подразбраната имплементация на ModuleInit, генерирана с SCSF, създава автоматично работен обект наименуван ModuleController. Затова ModuleController е основната входна точка за всички работни обекти, доставяни от модула

Представител (Presenter) е клас, имплементиращ логиката за единичен SmartPart. Класът -представител е базиран на Модел Изглед Представител (MVP) шаблона, който е просто опростен вариант на Модел Изглед Контролер (MVC) шаблона. Основната разлика между двата е, че с MVP изгледът е изцяло контролиран от представителя, докато в MVC, контролерът и моделът могат да обновяват изгледа.

4.3.Процес на разработване

Скелетът е конкретното приложение, отговорно за зареждане и инициализация на базовите услуги на клиента, зареждане и инициализиране на модули, осигуряващо базовия потребителски интерфейс на приложението, хостващо SmartParts, заредени в първото ниво работни обекти(първото ниво работни обекти нямат родител друг освен RootWorkItem или ModuleController).

Базовият потребителски интерфейс е еднакъв за всеки модул, зареден в приложението. Като такъв, трябва да изпълни изискванията, които са общи за всички случаи на употреба. Типични примери за общ потребителски интерфейсен елемент, който трябва да бъде добавен в скелета, са менютата и лентите за инструменти, навигационните контроли, общ прозорец са съобщения, за да има централизирана точка за обработване на съобщенията към потребителя като забележки и грешки, контекстни панели, които показват допълнителна информация в зависимост от ситуацията, в която се намира потребителят. Това са просто няколко примера за добри кандидати, които да бъдат интегрирани в скелета. С работните места и UIExtensionSites, CAB осигурява готова за използване инфраструктура за разширяеми места в скелета. Работните места се използват, за да заменят част от интерфейса, докато UIExtensionSites се използват за разширяване на съществуващи части от скелета, като добавянето на полета в менютата или добавяне на инструменти в лентата за инструменти. UIExtensionSites са статични части от потребителския интерфейс. Те могат и въобще да не се изменят. Те се разширяват от модулите, които се зареждат в умния клиент. Обикновено се използват за стартиране на случай на употреба чрез работен обект, затова най-често се добавят от ModuleInit компонентите. Когато се говори за общите части на приложението, обикновено архитектите трябва да решат дали искат да ги заредят директно в главният изглед на модула или да ги добавят в отделен инфраструктурен модул. Основният въпрос тук е преизползването. Искаме ли да използваме общата потребителска част някъде другаде. Може би да я използваме в друг умен клиент. Ако да, тогава разбира се, тя трябва да бъде добавена в отделен модул вместо директно да е интегрирана в скелета. Други фактори, влияещи на решението, са конфигурацията и сигурността. Ако трябва динамично да се зарежда потребителски интерфейс или трябва да се зареди потребителският интерфейс на базата на конфиденциалния контекст на потребителя, общият потребителски интерфейс трябва да се сложи в отделен модул.

Инфраструктурни услуги – както бе споменато по рано, инфраструктурните услуги капсуловат функционалност, обща за всички или обща за много модули и компоненти. За разлика от скелетната инфраструктура, тези услуги не са свързани със специфични за потребителския интерфейс задачи. Те капсулират логика на клиента, например специфична за клиента

бизнес логика или поддържане на офлайн функционалност. CAB подържа множество готови услуги като услуга за идентифициране или услуга за зареждане на каталог от модули. Разбира се, винаги могат да се добавят собствени услуги. Типичен пример за такава услуга не предоставена от CAB е услуга за авторизация. CAB и SCSF не осигуряват такава готова, но примерната имплементация Bank Branch Workbench, част от SCSF, демонстрира как това може да бъде постигнато на базата на ActionCatalog, осигурен от SCSF. Други примери са агенти за уеб услуги. Проксита, капсулиращи заявките към уеб услуги и изпълняващи офлайн функционалности, услуга за поддържане на контекста, която да управлява контекста между работните обекти, конфигурационна услуга за достъп до приложно насочената конфигурация, услуга за разгръщане чрез използване на ClickOnce. Инфраструктурните услуги е необходимо да бъдат капсулирани в отделни инфраструктурни модули и заредени преди другите модули, реализиращи конкретни случаи на употреба.

Идентифициране на работни обекти – работните обекти са компонентите, отговорни за капсулиране на логиката за специфичните задачи, които потребителят иска да извърши чрез приложението. Като такива, работните обекти са централните и най-важни части на умното приложение, защото те осигуряват действителната бизнес функционалност. За тази цел, работният обект съдържа или знае за една или повече SmartParts (изгледи) с техните класове контролери и модели. Работният обект знае кой SmartPart трябва да бъде показан във всяко едно време и кои подчинени работни обекти трябва да бъдат стартирани и по кое време. Той управлява състоянията, необходими на SmartParts и подчинените работните обекти. Всяко CAB приложение има един RootWorkItem, който играе ролята на главна входна точка за основните услуги и работните обекти, добавени от модула. Със SCSF всеки модул автоматично, зарежда ModuleController работен обект, който играе ролята на основен работен обект за модула. Основно има два начина да се идентифицират работните обекти за умния клиент: единият е основан на случаи на употреба, а вторият на бизнес обекти.

Подход базиран на случаите на употреба. – едно от най-големите предимства на CAB архитектура е, че позволява работните обекти да бъдат проектирани чрез диаграми на случаи на употреба. В действителност, в повечето случаи, имаме съответствие едно към едно между случаите на

употреба и работните обекти, обикновено един случай на употреба се реализира от един работен обект. Затова работните обекти не са нищо друго освен контролери на случаите на употреба, имплементиращи необходимата обработка на ниво потребителски интерфейс, за реализиране на случай на употреба (извършване на задача) и обхващащо всички необходими части за това.

Първо се започва с изграждане на съответствие едно към едно, между един случай на употреба и един CAB работен обект. Връзката между случаите на употреба може да бъде два типа: случаят на употреба може да бъде или подслучай на друг случай или той може да се използва от много случаи на употреба. Разбира се случай на употреба, който е под случай и не се използва от други резултира в подчинен работен обект. Чистите подслучаи на употреба са подчинени работни обекти и не могат да бъдат достъпвани извън техните родителски обекти. Случаите на употреба, използвани от други случаи на употреба, не само от родителските им, трябва да бъдат достъпвани директно или чрез родителският им работен контрол

Основните случаи на употреба се проектират в първо ниво работни обекти, ако има само един такъв, той може директно да бъде имплементиран чрез ModuleController. Те са входните точки за подчинените работни обекти. Те са отговорни за управление на състоянието, необходимо за всички подчинени работни обекти. Препоръчително е работните обекти от първо ниво да изпълнят следните стъпки, когато се зареждат: да добавят услугите необходими за този набор работни обекти, да регистрират командите за стартиране на подчинен работен обект, да добавят UIExtensionSite, да създадат инстанции на подчинените работни обекти, когато е необходимо, да създадат и регистрират инстанции на смарт частите изисквани от самия работен обект

Обикновено простите работни обекти без подчинени работни обекти или самите подчинени обекти изпълняват същия набор от стъпки, единствено не регистрират услуги и UI extension sites. Подчинените работни обекти не регистрират команди, тъй като те се извикват от родителските класове. Работните класове на първо ниво се създават от ModuleInit на модула, в който се съдържат. ModuleInit на модула регистрира командите и UIExtensionSites за първоначалната функционалност, която е капсулирана в работните обекти на първо ниво.

Не всичките случаи на употреба се трансформират в работни обекти, например случаят офлайн употреба. Това изискване не променя бизнес логиката, но е индикатор за нещо напълно различно. Това е нещо, което трябва да бъде имплементирано на ниво инфраструктурни услуги, например агентът за уеб услуги трябва да поддържа разпознаване на връзката, да има офлайн хранилище и да обновява съобщенията.

Подчинен работен обект или работен обект. Някои от работните обекти стават работни обекти на първо ниво, въпреки че изглеждат като подчинени в диаграмата на случаи на употреба. Обикновено, това се случва когато случаят на употреба принадлежи логически на родителския случай на употреба, но в детайлния анализ се вижда че той не споделя състояние, контекст или нещо друго с другите подчинени случаи на употреба и не зависи от споделено състояние или контекст. Този случай на употреба, също трябва да се проектира като работен обект на първо ниво.

Могат да се спазват следните стъпки за идентифициране на работни обекти

1. Създаване на диаграма на случаите на употреба
2. Да се направи съответствие между всеки случай на употреба и работен обект в CAB.
3. Да се изключат случаите на употреба, които не са директно свързани с умния клиент.
4. Да се анализират връзките между случаите на употреба. Ако случаите на употреба се използват от други случаи на употреба освен родителските им, те са кандидати за работни обекти от първо ниво.
5. Да се анализират изискванията на случаите на употреба. Ако случаите на употреба не зависят от състоянието и контекста на техните родители и обратно, то това са кандидати за работни обекти на първо ниво.

В предходния лист стъпки 4 и 5 са части от процеса на усъвършенстване, в който се анализират отново диаграмите със случаи на употреба, за да се идентифицират връзките между случаите на употреба.

Стратегията, базирана на бизнес обектите е по-прост подход. Използва се за малки, прости приложения. Работните обекти се идентифицират и структурират на базата на бизнес обектите, обработвани от клиентското приложение. Създава се лист с бизнес обекти, притежавани от приложението. За

всеки бизнес обект се създава работен обект. Ако даден обект е комбинация от няколко бизнес обекта, то той се състои от няколко подчинени работни обекта.

След като работните обекти се идентифицират с помощта на някой от двата подхода, те трябва да се пакетират в модули. Модулът е единица за разгръщане на CAB базираните умни клиенти. В един модул се пакетират логически свързани работни обекти, които адресират едно пространство от бизнеса. За да се реши финално, какви ще бъдат пакетите, трябва да се вземе предвид сигурността, конфигурируемостта и преизползването на компонентите.

Модулите се конфигурират чрез профилен каталог. По подразбиране, този каталог може да бъде в ProfileCatalog.xml файла в директорията на приложението. Тези модули могат да бъдат конфигурирани на базата на ролята на потребителя. Затова сигурността играе главна роля при решението как да се пакетират модулите. Ако дадена функционалност трябва да се използва независимо от другите, то тя трябва да се сложи в отделен модул. Ако е необходимо работен обект да бъде конфигуриран независимо от другите, той също трябва да е сложен в отделен модул. Ако конфигурационните изисквания, свързани със сигурността и преизползваемостта, са еднакви или подобни за някои от работните обекти, те могат да бъдат обединени в един модул.

Връзките между случаите на употреба представляват взаимодействия между случаите на употреба. CAB поддържа два вида взаимодействия между компонентите –активна комуникация чрез имплементиране на шаблона команда или пасивна комуникация базирана на посредник на събития. Но кога трябва да се използва всяка от тях? Посредникът на събития осигурява вградена инфраструктура за слабо свързани съобщения. Това означава, че всеки компонент зареден в CAB, може да публикува съобщения и други компоненти, да се подписват за тях, без да ги интересува кой е издателят. Връзката между издатели и абонати е построена на базата на URI на съобщението. Посредниците на съобщения трябва да се използват за слабо свързани типове комуникация. Това означава, че ако CAB компонент иска да осигури информация на другите, без да получи незабавен отговор и без да изисква знание кой ще го получи, посредникът на събития е правилният избор. Ако трябва да се получи отговор веднага или трябва да се знае с кой компонент се общува, посредникът на събития не е подходящ. Незабавен отговор не трябва да се имплементира чрез съобщения между компонентите. Шаблонът команда

може да се използва, ако компонент се нуждае незабавно някой друг компонент да извърши нещо или трябва да предаде контрола на друг компонент. Този шаблон се използва, за да се стартират работни обекти или подчинени работни обекти, когато не трябва да се използват съобщения.

4.4.ClickOnce

ClickOnce е механизъм за разгръщане, разработен за нуждите на умните клиент приложения и е част от .NET Framework 2.0. Той позволява автоматично разгръщане и актуализиране на умните клиентите от сървър за разгръщане. За да се разгърне или обнови умно клиентско приложение чрез ClickOnce, трябва само да се публикува приложението на сървър за разгръщане и да се предостави съответстващия му линк на потребителя. Когато потребителят стартира приложението чрез линка, то автоматично се разгръща и кешира на клиентския компютър. По-нататъшните стартирания не се извършват през мрежата, а приложението се зарежда и работи от клиента.

ClickOnce третира всяка публикувана версия на приложението като атомарна единица. Публикацията се състои от манифест на разгръщане, манифест на приложението и всичките файлове на приложението за определена негова версия. Версията се специфицира в манифеста на разгръщане. Всеки манифест на разгръщане реферира един манифест на приложението и асоцииранатa с него колекция от файлове. Клиентския компютър трябва да разполага с .NET Framework 2.0, за да се извърши ClickOnce разгръщане на този компютър. Единствено платформено изискване към сървъра е възможност да се свалят манифест файлове и необходимите приложни файлове.

ClickOnce поддържа множество опции. Приложенията могат да се разгръщат чрез HTTP, UNC или CD/DVD. Те могат да се стартират чрез URL или Start меню. Приложенията могат да се използват само в онлайн режим или в комбинация от онлайн и офлайн режим. Проверката за нови версии може да бъде направена, когато клиентът пожелае .Новата версия може да бъде взета от място, специфицирано от клиента. Новите версии могат да бъдат задължителни или опционални. Може да се използва предоставеното от Microsoft API, за да се контролира процеса на обновяване

ClickOnce поддържа два режима за работа – инсталиран и онлайн. При онлайн режима приложението е достъпно само през мрежата, затова този режим е неприложим за непостоянно свързани умни клиенти. Вторият режим е инсталираният режим, при който приложението е достъпно в офлайн и в онлайн режим. Инсталираното приложение позволява на потребителя да стартира умния клиент от Start менюто, след първоначално разгръщане на потребителската станция.

ClickOnce поддържа автоматични и програмни актуализации. Автоматичните актуализации се извършват от ClickOnce runtime на клиента преди или след стартиране на приложението. Програмните актуализации се извършват чрез ApplicationDeployment класа в System.DeploymentApplication пространството.

Автоматичните актуализации включват възможност за: проверка за нови версии преди или след стартиране на приложението, или комбинация от двете; новата версия може да бъде обозначена като опционална или задължителна; проверка за нови версии по план; проверка за актуализации на местоположение, различно от оригиналната инсталация. Автоматичното обновяване сваля новата версия при първа възможност и я прилага при следващо стартиране на програмата. За отлагане на обновяването, може да се постъпи по един от следните начини:

- Да се използва програмното API на ClickOnce, което отчита наличието на нова версия и програмно да се забави нейното сваляне от сървъра, докато критерият за забавяне бъде удовлетворен.
- Да се използва динамичен начин за зареждане, осигурен от Composite UI Application Block профилния каталог, за да се определи кога даден файл трябва да бъде зареден и стартиран. ClickOnce може да сваля новите версии, веднага след като са на разположение, но профил каталогът определя, кога новите версии ще бъдат приложени.
- ClickOnce поддържа следните сценарии за разгръщане
- Разпръскващо разгръщане (Broadcast Deployment). – Всички потребители започват да свалят новата версия едновременно, в същия момент, в който тя е публикувана на сървъра.

- Етапно разгръщане- Новата версия се качва на сървъра, но се очаква различни групи да започнат със свалянето и по различно време. Това налага ограничението всяка група да ползва различно стартово URL, за да могат да бъдат създадени различни физически представяния на приложението върху клиентския компютър.
- Бъдещо разгръщане – новата версия е достъпна, но трябва да бъде приложена в определен момент от време. В този случай трябва да се позволи на клиента да свали новата версия и да я кешира. По този начин смяната на версиите в определен бъдещ момент няма да бъде афектирана от натоварването на сървъра за разгръщане.
- Разгръщане за множество групи - често срещано е изискването да се предоставя различна приложна функционалност на различни типове потребители, спрямо тяхната роля. Благодарение на базираната на роли сигурност, която предоставя .NET Framework, може да се определи ролята на потребителя по време на изпълнение и дадени опции да се забранят или разрешат, в зависимост от нея. При използването на Composite UI Application Block, могат да се сложат роли за достъп на различните модули в профил каталога. Каталогът може да се използва, за да се определи кои модули са разрешени за даден потребител.

Една от задачите на дипломната работа бе да се намери решение на проблема как умният клиент трябва да бъде проектиран така, че да разпознава потребителя и да зарежда само тези модули, за които потребителят има права. Използването на Composite UI Application Block и неговите профилни каталози комбинирано с използването на ClickOnce сценария, разгръщане за множество групи, дават решението на този проблем.

При ниски до средни натоварвания, публикуването на приложението на един сървър е приемлив сценарий, но при големи компании със стотици работници натоварването върху сървъра за разгръщане при актуализация ще е огромно и ще зависи от големината на актуализацията и броят на потребителите. Това ще се отрази неблагоприятно върху времето за издърпване на обновяването и стартиране на приложението. Има няколко варианта за справяне с този проблем:

- Мрежово балансиране на натоварването – представлява разполагане на ClickOnce публикацията на няколко идентично конфигурирани сървъра със един и същи мрежов адрес. Това, че натоварването се разпределя между няколко сървъра, ще е прозрачно за потребителя и за ClickOnce адресирането. Този вариант може да се приложи, както за първоначалното разгръщане, така и за последващи актуализации.
- Публикуване на множество сървъри – Публикацията може да е разположена на няколко сървъра и отделните части на организацията да свалят и стартират приложението от различни физически адреси. Този вариант също се отнася за първоначалното разгръщане и за по нататъшни актуализации. По този начин може да се адресира поетапното разгръщане, което бе разгледано по-горе.
- Сваляне на новите версии и подобренията във фонов режим– чрез програмното API ClickOnce може да проверява за актуализации и те да се свалят във фонов режим. Това няма да намали натоварването на сървъра. Ползата е най-вече за потребителя, на когото няма да му се налага да чака за обновяване при стартиране на приложението. Този подход може да се използва само за актуализации. Неприложим е за първоначално разгръщане на приложението.
- Сваляне на новите версии във фонов режим по план – Може да се използва ClickOnce програмното API, за да се проверява за обновявания и те да се свалят във фонов режим. Може да е необходимо проверките да се случват в различно време на различни клиентски компютри. Това може да бъде реализирано чрез случаен период на изчакване в кода на актуализацията или чрез отдалечена планираща услуга, която клиентският компютър извиква преди да се опита да провери дали има налична актуализация. Този вариант се отнася само за актуализациите, не покрива първоначалните разгръщания. За разлика от предния вариант, той води до редуциране на натоварването върху сървъра.

Повече информация по въпроса и примерни реализации могат да бъдат намерени в онлайн библиотеката MSDN (MSDN, Click Once Deployment, 2005) и книгата на Браян Ноулс (Noyes, 2006).

4.5.Обобщение

В тази глава се разгледа CAB и SCSF и как с тяхна помощ могат да се разработват модулни умни клиенти. CAB предоставя ясен подход за изграждане на: скелет на приложението, общ за всичко модули; подход за разработване на модули; - услуги, локализиращи модулите и - готова инфраструктура, за поддържането на такъв модел. Той позволява потребителски елементи като менюта, ленти за инструменти и работа, да бъдат разширени с елементи от добавените към приложението модули. SCSF надгражда CAB. Фабриката автоматизира еднотипни дейности, които програмистът всеки път трябва да извършва. Благодарение на сигурността базирана на роли, профилните каталози, поддържани от CAB и ClickOnce технологията може да реализира сценарий, в който да се зареждат само тези модули, за които клиентът има права. Но ClickOnce не поддържа само този сценарий на разгръщане, той поддържа много повече варианти и възможности за разгръщане, отколкото може да предостави едно уеб приложение.

5. Примерно приложение

Task Viewer е примерен умен клиент за управление на проекти. То е изградено на базата на Smart Client Software Factory, Composite Application Block, използва WCF услуги, реализирани с помощта на Web Service Software Factory (Microsoft, Web Service Software Factory, 2007). Имплементирано е на C#, използва .NET Framework 2.0. За база данни се използва Microsoft SQL Server 2005 (Microsoft SQL Server, 2005). Състои се от два модула административен и приложен. Административният се използва за добавяне на нови потребители, преглед и изменение на данните им. Приложният модул позволява разглеждане, модифициране и добавяне на проекти и задачи, които се споделят между потребители. Приложението може да бъде използвано при разнообразни сценарии от система за следене на дефекти или управление на проекти, до система за отчитане на статуса на даден проект пред клиенти. Основната цел на приложението е да илюстрира решение на проблемите свързани с разработката на непостоянно, свързани умни клиенти.

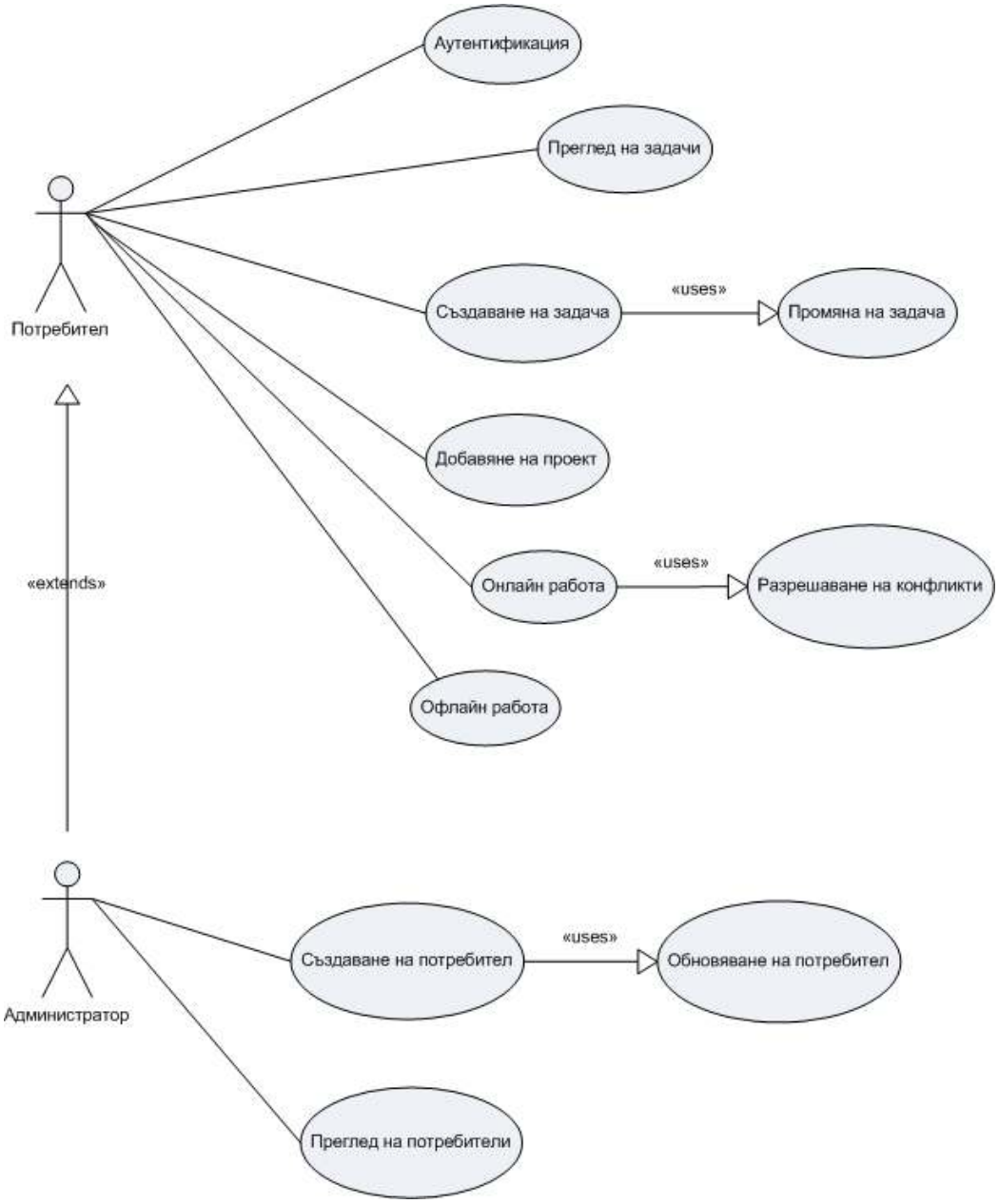
Task Viewer илюстрира следните техники:

- Подход ориентиран към услуги
- Имплементиране на WCF услуга за работа със умен клиент
- Разработване на модулни приложения
- Използване на асинхронна комуникация с уеб услугите
- Работа в офлайн и онлайн режим
- Използване на профилен каталог за специфициране на условията на разгръщане
- Използване на агент на услугите
- Използване на MVP шаблон
- Справяне с конфликтите в данните

5.1. Бизнес сценарии, поддържани от Task Viewer

Приложението поддържа различни случаи на употреба свързани с управлението на проекти, задачи и работата с потребители. Има два актьора. Това са потребител и администратор, който разширява задълженията на

потребителя. Фигура 3 показва диаграмата на случаите на употреба, които примерното приложение адресира.



Фигура 3: Task Viewer - случаи на употреба

Таблица 5 описва детайлно случаите на употреба. Всеки случай на употреба има име, което го отличава от другите случаи на употреба. Той е описан чрез входни условия, участващи актьори, действия, които трябва да бъдат изпълнение и алтернативни варианти на случая на употреба, ако има такива.

Таблица 5: Детайлно описание на случаите на употреба

Име		
1. Аутентификация	Входни условия	Приложението не е стартирано.
	Актьор	Потребител, Администратор
	Действия	Потребителят стартира приложението. То изисква потребителско име и парола. Потребителят въвежда потребителско име и парола. Приложението ги използва, за да определи ролята на потребителя. Валидни роли са служител и администратор. Приложението използва профил каталога и зарежда необходимите модули. Зареждат се само тези модули, за които потребителската роля има права.
	Алтернатива	Потребителят не съществува. Система извежда съобщение за невалиден потребител. Потребителят натиска Cancel бутона и приложението се затваря
	Изходни условия	Приложението стартира подходящия екран за ролята на актьора
2. Създаване на задача	Входни условия	Потребителят е влязъл в модула Task Manger и е избрал проект.
	Актьор	Потребител
	Действия	От менюто се избира Създай нова задача. Отваря се формата Task, в която потребителят може да попълни информация за задачата, обобщение, описание, приоритет, отговорен за задачата човек, краен срок на задачата Потребителят натиска Ok, задачата се създава
	Алтернативни действия	Потребителят натиска Cancel, прозорецът се затваря, системата не създава задача.
3. Промяна на задача	Входни условия	Потребителят е влязъл в модула Task Manger, заредил е задачите за определен проект.
	Актьор	Потребител
	Действия	Потребителят кликва два пъти върху задача, системата зарежда детайлите за

		задачата. Потребителят има възможност да промени крайния срок на задачата, отговорния човек за задачата, приоритета, обобщението, описанието и прогреса. Формата е подобна на формата за създаване на задача. Потребителят натиска Ok, информацията за задачата се обновява.
	Алтернативни действия	Потребителят натиска Cancel, прозорецът се затваря, системата не обновява задачата.
4. Добавяне на проект	Входни условия	Потребителят е влязъл в модула Task Manger.
	Актьор	Потребител
	Действия	Избира от менюто Създай проект. Системата отваря формата Project. Потребителят въвежда информацията за проекта и натиска Ok. Системата затваря формата и създава проекта.
	Алтернативни действия	Потребителят натиска Cancel. Системата затваря прозореца и не създава проект.
5. Онлайн работа	Входни условия	Потребителят работи в офлайн режим.
	Актьор	Потребител
	Действия	Потребителят натиска бутона за преминаване в онлайн състояние. Системата синхронизира данните си със сървъра.
	Изходни условия	Системата работи онлайн, данните на клиента са синхронизирани със сървъра. Индикацията показва, че приложението работи в онлайн режим
6. Офлайн работа	Входни условия	Приложението работи онлайн.
	Актьор	Потребител
	Действия	Потребителят натиска бутон за работа в офлайн режим. Показва се форма, в която на потребителят се дава възможност да избере, кои проекти да бъдат кеширани на клиента.
	Изходни условия	Клиента работи в несвързано състояние. Само със

		проектите, които е избрал по-рано. Приложението индикира, че клиентът работи в офлайн състояние.
7. Разрешаване на конфликти	Входни условия	Настъпил е конфликт
	Актьор	Потребител
	Действия	Приложението изкарва форма за разрешаване на конфликти. Потребителят избира своя или сървърният вариант. Системата запазва избора на потребителя.
	Изходни условия	Съхранен е вариантът, който потребителят е избрал на фаза разрешаване на конфликти.
8. Преглед на задачите	Входни условия	Потребителят е влязъл в модула Task Manger.
	Актьор	Потребител
	Действия	Екранът е разделен на три части. В дясно е показано дърво с всички налични проекти. В дясно екрана е разделен на две, горният е Task Summary, а долният Task Details двата са празни. Потребителят кликва два пъти върху даден проект, в Task Summary се зареждат задачите за него. Детайлите остават празни. В Task Summary се избира задача, нейните детайли се зареждат в Task Details
9. Преглед на потребители	Входни условия	Потребителят е влязъл в модула User Management. В дясната горна част се зареждат всичките потребители в системата. User Details частта е празна.
	Актьор	Администратор
	Действия	Актьорът кликва върху потребител. Неговите детайли се зареждат в User Details
10. Създаване на потребител	Входни условия	Потребителят е влязъл в модула User Management.
	Актьор	Администратор
	Действия	От менюто потребителят избира Създай нов потребител. Системата зарежда User форма. Актьорът запълва информацията за потребителя и натиска Ok. Системата затваря прозореца и създава потребителя

	Алтернативни действия	Актьорът натиска Cancel. Системата затваря прозореца и не създава потребителя
11. Обновяване на потребители	Входни условия	Потребителят е влязъл в модула User Management.
	Актьор	Администратор
	Действия	Актьорът два пъти кликва върху името на потребителя. Отваря се User формата. Актьорът променя информацията за потребителя и натиска Ok. Системата затваря формата и обновява информацията за потребителя.
	Алтернативни действия	Актьорът променя информацията за потребителя и натиска Cancel. Системата затваря формата без да обновява информацията за потребителя.

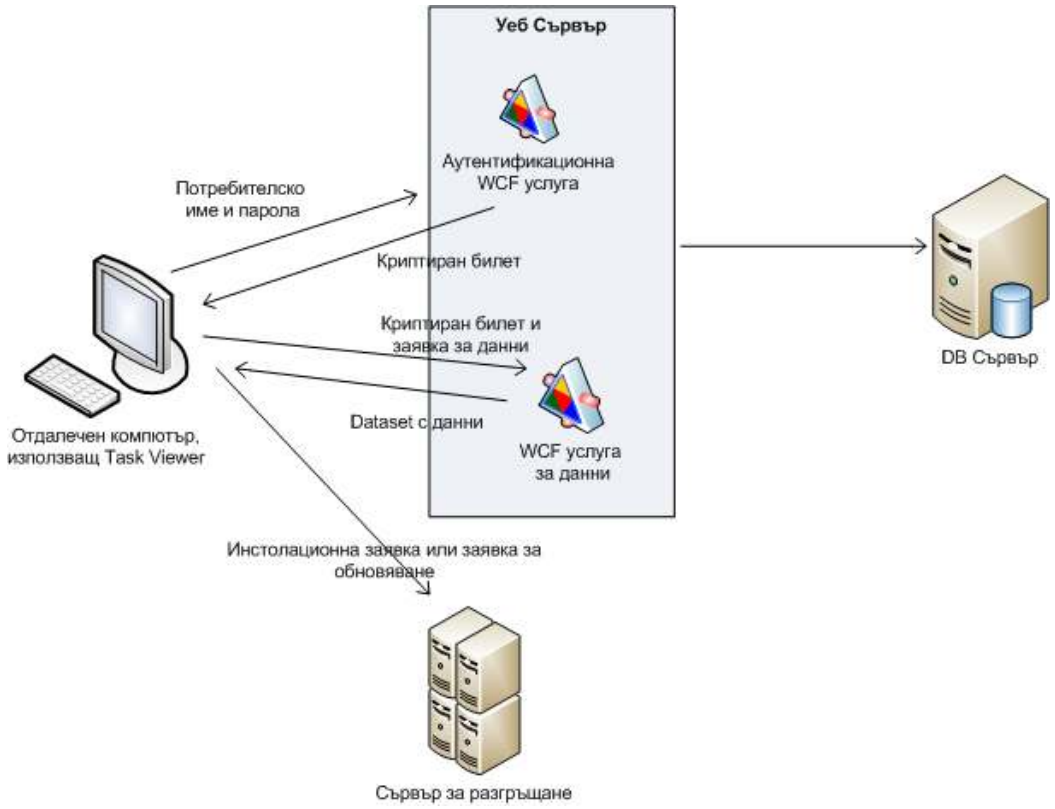
5.2. Архитектура на приложението

Task Viewer се състои от 3 главни компонента, умно клиент приложение, разположено на клиентски компютри; WCF уеб услуги, с които приложението комуникира и база данни, с която комуникират уеб услугите.

За проектиране на умното приложение е избран подхода ориентиран към услуги. Двете проектирани и имплементирани услуги са разработени с CRUD интерфейс и поддържат атомарни методи. Приложението имплементира ръчен мениджмънт на връзката и изглаждане на конфликтите на клиента. Аутентификационната услуга се грижа за аутентифициране на потребителя и управление на потребители, използва се основно от административния модул. Клиентът се аутентифицира като изпраща потребителско име и парола, а му се връща билет, който той изпраща при последващи заявки. Услугата за данни се използва от приложният модул. Тя предоставя метода за управление на проектите.

Приложението се разгръща онлайн, като версиите и актуализацията се свалят от сървър за разгръщане, на който е публикувано приложението.

Фигура 4 илюстрира архитектурата на системата.

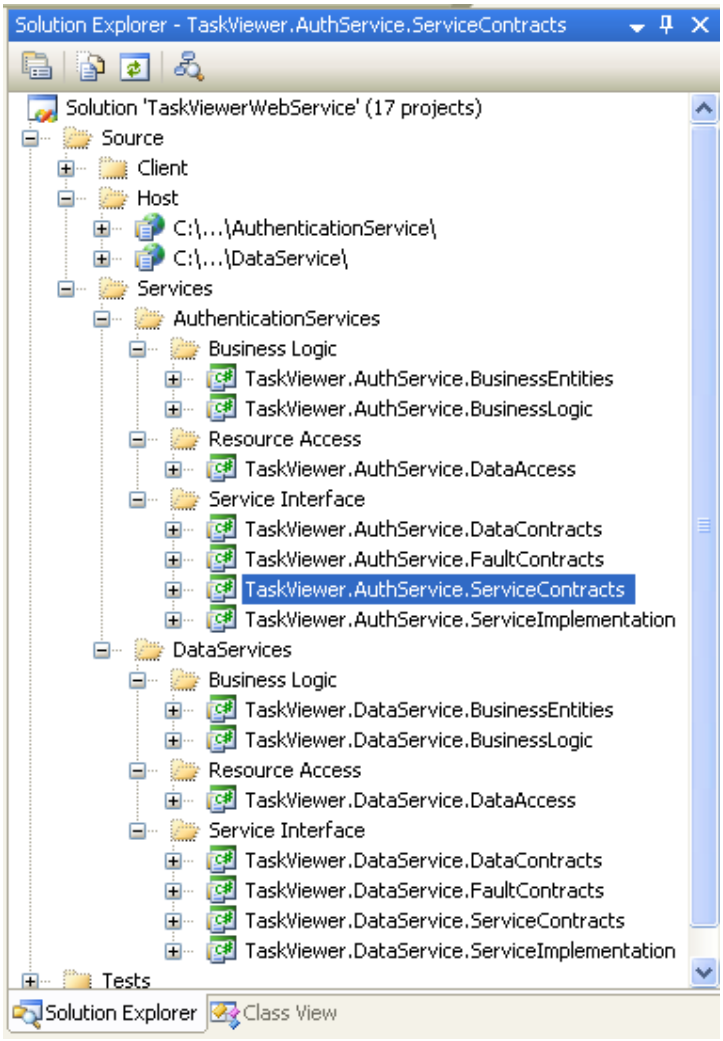


Фигура 4: Task Viewer – архитектура на приложението

5.3.Проектиране на уеб услугите, използвани от примерното приложение

5.3.1. Изглед на разработката

Примерното приложение използва WCF уеб услуги. Решението Task Viewer Web Services предоставя две услуги: услуга за аутентифициране и услуга за данни. Проектите реализиращи WCF услугите са създадени с помощта на Windows Communication Foundation (WCF) пакет за разработка предоставен от Web Service Software Factory (MSDN, Web Service Software Factory, 2006). С помощта на този пакет е изградена структурата от проекти показана на фигура 5.

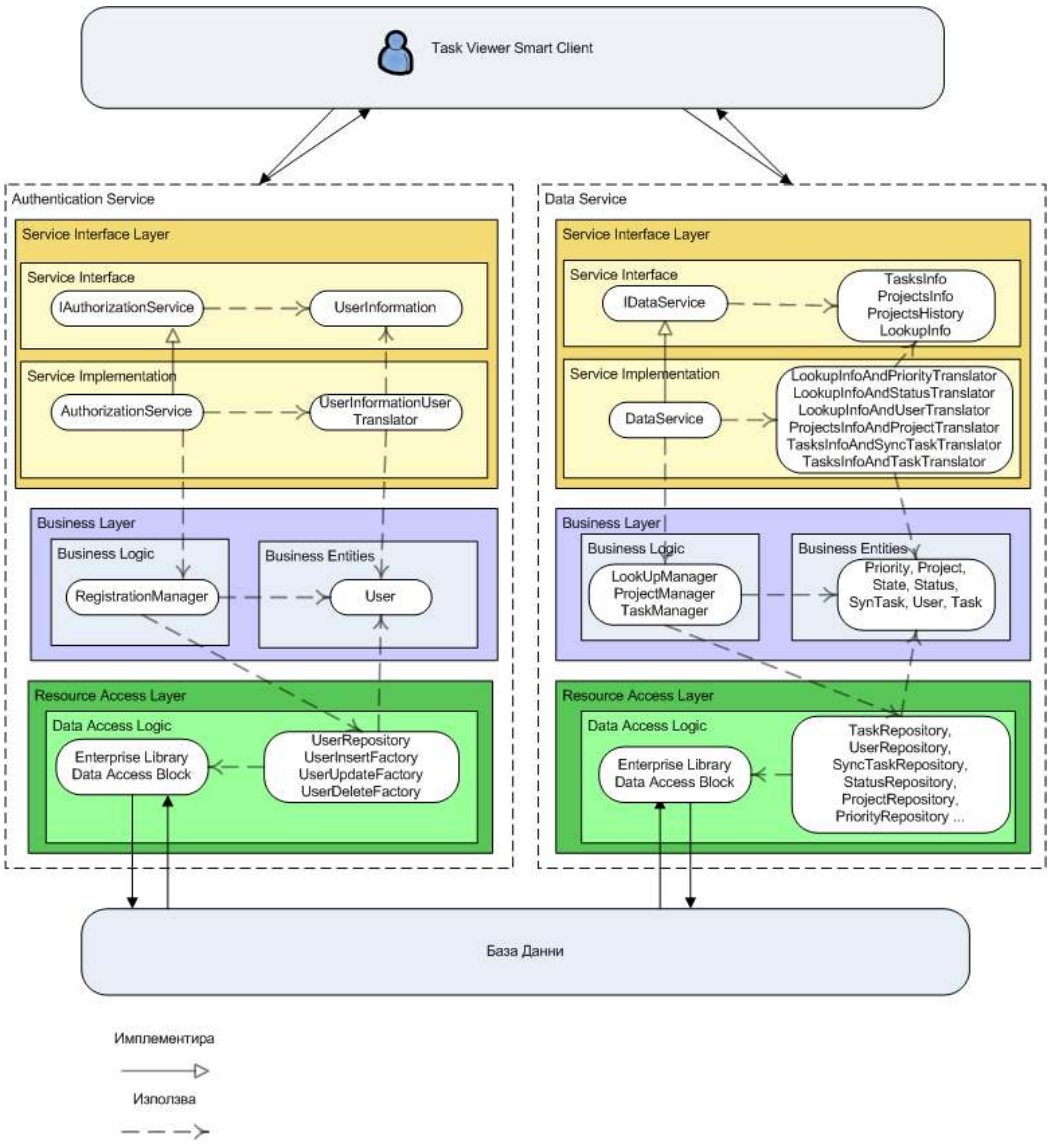


Фигура 5: Структура на проектите на Task Viewer Web Service имплементацията, изградена с използването на WCF пакети

Както е показано на фигура 5, проектите са разделени в три папки, всяка представяща отделен слой от архитектурата. Във всяка папка има няколко проекта използвани, за да се генерират библиотечни асемблита. Класовете имплементиращи дизайн шаблоните допълнително са добавени във подходящите проекти.

5.3.2. Контекстен Изглед

WCF проекта съдържа две WCF Услуги, всяка от тях е трислойна, състои се от слой на услугата, бизнес слой, и слой за достъп до ресурсите. Фигура 6 илюстрира слоевете и показва основните класове включени в тях.



Фигура 6: Слоеве в контекста на Task Viewer услугите

5.3.2.1. Слой на услугата

Този слой осигурява публичен интерфейс, който клиентското приложение използва, за да си взаимодейства с услугата. Слоя на услугата се състои от следните проекти.

Fault Contracts – този проект съдържа контрактите, използвани от операциите на услугата, в случай на грешка.

Data Contracts – този проект съдържа компонентите, представляващи структурата на входно-изходните съобщения. Тези структури от данни

идентифицират специфични типове данни и връзки, които се използват, за да се дефинира съобщението.

Service Contracts – този проект съдържа контрактите за услугата, които дефинират операциите, поддържани от услугата. Всяка операция се състои от контракт на входно-изходните съобщения и данни и политики за обработване на грешката.

Service Implementation –съдържа компоненти, които имплементират контрактите за услуги и взаимодействат с бизнес компонентите чрез използването на шаблона за транслиране на обекти. Използват се класове транслатори, които трансформират контрактите за данни до бизнес обектите и обратно.

5.3.2.2. Бизнес слой

Бизнес слоя съдържа компоненти, които имплементират бизнес логиката и дефинират бизнес обектите, използвани от бизнес логиката. Той обединява следните елементи, отделени в различни проекти:

BusinessLogic – съдържа имплементация на дефинираните бизнес правила и бизнес процеси.

Business Entity project – съдържа бизнес класовете използвани от бизнес домейна.

5.3.2.3. Слой за достъп до ресурси

Този слой съдържа елементите, които взаимодействат с базата данни. Проекта DataAccess отделя базата данни от приложението чрез шаблоните Repository и Фабрика. Repository компонентите използват фабриките, за да създават обекти заявки, които изпълняват операции срещу базата данни. Два типа на фабрични компоненти се използват в слоя за достъп до ресурси: едните създават бизнес обекти, а другите, се използват за създаване на обекти заявка.

5.3.3. Логически изглед

Тук ще се разгледа само слоя за услуги и първостепените класове във него за ауторизационата услуга. Класовете във всеки от слоевете са показани чрез клас диаграми в Приложение

Приложение А: Клас диаграми на аутентификационната услуга и Приложение Б: Клас диаграми на услугата за данни към дипломната работа.

Примерът на Фигура 7 използва шаблона транслатор. Контракта за данни използва още тип на съобщението за да обработи съобщението заявка и да върне съобщение отговор Вместо да се пращат индивидуални типове данни като параметри в отговора, типа съобщение осигурява контейнер, който се използва за да капсулира данните.

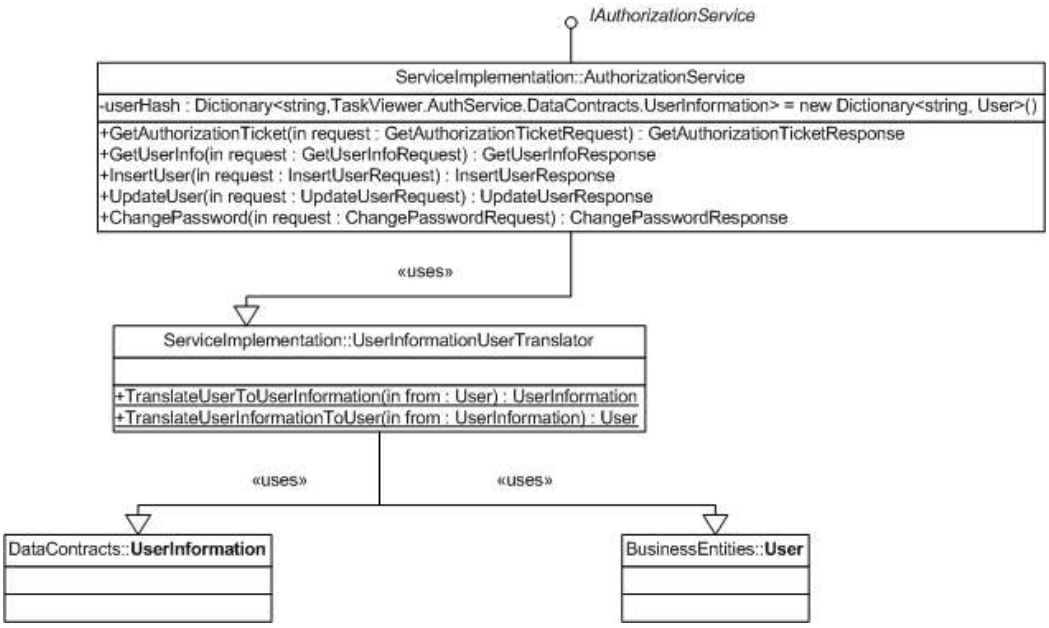
Следният лист описва класовете илюстрирани на Фигура 7

AuthorizationService е имплементация на аутентификационнната улсуга дефинирана чрез използването на интерфейса IAuthorizationService.

AuthorizationService класа използва UserInformationUserTranslator, за да направи транслация между UserInformation и User обектите.

User – бизнес обект, с който работи бизнес слоят.

UserInformation – контракт за данни на услугата.

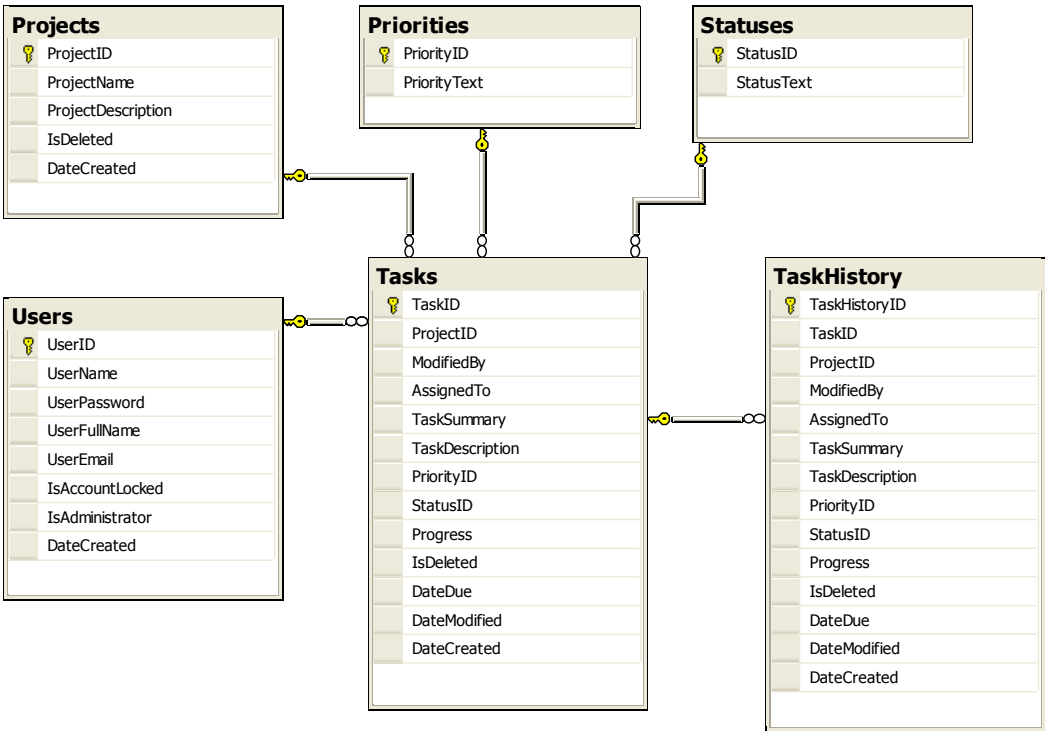


Фигура 7: Клас диаграма за AuthorizationService слоя на услугата

5.4. База данни

Цялата информация, която се споделя между потребителите, се съхранява в Microsoft SQL Server 2005 база данни.

Фигура 8 показва схема на базата данни.



Фигура 8: Task viewer - схема на базата данни

Task Viewer използва съхранени процедури за извличане и обновяване на данните в таблиците. Съхранените процедури осигуряват разделяне на базата от слоя за достъп. Това осигурява лесно поддържане, тъй като промяна в структурата на базата данни, ще бъде прозрачна за компонентите за достъп. Използването на съхранени процедури осигурява предимства при производителността, благодарение на кеширането на ниво база данни и намаляване на заявките към базата, благодарение на капсулирането на логиката в съхранените процедури.

5.5. Task Viewer

Примерното приложение се състои от стандартен скелет на приложение и два модула, които могат да бъдат включвани и изключвани във него. Приложението зарежда модулите си на база конфигурационен профайл каталог.

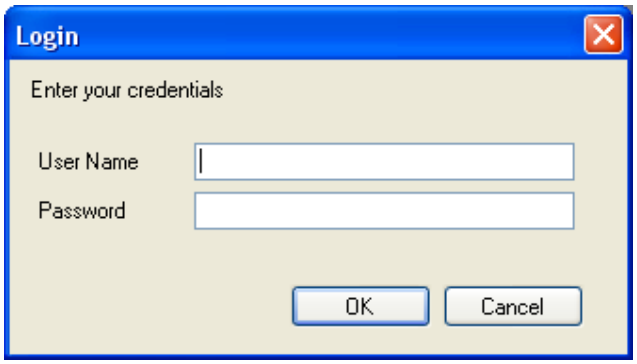
Компонентите си комуникират чрез използването на събития. Приложението използва споделени услуги, за да осигури обща функционалност на модулите. Класовете на потребителския интерфейс имплементират Модел Изглед Представител шаблон.

Функционалността предоставен чрез потребителския интерфейс се определя от ролята на текущия потребител. Структурата на скелета се състои от три работни места. Приложението показва изгледи във всяко работно място и в допълнителен прозорец. Действията на потребителя в един изглед, водят до промяна на друг изглед (навигация между изгледи). Статус лентата описва състоянието на комуникацията с уеб услугата. Менютата и лентите за инструменти стартират общите за приложението команди.

Приложението демонстрира кеширане на данни и поддържа Click Once разгръщане. Приложението извиква асинхронно уеб услугите.

5.5.1. Изглед на потребителският интерфейс

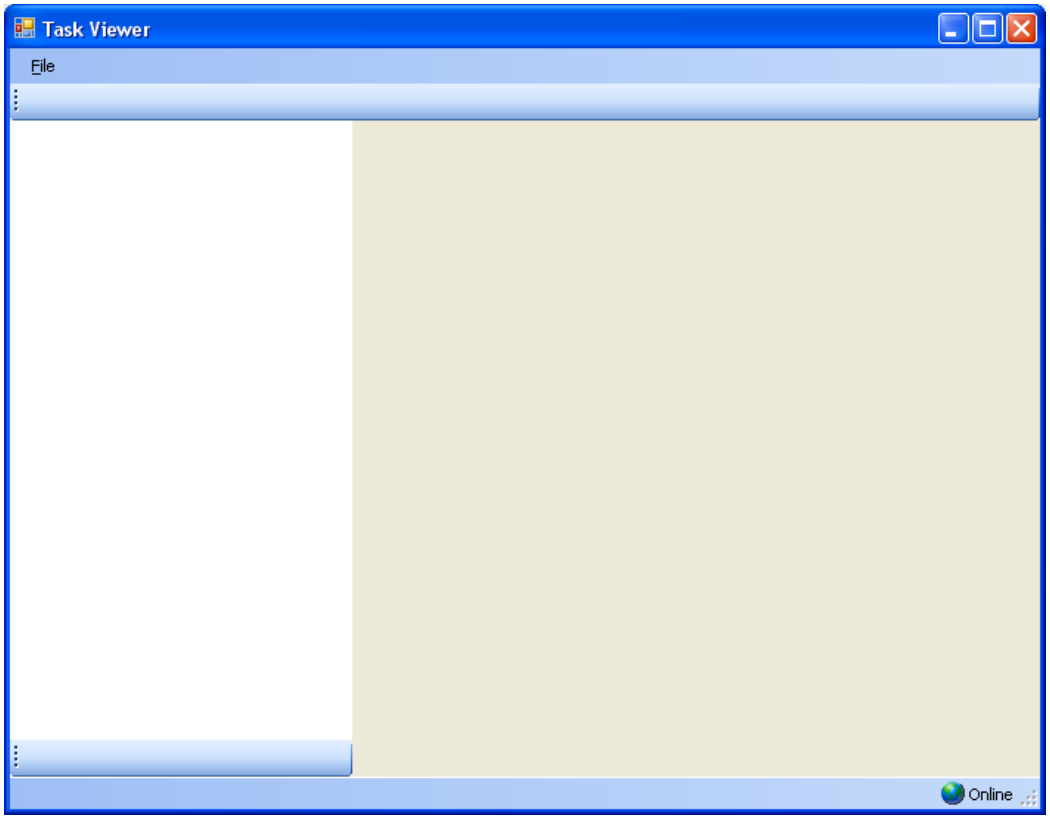
Когато Task Viewer се стартира, се показва форма за аутентифициране. Приложението подменя подразбраната имплементация на IAuthenticationService интерфейса с имплементация използваща WCF услугата за имплементиране. Ако потребителя е валиден се зарежда приложението със съответстващите за този потребител модули. Фигура 9 показва формата за аутентифициране.



Фигура 9: Аутентификационна форма

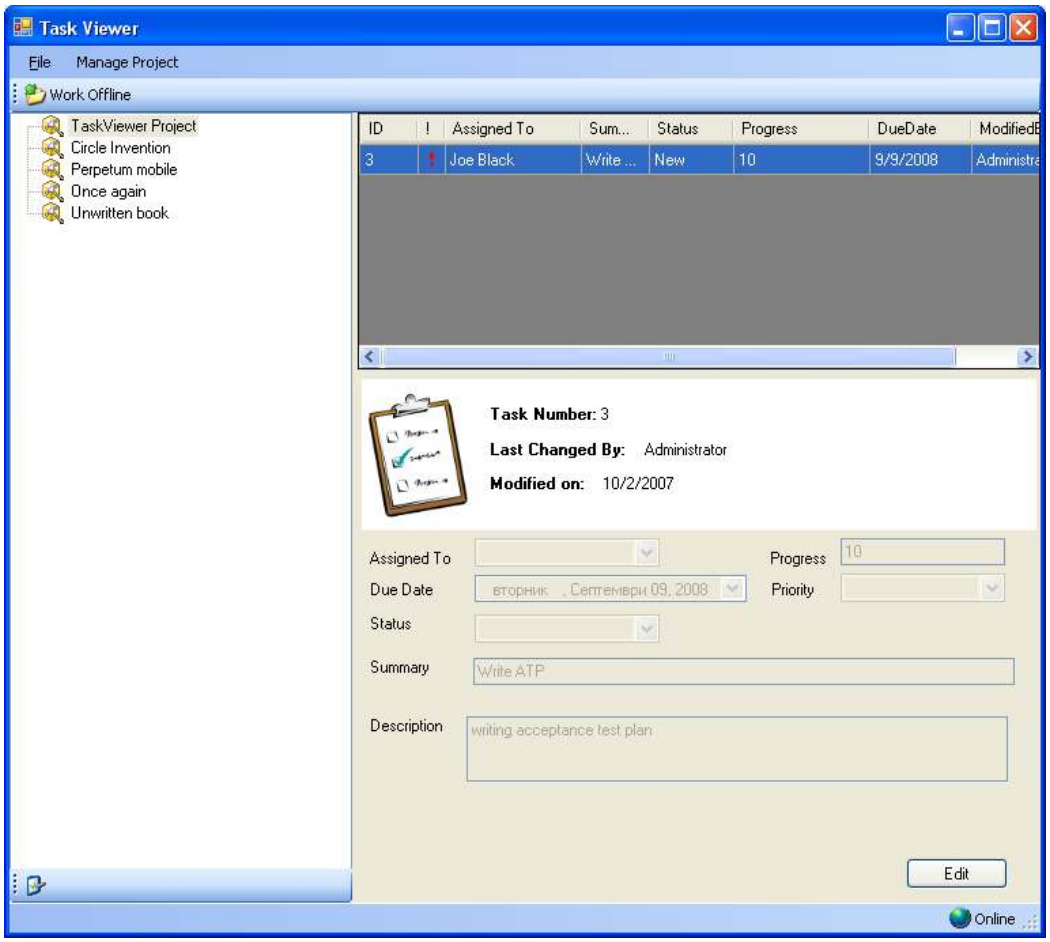
Скелетът на приложението се състои само от един DeckWorkspace, скелетът на приложението се определя от Infrastructure.Layout проекта. Той се състои от 3 DeckWorkspace форми: _navigationWorkSpace, _summaryWorkspace и _detailWorkspace.Това са работните места, които се подменят с излгедие на съответния модул. Четири са UIExtensionSite, котио могат да бъдат разширени

от модулите. Това са: _mainMenuStrip, _mainToolStrip, _navigationToolStrip и Status Strip. Скелетът съдържа едно File меню с опция Exit. На Фигура 10 е показано приложението заредено без модули, за да се види потребителската структура на приложението.



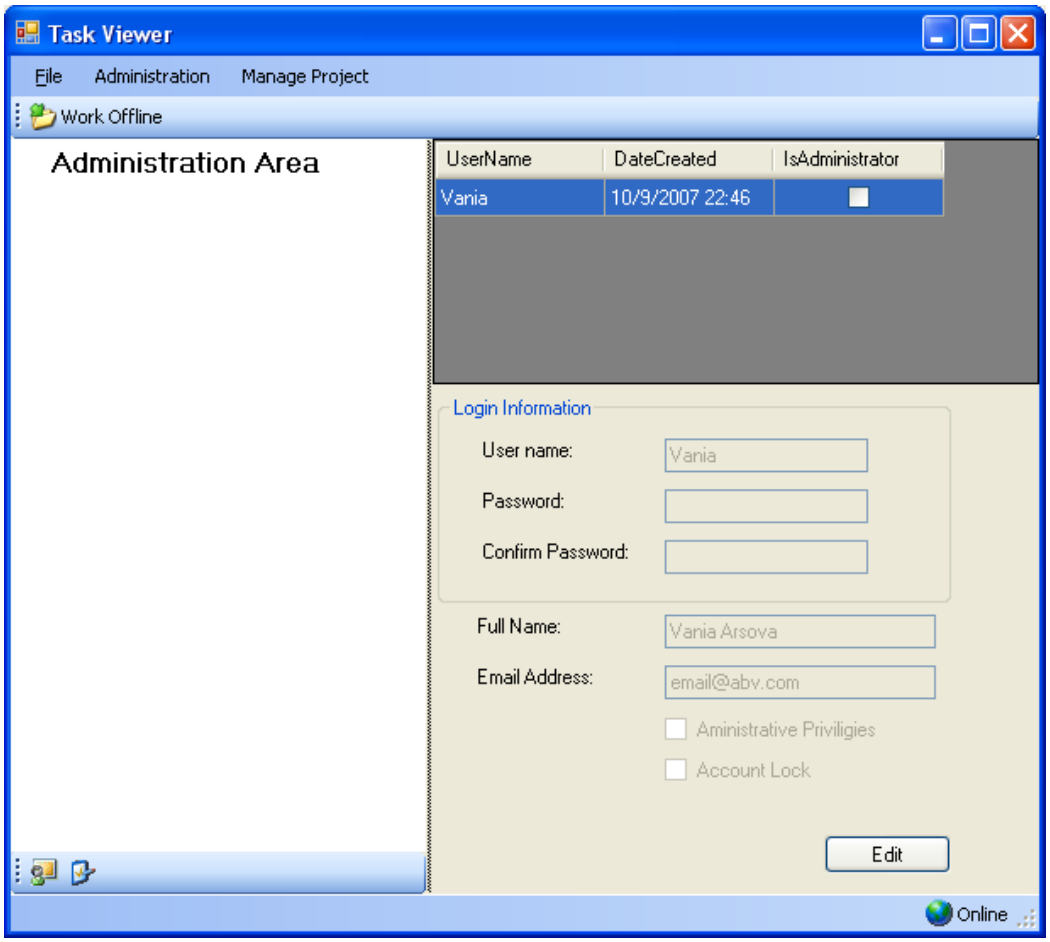
Фигура 10: Скелет и потребителска структура на приложението

На Фигура 11 е показано как ще изглежда приложението, ако бъде зареден Task Management модула. Той добавя към главното меню допълнително меню Manage Project с две опции: New Task и New Project. Към mainToolStrip се добавя бутонът Offline Selection, а към навигационната лента с инструменти иконка на модула. При кликване върху нея работните места на структурата на приложението се подменят с изгледите на Task Management модула(ProjectNavigation, TaskDetailsView, TasksReview).



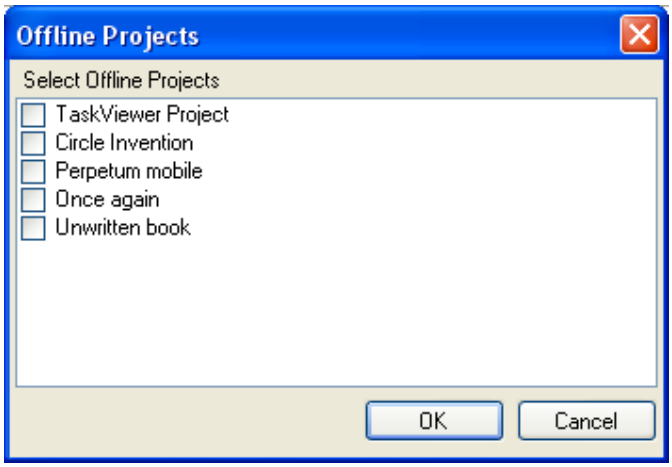
Фигура 11: Task Management Module

На Фигура 12 е показан изгледа на приложението, когато са заредени двата модула в него. Административният модул добавя ново меню: Administration, с опция New User, а в навигационната лента за инструменти добавя иконка на модула. При кликване върху нея в работните места на потребитеската структура на приложението се зареждат изгледите на административния модул (NavigationArea, UserDetails, UserList).



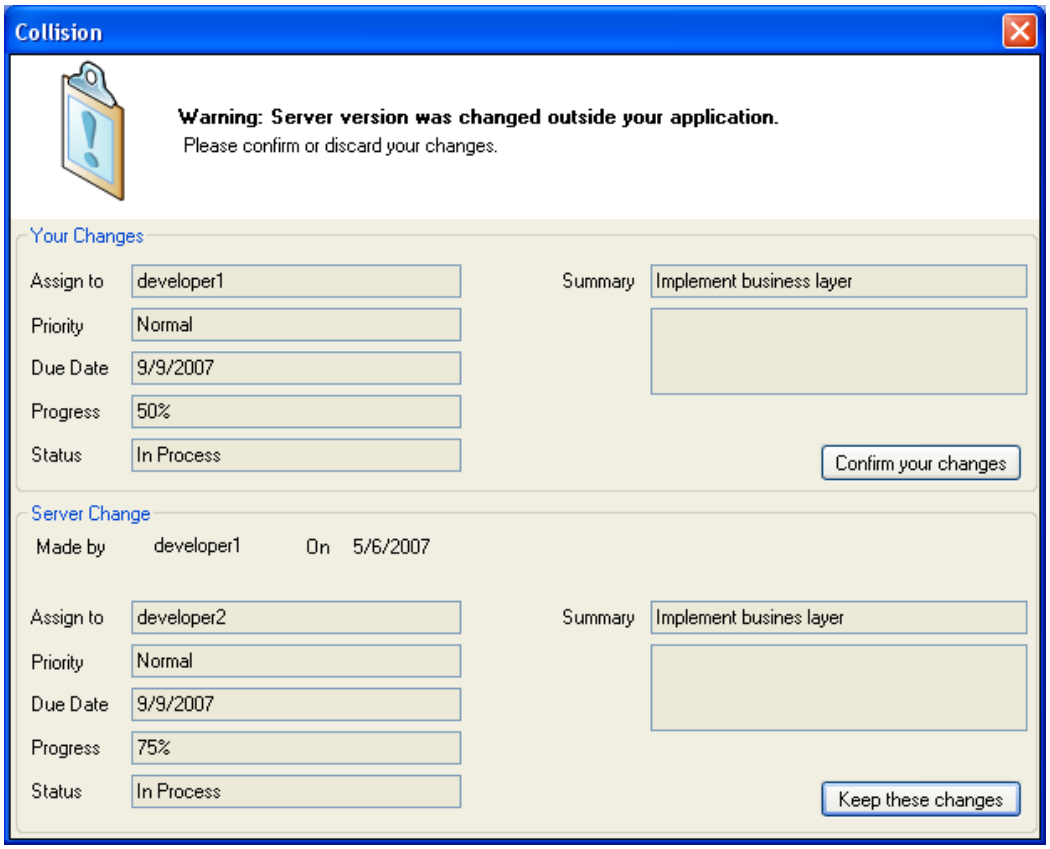
Фигура 12: Task Management и Administration module

Натискането на Work Offline бутона води до зареждане на изгледа OfflineSelectionView. Той се състои от CheckedListBox списък, в който може да се избере кои проекти ще бъдат достъпни в офлайн. При натискане на Ok, приложението преминава в офлайн режим. Преди това то зарежда избраните проекти в кеша. Ако в кеша присъстват допълнителни проекти, ги изтрива от кеша. В статус лентата се сменя статуса на приложението, а лентата за инструменти бутона Work Offline, се подменя с бутон Work Online. Докато е в офлайн режим приложението започва да прави всички заявки към локалния си кеш. Формата за избиране на офлайн проекти може да се види на Фигура 13.



Фигура 13: Форма за избор на проектите, които ще са достъпни в офлайн режим

Кликването на бутона Work Online, води до преминаване на приложението в Онлайн режим. Кеша на клиента се синхронизира с базата данни на сървъра. При възникване на колизия приложението информира потребителя като показва данните, които си противоречат. За тази цел CollisionView изгледът се зарежда в модален прозорец, в който потребителят може да избере, кои данни да запази.

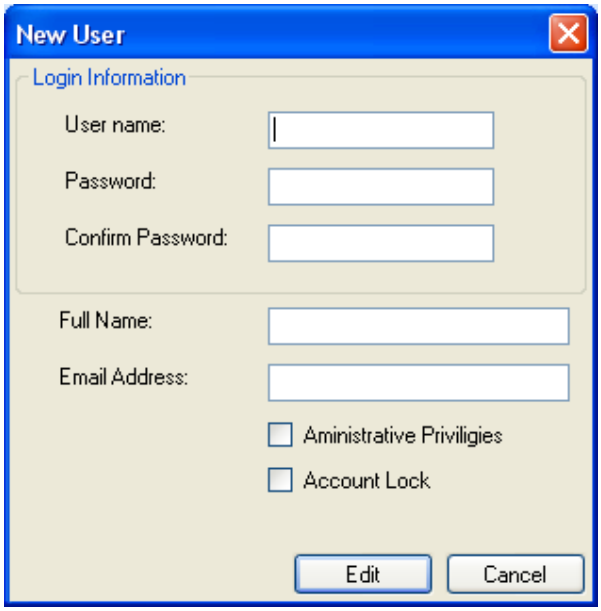


Фигура 14: Форма за разрешаване на възникнали проблеми

На фигура 15 е показан модалният диалог, който се показва при избор на опцията New Project. Изгледът NewProjectView се зарежда в модалният диалог, а неговият представител се грижи за създаването на нов проект

Фигура 15: Форма за създаване на нов проект

На фигура 16 е показана формата за създаване на нов потребител, тя преизползва UserDetails изгледът. Формата New Task преизползва TaskDetailsView. Техните представители се грижат за създаването на съответните обекти User и Task.



Фигура 16: Форма за създаване на нов потребител

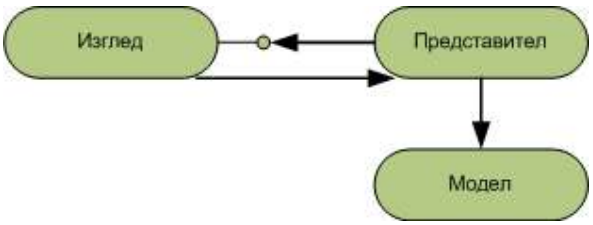
5.5.2. Логически изглед

Логическият изглед съдържа описани на шаблоните, които са имплементирани в примерното приложение. Това са Модел Изглед

Представител, агент на услуги, асинхронни комуникации с уеб услугите и навигация между изгледи.

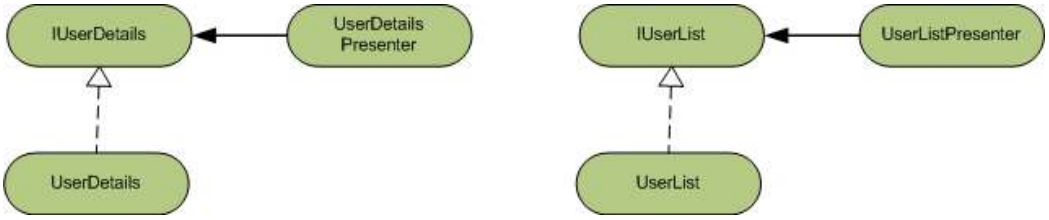
5.5.2.1. Шаблон Модел Изглед Представител

Формите в умното приложение, често съдържат различни контроли, обработват потребителски събития и съдържат логика променяща контролите в отговор на потребителските взаимодействия. Ако този код се напише директно в класът на формата, това ще направи класът сложен и труден за тестване. Този проблем би се решил, ако отговорностите за визуалното показване и поведението се разделят в различни класове. Изгледът управлява контролите във формата и препраща събитията към класа представител. Представителят съдържа логика за обработка на събитията и манипулира състоянието на изгледа. Представителят използва модела, обикновено моделът е състоянието на приложението представено от бизнес обектите. Логическият изглед на шаблона е показан на фигура 17.



Фигура 17: Логически изглед на шаблона Модел Изглед Представител

Всеки изглед в примерното приложение има асоцииран представител. Когато обектът изглед се създава, Object Builder създава представител, който се асоциира с този изглед. Представителят комуникира с изгледа чрез неговия интерфейс. Когато изглед обектът се добавя към работен обект, се добавя и представителят. Когато изгледът се унищожава, той унищожава представителя си. Изгледите и представителите за Administration Module могат да бъдат видени на фигура 18. Изгледите и представителите на другите модули могат да бъдат намерени в Приложение B: Task Viewer диаграми.

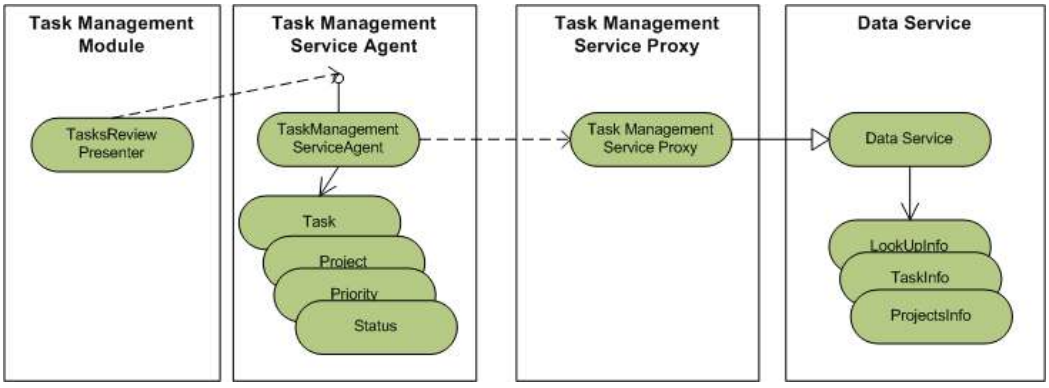


Фигура 18: Изгледи и представители на Administration Module

5.5.2.2. Агент на услуги

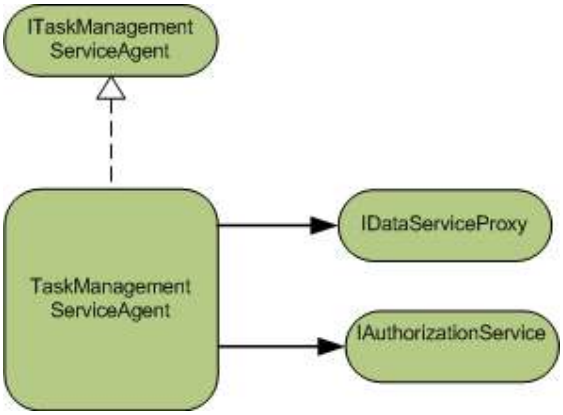
С Visual Studio може да се генерират проксита за използваните уеб услугите. Примерното приложение не използва DataService директно, а вместо това използва междинен клас, агент на услуги(Task Management Service Agent).

Агентите на услуги са класове, които осигуряват допълнителна функционалност, която помага на клиента да взаимодейства с уеб услугата. Агента на услуги в примерното приложение имплементира кеширането и офлайн операциите. Той дефинира опростен интерфейс, не предоставя всички методи, които Visual Studio е генерирал в проксито за уеб услугата. Например GetLookupTables е разбито на GetStatuses, GetPriorities, GetUsers. Агентът на услуги предпазва клиента от рефериране на типовете данни предоставени от уеб услугите. Генерираното от Visual Studio прокси съдържа типове данни дефинирани от уеб услугата като LookupInfo, TaksInfo. Клиентът има собствени типове данни като Priority, Status, Task. Агентът се грижи за трансформирането на обектите от услугата в обекти на клиента(например трансформацията от LookUpInfo обект в обекти Priority и Status). Разделението на логиката е показано на фигура 19.



Фигура 19: Разделяне на агента на услуги от прокси интерфейса

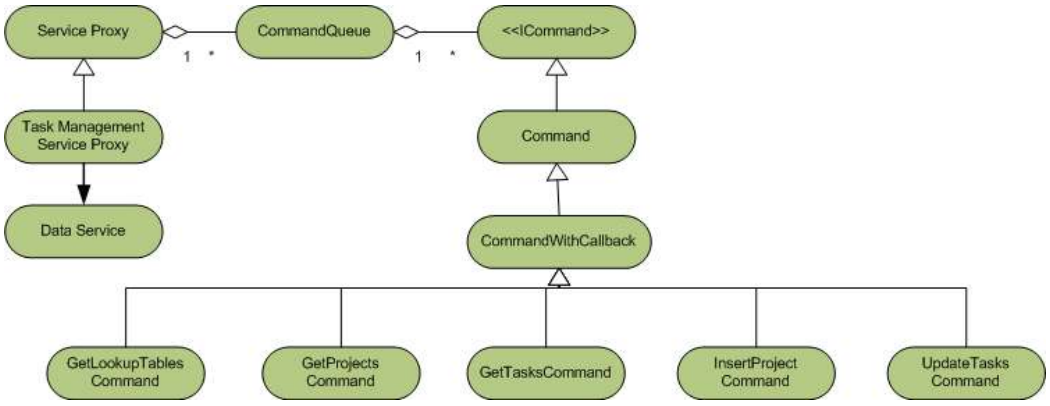
Интерфейсът ITaskManagementServiceAgent дефинира функционалността предоставена на клиента, той се имплементира от TaskManagementServiceAgent класа. Агента на услуги комуникира с две услуги, едната имплементираща IDataServiceProxy, а другата IAuthorizationService. Фигура 20 илюстрира тази имплементация.



Фигура 20: Task Management Service Agent

5.5.2.3. Асинхронни комуникация с уеб услуги

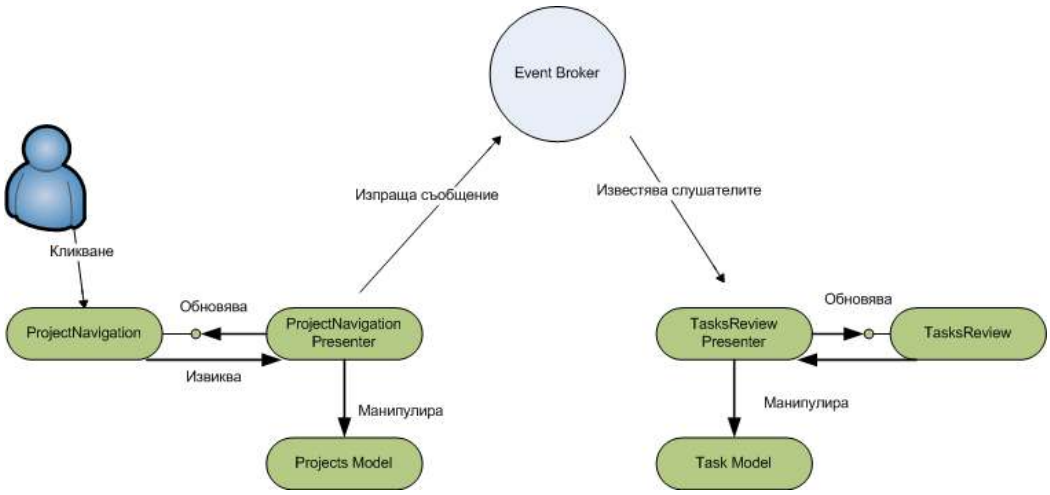
Уеб услугите могат да бъдат извиквани асинхронно на базата на генерираният прокси клас. Той имплементира общия за .Net Framework шаблон, асинхронно извикване, но той не дава възможност за специфициране на time-out стойности за асинхронни уеб услуги. Класът ServiceProxy, който се съдържа в приложението, осигурява шаблон асинхронно извикване, който поддържа time-out. Това е реализирано чрез опашка на командите. Класът TaskManagementServiceProxy наследява ServiceProxy и добавя команда за всяка уеб заявка в опашката на командите. Опашката изпълнява командите в отделна нишка. Командите използват генерираният прокси клас, за да извършат заявките. Тази логика е илюстрирана на фигура 21.



Фигура 21: Асинхронни уеб заявки със поддържане на time-out

5.5.2.4. Навигация между изгледи

Честа задача в потребителския интерфейс е обновяването на един изглед като резултат от действията на потребителя върху друг. Например TaskReview изгледът трябва да се обновява в зависимост от проекта избран в Project Navigation изгледа. Решението на тази задача е комуникация чрез събития между представителите на изгледите. Решението е представено на фигура 22. Project Navigation изгледа отчита, че потребителя е избрал някакъв проект и уведомява представителят си. Той изпраща събитие на Event Broker-a. Event Broker – а уведомява представителя на TaskReview изгледа, че е настъпило събитие. Като отговор на събитието представителят на TaskReview извлича информацията от модела и я показва в изгледа.



Фигура 22: Навигация между изгледите Project Navigation и TaskReview

5.5.2.5. Кеширане, съгласуване, справяне с конфликти и управление на връзката

Административният модул работи само в онлайн режим. Модулът за управление на задачи работи в офлайн и онлайн режим. Кеширането и офлайн функционалността са реализирани в агента на услуги. Класовете user, priority, status са реферирани данни, които рядко се променят. Затова за тях е избрано предварително дълготрайно кеширане. Зареждат при зареждане на приложението и се обновяват при преминавания в офлайн и онлайн режим. Краткотрайни данни са project и task, те се кешират противодействащо при поискване. Като при второ поискване се връща стойността от кеша и се прави заявка за обновяване към уеб услугата.

За съгласуване се използва оптимистично съгласуване реализирано в базата данни и уеб услугите. Използват се типизирани DataSets за предаването на данни. Предават се само променените данни, като се пази тяхната версия. На сървъра се използват оригиналната и променената версия за проверка дали е настъпила колизия с промените на други потребители.

Справянето с конфликти при краткотрайните данни се разрешава на клиента. Конфликтиращата информация се показва във форма, даваща възможност на потребителя да се запознае с конфликта и да реши коя версия на данните да запази.

Управлението на връзката се извършва ръчно. Потребителят има възможност да определи кога ще работи офлайн и кога онлайн. Може да определи коя информация да се кешира при офлайн работата му.

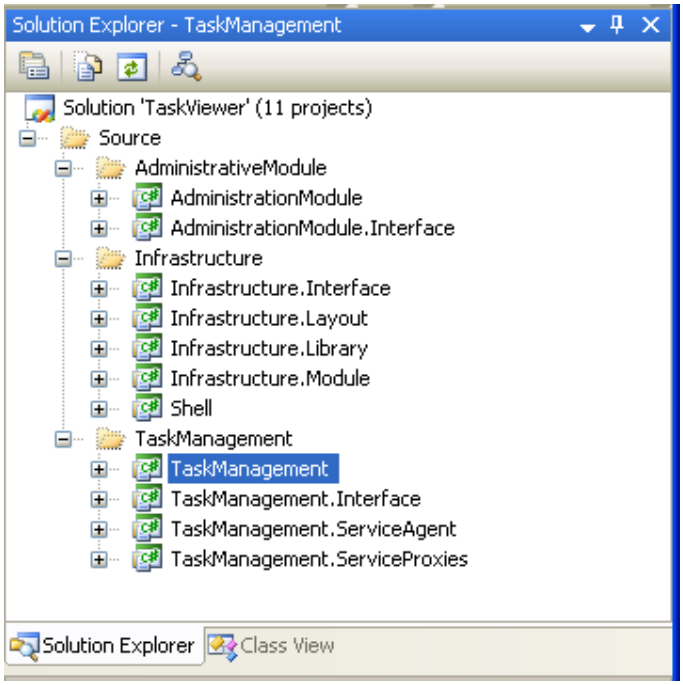
5.5.3. Изглед на имплементацията

Умният клиент Task Viewer е логически изграден от три части: скелет и два модула. Затова решението е разделено в 3 папки на решението:

- Administrative Module – тази папка съдържа проектите, които изграждат административния модул.
- Infrastructure – тази папка съдържа кодовете на скелета и общите компоненти, използвани от двата модула.

- Task Management – съдържа модула имплементиращ управлението на проекти и задачи.

Структурата на проекта може да се види на фигура 23.

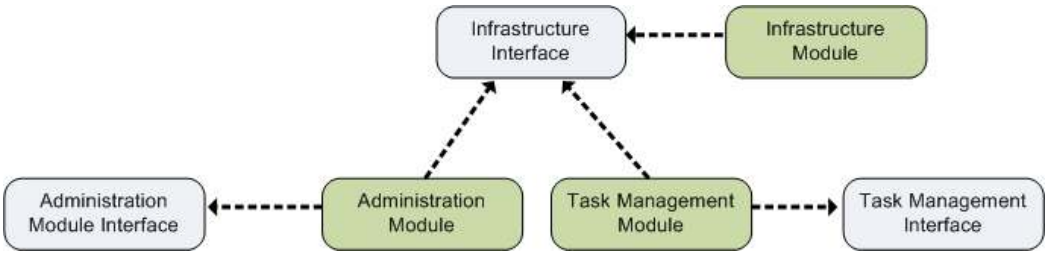


Фигура 23: TaskViewer - структура на проектите

Разделянето на модулите

Обикновено умните клиенти се състоят от множество модули разработвани от различни екипи. Някои модули имат връзки помежду си. Например административния модул използва аутентификационнат услуга предоставена от инфраструктурния модул. За да достъпи тази услуги, административният модул има референция към асемлито на интерфейсния модул. Всеки път, в който се промени инфраструктурният модул, административният трябва да бъде прекомпилиран. Този проблем е разрешен чрез разделяне на модула на две асеблита: интерфейсно и асембли, съдържащо имплементацията. Интерфейсното асембли съдържа: интерфейси на услугите, команди, константи (имена на команди, имена на съобщения, имена на UI extension sites, имена на работни пространства) и бизнес обекта User, който се използва от административния и модула, управляващ проекти. Чрез това разделяне, имплементацията може да се променя, без да се изисква прекомпилиране на зависимите модули. Зависимите модули трябва да се прекомпилират само ако се промени интерфейсното асембли. Разделянето на

интерфейсите от имплементациите за разработеното приложение е показано на фигура 24.



Фигура 24: Разделяне на интерфейсите от имплементацията на модулите

По долу са разгледани проектите във всеки един от модулите

Administration Module папка:

Administration Module – този проект съдържа елементите на потребителският интерфейс и съдържа компонентите свързани с управлението на информацията за потребителите

Administration Module Interface – проекта съдържа наименования на действия, команди, имена на съобщения, UI extension site наименования, и имена на работни пространства, които са публични интерфейси на Administration Module.

Infrastructure папка:

Infrastructure Interface – съдържа константи, класове и интерфейси, които се използват във всички модули на приложението.

Infrastructure Layout – дефинира структурата на потребителският интерфейс на приложението.

Infrastructure Library – съдържа общи компоненти използвани от модулите. Например съдържа прокси за ауторизационната услуга използвана от административния модул.

Infrastructure Module – проекта съдържа общите услуги за приложение и общи интерфейсни елементи като формата за ауторизация, появяваща се при стратиране на приложението.

Shell – съдържа скелета на приложението. В основата на имплементацията стои ShellApplication класа. Този клас наследява от SmartClientApplicationBase класа в Infrastracture.Library проекта. SmartClientApplicationBase създава работният обект и ShellForm. Shell Form е

стартиращата форма. Тя се състои от DeckWorkspace обект. Структурата на стартиращата форма се определя от Infrastracture.Layout модула, за да се демонстрира как структурата може да бъде изнесена в отделен модул, тъй че да се ползва и в други приложения.

Task Management папка

Task Management – съдържа Module и Module Controller класове. Класът Module Controller е отговорен да извърши инициализацията на модула, като добави необходимите услуги и елементи на потребителският интерфейс към скелета. Проекта съдържа всички изгледи и съответстващите им представители за този модул.

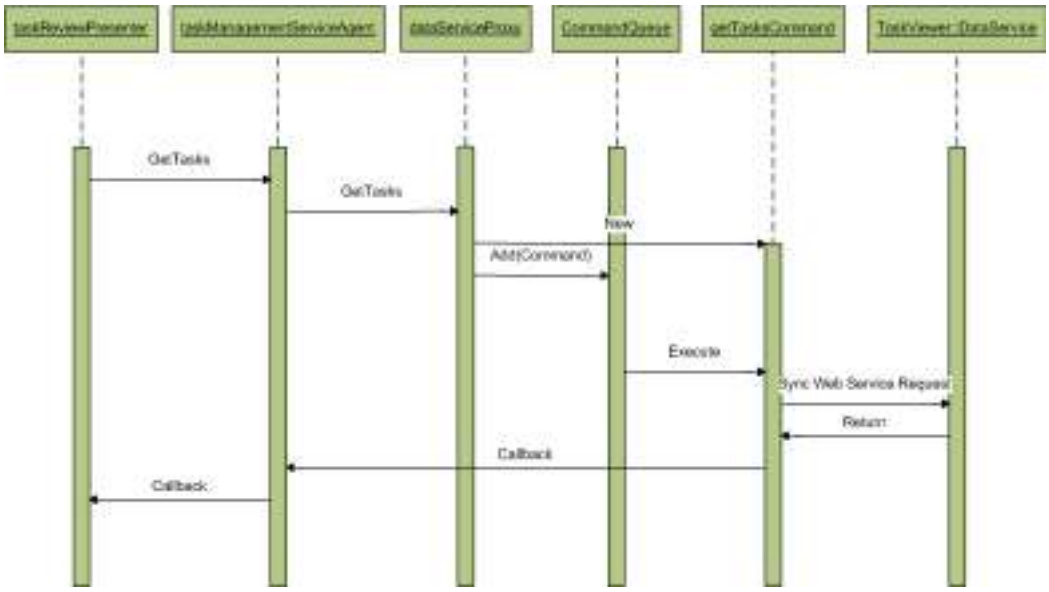
Task Management Interface съдържа константи именоващи действия, команди, съобщения, UI extension site и работни пространства, които са публичния интерфейс на Task Management Module.

Task Management Service Agent - съдържа бизнес обектите и логиката обгръщаща уеб услугите. Клиентският код използва TaskManagementService класа да изпълни заявки към уеб услугата, ако клиентът е в офлайн режим, агента за услуги изпълнява заявката спрямо локалното хранилище за данни.

Task Management Service Proxy – този проект съдържа прокси за използваните уеб услуги. Този проект съдържа и класове за извикване на уеб услугата асинхронно с използването на time-out стойности.

5.5.4. Изглед на потока на управление

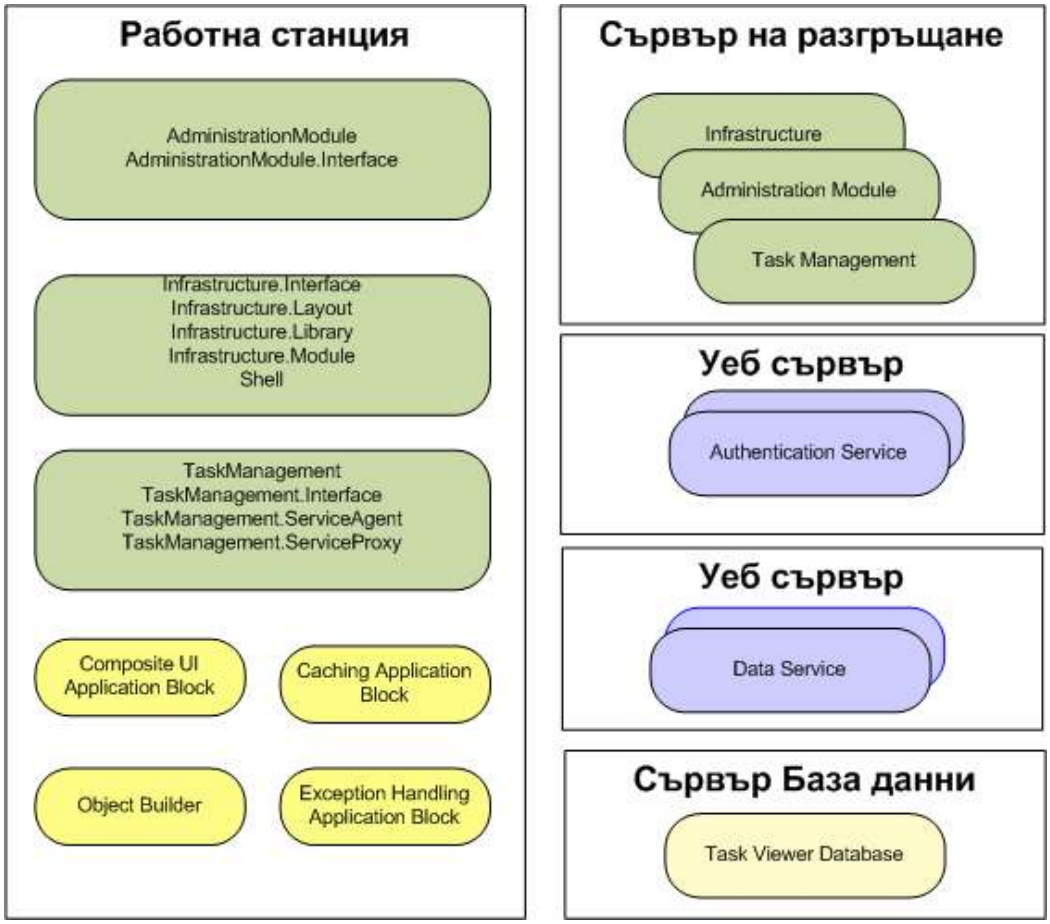
На фигура 25 е представена диаграма на последователност, показваща използването на агента на услуги. Агента на услуги използва smart Web Reference, за да изпрати асинхронни уеб заявки.



Фигура 25: Диаграма на последователност при използването на агент на услуги

5.6.Изглед на разгръщане

Кодът, който се разгръща на работната станция, използва следните програмни блокове на Microsoft Enterprise Library: Caching Application Block, Exception Handling Block. Изисква още Composite UI Application Block и Object Builder Application Block. Приложението използва две услуги, те могат да бъдат разгърнати на два отделни сървъра. То използва Click Once за разгръщане, затова е необходим отделен сървър, на който се разполагат файловете необходими за разгръщането. Фигура 26 показва компонентите и тяхното разгръщане.



Фигура 26: Изглед на разгръщане за примерното приложение

5.7.Обобщение

Настоящата глава от дипломната работа разгледа примерен офлайн умен клиент, разработен на базата на Microsoft технологии. Представено бе цялостно решение, включващо база данни, уеб услуги и клиентско приложение. Разработените WCF уеб услуги са трислойни, за да бъдат максималко гъвкави, и с CRUD интерфейс. Умният клиент е изграден чрез ориентиран към услуги подход. Работата със услугите е централизирана в агент на услуги. Агентът на услугите се грижи за кеширането на данни и офлайн работата. Умният клиент използва шаблоните: Модел Изглед Представител, агент на услуги, асинхронна комуникация с уеб услуги, навигация между изгледи. Шаблоните са разгледани със примери от разработеното приложение.

6. Тестване

Тестването е важна част от разработването на приложения. При разработването на умни клиенти би трябвало да се използват всички техники за тестване, които се прилагат при настолните приложения и уеб клиентите. Тестването трябва да включва тестване на единици, реализирано от програмистите. Умните клиенти се разработват по етапно. Отделни тимове изграждат уеб услугите, скелета, отделните модули на приложението. Важно е да се изтестват услугите и интерфейсите, които те предлагат, защото това е контрактът за комуникация с настоящи и бъдещи системи. При тестване на уеб услугите, трябва да се извърши сравнение с данните налични в базата данни, за да се провери дали върнатите данни са коректни. Уеб услугите се определят от wsdl контракти, трябва да се провери, че съобщенията, които се предават и приемат от методите на уеб услугите отговарят на дефинираните в wsdl файла.

При умните клиенти трябва да се приложи модулно тестване и тестване на интеграцията. Тъй като умните клиенти са модулни приложения, трябва да се изтестват модулите самостоятелно отделно един от друг и след това как се държат при едновременна си работа. Всеки от модулите на умния клиент трябва да бъде подложен на функционално тестване, което да осигури, че сценариите дефинирани със случаи на употреба са имплементирани успешно. Трябва да се изяснят изискванията към сигурността, производителността и бързодействието и да се дефинират и изпълнят тест случаи, които ги покриват.

Освен тестване на базата данни, уеб услугите и приложението трябва да се провери и разгръщането. В някои случаи определени обстоятелства могат да пречат на безпроблемното разгръщане или актуализация. Трябва да се изтества разгръщането, за да се потвърди, че то изпълнява първоначално заложените сценарии за разгръщане.

Тъй като тестването е извън обхвата на дипломната работа, тази глава ще дефинира кратък тест план свързан с функционалността заложена в случаите на употреба. Таблица 6 съдържа тест плана за примерното приложение.

Таблица 6: Task Viewer тест план

Тестов случай		
Аутентификация и ауторизация с	Предварителни условия	Приложението не е стартирано

потребител, който е администратор	Стъпки за тестване	1. Потребителят стартира приложението. 2. Потребителят въвежда потребителско име и парола, за които има валиден потребител с роля администратор
	Очакван резултат	Зареждат се административният и приложният модул. Приложението стартира подходящ екран за ролята на актьора
Аутентификация и ауторизация с потребител, който е служител	Предварителни условия	Приложението не е стартирано
	Стъпки за тестване	1. Потребителят стартира приложението. 2. Потребителят въвежда потребителско име и парола, за които има валиден потребител с роля служител
	Очакван резултат	Зарежда се приложният модул. Приложението стартира подходящия екран за ролята на актьора
Аутентификация с несъществуващ потребител или с потребител с грешна парола	Предварителни условия	Приложението не е стартирано
	Стъпки за тестване	1. Потребителят стартира приложението. 2. Потребителят въвежда грешно потребителско име или парола
	Очакван резултат	Системата извежда съобщение за невалиден потребител. При натискане на Cancel приложението се затваря.
Тестване на функционалността създаване на задача	Предварителни условия	Потребителят е влязъл в модула Task Manger, и е избрал проект.
	Стъпки за тестване	1. Потребителят избира от менюто Създай нова задача. 2. В отворилата се форма Task, потребителят може да попълни информация за задачата, обобщение, описание, приоритет, човек отговорен за задачата, краен срок за приключване на задачата 3. Потребителят натиска Ok
	Очакван резултат	Задачата се създава, появява се във списъка със задачи на проекта
Тестване на функционалността създаване на задача, алтернативен вариант	Предварителни условия	Потребителят е влязъл в модула Task Manger, и е избрал проект
	Стъпки за тестване	1. Потребителят избира от менюто Създай нова задача. 2. В отворилата се форма Task, потребителят може да попълни информация за задачата, обобщение, описание, приоритет, човек отговорен за задачата, краен срок за приключване на

		задачата 3. Потребителят натиска Cancel
	Очакван резултат	Прозорецът се затваря и системата не създава задача.
Тестване на функционалността промяна на задача	Предварителни условия	Потребителят е влязъл в модула Task Manger, заредил е задачите за определен проект.
	Стъпки за тестване	1. Потребителят кликва два пъти върху задача, системата зарежда детайлите за задачата. 2. Потребителят променя крайния срок на задачата, човекът отговорен за задачата, приоритета, обобщението, описанието или прогреса. Формата е подобна на формата за създаване на задача. 3. Потребителят натиска Ok
	Очакван резултат	Информацията за задачата се обновява.
Тестване на функционалността промяна на задача, алтернативен вариант	Предварителни условия	Потребителят е влязъл в модула Task Manger, заредил е задачите за определен проект.
	Стъпки за тестване	1. Потребителят кликва два пъти върху задача, системата зарежда детайлите за задачата. 2. Потребителят променя крайния срок на задачата, човекът отговорен за задачата, приоритета, обобщението, описанието и прогреса. Формата е подобна на формата за създаване на задача. 3. Потребителят натиска Cancel
	Очакван резултат	Прозорецът се затваря, системата не обновява задачата.
Тестване на функционалността създаване на проект	Предварителни условия	Потребителят е влязъл в модула Task Manger.
	Стъпки за тестване	1. Потребителят избира от менюто Създай проект. 2. Системата отваря формата Project. Потребителят въвежда информацията за проекта и натиска Ok.
	Очакван резултат	Системата затваря формата и създава проекта
Тестване на функционалността създаване на проект, алтернативен вариант	Предварителни условия	Потребителят е влязъл в модула Task Manger.
	Стъпки за тестване	1. Потребителят избира от менюто Създай проект. 2. Системата отваря формата Project. Потребителят въвежда информацията за проекта и натиска Cancel.
	Очакван резултат	Системата затваря формата и не създава

		проекта
Тестване на функционалността преминаване в онлайн режим	Предварителни условия	Потребителят работи в офлайн режим.
	Стъпки за тестване	1. Потребителят натиска бутона за преминаване в онлайн състояние. Системата синхронизира данните си със сървъра. 2. Зареждат се данните от сървъра, които не са били кеширани на клиента, по време на офлайн работата.
	Очакван резултат	Системата работи онлайн, данните на клиента са синхронизирани със сървъра. Индикацията показва, че приложението работи в онлайн режим
Тестване на функционалността преминаване в офлайн режим	Предварителни условия	Потребителят работи в онлайн режим.
	Стъпки за тестване	1. Потребителят натиска бутон за работа в офлайн режим. Показва се форма с проектите. 2. Потребителят избира кои проекти да бъдат кеширани на клиента. 3. Потребителя натиска Ok
	Очакван резултат	Клиента работи в несвързано състояние. Само с проектите, които е избрал по рано. Приложението индикира, че клиентът работи в офлайн състояние.
Тестване на функционалността преминаване в офлайн режим, алтернативен вариант	Предварителни условия	Потребителят работи в онлайн режим.
	Стъпки за тестване	1. Потребителят натиска бутон за работа в офлайн режим. Показва се форма с проектите. 2. Потребителят избира кои проекти да бъдат кеширани на клиента. 3. Потребителя натиска Cancel
	Очакван резултат	Приложението продължава да работи онлайн, индикацията показва, че приложението работи онлайн.
Тестване на функционалността разрешаване на конфликти	Предварителни условия	Настъпил е конфликт. Приложението изкарва форма за разрешаване на конфликти
	Стъпки за тестване	Потребителят избира своя или сървърния вариант. Системата запазва този избор.
	Очакван резултат	Съхранен е вариантът, който потребителят е избрал на фаза разрешаване на конфликти.

Тестване на функционалност разглеждане на задачи	Стъпки за тестване	1. Потребителят влиза в модула Task Manger. 2. Потребителят кликва два пъти върху даден проект. В Task Summary се зареждат задачите за този проект. Детайлите остават празни. 3. В Task Summary се избира задача, нейните детайли се зареждат в Task Details
	Очакван резултат	При влизане в модула: Екранът е разделен на три части. В дясно е показано дърво с всички налични проекти. В дясно екранът е разделен на две, горният е Task Summary, а долният Task Details двата са празни. След изпълнение на теста в Task Summary присъстват задачите за избрания проект, а в Task Details детайлите на избраната задача
Тестване на функционалност преглед на потребители	Стъпки за тестване	1. Потребителят влиза в модулът User Management 2. Актьорът кликва върху потребител
	Очакван резултат	При влизане в модула в дясната горна част се зареждат всичките потребители в системата. User Details частта е празна При кликването детайлите на потребителят се зареждат в User Details
Тестване на функционалността създаване на потребител	Предварителни условия	Потребителят е влязъл в модула User Management.
	Стъпки за тестване	1. От менюто потребителят избира Създай нов потребител опция 2. В заредената User форма, актьорът запълва информацията за потребителя. 3. Натиска Ok..
	Очакван резултат	Системата затваря прозореца и създава потребителя.
Тестване на функционалността създаване на потребител, алтернативен вариант	Предварителни условия	Потребителят е влязъл в модула User Management.
	Стъпки за тестване	1. От менюто потребителят избира Създай нов потребител опция 2. В заредената User форма, актьорът запълва информацията за потребителя. 3. Актьорът натиска Cancel.
	Очакван резултат	Системата затваря прозореца и не създава потребител
Тестване на функционалността променяне на информацията за потребител	Предварителни условия	Потребителят е влязъл в модула User Management.
	Стъпки за тестване	1. Актьорът кликва два пъти върху името на потребителя. 2. В отворилата се User форма

		актьорът променя информацията. 3. Натиска бутона Ok
	Очакван резултат	Системата затваря формата и обновява информацията за потребителя.
Тестване на функционалността променяне на информацията за потребител, алтернативен вариант	Предварителни условия	Потребителят е влязъл в модула User Management.
	Стъпки за тестване	1. Актьорът кликва два пъти върху името на потребителя. 2. В отворилата се User форма актьорът променя информацията. 3. Натиска бутона Cancel
	Очакван резултат	Системата затваря формата без да обновява информацията на потребителя.

В главата се разгледаха типовете тестване, които могат да се приложат, за тестване на умни клиенти. Бе направен кратък тест план за функционално тестване на приложението, който се използва в процеса по осигуряване качеството на приложението. Изпълнението му гарантира, че приложението отговаря на поставените условия.

7. Алтернативни разработки и бъдещи насоки

До тук дипломната работа обсъди какво е умен клиент, проблеми свързани с умните клиенти, проектиране на умен клиент с Microsoft технологии. В тази глава се разглежда въпросът има ли алтернативи на готовите блокове и фабрики, които предлагат Microsoft. За да му се отговори, се преглеждат инструментите и решенията предлагани от iAnywhere. Втората част от главата показва няколко компании, които използват такъв тип приложения, и причините, които са ги накарали да пожелаят такъв тип софтуер. Последната част предлага различни варианти за разширяване на дипломната работа.

7.1. Алтернативни разработки

Освен Microsoft и други фирми предлагат гама от инструменти и технологии предназначени за създаване на умни клиенти. Една от тях е Sybase iAnywhere. Тук ще направим преглед на случаите, които тя адресира и решенията, които предлага. Те са взети от документа Smart Client Architectures for the Mobile Developer (iAnywhere Solutions, 2006).

Основните архитектури за разработване на умни клиенти, за които iAnywhere предоставя готови технологии, са шест. Първите четири са свързани с начина за управление на данни и типа системи, с които умните клиенти ще се интегрират. Това са:

- Синхронизация на ниво база данни
- Подход, базиран на съхрани и препрати механизма
- Комбинация от синхронизиране на ниво база данни и подход базиран на съхрани и препрати механизъм
- Обмен на данни чрез репликация на файлове

iAnywhere предлага две допълнителни архитектури, като варианти комбиниращи предимствата на умния клиент с ползите от разгръщането и гъвкавостта, които са характерни за тънкия клиент.

- Офлайн уеб приложения
- Офлайн уеб приложения със синхронизация на ниво база данни

7.1.1. iAnywhere технологии

iAnywhere предлага кръг от технологии, предназначени за разработчиците на умни клиенти, и готови продукти, приложими в мобилни сценарии. За да направят своите мобилни потребители по-продуктивни, компании като Pepsi Bottling Group, McKesson, Britannia Airways, BNSF Railway, Harvard Medical School и U.S. Military използват продукти разработени с технологиите предоставени от iAnywhere. Два са основните продукти, предлагани от компанията за разработване на умни приложения: SQL Anywhere и M-Business Anywhere.

Пакетът SQL Anywhere осигурява функционалност за управление и синхронизиране на данни, позволяваща бърза разработка и разгръщане на мобилни приложения. Той съдържа:

- SQL Anywhere database server – това е RDBM сървър, който осигурява висока производителност и надеждност. Той включва: обработка на транзакции, поддържа интегритет на референциите, SQL и Java съхранени процедури, тригери, заключване на редове, автоматично планиране на събития и автоматично възстановяване.
- UltraLite - база данни предназначена за мобилни устройства, с ограничения на паметта. Тя изисква само 150KB памет. Въпреки това UltraLite осигурява функционалност включваща: интегритет на референциите, обработка на транзакции, шифриране и вграден синхронизационен клиент.
- MobiLink – е силен и гъвкав синхронизационен сървър. Използва се за реализиране на синхронизация между бази данни. Той поддържа еднакво добре малък брой от потребители и хиляди, синхронизиращи се едновременно клиента.
- QAnywhere – осигурява съхрани и препрати механизъм за мобилни потребители. Той обогатява MobiLink като му позволява да функционира като сървър за съобщения

M-Business Anywhere е платформа за доставяне на уеб базирано съдържание и уеб приложения на широк кръг от мобилни устройства. С негова помощ се изграждат уеб приложения, които работят локално на мобилното устройство.

7.1.2. iAnywhere решения

За всяка архитектура, iAnywhere дефинира решение, което обхваща: софтуера на мобилният клиент; междинен софтуер, който се налага да бъде използван; софтуера на сървърите в организацията, с които трябва да се извърши интеграция; езиците за разработка, които могат да бъдат използвани за реализиране на необходимото решение; целевите платформи, за които се разработва клиентът и мрежите, които се поддържат. Решенията са наречени обща постановка и за всяка архитектура са систематизирани в таблици.

7.1.2.1. Синхронизация на ниво база данни

Синхронизацията на ниво база данни е вече разгледаният подход, базиран на данни. Решението използва SQL Anywhere и MobiLink. Базата SQL Anywhere се грижи за консистентността на данните в базите на клиента и на сървъра. Тя осигурява минимизиране на количество предавана информация. MobiLink филтрира данни по ред и колона. По този начин осигурява, че мобилните потребители разполагат само с данните, от които се нуждаят. Мобилните приложения могат да бъдат написани с използването на различни езици и инструменти. Устройствата, на които могат да се използват, така разработените решения са: мобилни компютри, таблет компютри , PDA, умни телефони и настолни компютри. Таблица 7 обобщава решението предложено от iAnywhere.

Таблица 7: Обща постановка при синхронизирането на ниво база данни

Тип софтуер	**Софтуерно решение**
Софтуер, намиращ се на мобилния клиент	UltraLite или SQL Anywhere MobiLink синхронизационен клиент Mobile software applications
Междинен софтуер(Middleware)	MobiLink синхронизационен сървър
Софтуер в организацията	Централната база данни може да е SQL Anywhere server, Oracle, Microsoft SQL Server, IMB DB2 или Sybase ASE
Езици за разработка	Мобилното приложение може да бъде разработено на C/C++, VB, VB.NET, C#, ASP, PHP, Perl, PowerBuilder, PocketBuilder, Java, Python, Delphi
Поддържани отдалечени платформи	Pocket PC, Palm OS, Smartphones, Windows, Solaris, Linux
Поддържани мрежи	безжични, 802.11, WiFi, W-LAN, GPRS, GSM, CDMA, CDPD, Ethernet, device

	cradle, модем, ActiveSync, HotSync

7.1.2.2. Подход, базиран на съхрани и препрати механизъм

Това е разновидност на подхода ориентиран към услуги, имплементиращ съхрани и препрати механизъм. Използва се асинхронна обмяна на съобщения.

Данните се пренасят чрез съобщения, които се кешират първоначално в опашки. Тук обаче, има ценрализиран сървър грижещ се за съобщенията и тяхната обмяна в системата. Всяка опашка има едно или повече приложения, които консумират съобщенията поставената в нея.

Софтуера QAnywhere, част от SQL Anywhere пакета, е решение имплементиращо механизма съхрани и препрати и се грижи за детайлите при работата на приложенията в мобилна и безжична среда. Той позволява на разработчиците да оптимизират производителността, широчината на канала за предаване на съобщения, да компресират предаваната информация и да осигурят гарантирано предаване на съобщенията, независимо от липса на свързаност или системни грешки. QAnywhere се интегрира с всякакви системи за съобщения на ниво организация, поддържа Java Message Service(JMS) като MQSeries и J2EE приложни сървъри, като IBM WebSphere, BEA WebLogic и Sybase EAServer. Допълнителни системи могат лесно да бъдат свързани на базата на Sybase Unwired Orchestrator. Таблица 8 обобщава характеристиките на това решение.

Таблица 8: Обща постановка при подхода, базиран на съхрани и препрати механизъм

Тип софтуер	**Софтуерно решение**
Софтуер, намиращ се на мобилния клиент	SQL Anywhere database engine QAnywhere синхронизационен клиент Мобилно приложение
Междинен софтуер(Middleware)	MobiLink синхронизационен сървър с QAnywhere опции
Софтуер в организацията	Всяка базирана на JMS система за съобщения, като MQ Series или JMS базиран приложен сървър, като Sybase EAServer, BEA WebLogic или IBM WebSphere TiBCO, SAP, PeopeSoft, Siebel или други използващи Sybase Unwired Orchestrator
Езици за разработка	Мобилни приложения написани на C/C++/C#
Поддържани отдалечени платформи	Pocket PC, Smartphones, Windows
Поддържани мрежи	безжични, 802.11, WiFi, W-LAN, GPRS, GSM, CDMA, CDPD, Ethernet, device

	cradle, модем, ActiveSync, HotSync

7.1.2.3. Комбинация между синхронизация с база данни и подхода съхрани и препрати

Докато едни организации позволяват данните да бъдат синхронизирани на ниво бази данни, други организации контролират промените в базите като изпращат всички изменения чрез междинен бизнес слой работещ на приложен сървър.

При комбинирането на синхронизацията на ниво база данни и съхрани и препрати метода, промените се синхронизират на ниво база данни, но контролът се осъществява от междинния слой. Използва се комбинацията от SQL Anywhere, MobiLink и QAnywhere. Пълната обща постановка е дадена в таблица 9.

Таблица 9: Обща постановка при метод, комбиниращ синхронизация на ниво база данни и подход съхрани и изпрати

Тип софтуер	**Софтуерно решение**
Софтуер, намиращ се на мобилния клиент	SQL Anywhere database engine QAnywhere клиент MobiLink синхронизационен клиент
Междинен софтуер(Middleware)	MobiLink синхронизационен сървър с QAnywhere опции
Софтуер в организацията	Централизираната база данни може да е SQL Anywhere server, Oracle, Microsoft SQL Server, IMB DB2 или Sybase ASE Изпращане и получаване на съобщения с базирани на JMS системи за съобщения, като MQ Series или JMS базирани приложни сървъри, като Sybase EAServer, BEA WebLogic или IBM WebSphere Изпращане и получаване на съобщения с TiBCO, SAP, PeopeSoft, Siebel или други използващи Sybase Unwired Orchestrator
Езици за разработка	C/C++/C#
Поддържани отдалечени платформи	Pocket PC, Smartphones, Windows
Поддържани мрежи	безжични, 802.11, WiFi, W-LAN, GPRS, GSM, CDMA, CDPD, Ethernet, device cradle, модем, ActiveSync

7.1.2.4. Обмен на данни чрез репликация на файлове

В някои случаи разработчиците използват файлове, за да управляват и обменят информация. Файловете са прост подход за съхраняване на данни. При него отговорността за организирането, управлението на структурата на файла и

сигурността му са в ръцете на разработчикът. Файловете с данни могат да бъдат пренасяни от междинния организационен сървър към другите системи в организацията чрез специализирани скриптове и софтуерни програми изпълнявани от Session Manager.

Afaria Session Manager осигурява инструменти за писане на скриптове и се грижи за изпълнението им. Тези скриптове трябва да поддържат репликационния процес, отдалеченото управление на файлове, автоматизирането на комуникационните сесии. Решението базирано на Afaria е систематизирано в таблица 10.

Таблица 10: Обща постановка при обмен на данни чрез репликация на файлове

Тип софтуер	**Софтуерно решение**
Софтуер, намиращ се на мобилния клиент	Afaria клиент със Session Manager Мобилно софтуерно приложение
Междинен софтуер(Middleware)	Afaria сървър със Session Manager
Софтуер в организацията	Данните могат да бъдат обменяни с други системи чрез специално разработени за тази цел скриптове написани на JScript или VBScript, или специално разработен за тази цел софтуер
Езици за разработка	C/C++, C#, VB, VB.NET,ASP, PHP, Perl, PowerBuilder, PocketBuilder, Java, Python, Delphi или други езици за разработка поддържани от отдалеченото устройство
Поддържани отдалечени платформи	Windows CE, Pocket PC (Windows Mobile), Windows XP, Windows 2000, NT, Windows 95&98, Tablet PC Edition, Palm OS, Symbian Smartphones, RIM Blackberry
Поддържани мрежи	безжични, 802.11, WiFi, W-LAN, GPRS, GSM, CDMA, CDPD, Ethernet, device cradle, модем, ActiveSync, HotSync

7.1.2.5. Офлайн уеб приложения

Офлайн уеб приложенията комбинират предимствата на умния клиент и архитектурата на тънките клиенти. Офлайн уеб приложението представлява уеб базирано съдържание. То е изградено с използването на стандартни уеб технологии и допълнителни компоненти за достъп до периферията на работната станция. Многоплатформената природа на уеб приложенията гарантира, че приложението, веднъж написано, може да бъде разгърнато на множество от мобилни устройства.

M-Busines Anywhere осигурява платформа за доставяне на уеб базирани приложения на многообразие от мобилни устройства. С негова помощ уеб

разработчиците могат да приложат техните знания и опит, за да разработят и разгърнат динамично уеб приложение със sync-and-go механизъм или безжични възможности.

Файловете, изграждащи мобилното приложение, се свалят от един или повече сървъри, където се съхраняват. Промените на данните направени от клиентското приложение се качват автоматично от M-Business Anywhere клиента всеки път, когато мобилно приложение извършва синхронизация. Офлайн природата на тези приложение позволява на потребителя да продължи да работи с уеб приложението, дори когато мобилното приложение не е свързано към локалната или глобалната мрежа.

Вградените синхронизационни възможности на M-Business Anywhere гарантират, че данните предадени на приложението ще бъдат предадени към уеб сървъра, когато приложението извърши синхронизацията.

Таблица 11: Обща постановка при офлайн уеб приложения

Тип софтуер	**Софтуерно решение**
Софтуер, намиращ се на мобилния клиент	M-Business Anywhere клиент
Междинен софтуер(Middleware)	M-Business Anywhere сървър
Софтуер в организацията	Всякакъв уеб или приложен сървър чрез HTTP или HTTPS Siebel 7 XML – форматирани данни TiBCO, SAP, PeopleSoft, Siebel, и други, които използват Sybase Unwired Orchestrator
Езици за разработка	DHTML, SSL, HTML 4.01, CSS, XHTML, DOM, Javascript
Поддържани отдалечени платформи	Pocket PC(Windows mobile), Palm OS, Windows XP, Windows XP Tablet PC Edition
Поддържани мрежи	безжични, 802.11, WiFi, W-LAN, GPRS, GSM, CDMA, CDPD, Ethernet, device cradle, модем, ActiveSync

7.1.2.6. Офлайн уеб приложения със синхронизация на ниво база данни

Комбинирането на офлайн архитектурата със синхронизация на ниво база данни дава възможност на офлайн уеб приложенията да предоставят интегритета, скалируемостта и управлението на данни, които се асоциират с подходите, базирани на данни. Таблица 12 обобщава технологиите адресиращи този сценарий.

Таблица 12: Офлайн уеб приложения със синхронизация на ниво база данни

Тип софтуер	**Софтуерно решение**
Софтуер, намиращ се на мобилния клиент	M-Business Anywhere клиент UltraLite база данни за M-Business Anywhere
Междинен софтуер(Middleware)	M-Business Anywhere сървър MobiLink синхронизационен сървър
Софтуер в организацията	Всякакъв уеб или приложен сървър чрез HTTP или HTTPS Siebel 7 XML – форматирани данни TiBCO, SAP, PeopleSoft, Siebel, и други, които използват Sybase UnwiredOrchestrator Централизираната базата данни може да е SQL Anywhere database server, Oracle, Microsoft SQL Server, IBM DB2 или Sybase ASE
Езици за разработка	DHTML, SSL, HTML 4.01, CSS, XHTML, DOM, Javascript C++ за разработване на POD разширения Стандартен SQL
Поддържани отдалечени платформи	Pocket PC(Windows mobile), Palm OS, Windows XP, Windows XP Tablet PC Edition
Поддържани мрежи	безжични, 802.11, WiFi, W-LAN, GPRS, GSM, CDMA, CDPD, Ethernet, device cradle, модем, ActiveSync

7.2.Компании, използващи умни клиенти

dbMotion е софтуерна компания, занимаваща се с медицинска информатика. Нейният софтуер позволява на здравните организации да споделят сигурно медицинска информация чрез създаването на виртуални записи за пациентите. Те позволяват обмяна на информация без да централизират съхранението на данните и не подменят съществуващата информационна система. Чрез споделянето в реално време на текуща информация, медицинския персонал може да прави точни преценки, осигурявайки по-сигурна и ефикасна грижа на пациентите. dbMotion разполага с множество Windows инструменти за разработване и тестване на своята система. Инструментите нямат единен интерфейс, и са разпръснати в организацията. DbMotion иска да разработи нов набор от инструменти, за да улесни своите бизнес анализатори и разработчици, като цели по този начин да скъси времето за разработване на отделните части на системата. Тя разполага с уеб услуги,

които могат да бъдат консумирани, но едно уеб приложение не може да задоволи напълно нуждите на набора от инструменти, който компанията използва. Компанията иска новият набор от инструменти да използва наличните уеб услуги и да може да се възползва от клиентски ресурси. Решението бе умен клиент изграден чрез CAB, Smart Client Software Factory, кеширащия приложен блок на Microsoft и ClickOnce технологията. Решението бе имплементирано от екип на Unicoders. Първата фаза включваше разработване на скелета на умния клиент и два модула, които могат да бъдат включвани и изключвани от него. Ползата, която това решение донесе, бе единен подход и потребителски интерфейс за работа с останалите части на системата, намаление на разходите необходими за обучаване на потребителите да работят с отделните инструменти.

Според документа Smart Clients in Business (Microsoft Corporation, 2005) други примери за компании използващи умни клиенти като решение на специфичен проблем, са:

Компанията ICEE произвежда, разпространява и поддържа машини, използвани за правене на замразени стоки, доставя хранителни добавки за хранителната индустрия. Компанията разполага с 450 техници, които се грижат да осигурят безпроблемната работа на сайтовете и поддръжка на множеството от потребители, прекупвачи и магазини, с които тя работи. Проблемът, с който се сблъска ICEE, бе неефективни процеси на комуникация между служителите. Затова тя реши да автоматизира комуникацията, разпределянето, събирането и таксуването чрез използване на мобилна автоматизирана услуга. Решението бе построено на базата на Mobile Intelligence Platform, която е разработена с помощта Microsoft Visual C# инструменти за разработка и Microsoft .NET Compact Framework. Ползата, която донесе приложението на ICEE, е придобиването на бързо разгръщаемо и евтино мобилно приложение, което оптимизира мрежовата работа при голяма загуба на пакети и възможност да се свързва с множество видове мрежи: LAN, WLAN и WWLAN.

Amazon.com е една от най-големите електрони библиотеки. Тя е една от компаниите Fortune 500. Amazon смятат, че потребителското доволство, лоялността и продажбите ще се увеличават, ако потребителят може да достъпва сайта им през едни от най-често използваните приложения като Microsoft Word и Microsoft Outlook. Решението, на което те се спряха, бе офис умен клиент.

Чрез използването на Microsoft .NET Framework, Amazon.com създава XML базирани документи, които се зареждат в клиентското приложение. То обработва детайлите и вида им, спестявайки време за разработка на други приложения и ресурси. По този начин потребителите могат да пазаруват от Amazon, без да отварят браузъра, чрез Office пакета на Microsoft. Компанията мисли, че това помага за по-голямото и разрастване.

Една от най-големите компании за анализиране на данни във финансовата сфера в Америка е Thomson Financial. Тя се състои от множество бизнеси, като всеки придобит бизнес работи сравнително независимо от другите, предлагайки свои собствени разработки. Thomson поде инициатива за ново софтуерно решение, наречено Thomson ONE, разработено с помощта на Microsoft .NET Framework. От една страна Thomson трябваше да поддържат набор от потребители, а от друга разработчици, които да обслужват системата и администратори, които да я разгръщат и поддържат.

Thomson се фокусираха върху разработването на умен клиент на базата на Windows Forms класовете включени в .NET Framework. Ползата, която това приложение донесе, бе намаляване на натоварването на сървърите, тъй като голяма част от обработката на данни се пренесе на клиентските станции. Натовареността на мрежата се намали и това намали разходите за поддръжка на сървърите и мрежата.

7.3.Бъдещи насоки

Дипломната работа може да бъде разширена в няколко насоки, обхващащи разширяване от гледна точка на необхванати детайли в разработката и такива, които разглеждат подменянето на части на умния клиент с нови технологии и стратегии, в областта на компютърните науки.

Първата глава представи сравнителна характеристика между тънките и умните клиенти. В Таблица 1 са систематизирани основните случаи, за да може тя да бъде използвана като бърз справочник, кой от двата варианта е по-подходящ. Проблемните области на двата типа приложения могат да бъдат разгледани още веднъж и да се потърсят допълнителни потребителски изисквания, които биха повлияли при избора на архитектура.

В отделни части на дипломната работа бяха разгледани бегло отделни аспекти на сигурността на умните приложения. Надеждността и сигурността на данните в едно приложение е комплексен въпрос, към който трябва също да се подходи систематизирано и централизирано. Дипломната работа може да се разшири като се добави допълнителна глава, изцяло посветена на сигурността и нейните аспекти, като авторизация, аутентификация, валидиране на данните, управление на грешки, сигурност на средата и т.н.

В последната глава са включени резултатите от функционалното тестване на разработеното приложение. Главата може да се разшири, като се добави тестване на програмни единици(unit testing).

Умните приложения не са статични, технологията се развива и те трябва да отговарят на новите изисквания и желания на потребителите. Преди около година Microsoft представиха ново поколение операционни системи Windows Vista (Windows Vista Developer Center), това разбира се, се отрази и на разработваните приложения. Следваща версия на пакета за разработка Visual Studio е вече в бета версия и тя може да бъде свалена от официалния си сайт Visual Studio 2008 (Visual Studio 2008 Downloads). Инструментите за разработка са насочени специално за разработка на приложения, възползващи се от нововъведенията в Windows Vista. Подробното им описание може да се намерени на следният сайт (Microsoft, Windows Vista Features). Едно от първите неща, към които ще се ориентират желанията на потребителите, са приложения, възползващи се от интерфейсните подобрения в Windows Vista. Едно от основните предимства на умните клиенти е, че могат да се възползват от богатия десктоп интерфейс. Разгледаната имплементация използва Windows Forms за реализирането на интерфейса, използва се MVP (Model View Presenter) подход. Потребителският интерфейс на Vista се базира на Window Presentation Foundation. Може да се изследва колко време би отнела подмяната на реализираните чрез Windows Forms гранични класове с такива използващи Window Presentation Foundation(WPF). Новото студио дава възможност за взаимодействие между Windows Forms и WPF като тази функционалност носи кодовото наименование CrossBow. В бъдеща разработка, част от екраните може да се оставят такива каквито са, за да се разгледа как вече съществуващи умни клиенти могат да се възползват от новият потребителски интерфейс, запазвайки съществуващите потребителски форми. Може да се разгледа дали е удачно

пренаписване на потребителския интерфейс изцяло с WPF или двата вида работят без проблемно и това би било излишно.

Освен това нововъведение, новото студио предлага редица улеснения при разработката на умни клиенти. В блога си Saurabh Pant (Pant, 2007) разглежда нововъведенията, които предлага новото студио, което носи кодовото название Orcas. ClickOnce технологията, не е вече ограничена само да браузъра на Microsoft Internet Explorer. В новата версия технологията поддържа FireFox. ClickOnce може да работи вече чрез аутентификационни проксита. Поддържа сценарий за копиране XCopy. Поддръжката за разработка на умни клиенти е вградена в студиото и са изчистени дефектите свързани с тази функционалност. Новото студио и .NET Framework 3.5 разширява ADO.NET функционалността като добавя синхронизиращи услуги. В бъдеща разработка е подходящо да се разгледа как те могат да се използват и какви ползи носят.

7.4.Обобщение

Освен Microsoft на пазара на готови блокове и технологии за разработване на умни клиенти има множество компании. Умните клиенти са атрактивна ниша, към която те се стремят. Една от тези компании е iAnywhere, която предлага поддръжка на подхода базиран на данни и на подхода ориентиран към услуги. В добавка към архитектурите поддържани от Microsoft, iAnywhere предлага умни клиенти използващи уеб потребителският интерфейс в офлайн режим.

Технологиите за разработването на умни клиенти произлизат от наложената нужда на пазара за такива приложения. Много фирми са избрали вътрешния за компанията софтуер да бъде базиран на идеята за умни клиент приложения. Едни от тях като Pepsi Bottling Group, McKesson, Britannia Airways, BNSF Railway и Harvard Medical School са избрали решенията на iAnywhere, друга част като dbMotion, ICEE, Amazon и Thomson Financial разчитат на Microsoft технологиите.

Дипломната работа обхваща голяма част от въпросите вълнуващи разработчиците и софтуерните архитекти, занимаващи се с разработка на умни клиенти. На пазара излизат нови технологии, които се отразяват на разработката на софтуер като WCF, WPF и др. , нови платформи като Windows Vista. Бъдещо

разширение на темата може да включва обзор на новите методи и технологии и проблемите свързани с тях. Аспект, който трябва да бъде разширен и систематизиран в отделна глава е сигурността и как тя се отразява на разработката на умни клиенти.

8. Заключение

При разработването на умни клиенти архитектите и програмистите се сблъскват с много въпроси. В някои случаи те са запознати с този термин, а в други това е просто каприз, наложен от клиентите. Кратко време след това, те разбират, че умният клиент е един нов начин на мислене, сравнен с обикновените десктоп приложения. Умният клиент не само използва уеб услуги, но и е многослойно модулно приложения, в чиято имплементация е залегнало слабото свързване между модулите и тесните връзки между класовете в модула.

Smart clients територията е много обширна. Тя обхваща всички аспекти при разработването на декстоп приложения, предимствата на уеб технологиите, адресира въпросите на свързаността и липсата на такава. Умните клиенти са сравнително нова материя и литературата посветена на тях е малко. Дипломната работа прави крачка в систематизирането на аспектите, проблемите и решенията свързани с умните клиенти. Систематизираната информация е представена в текст, таблици и фигури. Дипломната работа се опита да разгледа възможно най-много проблеми и разнообразни варианти за решение им.

Умните клиенти са хибрид между положителните черти на десктоп и уеб приложенията. Microsoft предоставя удобна платформа за разработването им. В зависимост от бизнес целите, Microsoft базираните умни клиенти се разработват за десктоп, офис пакет приложения или устройства с малки размери.

Умните клиенти се предпочитат пред уеб приложенията, когато трябва да се предостави офлайн функционалност или интеграция със специализиран софтуер и хардуер. Те не са добро решение, когато целевите потребители са разнородна публика вън от организацията.

Централен аспект на умните клиенти се явяват данните и кеширането. За да бъде успешна имплементацията, проектирането трябва да им обърне сериозно внимание. Трябва да се реализира централизиран подход, чрез изграждането на кешираща инфраструктура. Тя трябва да поддържа техники за съхранение на данните, стратегии за загуба на давност и стратегии за изчистване на кеша. Данните са сравнително постоянни във времето или краткотрайни. Подходящо за краткотрайните данни е оптимистично

противодействащо кеширане за кратък период, а за данните само за четене, предварително дълготрайно кеширане.

Основните подходи при разработване на офлайн умни клиенти са два: базиран на данни и ориентиран към услуги. Подходът базиран на данни използва релационна база данни за поддържането на офлайн функционалността, синхронизирането, заключването и изглаждането на. Разработчикът е отговорен за дефиниране на бизнес правилата. При подхода ориентиран към услуги обаче, разработчикът трябва да се погрижи за много повече аспекти – реализиране на асинхронната комуникация, минимизиране на сложните мрежови взаимодействия, кеширане, отчитане на връзката и промяна на поведението на приложението в зависимост от нея. Подходът базиран на данни, дава бързо като имплементация решение, а подходът базиран на услуги по-гъвкаво решение на цената на допълнителен труд.

CAB и SCSF са удачен избор за разработване на умни клиенти. CAB предоставя ясен подход за изграждане на скелет на приложението, общ за всичко модули, подход за разработване на модули, услуги, локализиращи модулите и готова инфраструктура, за поддържането на такъв модел. SCSF автоматизира еднотипните дейности като по този начин ускорява времето за разработка. Клиентът получава приложението, което е поръчал, в по-кратки срокове и най-вече навреме. Благодарение на сигурността базирани на роли, и профилните каталози, поддържани от CAB и ClickOnce технологията, може да реализират множество разнообразни сценарии за разгръщане. Дипломната работа представи имплементация на примерен умен клиент базиран на Microsoft технологиите.

Пазарът на умни клиенти е ниша, която се експлоатира не от една фирма. Компанията iAnywhere също предлага на пазара решения насочени към разработването на умни клиенти. Тя също адресира подхода базиран на данни и този на ориентиран към услуги, дава възможност за имплементиране на офлайн функционалност. Множество фирми, като dbMotion, Amazon и др., са се възползвали от предимствата давани от умните клиентите. Удовлетвореността на техните потребители расте, поради удобствата, които те им предоставят като работа в офлайн режим или при слаба свързаност, бързо имплементиране на нови изисквания и гъвкавост на решенията.

9. Използвана литература

1. Microsoft Corporation. (2006). Selecting a Backing Store. In *Enterprise Library Documentation.*

2. Hill, D. (2 Февруари 2004 г.). *What is a Smart Client anyway?* Изтеглено на 19 Септември 2007 г. от David Hill's WebLog: http://blogs.msdn.com/dphill/articles/66300.aspx

3. iAnywhere Solutions. (11 Март 2006 г.). *Smart Client Architectures for the Mobile Developer.* Изтеглено на 8 Март 2007 г. от iAnywhere: www.iAnywhere.com

4. Lasker, S. (16 Декември 2006 г.). *Choosing Between SQL Server 2005 Compact Edition and SQL Server 2005 Express Edition.* Изтеглено на 27 Септември 2007 г. от www.Microsoft.com: http://www.microsoft.com/sql/editions/compact/sscecomparison.mspx

5. Microsoft. (11 Юни 2006 г.). *.NET Compact Framework 2.0 Service Pack 1 Redistributable*. Изтеглено на 19 Септември 2007 г. от www.Microsoft.com: http://www.microsoft.com/downloads/details.aspx?familyid=0C1B0A88-59E2-4EBA-A70E-4CD851C5FCC4&displaylang=en

6. Microsoft. (20 Ноември 2006 г.). *.NET Framework 2.0 Software Development Kit (SDK)*. Изтеглено на 19 Септември 2007 г. от www.Microsoft.com: http://www.microsoft.com/downloads/details.aspx?familyid=FE6F2099-B7B4-4F47-A244-C96D69C35DEC&displaylang=en

7. Microsoft Corporation. (8 Март 2005 г.). *Smart Clients in Business.* Изтеглено на 26 Септември 2007 г. от Smart Clients in Business White Paper: http://www.microsoft.com/net/smartclient_businessvalue.mspx

8. Microsoft. (5 Април 2007 г.). *Enterprise Library 3.0 – April 2007*. Изтеглено на 19 Септември 2007 г. от Enterprise Library: http://www.microsoft.com/downloads/details.aspx?familyid=62ef5f79-daf2-43af-9897-d926f03b9e60&displaylang=en

9. Microsoft. (22 Януари 2006 г.). *Microsoft .NET Framework Version 2.0*. Изтеглено на 19 Септември 2007 г. от www.Microsoft.com: http://www.microsoft.com/downloads/details.aspx?FamilyID=0856EACB-4362-4B0D-8EDD-AAB15C5E04F5&displaylang=en

10. Microsoft. (н.д.). *Microsoft Visual Studio 2005*. Изтеглено на 19 Септември 2007 г. от www.Microsoft.com: http://www.microsoft.com/bulgaria/vstudio2005/default.mspx

11. Microsoft Patterns & Practices. (2004). *Smart Client Architecture and Design Guide.* Microsoft Press.

12. Microsoft. (12 Май 2005 г.). *Smart Client – Composite UI Application Block*. Изтеглено на 19 Септември 2007 г. от www.Microsoft.com: http://www.microsoft.com/downloads/details.aspx?familyid=7b9ba1a7-dd6d-4144-8ac6-df88223aee19&displaylang=en

13. *Microsoft SQL Server*. (2005). Изтеглено на 21 Септември 2007 г. от www.Microsoft.com: http://www.microsoft.com/sql/default.mspx

14. Microsoft. (23 Февруари 2007 г.). *Web Service Software Factory*. Изтеглено на 19 Септември 2007 г. от www.Microsoft.com: http://www.microsoft.com/downloads/details.aspx?familyid=db996113-6e92-4894-9b7e-0debb614d72f%20&displaylang=en

15. Microsoft. (2007). *Windows Mobile 6*. Изтеглено на 20 Септември 2007 г. от www.Microsoft.com: http://www.microsoft.com/windowsmobile/6/default.mspx

16. Microsoft. (1 Май 2007 г.). *Windows Mobile 6 Professional and Standard Software Development Kits Refresh*. Изтеглено на 20 Септември 2007 г. от http://www.microsoft.com/downloads/details.aspx?familyid=06111a3a-a651-4745-88ef-3d48091a390b&displaylang=en

17. Microsoft. (н.д.). *Windows Vista Features*. Изтеглено на 26 Септември 2007 г. от www.Microsoft.com: http://www.microsoft.com/windows/products/windowsvista/features/default.mspx

18. MSDN. (2005). *Click Once Deployment*. Изтеглено на 19 Септември 2007 г. от http://msdn2.microsoft.com/en-us/library/t71a733d(vs.80).aspx

19. MSDN. (Юни 2006 г.). *Mobile Client Software Factory – July 2006*. Изтеглено на 20 Септември 2007 г. от http://msdn2.microsoft.com/en-us/library/aa480471.aspx

20. MSDN. (2005). *Smart Client – Composite UI Application Block*. Изтеглено на 19 Септември 2007 г. от http://msdn2.microsoft.com/en-us/library/aa480450.aspx

21. MSDN. (Май 2007 г.). *Smart Client Software Factory*. Изтеглено на 19 Септември 2007 г. от MSDN: http://msdn2.microsoft.com/en-us/library/aa480482.aspx

22. MSDN. (н.д.). *Visual Studio Tools for Office*. Изтеглено на 19 Септември 2007 г. от Visual Studio Tools: http://msdn2.microsoft.com/en-us/office/aa905533.aspx

23. MSDN. (Декември 2006 г.). *Web Service Software Factory*. Изтеглено на 19 Септември 2007 г. от http://msdn2.microsoft.com/en-us/library/aa480534.aspx

24. Noyes, B. (2006). *Smart Client Deployment with ClickOnce - A Developers Guide to Deploying Windows Forms Applications.* Addison-Wesley Professional .

25. Pant, S. (26 Март 2007 г.). *Smart Client: Whats new in Orcas?* Изтеглено на 26 Септември 2007 г. от Saurabh Pant's Web Log: http://blogs.msdn.com/saurabh/archive/2007/03/26/smart-client-whats-new-in-orcas.aspx

26. *Patterns & Practices: Web Service Software Factory - Home*. (2007). Изтеглено на 19 Септември 2007 г. от CodePlex: http://www.codeplex.com/servicefactory

27. *Visual Studio 2008 Downloads*. (н.д.). Изтеглено на 26 Септември 2007 г. от MSDN: http://msdn2.microsoft.com/en-us/vstudio/aa700831.aspx

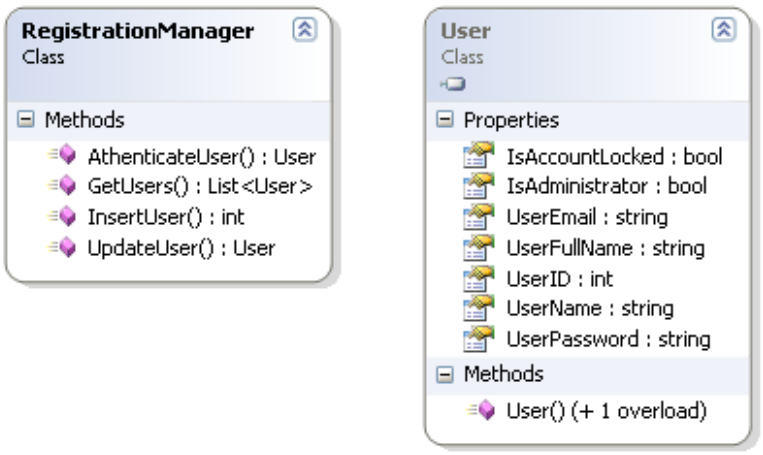
28. *Windows Vista Developer Center*. (н.д.). Изтеглено на 26 Септември 2007 г. от MSDN: http://msdn2.microsoft.com/en-us/windowsvista/default.aspx

10.Приложение

Приложение А: Клас диаграми на аутентификационната услуга

Бизнес слой: Business Logic и Business Entity проекти

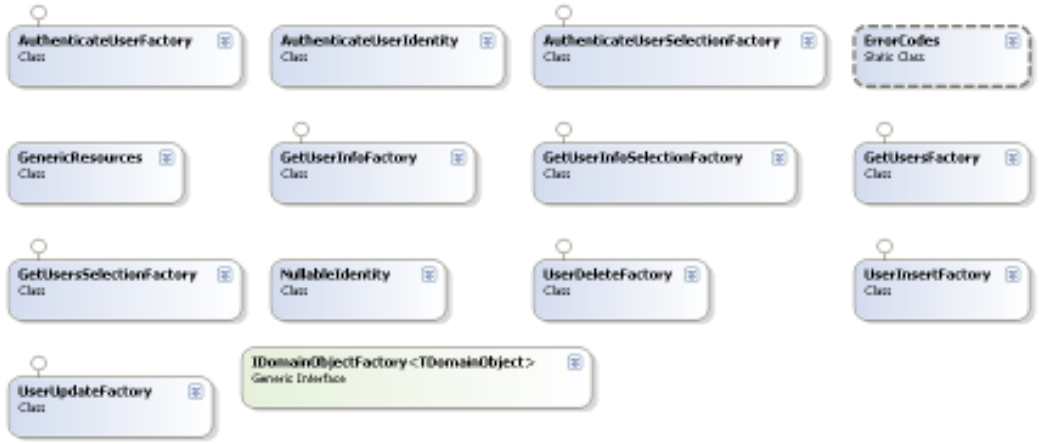
Бизнес слоят е прост. В проекта на бизнес логиката се съдържа един обект, който се грижи за аутентифицирането на потребителя, извличането на лист със потребители, създаване и обновяване на потребителя. Проектът на бизнес обектите съдържа един User обект. Клас диаграмата за бизнес слоя е показана на фигура 27.



Фигура 27: Клас диаграма на бизнес слоя на аутентификационната услуга

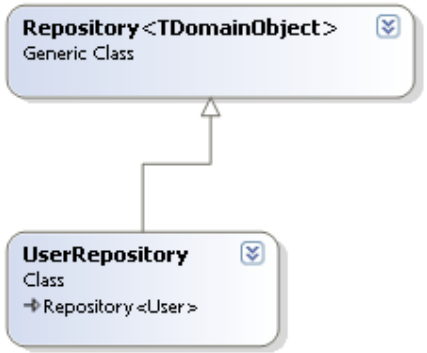
Слой за достъп до ресурси: Data Access

На фигура 28 и фигура 29 могат да се видят всичките класове в DataAccess проекта. Фигура 28 показва класовете имплементиращи шаблона фабрика.



Фигура 28: Клас диаграма на класовете фабрики използвани в аутентификационната услуга

Фигура 29 показва класът Repository за User обекта.

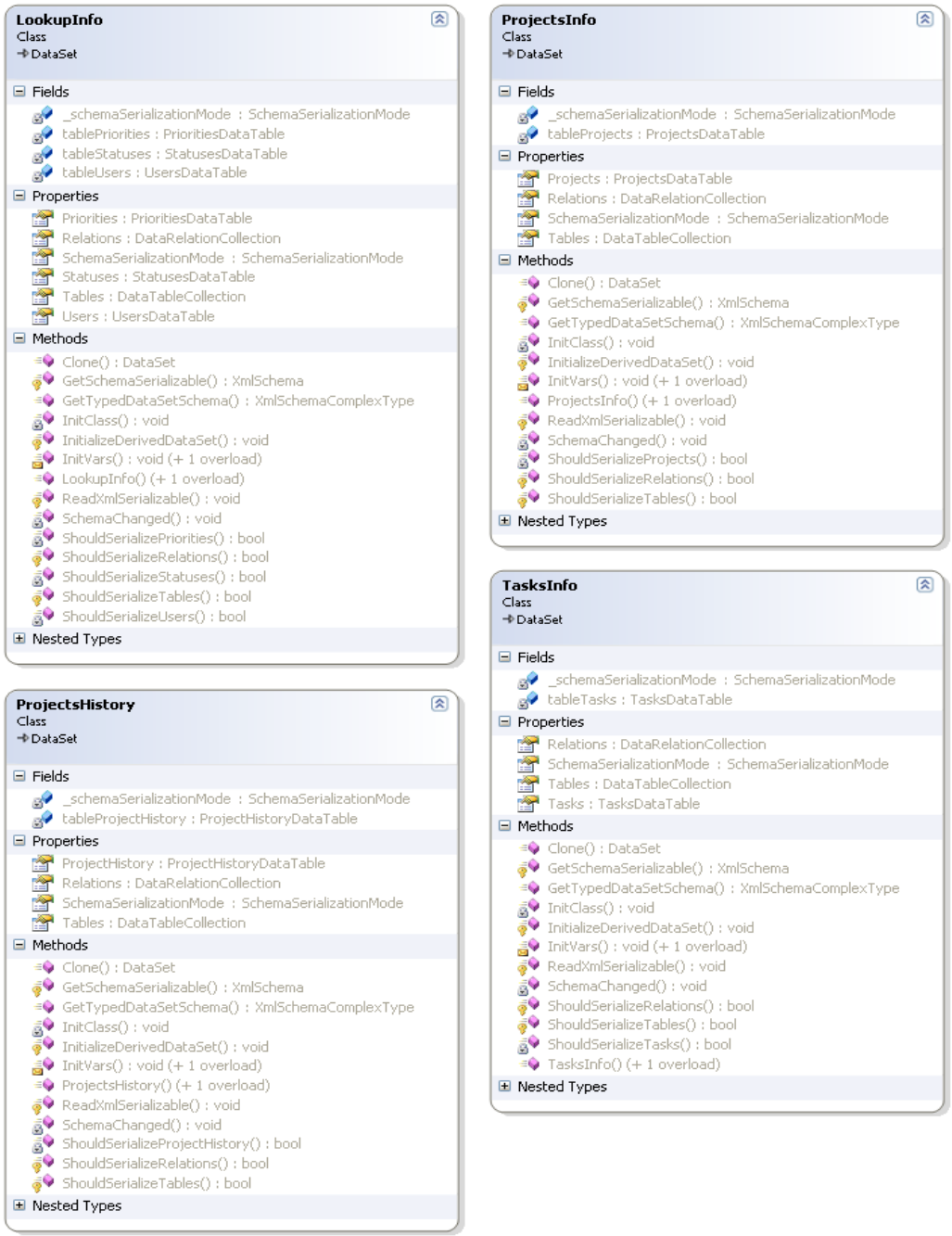


Фигура 29: Клас диаграма на класовете Repository използвани в аутентификационната услуга

Приложение Б: Клас диаграми на услугата за данни

Слой на услугата: Data Contracts проект

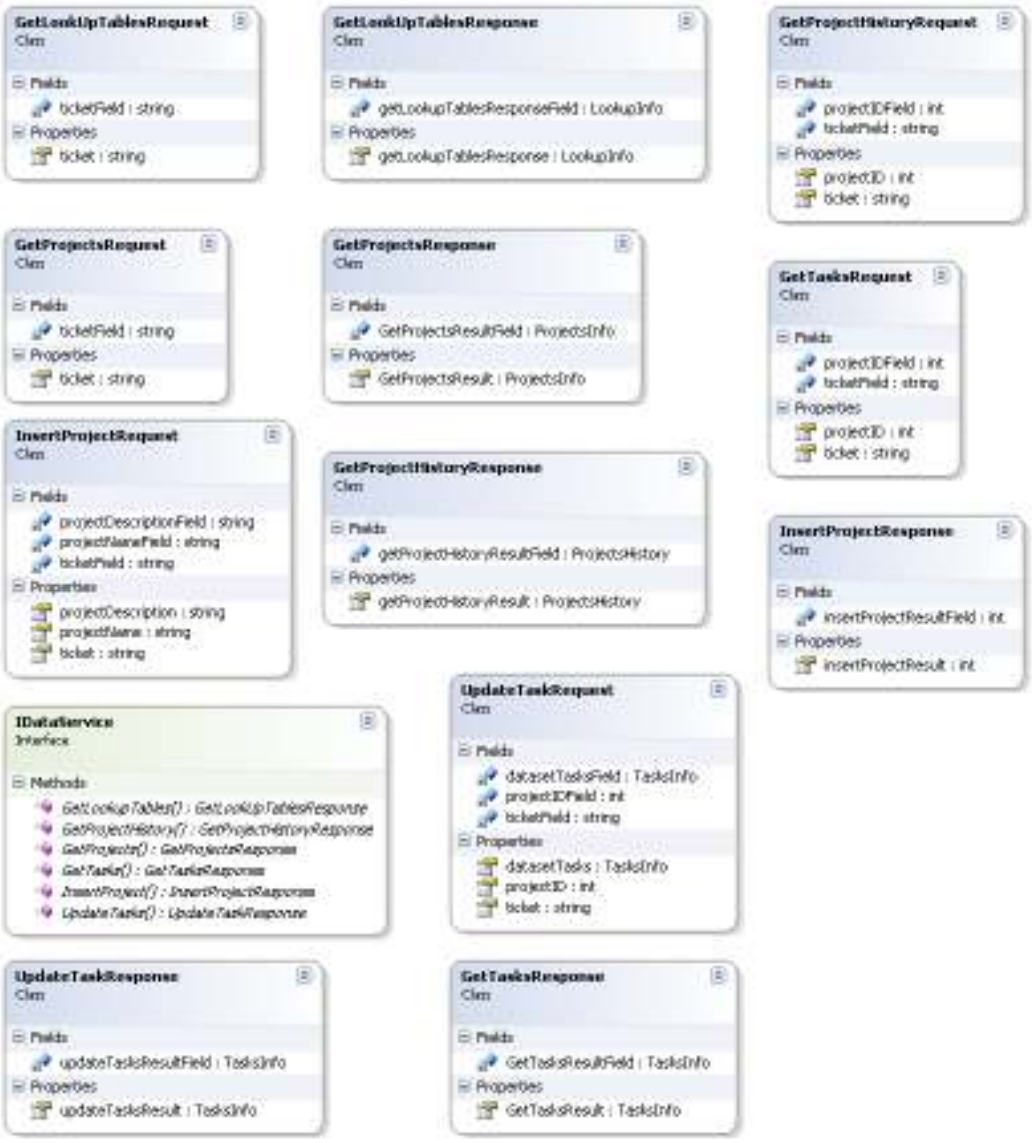
Data Contracts съдържа класовете определящи контрактите за данни на услугата за данни. Те са показани на фигура 30.



Фигура 30: Клас диаграма на Data Contract проекта, част от услугата за данни

Слой на услугата: Service Contracts проект

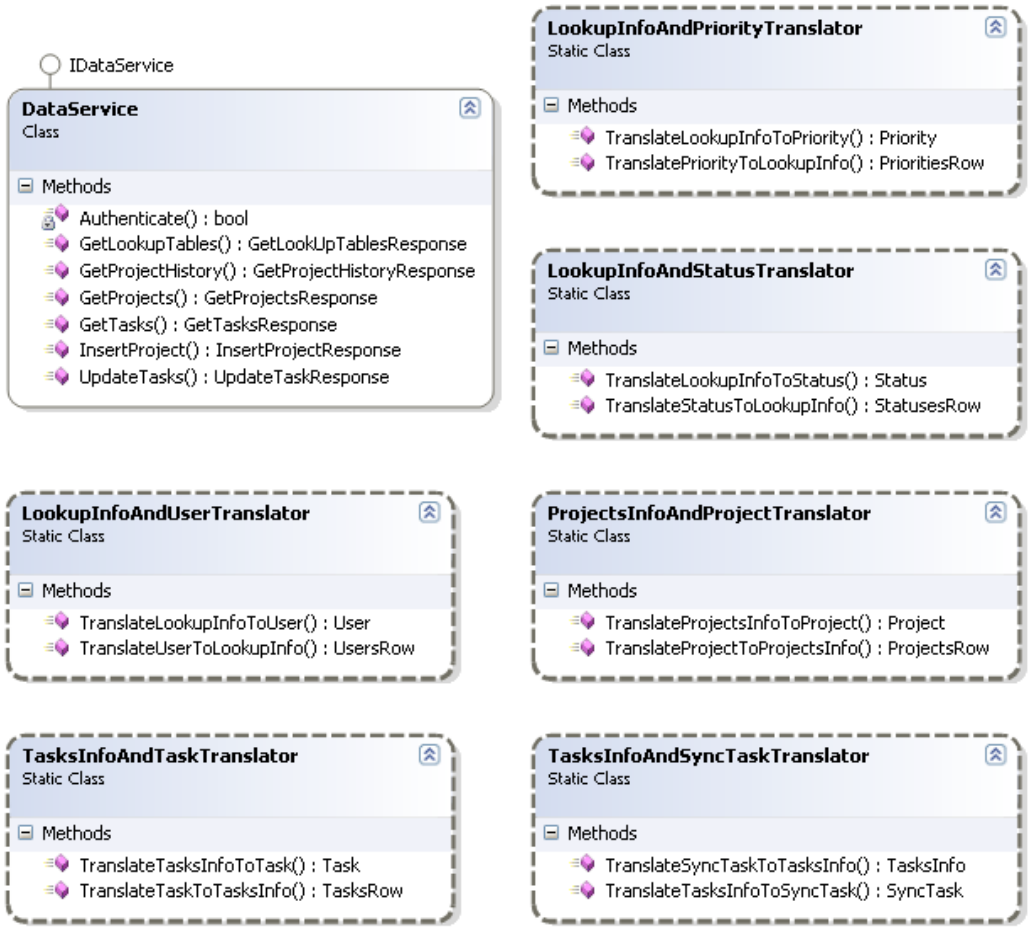
Фигура 31 показва клас диаграма на контрактите за входните и изходните съобщения и интерфейса на услугата за данни IDataService.



Фигура 31: Клас диаграма на Service Contracts проекта на услугата за данни

Слой на услугата: Service Implementation проект

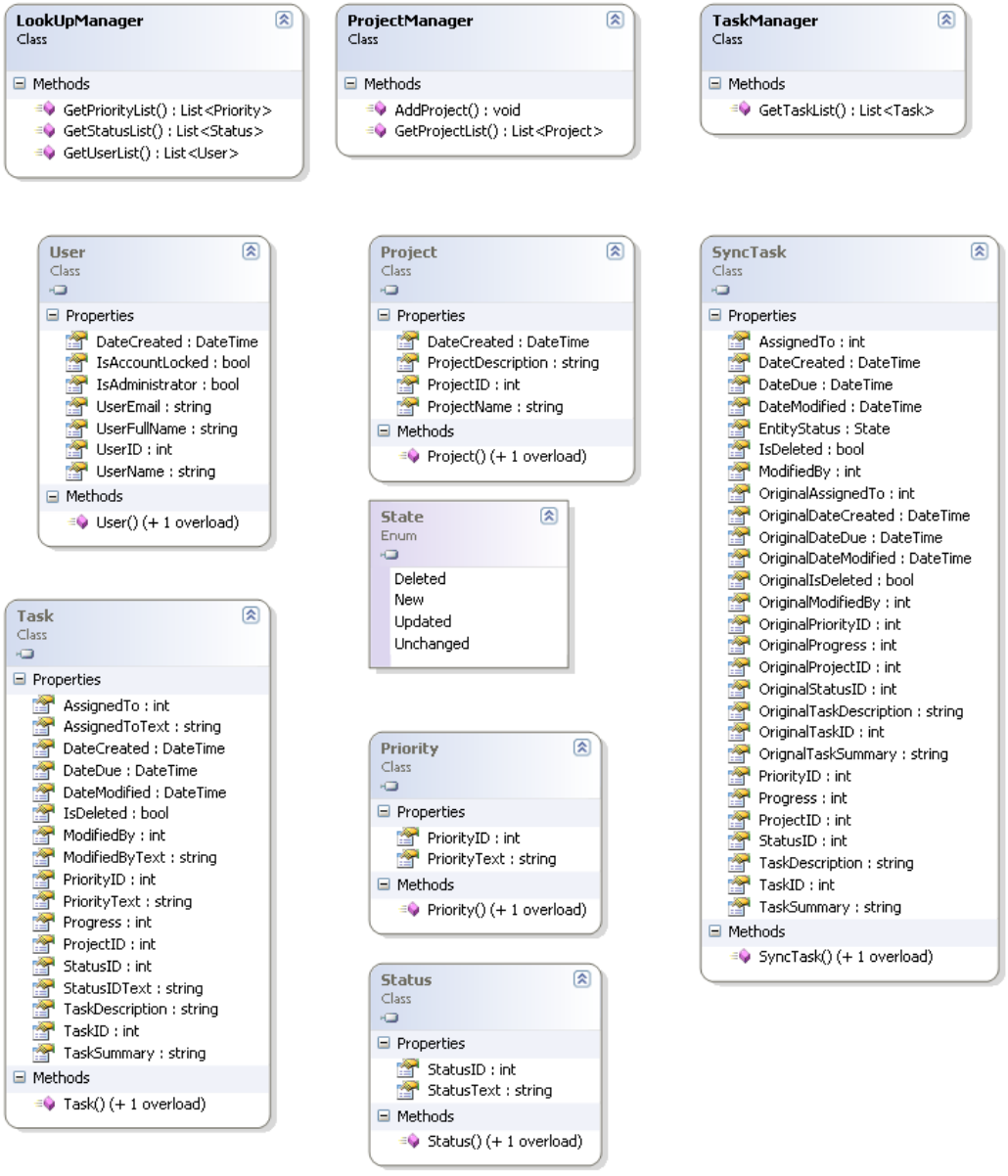
Имплементацията на услугата се състои от клас имплементиращ IDataService интерфейса и класове транслатори. За всеки от бизнес обекти има транслаторен клас, който го трансформира в съответния контракт за данни. Всеки транслатор има два метода за трансфомиране на обекта в контракт за данни, и за транслиране на данните от контракт в обект. Фигура 32 показва класовете със съответните им методи.



Фигура 32: Клас диаграма на Service Implementation проекта на услугата за данни

Бизнес слой: Business Logic и Business Entity проекти

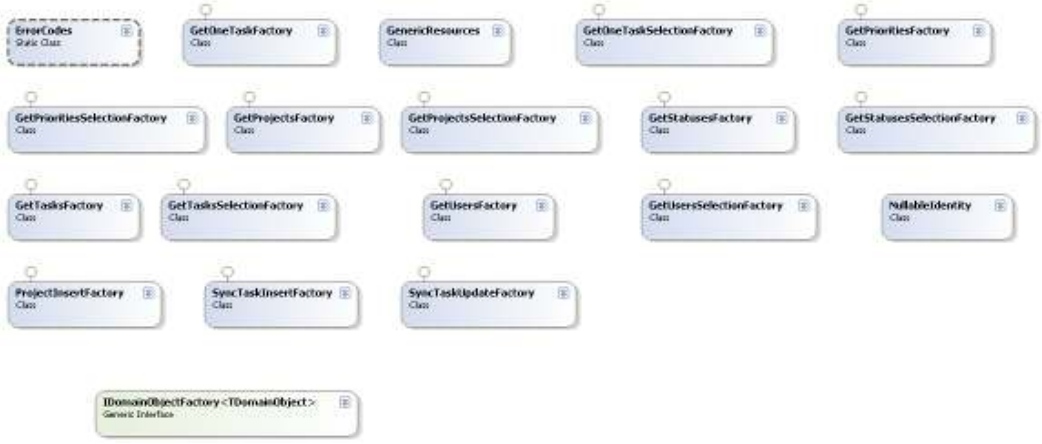
За всеки от контрактите за данните е дефиниран Manager клас. Той се използва от услугата за попълване на контрактите за данни. Manager класът изпълнява бизнес логиката свързана с бизнес обектите. На фигура 33 са показани бизнес класовете и Manager класовете, които ги използват.



Фигура 33: Клас диаграма на бизнес слоя на услугата за данни

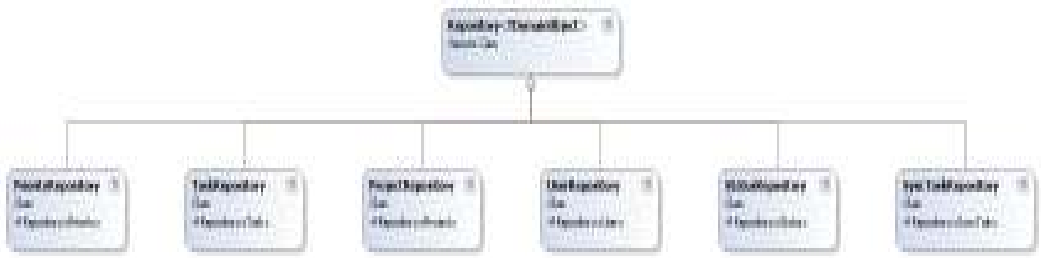
Слой за достъп до ресурси: Data Access

На фигура 34 и фигура 35 могат да се видят всичките класове в DataAccess проекта. Фигура 34 показва класовете имплементиращи шаблонът фабрика.



Фигура 34: Клас диаграма на класовете фабрики на услугата за данни

Фигура 35 показва Repository класовете за услугата на данни. Всеки един от бизнес класовете има асоцииран Repository клас .

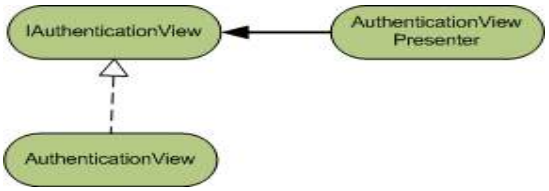


Фигура 35: Клас диаграма на класовете Repository на услугата за данни

Приложение B: Task Viewer диаграми

Изгледи и представители на Infrastructure.Module

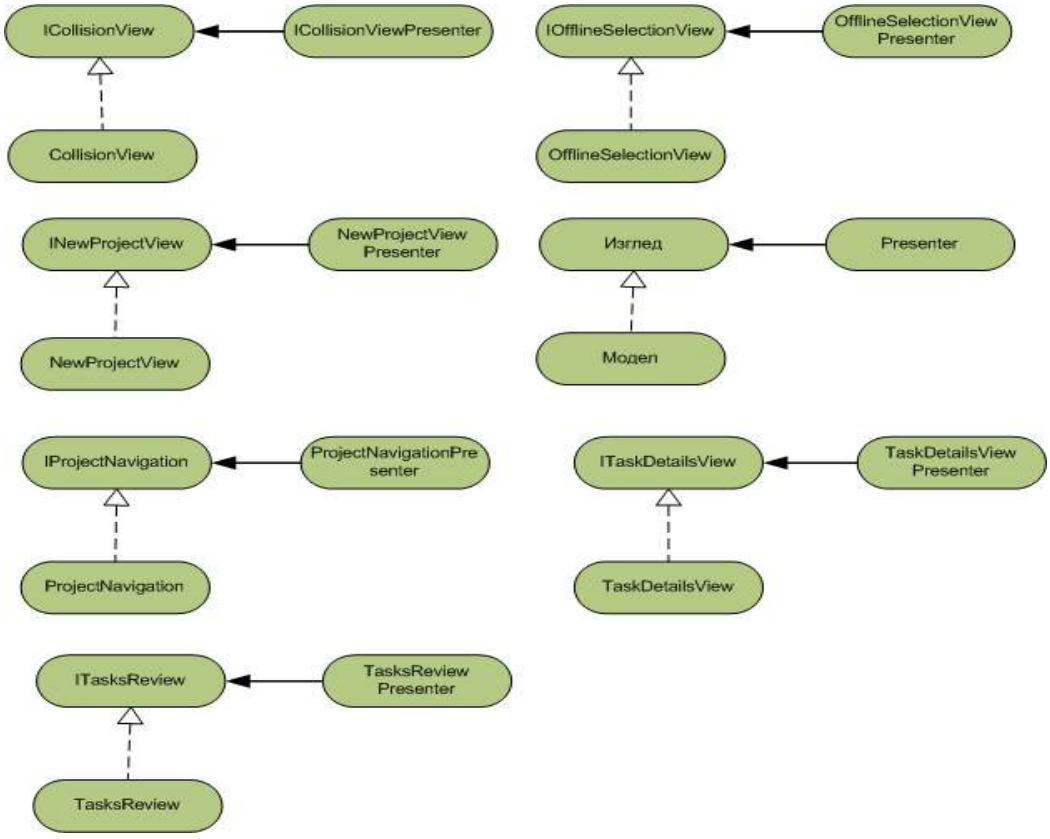
На фигура 36 е представен изгледът и асоциирания представител за аутентикационния излгед в Infrastructure.Module.



Фигура 36: Изгледи и представители на Infrastructure.Module

Изгледи и представители на Task Management модула

На фигура 37 са представени излгедите и представителите в TaskManagement модула.



Фигура 37:Изгледи и представители на Task Management проекта