Mendelova univerzita v Brně
Provozně ekonomická fakulta

Migrácia informačného systému na Microsoft SQL Server

Bakalárska práca

Vedúci práce:
Ing. Vít Ondroušek, Ph.D.

Autor práce:
Ján Repka

Brno 2016

Rád by som poďakoval vedúcemu tejto bakalárskej práce Ing. Vítu Ondrouškovi, Ph.D. za čas a odborné rady ktoré mi venoval.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Migrácia informačného systému na Microsoft SQL Server**
vypracoval/a samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací.*

Jsem si vědom/a, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.

Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 4. ledna 2016 _____

Abstract

Repka, J. Informatics system migration to Microsoft SQLServer. Brno: Mendel University in Brno, 2016.
This bachelor thesis contains analysis, design, implementation and test of solution serving to migrate persistence layer of selected module of the information system to Microsoft SQL Server through replication of data changes. The theoretical part contains an analysis of the current state of the information system with a focus on work with data, analysis of requests from the owner of the information system and the subsequent research of selected technologies. The practical part includes the design, implementation and test of the functionality of the implemented solution.

Keywords

Informatics system, SQL, Server, migration, .Net, WCF, WPF

Abstrakt

Repka, J. Migrácia informačného systému na Microsoft SQL Server. Brno: Mendelova Univerzita v Brne, 2016.
Táto bakalárska práca obsahuje analýzu, návrh, implementáciu a test riešenia slúžiaceho na migráciu perzistenčnej vrstvy vybraného modulu informačného systému na Microsoft SQL Server formou replikácie zmeny dát. Teoretická časť obsahuje analýzu aktuálneho stavu daného informačného systému so zameraním na prácu s dátami, analýzu požiadavkov zo strany vlastníka informačného systému a následnu rešerš vybraných technológii. Praktická časť obsahuje návrh, popis implementácie a test funkčnosti implementovaného riešenia.

Klíčová slova

Informačný systém, SQL, server, migrácia, .Net, WCF, WPF

Obsah

1 Úvod a cieľ práce

1.1 Úvod

Migrácia informačného systému na novú platformu, jazyk alebo dátové úložisko predstavuje rozsiahly a komplexný projekt ako zo strany návrhu a implementácie tak aj zo strany kapitálu. Je to investícia do budúcnosti firmy a zároveň príležitosť ako využiť know-how z pôvodného systému, nadchnúť pre neho nových ľudí a zákaznikom predviesť nový produkt. Migrovať sa môže len určitá komponenta systému (napr. Webové rozhranie) alebo systém celý. Pre migráciu celého systému je z pohľadu organizácie a plánovania potrebné najprv identifikovať kľúčove prvky a komponenty pôvodného systému, rozhodnúť ktoré, a v akej podobe, budú dostupné v novom systéme a následne ich modul po module do nového systému integrovať. Migrácia zväčša vyžaduje zachovanie simultánnej funkčnosti oboch systémov, aby počas projektu migrácie bol pôvodný systém stále plnohodnotne využiteľný zákazníkmi. Tato bakalárska práca rieši problém migrácie databázovej vrstvy konkretného informačného systému na Microsoft SQL Server formou near-real-time replikácie.
Predmetom zájmu je InNET ERP systém, což je ekonomický zákazkovo orientovaný informačný systém s modulárnou a flexibilnou štruktúrou. Je určený pre malé a stredne veľké firmy podnikajúce v oblastiach výroby, distribúcie, maloobchodu, veľkoobchodu a ekonomiky na spracovávanie obchodnej, ekonomickej a výrobnej agendy danej firmy.

1.2 Cieľ práce

Cieľom tejto práce je spraviť analýzu vybranej časti IS konkrétnej firmy so zameraním na databázovu vrstvu a spôsob práce s dátami.
Následne je potrebné získať požiadavky firmy.
Na základe analýzy požiadavkov je potrebné navrhnúť a implementovať vlastné riešenie pre tvorbu a ukládanie definícii modelu relačnej databáze.
Na základe týchto definícii sa urobí konverzia dát z aktuálneho IS do systému nového.
Po konverzii sa otestuje konzistencia dát v novom systéme.

2 Vybrané technológie .Net framework

V tejto kapitole je teoretický výpis k používaným metodikam, technológiam a knižniciam.

2.1 Technológia ADO .NET

ADO .Net je knižnica ktorá obsahuje objekty na prácu s relačnými dátami a vrstvou persistencie. Využívajú sa 3 spôsoby prístupu k dátam, a to vrstva pripojená, vrstva odpojená a entity framework.

Odpojená vrstva využíva objekt typu `DataSet` a vnorené objekty typu `DataTable` ktoré obsahujú kópiu dát (tabuliek) z dátoveho úložiska. Tie sa, za predpokladu korektného reťazca pripojenia, automaticky inicializujú pre každú novo-vytvorenú inštanciu. To umožňuje pristupovať k dátam a aplikovať zmeny na tieto objekty a ich dáta bez potreby komunikovať priamo s databázovým serverom. Týmto sa zmenšuje náročnoť na sieťový prevoz a redukuje sa množstvo spojení. Objekt `DataSet` udržuje záznam o zmenených dátach, ktoré po dokončení tranzakcie programátor aplikuje na databázový server zavolaním metódy `update()` ktorá využije už predpripravený generátor SQL príkazov a reťazec pripojenia pre poslanie kódu naô databázovy server. Samozrejmosťou je kontrola primárných a unikatných kľúčov, udržiavanie referencii a korektné dátove pretypovanie na úrovni práce s objektom. Nepredpokladá sa, že k dátovemu uložisku bude prístupovať a meniť dáta iná aplikácie, to obmedzuje využitie pre paralelne spracovanie. Tento prístup poskytuje relatívne konzistentnú a jednoduchú prácu s externými dátami. Nevyužíva sa pri tabuľkách s veľkým počtom záznamov a veľkých databázach kvôli náročnosti na RAM a zpomalenie inicializácie aplikácie.

Pripojená vrstva necháva prístup k databázovemu serveru a jeho dátam explicitne volať programátora. Využívajú sa pre to objekty `DataReader`, `Connection`, `Command` a `Parameter`. Načítane dáta sa ako u vrstvy odpojenej môžu udržiavať v objektoch typu `DataSet` a `DataTable` ale to len pre účely dočasného relačného úložiska v pamäti aplikácie. Jednotlive zmeny v štruktúre a príkazy pre databázovy server už programátor explicitne definuje a volá metodami `ExecuteReader()`, `ExecuteScalar()`, `ExecuteNonQuery()`. Výhodou u tohto prístupu je kontrola nad posielanými dotazmi a možosť paralelného prístupu k dátovemu úložisku z aplikácie, keďže zodpovednosť za správu tranzakcii sa prenáša na SQL server.

2.2 Technológia SMO

Shared Management Objects je .Net knižnica publikovaná firmou Microsoft. Slúži na správu Microsoft SQL Servera priamo z prostredia aplikácie. Obsahuje objekty ako `Server`, `Database`, `Table`, `View`, `Function`, `Column`, `Index` a iné.

Spolu s pripravenými atribútmi a metodami nad týmito objektami umožňuje programátorovi spravovať dané objekty bez potreby písania T-SQL kódu. Pri použití vývojového software Microsoft Visual Studio objekty podporujú funkciu IntelliSense čo ešte viac zjednodušuje prácu s objektami.

2.3 Komunikácia klient-server

Windows Communication Foundation je rozhranie (API) ktoré ponúka jednotný, unifikovaný a rozšíriteľný programový model objektov. Môže byť použitý pre interakciu so širokou škálou diverzných technológii. (TROELSEN, 2012)
Čo robí WCF unikátnym je využívanie tzv. Endpoint-ov. Tie slúžia ako tunely pre komunikáciu s externými entitami. Vo WCF je možné nakonfigurovať týchto endpointov viac a tým je schopné paralelne komunikovať formov binárných správ s Windows aplikáciou a zároveň pomocov JSON/XML a HTTP/TCP komunikovať s Java clientom alebo PHP webstránkou. Ďalšiou z unikátnostni tohto rozhrania je manažment inštancii a sessions. To sa dá nakonfigurovať pred spustením WCF služby ako atribút. Môže to byť jedná inštancia ktorá obsluhuje všetky klientske pripojenia sériovo, môže to byť nová inštancia pre každého pripojeného klienta(paralelne a nezávislé spracovanie) a dokonca nová inštancia pre každé dotázanie(call) od klienta.

2.4 Uživateľské rozhranie

Windows Presentation Foundation je framework ktorý sa snaží nahradiť zastaralý Windows Forms, 2D/3D modelovanie a unifikovať vývoj UI do jediného nástroja a to do miery rendrovania iba cez DirectX bez potreby písať kód pre DirectX API. Využíva jazyk XAML, ktorý umožňuje real-time renderovanie vo Visual Studiu bez potreby buildovania aplikácie. XAML ponúka data-binding, čo extrémne uľahčuje asynchrónnu prácu s UI prvkami. WPF ponúka širokú škálu animácii, štýlov a taktiež pokročilé grafické technológie ako je anti-aliasing, priehľadnosť, blur a podobné.

2.5 Windows služba

Windows Service zabezpečuje dlhodobé spustenie aplikácie vo vlastnej session, pod definovaným uživateľom a s automatickým spustením/vypnutím pri štarte/vypnutí operačného systému. Je potrebné naimplementovať metódy `OnStart()` a `OnStop()` ktoré zabezpečia vytvorenie inštancie danej aplikácie pri štarte a korektné zastavenie pri vypnutí. Visual Studio ponúka GUI pre vytvorenie inštalačného balíčku danej služby spolu s aplikáciou.

3 Metodika

Táto bakalárska práca sa riadi Unified process alebo Unified Software Development Process. Je to jednotný, iterárny a inkrementálny proces vývoja informačných systémov. Cieľom v tomto processe je informačný system navrhnuť, identifikovať ako rizika tak aj časovú a zdrojovú náročnosť vývoja.
Ako prvy bude analýzovaný aktuálny stav riešenia u zákaznika. Následne budú spísane ako funkčné tak aj nefunkčné požiadavky zákaznika. Analýza riešenia bude vytvorená po konkretizovaní cieľov vychádzajúcich z požiadavkov zákaznika. Jej súčasťou je tvorba analytických tried. Následne bude vytvorený návrh riešenia. Po ukončení návrhu bude započatý proces implementácie. FoxPro 2.X je databázový systém, a teda bude mať preddefinovanú hlavičku každej tabuľky. Zistíme či existuje OLE DB provider pre daný tabuľkový systém a či je využiteľný v programovaciom jazyku C#. Pokiaľ neexistuje, tak bude naimplementový prekladač binárných dát ktorý zabezpečí načítanie hlavičky tabuľky a prístup k dátam podľa dokumentácie štruktúry súborov typu dbf.

Následne bude vytvorený prvý modul aplikácie Data Replicator ktorý je zodpovedný za prístup k zdrojovým dátam. Do tohto modulu budú následne pridané funkcie na prekladanie štruktúry zdrojovej tabuľky do internej štruktúry vytváranej aplikácie, funkcie na analýzu vlastností zdrojových dát a generátor dotazov pre selektívny prístup k dátam.

Po otestovaní funkčností modulu pre prístup k zdrojovým dátam bude vytvorený modul ktorý bude pristupovať k cieľovému databazovému systému. Ten bude obsahovať ako metódy pre inicializáciu systémových objektov na strane databázového systému tak aj metódy pre modifikáciu replikovaných dát a metody pre prístup k metadátam o replikovaných tabuľkách v systémovej databáze.

Modul zodpovedný za replikáciu bude pridaný ako posledný. Ten bude obsahovať inštancie ako zdrojového tak aj cieľového systému. Z dát ktoré mu tieto systémy persistencie poskytnú následne vygeneruje sadu príkazov pre cieľový systém ktoré zabezpečia konzistentný stav voči zdrojovému systému.

Triedy a objekty týchto troch modulov budu následne pridané do knižnice objektov.

Následne budu vytvorené objekty potrebné pre Windows Communication Foundation framework spolu s korektnou konfiguráciu pre akceptovanie klientskej komunikácie cez .net.tcp protokol a správnym manažmentom inštancii.

Potom bude vytvorená služba Windows Service, ktorá bude slúžiť ako hostiteľ pre inštanciu Data Replicatora.

Následne začne implementácia grafického rozhrania. Ako prvý bude naimplementovaný modul pre komunikáciu so službou. Následne budu vytvorené grafické objekty pre prezentáciu a úpravu metadát replikácie spolu s funkčnými prvkami.

Finálne, budú do replikačného procesu pridané tabuľky definované zákazníkom a overená integrita dát.

4 Analýza aktuálneho stavu

InNET ERP Systém má modulovú štruktúru a je vyvíjaný v programovaciom jazyku Visual FoxPro. Visual FoxPro je databázovo a objektovo orientovaný programovací jazyk v minulosti vyvíjaný firmou Microsoft. Vývoj sa zastavil v roku 2004 a posledná oficiálna verzia programovacieho prostredia je 9.0 Service Pack 2. Dáta sú ukladané v tabuľkovom systéme FoxPro 2.x, kde je každá tabuľka reprezentovaná samotným súborom s príponou .dbf, ktorý obsahuje ako štruktúru tabuľky tak aj samotné dáta. Zákazník má vyvinuté upravené vývojové prostredie softwaru Visual FoxPro 9.0 s vlastným Dátovým managerom špecificky navrhnutým pre vývoj modulov InNET ERP Systému. Dátový Manager sa stará o priradenie tabuliek k modulu, kompatibilnú štruktúru tabuliek a indexy nad atribútmi.

4.1 Dátová vrstva

InNET ERP Systém používa dve módy pre dátové úložisko. Prvý mód je určený pre jedného uživateľa. Dáta sú uložené v inštalačnom priečinku klienta.

Druhý mód umožňuje zdieľať dáta pre viacero uživateľov ako aj simultánny prístup k dátam. Centralizované a zdieľané úložisko slúži ako sieťový disk na ktorý sa uživatelia v rámci firemnej siete pripájajú. Po inštalácii klienta na lokálnom počítači je potrebné uživateľovi natrvalo pripojiť sieťový disk so zdieľanými dátami.

4.1.1 Bezpečnosť

Pre prístup k dátam z vnútra systému je potrebný uživateľ s dostatočným opravnením v prostredí aplikácie. Správa uživateľov a ich prístupové práva sú riešené licenčným modulom, ktorý na základe administratorom definovaných práv a prístupov púšťa uživateľov do jednotlivých modulov. Každý modul ma dostupnú sadu nástrojov na prácu s dátami, ktoré však môžu využívať všetci uživatelia daného modulu. Možnosť obmedziť práva uživateľov v rámci modulu na presne definované činnosti systém neumožňuje.

Pre prístup k dátam z vonkajšieho prostredia je potrebné mať prístup k dátovým súborom na úrovni operačného systému. Dáta v súboroch a ani samotné súbory nie-sú šifrované. V prípade používania softwaru v móde so zdieľaným dátovým úložiskom ma každý užívateľ IS prístup ku všetkým dátam a tabuľkám, čo je nevyhovujúce pre moduly s citlivými dátami ako sú mzdy, dph a banka.

Inštaláciou na terminálový server a spúšťaním softwaru lokálne cez pripojenie vzdialenej plochy sa dá prístup k súborom obmedziť avšak nie znemožniť.

4.1.2 Štruktúra tabuliek

FoxPro 2.X ponúka dostatočný súbor dátových typov ako aj možnosti indexácie. Maximálna veľkosť tabuľky je 2GB. Chýba však implementácia cudzieho kľúča, hodnoty null a komputovaných atribútov. Určitá forma cudzieho kľúču je implementovaná priamo v kóde programu. Nedostatočná implementácia cudzieho

kľúču a aj zjednodušenie prezerania a jednoduchosť ojedinelých servisných zásahov do dát spôsobila veľkú duplicitu údajov.

4.1.3 Údržba

O aktualizáciu indexov, prepočet komputovaných hodnôt atribútov, aktualizovanie štruktúry tabuliek, kontrolu cudzích kľúčov a kontrolu konzistencie sa stará modul Údržba a reindexácia. Využíva sa hlavne pri neočakávanom ukončení programu, pri podozreniach na chybne komputované dáta a tiež na úpravu štruktúr po vydaní aktualizácie.

4.1.4 Aliasy

Software môže mať danú tabuľku otvorenú iba jeden krát. To znamená, že ak potrebujeme zistiť či sa práve spracovávaná hodnota v tej istej tabuľke už nachádza, musíme si tabuľku otvoriť druhý krát pod iným aliasom, teda aj iným menom. Existuje možnosť zložených SQL dotazov, to však presunie kurzor na iný záznam aký aktuálne spracovávame. Otvorením danej tabuľky pod aliasom nám v pôvodnej tabuľke ostane kurzor na spracovávanej hodnote a duplicitu kontroluje v tabuľke otvorenej pod aliasom.

4.2 Grafické uživateľské rozhranie

Grafické rozhranie je pre užívateľov je veľmi priateľské a doba odozvy je minimálna. Odozvy tabuliek sú vzhľadom na relatívne malo záznamov (max 200000 na tabuľku) rýchle, ale to za predpokladu prístupu obmedzeného počtu užívateľov k tabuľkám. Pre plánované rozšírenie funkčnosti a prístupov do systému teda aktuálny tabuľkový systém funkciami neodpovedá. Visual FoxPro využíva na prehliadanie dát techniku Rushmore, ktorá exemplárne optimalizuje SQL dotazy, efektívne využíva indexy a užívateľovi poskytuje veľmi dobrú odozvu.
Systém disponuje vlastným grafickým builderom objektov a teda tvorba základných formulárov a mriežky na zobrazovanie dát je relatívne jednoduché.

4.3 Modul Zákazky

Modul je určený na evidenciu a spracovanie zákaziek – prijatých objednávok v obchodných alebo výrobných firmách a sledovanie ich postupnej expedície. Modul k svojej funkčnosti využíva nasledujúce tabuľky:

- ZHLA – Hlavičky Dokladov
- ZPOL – Položky Dokladov
- ZSPLN – Splnenie Položiek
- ZCTYP – Číselník typov zákaziek

K tomu využíva číselníky Základného modulu: HPARTNER – Číselník partnerov, HSTRED – Číselnik stredísk.

5 Požiadavky zákazníka

Software tvoríme pre zákazníka a jedine ten vie špecifikovať čo má software robiť a akú funkcionalitu má ponúkať. V závislosti od komplexnosti projektu sa využívajú iteračné postupy v ktorých sa postupne definuje cieľova podoba systému. Keďže od zákazníkom špecifikovaných požiadavkov sa odvíjajú všetky následujúce procesy návrhu, implementácie a testovania je tento proces najdôležitejšiou časťou vývoju software. Pre špecifikáciu zákaznikových požiadavkov využijeme metódu rozdelenia na funkčné a nefunkčné požiadavky. Funkčné požiadavky špecifikujú aké funkcie má software ponúkať. Nefunkčné požiadavky špecifikujú ako ma systém tieto požiadavky zabezpečovať a prípadne obmedzenia. Keďže sa riešenie skladá z aplikácii Data Replikátor a Data Manager, rozdelili sme funkčne a nefunkčne požiadavky do týchto skupín.

5.1 Data Replikátor

Primárnou funkciou data replikátora je podľa časového rozvrhu kontrolovať zdrojový systém a hľadať nové,zmazané a zmenené záznamy a replikovať ich zmenu na cieľový systém. Sekundárnou funkciou je ponúkať pre Data Manager prístup k zdrojovému systému vo forme schémy zdrojových tabuliek.

5.1.1 Funkčné požiadavky

Data Replicator bude windows služba a bude sa spúšťať pri každom štarte operačného systému. Operátor musí mať možnosť službu vypnúť a reštartovať. Data Replicator bude udržiavať cieľový systém konzistentný a aktualizovaný voči zdrojovému. Data Replicator bude využívať definície vytvorené Data Managerom uložené v systémovej databáze, pre kontrolu štruktúr tabuliek a transformovanie zdrojových dat do cieľovej podoby. Data Replicator bude využívať rovnaké rozhranie pre prístup k zdrojovým súborom ako využíva Data Manager. Data Replicator bude mať 2 módy kontroly zdrojových dát, a to: kontrola na základe porovnania časových známok a úplna kontrola. Časovo orientovaná kontrola bude hľadať iba dáta zmenené od poslednej úspešnej replikácie. O štruktúru sa nestará. Jedine v prípade chyby vloženia sa zavolá úplna kontrola. Úplná kontrola bude kontrolovať celý obsah a štruktúru cieľového systému. Data Replicator v prípade problémov informuje operátora cez e-mail.

5.1.2 Nefunkčné požiadavky

Data Replicator bude spúšťaný ako Windows služba. Data Replicator nesmie meniť dáta v zdrojovom systéme. Účet pod ktorým služba Data Replicator bude spustená musí mať opravnenia db_creator na úrovni Microsoft SQL Servera a oprávnenie run as a service na úrovni operačného systému

5.2 Data Manager

Data Manager je grafické rozhranie ponúkajúce operátorovi správu replikovaných tabuliek. Aj keď je Data Manager nezávislá aplikácia, k svojej činnosti potrebuje spustenú službu Data Replikátora.

5.2.1 Funkčné požiadavky

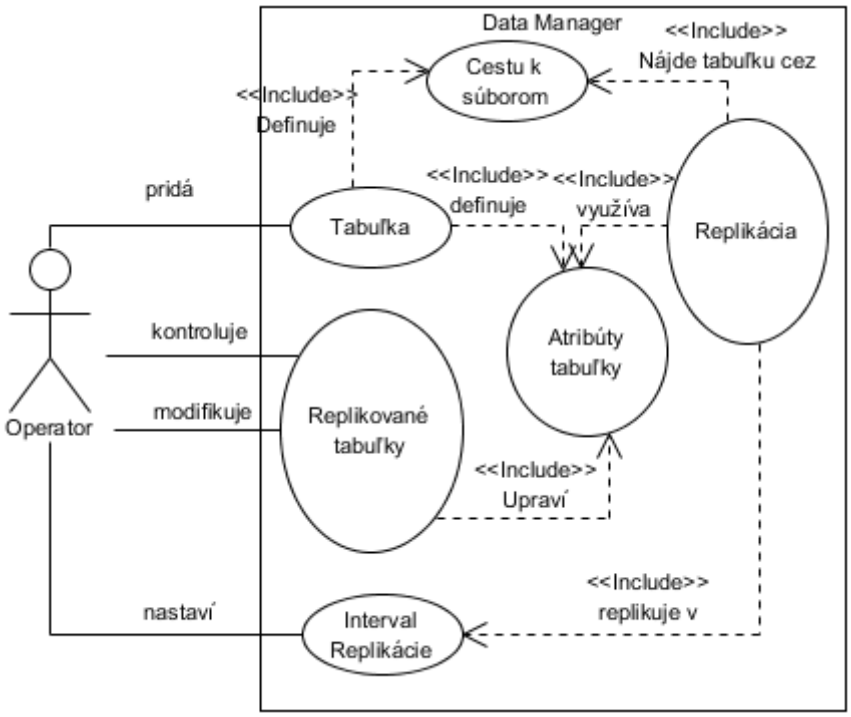
Engine budú ponuknuté atributy v C# dátovom type. Zdrojové dátové typy budú iba DateTime, Decimal, Ordinal, String, Logical. U typu String ponúkne aj maximálnu dĺžku hodnoty atribútu. Zdrojové dáta budú ošetrené časovou nálepkou poslednej zmeny. Zdrojové dáta budú prezentované DataManageru v pred-definovanej tabuľke nezávisle na zdroji. Zdrojová štruktúra dát bude prezentovaná DataManageru v pred-definovanej tabuľke nezávisle na zdroji. Data Manager bude obsahovať komponentu IDestinationFormat, na základe ktorej sa bude rozhodovať aké dátove typy navrhne pre cieľové úložisko a ako ich do cieľového uložiska vloží. Data Manager bude kontrolovať dostupnosť tabuľky v cieľovom umiestnení a prípadnu editáciu urobí online bez potreby zrušenia a znova vytvorenia. Data Manager bude na základe analýzy dát daného zdrojového atributu navrhovať vlastnosť „Nullable“. Data Manager bude v cieľovej aplikačnej tabuľke vytvárať primárny kľúč GUID za predpokladu, že v zdrojových tabuľkách ešte neexistuje. Data Manager vyhľadá v zdrojových dátach prítomnosť primárneho kľúča typu GUID podľa názvu atributu, dátoveho typu a maximálnnej veľkosti. Data Manager bude ukládať štruktúru zdrojovej tabuľky do systémovej databázy. Data Manager bude udržiavať v systémovej databáze informácie potrebné pre vyžiadanie si určitej tabuľky od zdrojového systému. Data Manager si bude ukládať pravidla pre transformáciu zdrojových dát do cieľových v systémovej databáze. DataManager vytvorí systémovú a aplikačnú databázu pri prvej dostupnosti cieľového umiestnenia a dostatočných informáciach o zdroji. GUI bude ponúkať prehľadávanie v lokálnom súborovom systéme na lokalizáciu súborov .dbf. GUI bude ponúkať prehľad zdrojových tabuliek dostupných vo vybranom zdrojovom priečinku. GUI bude umožňovať editáciu názvu cieľovej tabuľky. GUI bude umožňovať na úrovni definície cieľového typu atribútu editáciu názvu, dátového typu, veľkosti dátového typu, povolenie hodnoty null, definíciu primárneho kľúča a povolenie daneho atribútu na replikáciu. GUI bude obsahovať rozhranie pre definíciu primárnych, unikátnych a cudzích kľúčov. GUI bude obsahovať tlačidlo pre okamžité skontrolovanie konzistencie zdrojového a cieľového úložiska. GUI bude ponúkať prehľad aktuálne replikovaných tabuliek, dátum poslednej kontroly a detaily k tabuľke.

5.2.2 Nefunkčné požiadavky

Jednotlivé komponenty budú oddelené a bude zabezpečená znovu použiteľnosť kódu. Komunikácia s Data Replicatorom bude obmedzená na lokálny systém. Operátor musí mať oprávnenia administrátora v operačnom systéme. GUI bude navrhnuté pomocov Windows Presentation Foundation.

5.3 Vizualizácia požiadaviek

Na vizualizáciu požiadavok použijeme diagram USE CASE, ktorý vizuálne popisuje prípady užitia daného softwaru. Tento diagram znázorňuje iba interakciu medzi Operátorom a aplikáciou Data Manager. Data Replicator nebude mať uživateľské rozhranie a jediná interakcia uživateľa je potrebná v prípade spustenia, reštartovania alebo vypnutia služby z rozhrania pre správu služieb operačného systému Windows.



Obr. 1 USE CASE Diagram Data Manager

5.4 Konkretizácia cieľov

Vytvoriť aplikáciu Data Manager a v ňom vytvoriť grafické uživateľské rozhranie potrebné pre operátora na výber tabuliek k replikácii, definovanie ich štruktúry a výber atribútov k replikácii. Vytvoriť aplikáciu Data Replikátor ako Windows službu. Za pomoci vytvorených aplikácii pridať do replikácie následujúce tabuľky z modulu Zákazky: ZHLA – Hlavičky Dokladov, ZPOL – Položky Dokladov, ZSPLN – Splnenie Položiek, ZCTYP – Číselník typov zákaziek.

Pre úplnosť fungovania replikácie je potrebné pridať aj zdieľané tabuľky z modulu Startwin využívajúce modulom Zakázky: HPARTNER – Číselník Partnerov, HSTRED – Číselník stredísk, HOBLAST – Číselník tras, HPARTPB – Číselník pobočiek, HTOVAR – Číselník tovarov a služieb, HAKCIE – Predajné akcie, HTOVARD – Druh tovaru.

6 Analýza riešenia

Analýza riešenia nám modeluje základné správanie vytváraného systému. V návrhovej časti riešenia predpokládame rozšírenie o ďalšie triedy.

6.1 Komunikácia so zdrojovým systémom

Po zvážení dostupných riešení pre dotazovací prístup k zdrojovému databázovému systému bol vybraný prístup cez rozhranie OLEDB. To je navrhnuté pre unifikovaný prístup k dátam z rôznych zdrojov. Ponúka súbor operácii nad zdrojovým systémom, ktoré môže aplikácia využívať bez potreby vedieť s akým zdrojovým systémom komunikuje.
Firma Microsoft, ako hlavný aktér vo vývoji platformy Visual FoxPro, ponúka na stiahnutie svoj VFP OLEDB Provider. Je využiteľný iba na operačných systémoch Windows a pre tabuľky verzie FoxPro 2.0 a vyššie.
Prístup k dátam je dostatočný, provider správne prekladá dotazy a rýchlosť ktorou s aplikáciou komunikuje je postačujúca. Problém nastáva pri dotazovaní na schému tabuľky. Informácie ktoré provider poskytne sú iba dátový typ a maximálna dĺžka záznamu vypočítana z najdlhšieho reťazca v dátach daného atribútu. Vlastnosti ako primárny kľúč, unikátny kľúč alebo povolená hodnota `null` sú nedostupné. Tie však dokážeme definovať explicitne v uživateľskom rozhraní a za pomoci automatizovanej analýzy dát aj poskytnúť návrhy na vlastnosti atribútov.
Pre distribúciu s vytvorenou aplikáciou je potrebné tento provider nainštalovať na lokálny systém na ktorom bude spustená služba Data Replicator.

6.2 Analytické triedy

Funkciou triedy `DataSource` je poskytovať prístup k zdrojovému databázovému systému. Využíva OLEDB provider a pomocov neho získava ako samotné dáta tak aj schému tabuľky z ktorej dáta číta. Vzhľadom na požiadavok zákazníka nemeniť dáta v zdrojovom systéme, trieda neobsahuje žiadne metódy na ukládanie dát. Metóda `getSourceTableList()` je zodpovedná za výčet všetkých dostupných tabuliek ktoré provider VFPOLEDB našiel v zadanom priečinku.
Trieda `DataDestination` je zodpovedná za interakciu s cieľovým databázovým systémom. Podobne ako trieda `DataSource`, ponúka metódy pre čítanie cieľových dát a čítanie schémy tabuľky. Po tom čo uživateľ upraví definíciu atribútov v grafickom rozhraní, metóda `addTableToReplication()` zabezpečí uloženie potrebných metadát do systémovej databázy a vytvorenie aplikačnej tabuľky na základe týchto definícii. Metóda `populateTable()` je zodpovedná za pridávanie a aktualizovanie záznamov v aplikačnej tabuľke.

Za samotný proces replikácie zodpovedá `ReplicationManager`. Ten na základe časového intervalu kontroly sleduje dáta ktoré mu ponúka `DataSource` a

porovnáva ich s dátami z `DataDestination`. Metóda `replicate()` sa zavolá selektívne pre každú tabuľku a za využitia metód `comparePKeys()` a `compareTimeStamp()` rozozná zmenu v zdrojových dátach a pošle triede `DataDestination` zoznam záznamov ktoré musí upraviť.

`Service` je trieda ktorá slúži ako vstupná brána pre `DataManager`. Pri spustení služby vo Windows táto trieda vytvorí inštanciu ReplicationManager a otvorí komunikačné porty na lokálnom počítači cez ktoré sa bude dať k službe pripojiť.

Grafické uživateľské rozhranie je postavené na metódach triedy `UserInterFace`. Trieda priamo komunikuje s triedou `Service` cez protokol .NetTCP. Atribúty udržiavaju dáta o aktuálne spracovávanej tabuľke v grafickom rozhraní, ktoré získali zo služby cez metódy `getTables()` a `getColumns()`. Tabuľku spracováva operátor, ktorý upravuje jej štruktúru pred pridaním do replikovaných tabuliek. Po dokončení sa zavolá metóda `addTableToReplication()` triedy `Service` ktorej sa predájú operátorom upravené definície.

7 Návrh a implementácia

Návrhové triedy presne špecifikujú funkcie tried a spôsob ktorým tieto triedy medzi sebou komunikujú. Roztriedili sme ich do modulov, podľa ich funkčnosti a asociacie s analytickými triedami. Implementácia zahrňa bližšiu špecifikáciu algoritmov a procesov využívaných v aplikácii.

7.1 Obslužné triedy

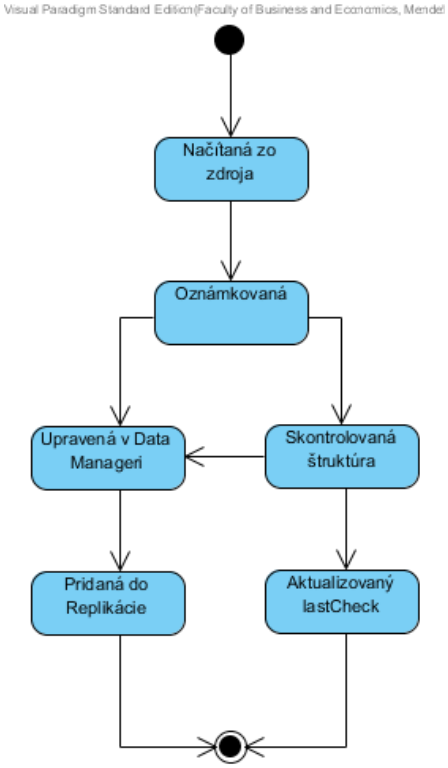
Obslužné triedy sú využívane naprieč celou aplikáciou. Statická trieda `Factory` je vytvorená ako návrhovy vzor, ktorý slúži na vytváranie zdieľaných objektov ako sú reťazce znakov pre pripojenie na server alebo inštancie tried implementujúce určite rozhranie. Metóda `serialize()` je využívaná pri posielaní dát od grafického rozhrania k službe. Je bližšie špecifikovaná v module DataSource. Jej návratovou hodnotou je reťazec znakov obsahujúci XML kód.

Obr. 2 Obslužné triedy

Metóda `deSerialize()` je využívana službou na prečítanie XML kódu predaného ako reťazec znakov v parametri. Výstupom je inštancia triedy splňujúca rozhranie `IDataSourceProp`. Pre využitie vytvorenej inštancie cieľovým systémom je potrebné ju pretypovať na špecifickú triedu pre daný cieľový systém, ktorá musí implementovať už spomenuté rozhranie. Metódy ktoré poskytujú databázovým

vrstvám potrebné reťazce znakov sú `getDataSourceConnection()` a `getDataDestinationConnection()`. Tieto generujú potrebné reťazce využitím atribútu príslušného systému a názvu projektu. Metóda `getDataSource()` vytvára inštanciu triedy `VFPDataSource` potrebnú pre prístup k zdrojovému systému. Na prístup k cieľovému systému využívame triedu `SQLServerDD` ktorej inštanciu vytvorí metóda `getDataDestination()`. Využitím návrhového vzoru Factory sme vylepšili nezávislosť vrstvy persistencie od vrstvy aplikačnej logiky.

Triedy `TableInfo` a `ColumnInfo` sú využívané ako dátové typy, ktoré slúžia na prenos potrebných metadát.



Obr. 3 Stavový diagram triedy TableInfo

7.2 Modul DataSource

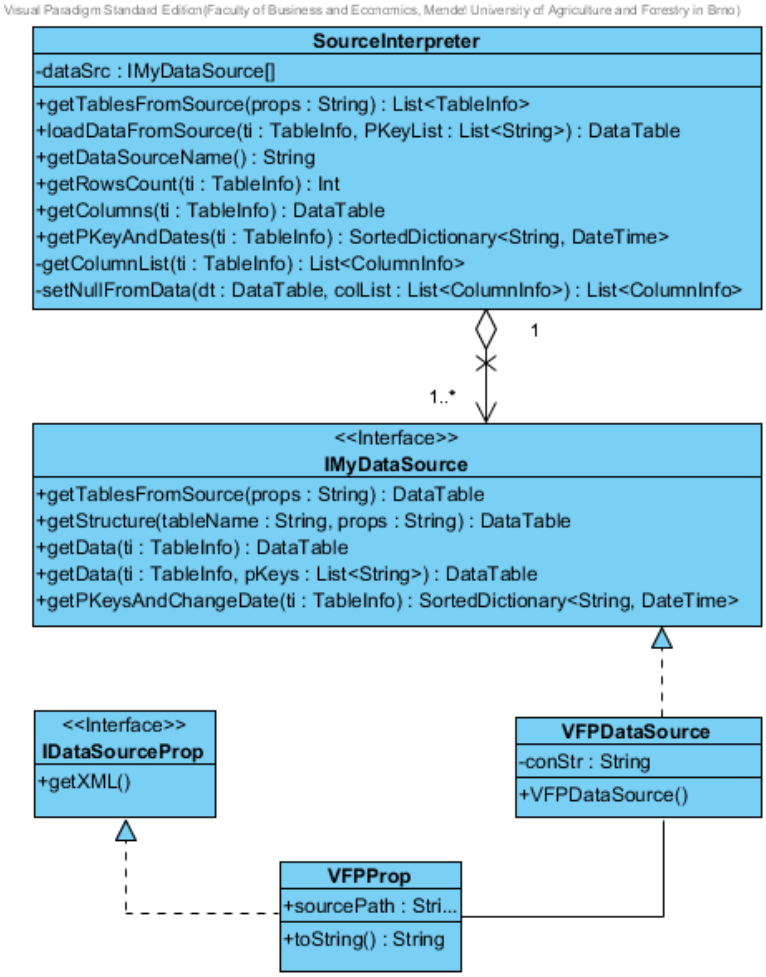
Modul DataSource mal v analytickom návrhu iba jednu analytickú triedu. Po hlbšej analýze problematiky pripájania k dátovemu zdroju sme vytvorili ďalšie štyri triedy.

Činnosť modulu môžeme rozdeliť na dve časti ktoré spolu úzko súvisia. Prvou časťou je prístup k samotným dátam a generované dotazovanie na zdrojový systém pre získanie potrebných záznamov k replikácii. Druhá časť sa stará o štruktúru zdrojových tabuliek, teda špecifikáciu jej atribútov. Kontrola štruktúry a jej prípadna úprava sa používa počas replikácie a je to plne automatizovaný proces ktorý ak zistí nový alebo zmazaný atribút, tak zmenu premietne do cieľovej

aplikačnej tabuľky. Prístup k štruktúre zdrojovej tabuľky potrebuje aj operátor pri pridávani novej tabuľky k replikácii.

Trieda `SourceInterpreter` je zodpovedná za upravenie a transformovanie zdrojových dát do internej štruktúry aplikácie. Privátny atribút dataSrc má v sebe uložené odkazy na všetky inštancie dátových zdrojov pridaných do aplikácie. Pre potrebu tejto bakalárskej práce pole obsahuje iba jeden odkaz a to na dátový zdroj pre systém VisualFoxPro. Verejná metóda `getTablesFromSource()` slúži na získanie dostupných tabuliek zo zdrojového systému na základe v parametri predanej cesty k zdrojovým dátam. Návratovou hodnotou je `List<TableInfo>`. Verejná metóda `loadDataFromSource()` sa využíva na nahrávanie zdrojových dát. Parametrom predávame informácie o tabuľke a zoznam primárnych kľúčov na základe ktorých sa načítajú iba potrebné záznamy. Štruktúra návratovej `DataTable` je definovaná zdrojovým systémom. Verejná metóda `getDataSourceName()` slúži na identifikovanie zdrojového systému pre potreby grafického rozhrania. Verejná metóda `getRowsCount()` vracia počet záznamov v cieľovej tabuľke. Využíva sa v grafickom rozhraní. Verejná metóda `getPKeysAndChangeDate()` získava zo zdrojových dát iba hodnoty primárneho kľúča a časovej známky. Tieto sú ďalej využívané v replikácii na kontrolu prítomnosti záznamov a identifikovanie zmenených záznamov. Verejná metóda `getColumns()` slúži na predanie štruktúry tabuľky špecifikovanej v parametri. Táto metóda nekomunikuje priamo so zdrojovým systémo ale iba transformuje získane dáta z metódy `getColumnList()`. Každý záznam vo výslednej `DataTable` reprezentuje jeden atribút. Štruktúra je presne definovaná, aby bola kompatibilná ako s grafickým rozhraním, tak aj s procesom pravidelnej kontroly vlastností zdrojových atribútov počas replikácie. Privátna metóda `getColumnList()` komunikuje zo zdrojovým systémom a analyzuje prípadne nedostatky v štruktúre zdrojovej tabuľky. Kontroluje prítomnosť primárneho kľúča a prípadnu absenciu sa snaží operátorovi navrhnuť pomocov analýzy zdrojových dát. Podobne sa správa aj metóda `setNullFromData()` ktorá ak nenájde aspoň jeden atribút v zdrojovej tabuľke so zákazom ukládania hodnoty null, tak usúdi, že zdrojová štruktúra tieto informácie o atribútoch neponúka a preto operátorovi analýzou dát ponúkne návrhy na atribúty ktoré by nemali mať povolené ukládanie hodnoty null.

Rozhranie `IMyDataSource` zastáva činnosť univerzálneho datového zdroja. Každá trieda ktorá správne implementuje toto rozhranie, sa za podmienky prístupu k zdrojového systému dá v celej aplikácii využívať ako dátovy zdroj. Týmto sme zabezpečili jednoduché pridávanie ďalších dátových zdrojov a oddelili databázovu vrstvu od vrstvy aplikačnej logiky. Metóda `getTablesFromSource()` musí zabezpečiť načítanie všetkých dostupných tabuliek a ich všetkých vlastnosti. Využíva parameter `props` ktorý upresňuje nastavenie daného dátoveho zdroja. V našom prípade je to cesta k súborom .dbf na lokálnom stroji. Metóda `getDataSourceName()` slúži na identifikovanie dátoveho zdroja. Metóda `getStructure()` načíta štruktúru danej tabuľky a vlastností všetkých jej atribútov. Vráti objekt `DataTable`, kde každý záznam reprezentuje vlastnosti jedneho atribútu zdrojovej tabuľky. Metóda `getData()` má dve definície. Jedna vracia objekt `DataTable` naplnený všetkými dátami z definovanej tabuľky.



Obr. 4 Návrhove triedy modulu DataSource

Preťažením metódy `getData()` a pridaním ešte jedneho parametru nesúceho zoznam primárnych kľúčov získame objekt `DataTable` naplnený iba dátami ktoré

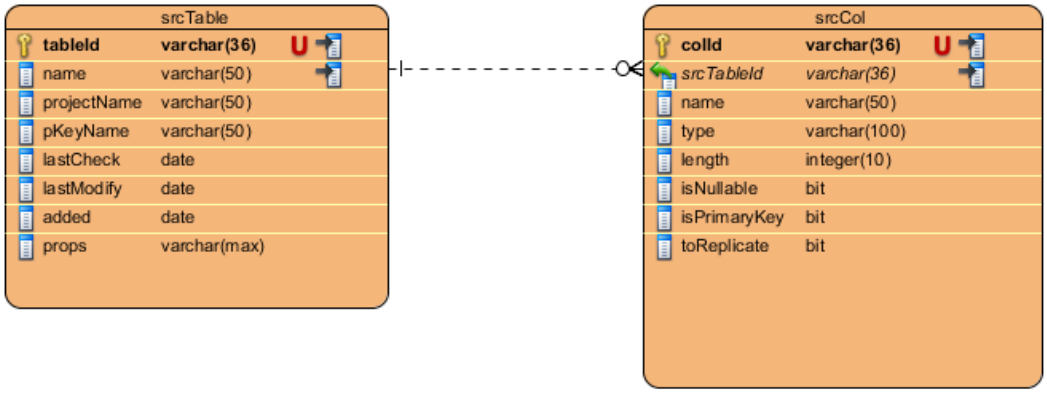
vyhovovali zoznamu v parametri. Metóda `getPkeysAndChangeDate()` načíta iba hodnoty primárnych kľúčov a časové známky poslednej zmeny v zázname. Zoradí ich do objektu `SortedDictionary` podľa hodnoty primárneho kľúča, čo zabezpečuje efektívnu kontrolu prítomnosti kľúčov v procese replikácie.

Trieda `VFPDataSource` implementuje rozhranie IMyDataSource a teda aj všetky metódy daného rozhrania. Atribút `conStr` obsahuje reťazec znakov špecifikujúcich využitie rozhrania VFPOLEDB. Konštruktor `VFPDataSource()` zabezpečí inicializáciu atribútu conStr a to hodnotou ktorú získa od triedy `Factory` a jej metódy `getConnStringSource()`. Týmto je zabezpečená prípadna prenositeľnosť na iné rozhranie poskytujúce prístup k dátam nachádzajúcim sa vo Visual FoxPro.

Rozhranie `IDataSourceProp` podporuje oddelenie databázovej vrstvy a vrstvy aplikačnej logiky. Slúži na prenášanie potrebných informácii o dátovom zdroji a prístupu k zdrojovým tabuľkám. Obsahuje metodu `getXML()` ktorá slúži na serializáciu triedy do XML. Návratovou hodnotou je teda reťazec znakov ktorý obsahuje vygenerovaný XML kód. Komunikácia medzi službou a užívateľským rozhraním musí mať presne špecifikovanú štruktúru objektov ktoré skrze komunikačný kanal posielajú. Trieda `VFPProp` implementujúca dane rozhranie má iba jeden atribút a to je `sourcePath` čo špecifikuje cestu k tabuľke v rámci lokálneho systému. To by už nešlo zabezpečiť v prípade ďalšieho dátového zdroja ktorého trieda implementujúca toto rozhranie by mala inú štruktúru atribútov. Serializáciou triedy do reťazca znakov na strane užívateľského rozhrania a deserializáciou tej istej triedy na strane služby a následným pretypovaním zabezpečíme korektný prenos týchto tried.

7.3 Modul DataDestination

Modul bližšie špecifikuje správanie analytickej triedy DataDestination. Po hlbšej analýze sme využili ďalšie tri návrhové triedy, ktoré nám opäť zabezpečujú oddeliteľnosť a znovuvyužiteľnosť kódu. Primárna činnosť modulu je správa cieľového systému. Pre potreby tejto bakalárskej práce sme zvolili ako cieľový systém Microsoft SQL Server vo verzii 2014 Developer.
Pre uchovávanie štruktúr zdrojových tabuliek a pre ukládanie metadát o procese replikácie sme vytvorili systémovú databázu ktorú spravuje trieda `SystemDB`.



Obr. 5 ERD diagram systémovej databázy

Entita `srcTable` reprezentuje metadáta o zdrojových tabuľkách a v aplikácii je reprezentovaná obslužnou triedou `TableInfo`. Entita `srcCol` reprezentuje jednotlivé atribúty zdrojových tabuliek, tie sú v aplikácii reprezentované obslužnou triedou `ColumnInfo`.

Pri vytvorení inštancie triedy `SystemDB` sa načíta aktuálny stav systémovej databázy z cieľového systému do objektu `DataSet` ktorý je dostupný cez atribút `systemDB`. Tento stav je pre správnu funkčnosť tejto triedy kľúčový, keďže všetky vlastné metódy dotazuju tento obraz databázy a nie databázový systém. Pri zmene tohto obrazu sa zmeny automatizovane premietnu do cieľového systému. Výhodu to ponúka vo výkonnosti a obmedzení potreby prístupovať priamo k databázi. Verejná metóda `addNewTable()` pridáva do systémovej databázy zdrojovú tabuľku pripravenú k replikácii. Jej zavolanie prebehne vždy keď operátor dokončí špecifikáciu atribútov v grafickom rozhraní. Táto metóda sa volá aj v prípade zistenia rozdielov v štruktúre už replikovanej tabuľky. Kontrola prítomnosti atribútov a metadát tabuľky prebieha cez privátne metódy. Verejná metóda `getTableColumns()` sa využíva primárne v uživateľskom rozhraní, pre kontrolu atribútov replikovanej tabuľky. Po úprave atribútov sa opäť zavolá metóda `addNewTable()` ktorá skontroluje prítomnosť a vlastnosti týchto zmenených atribútov a upraví ich v systémovej databáze. Návratovou hodnotou je rovnako ako pri čítani zo zdrojového systému objekt `DataTable`, kde každý záznam reprezentuje jeden atribút a jeho vlastnosti. Verejná metóda `getLastChangeAndPKeyName()` zisťuje v zozname replikovaných tabuliek ich aktuálny primárny kľúč a dátum ich poslednej úspešnej replikácie. Tieto informácie sú využívane pri generovaní dotazov na zdrojový systém. Privátna metóda `getPKeyNameFromColumns()` zisťuje názov primárneho kľúča v zozname atribútov predaného v objekte `DataTable`. Používa sa pri pridávaní novej tabuľky do replikácie, je volaná metódou `addNewTable()`, ktorej overí správne špecifikovaný primárny kľúč a vráti jej názov atribútu ktorý je primárnym kľúčom. Privátna metóda `rowExists()` kontroluje prítomnosť atribútov,tá následne volá

metódu `createColumnIfNotExists()` ktorá atribút pridá. Za vytvorenie záznamu o novej tabuľke pridanej do replikácie sa stará metóda `createTable()`.

O správu procesov v cieľovom systéme sa stará trieda `DestinationManager`. Tá si udržuje odkaz na inštanciu triedy `SystemDB` tak aj priamo na inštanciu cieľového systému. Trieda priamo spravuje aplikačnú databázu za využitia informácii získaných od `SystemDB`. Verejná metóda `createAppTable()` vytvára v cieľovom systéme aplikačnú tabuľku na základe definícii načítaných zo systémovej databázy. Zavolá sa vždy po pridaní novej tabuľky do replikácie a pred vytvorením overí všetky prerekvizity v systémovej databáze. Po ukončení kontroly predá inštancii cieľového systému presne definovaný zoznam atribútov v objekte `DataTable`. Verejná metóda `checkTableStruct()` sa využíva iba v procese replikácie už pridanej tabuľky. Zistí či je cieľová tabuľka pripravená a či sa jej štruktúra nelíši od práve načítanej zdrojovej tabuľky. Následne podnikne kroky potrebné k úprave cieľovej tabuľky. Automatizovaný proces úpravy je trochu odlišný od manuálnej úpravy štruktúry tabuľky definovanej operátorom v grafickom rozhraní. Automatizovaný proces pri zistení nového atribútu v zdrojovej tabuľky využije predpripravenú definíciu daného atribútu modulom `DataSource` a pridá ho k cieľovej tabuľke. Zmenu primárneho kľúča ignoruje až do chvíle keď nie je dostupný, vtedy informuje operátora o potrebe interakcie. Verejná metóda `getPKeyList()` pripraví pre proces replikácie návratovu hodnotu `SortedList<String,DateTime>` ktorú získa od cieľového systému. V nej sa nachádza zoradený zoznam dvojíc kľúč hodnota. Zoradenie výrazne pomáha pri prechádzaní kľúčov a porovnávaní s druhým zoradeným zoznamom. Verejná metóda `deleteSpecificRows()` predá zoznam primárnych kľúčov cieľovému systému ktorý následne podľa hodnôt kľúčov dohľadá záznamy a odstráni ich. Verejná metóda `insertSpecificRows()` slúži na vkladanie a aktualizovanie údajov v cieľovej tabuľke. Metóda predá cieľovému systému objekt `DataTable` obsahujúci potrebné dáta v štruktúre cieľovej tabuľky.

Rozhranie `IMyDataDestination` slúži, podobne ako rozhranie pre zdrojový systém, na oddelenie vrstvy aplikačnej logiky a databázovej vrstvy. Metódy v ňom zahrnuté slúžia pre správu ako aplikačnej tak aj systémovej databázy. Pre potreby tejto bakalárskej práce bola vytvorená trieda `SQLServerDD` ktorá implementuje dané rozhranie. Využitie Microsoft SQL Server pre potreby cieľového systému má výhody nielen v kompatibilite s knižnicou ADO .NET ale taktiež vo využití sady objektov na správu tohto servera pomenovanej Shared Management Objects. Tá dokáže generovať dotazy a spravovať server a objekty priamo z kódu aplikácie bez potreby písania SQL dotazov a procedúr.

Následujúce metódy slúžia na správu systémovej databázy. Verejná metóda `getSystemDataSet()` vráti objekt `DataSet` naplnený entitami zo systémovej databázy. Využíva ju iba trieda `SystemDB` a zavolá sa iba počas vytvárania inštancie tejto triedy. Dáta načíta pomocov objektu `SqlDataAdapter` ktorý automaticky vygeneruje do návratového objektu `DataSet` dvoje objekty `DataTable` so

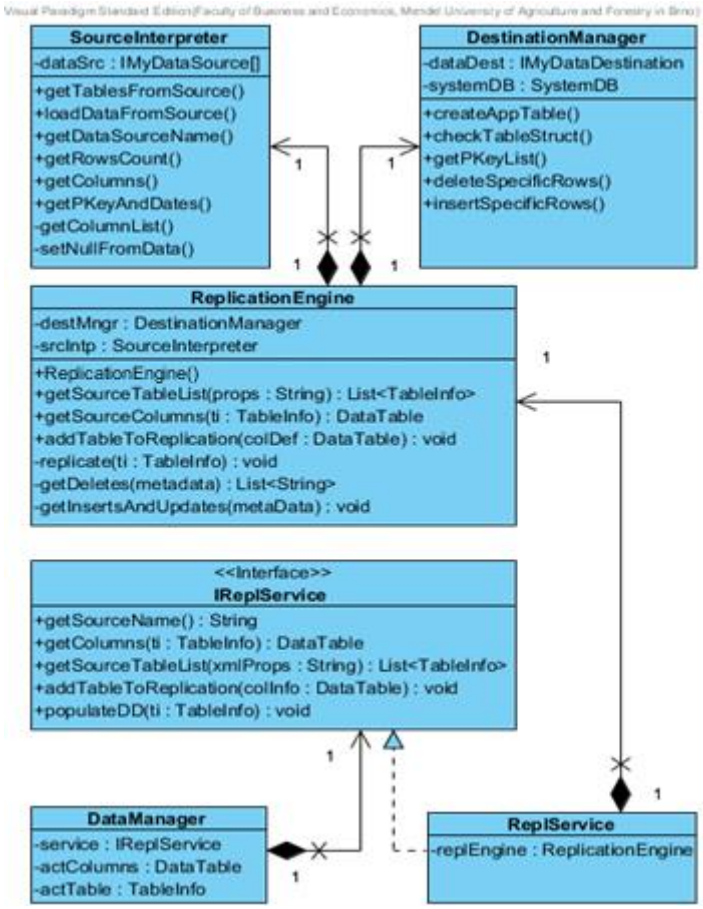
všetkými potrebnými atribútmi, vlastnostiami a dátami entít `srcTable` a `srcCol` znázornenými na obrázku 6. Privátna metóda `createSystemTables()` zabezpečí existenciu potrebných entít v systémovej databáze. Pokiaľ tam ešte neexistujú alebo ich pod nakonfigurovaným názvom nenajde tak ich vytvorí. Volá ju iba metóda `getSystemDataSet()` a to pred tým než začne načítavať dáta do objektu `DataSet`. To kontroluje prítomnosť systémových entít počas načítavania. Privátna metóda `createSystemDB()` má zodpovednosť za dostupnosť systémovej databázy pod nakonfigurovaným menom. Kontroluje dostupnosť servera a v prípade nedostupnosti servera vytvorí výnimku ktorá zabráni spusteniu aplikácie a informuje operátora o potrebe skontrolovať cieľový systém alebo reťazec znakov používaný na pripojenie k tomuto systému. Volá ju iba metóda `createSystemTables()` a to pred tým než začne vytvárať systémove entity. Previazaním týchto troch metod zabezpečíme dostupnosť všetkých prerekvizít na strane cieľového systému potrebných pre korektne správanie triedy `SystemDB`. Verejná metóda `updateSystemTable()` premietne zmeny vo v parametri-predanom objekte `DataSet` do systémových tabuliek. Využíva objekt `SqlDataAdapter` pre prístup k cieľovému systému. Pre porovnanie zmenených záznamov v objekte `DataSet` s cieľovou tabuľkou musí `SqlDataAdapter` načítať kompletný obsah tabuľky podľa špecifikovaného SQL dotazu, porovnať tieto dáta na strane aplikácie a rozhodnúť ktoré záznamy budú v cieľovom systéme zmenené. Potreba načítať do pamäte aplikácie celý obsah tejto entity, je pri malom počte záznamov ako majú systémove tabuľky, znesitelná. Tento spôsob porovnávania je nežiadúci pri tabuľkách s veľkým množstvom záznamov ako sú aplikačné tabuľky a mohlo by to znamenať zahĺtenie ako zdrojového tak aj cieľového systému. Generovanie SQL kódu na aktualizovanie, zmenenie alebo zmazanie záznamov v cieľovej databáze má na starosti objekt `SqlCommandBuilder` ktorý k tomu potrebuje pôvodný SQL dotaz na čítanie dát.

Správu aplikačnej databázy zabezpečujú nasledujúce metódy. Verejná metóda `createTable()` má za úlohu vytvárať aplikačné tabuľky na cieľovom systéme. Ako parametre dostane objekt `DataTable`, obsahujúci zoznam atribútrov tabuľky a ich vlastností, a objekt `TableInfo` obsahujúci detaily o tabuľke. Za využitia sady objektov Shared Management Objects analyzuje zoznam atribútov a pretransformuje ich do tabuľky ktorú následne vytvorí na cieľovom systéme. Na transformáciu vlastností atribútov špecifikovaných v objekte `DataTable` do jazyka SQL využije privátnu metódu `getDataType()` ktorej výstup presne špecifikuje ako SQL dátový typ, tak aj jeho maximálnu dĺžku. Následne zadefinuje povolenie hodnoty `null` a nakoniec za využitia privátnej metódy `createPKey()` vytvorí na atribúte s príznakom `isPrimaryKey` primárny kľúč tabuľky. Tabuľka je pripravená k vytvoreniu ale ešte nieje priradená k databáze. Pomocov privátnej metódy `getAppDB()` ktorá sa pokúsi na cieľovom systéme nájsť databázu podľa názvu z parametru metódy. Ak databázu s daným názvom nenájde tak zabezpečí jej vytvorenie. Návratovou hodnotou je objekt `Database` ktorý sa využije práve vytváranou tabuľkou na priradenie k databázi. Následne sa tabuľka zapíše do

cieľového systému a je pripravená na použitie replikačným procesom. Práve popísaný algoritmus metódy `createTable()` sa využíva aj pri aktualizovaní štruktúry cieľovej tabuľky bez potreby cieľovú tabuľku mazať. Verejná metóda `deleteSpecificRows()` je určená na zmazanie záznamov z cieľovej tabuľky ktoré vyhovujú niektorým z primárnych kľúčov nachádzajúcich sa v zozname predanom v parametri. Pre optimalizáciu výkonnosti mazania záznamov nieje použitý objekt `SqlCommandBuilder`. Namiesto toho je využitý preddefinovaný SQL kód, do ktorého sa generujú hodnoty primárného kľúča načítane zo zoznamu v parametri, názov tabuľky a názov atribútu ktorý je primárnym kľúčom v danej tabuľke. Následne sa kód posiela na server kde sa vykoná. Server pošle späť reálny počet zmazaných záznamov ktorého rovnosť s počtom hodnôt primárneho kľúča indikuje korektné vykonanie operácie zmazania. Verejná metóda `insertSpecificRows()` má ako jediná neocenitelné privilégium zapisovať do cieľovej tabuľky dáta zo zdrojového systému. V parametri dostane objekt `DataTable` s identickou štruktúrou akú ma cieľová tabuľka a to nám dáva právo využiť výhody objektu `SqlBulkCopy` ako je nahrávanie po dávkach a automaticke mapovanie atribútov. Kontrola vložených záznamov funguje na podobnom princípe ako pri mazaní, a to rovnosť počtu záznamov reálne vložených a počtu záznamov v objekte `DataTable`.

7.4 Modul ReplicationManager

Tento modul ostal iba pri jednej návrhovej triede rovnako ako v analytických triedach. Návrhova trieda nesie názov ReplicationEngine a jej metódy a atribúty na rozdiel od analytickej triedy však prešli zmenou a končná podoba je dostupná na obrázku 8, kde je spolu s touto navrhovou triedou zobrazené aj previazanie medzi ostatnými návrhovými triedami aplikácie. Keďže ReplicationEngine má ako jediná trieda odkaz na inštancie návrhových tried SourceInterpreter a DestinationManager, plní funkciu režiséra celého procesu replikácie a prenosu dát a teda tvorí jádro vrstvy aplikačnej logiky. Konštruktor `ReplicationEngine()` vytvára počas inicializácie inštancie tried SourceInterpreter a DestinationManager a uloží odkazy na tieto inštancie do atribútov `destMngr` respektíve `srcIntp`. Skrze tieto odkazy dokáže ReplicationManager získavať potrebné informácie z persistentných systémov. Metódy môžeme rozdeliť na dve časti, a to časť ktorá ponúka operátorovi rozhranie pre správu systémov persistencie a časť ktorá sa stará o replikáciu.



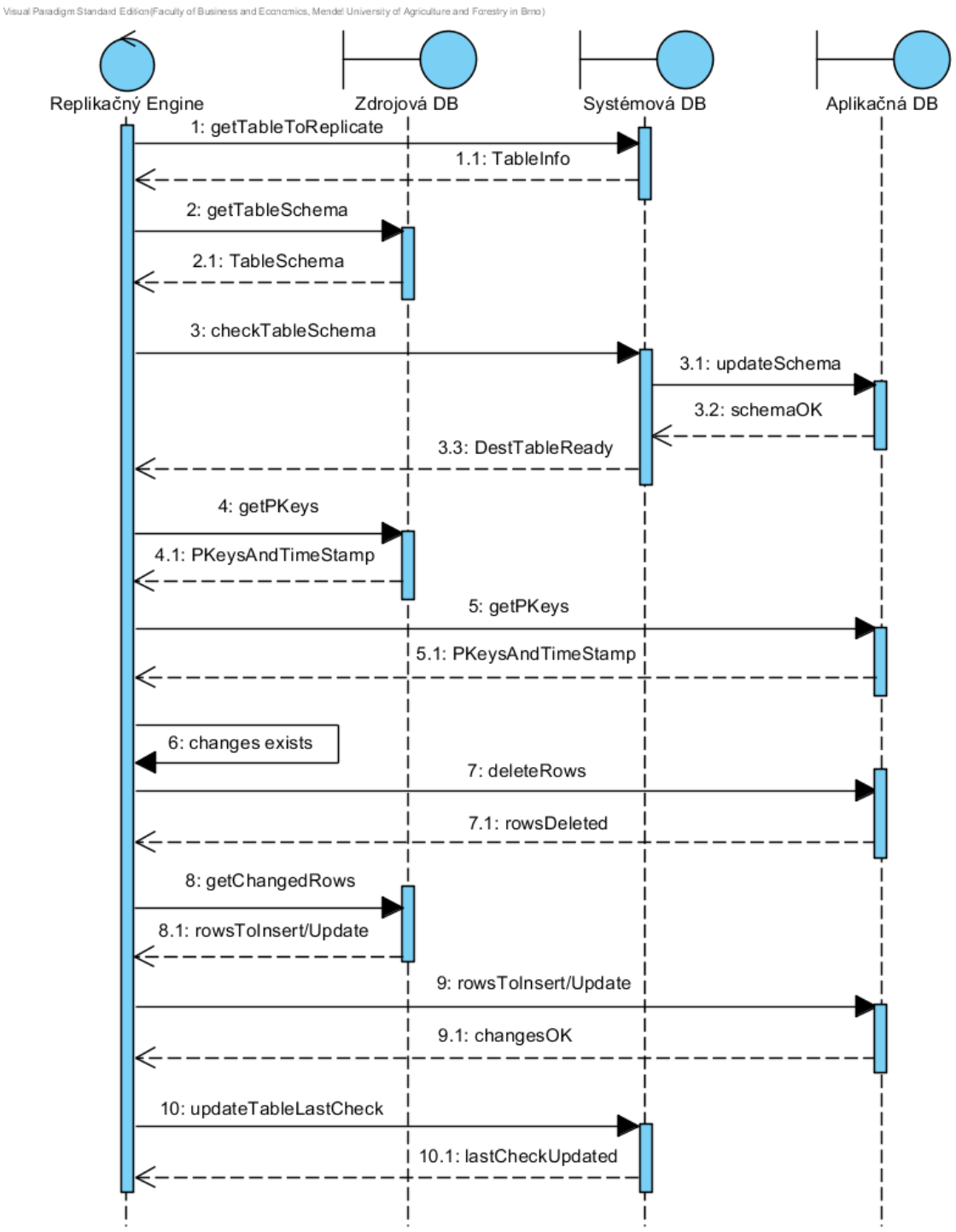
Obr. 6 Prepojenie návrhových tried

Metódy prvej časti sú verejné a nepriamo ich volá grafické rozhranie. Metóda `getSourceName()` získa od modulu DataSource reťazec znakov reprezentujúcich názov databázoveho systému v ktorom má operátor dostupné tabuľky. Po výbere cesty k databázovým súborom sa zavola metóda `getSourceTableList()`. Tá pripraví pre grafické rozhranie zoznam tabuliek dostupných v zdroji. Reťazec znakov `props` obsahuje XML kód v ktorom sú všetky potrebné informácie pre nasmerovanie zdrojového systému do priečinku obsahujúceho dbf súbory, ktoré vybral v grafickom rozhraní operátor. Tieto predá taktiež vo forme reťazca znakov metóde `getTablesFromSource()`, ktorej funkčnosť sme popísali v module DataSource.

Uživateľské rozhranie obdrží zoznam tabuliek ktoré premietne operátorovi ktorý po výbere jednej z nich pre pridanie do replikácie prejde do módu definície vlastností atribútov. Zoznam atribútov ako aj ich vlastností z vybranej tabuľky ponúkne grafickému rozhraniu metóda `getSourceColumns()` ktorá si uloží metadáta o poslednej načítanej tabuľke a k tomu zavolá metódu `getColumns()` patriacu návrhovej triede SourceInterpreter od ktorej získa objekt `DataTable`. Tento predá uživateľskému rozhraniu a to ho premietne operátorovi. Operátor

upraví vlastnosti atribútov a pridá tabuľku do replikácie čo nepriamo zavolá metódu `addTableToReplication()` ktorá v parametri dostane objekt `DataTable` od uživateľského rozhrania s upravenou definíciou atribútov. Obdržaný objekt sa predá metóde `createAppTable()` špecifikovanej v module DataDestination.

Druhá časť starajúca sa o proces replikácie má iba privátne metódy. Metóda `getInsertsAndUpdates()` porovnáva objekty obsahujúce hodnoty primárnych kľúčov a hodnoty časových známok. Objekt `SortedList<String,DateTime>` obsahuje všetky dostupné dvojice primárny kľúč a časová známka z cieľovej aplikačnej tabuľky. Časová známka je pri každom zázname rovnaká a reprezentuje hodnotu poslednej úspešne ukončenej replikácie danej tabuľky. `SortedList<Key,Value>` slúži na udržiavanie zoradených kľúčov a ich hodnôt. Bol vybraný pre svoju vlastnosť rýchleho pridávania už zoradených kľúčov do zoznamu, ktoré pre nás dokáže veľmi efektívne zoradiť databázovy engine ešte v cieľovom systéme. `SortedDictionary<String,DateTime>` obsahuje zoradený zoznam dvojíc primárny kľúč a časová známká zo zdrojového systému. Tu reprezentuje časová známka informáciu o poslednej zmene záznamu ku ktorému patrí primárny kľúč. Keďže nie-je zaručené, že zdrojový systém bude schopný dáta zoradiť použili sme objekt `SortedDictionary<Key,Value>`.Ten je rýchlejší v pridávani nezoradených dát do zoznamu než objekt `SortedList`. Rýchlosť prístupu k hodnotam je pri oboch objektoch skoro identická. Metóda porovnáva prítomnosť kľúčov zo zdrojového systému v cieľovom systéme. U zhodných kľúčov porovná časove známky. Ak je časová známka zdrojového záznamu novšia ako čas poslednej úspešnej replikácie tak sa tento kľúč pridá do zoznamu kľúčov na aktualizovanie. Ak zistí kľúč zo zdrojového systému ktorý sa nenachádza v zozname kľúčov cieľového systému tak ho pridá do zoznamu kľúčov na vkládanie. Výstupom tejto metódy sú dvoje objekty, každý obsahujúci zoznam primárnych kľúčov, jeden pre kľúče ktorých záznamy sa pridajú a druhý pre kľúče ktorých záznamy sa aktualizujú. Metóda `getDeletes()` funguje na podobnom princípe ako predošla metóda, s rozdielom, že kontroluje prítomnosť cieľových kľúčov v zdrojových dátach. Jej výstupom je zoznam kľúčov ktorých záznamy musia byť zmazané v cieľovej aplikačnej tabuľke. Asynchrónna metóda `runSchedule()` je zodpovedná za spustenie replikáčného rozvhrhu. Tá skontroluje či práve nebeží iný replikačný rozvrh a ak nebeží tak z `DestinationManager`-u získa zoznam potrebných tabuliek a z nich vytvorí zoznam `Task`-ov ktoré asynchronne spustí. Aktuálne vlákno uloží do spánku na štandardne 10 sekúnd. Táto hodnota ovplyvňuje ako často replikácia prebieha a môže byť menená z grafického rozhrania. Asynchrónna metóda `replicate()` riadi replikáciu tabuľky, ktorej metadáta dostala v parametri. Táto metóda ma návratový typ `Task`, čo využíva metóda `runSchedule()` ktorá takto vytvorí pre každú z replikovaných tabuliek inštanciu tejto metódy a dá jej metadáta o tabuľke. Proces replikácie znázorňuje sekvenčný diagram na obrázku 8.



Obr. 7 Sekvenčný diagram procesu replikácie tabuľky

7.5 Modul Service

Modul service slúži ako hostiteľ pre aplikáciu Data Replicator. Spravuje pripojenia, kontexty komunikácie a vytvára inštanciu modulu ReplicationManager, ktorému následne posiela požiadavky od pripojených klientov. Využíva sa technológia Windows Communication Foundation ktorá zabezpečuje paralelný prístup viacero klientov a prideľuje im jednotlivé inštancie podľa nastavenia a potreby. Pre potreby tejto bakalárskej práce bolo vybrané nastavenie jednej inštancie ktorá obsluhuje všetky pripojenia. To umožní komunikáciu viacerých klientov so službou ale nebude robiť problémy pri strete viacerých požiadavkov lebo budú vyriadené službou sériovo. So službou klient komunikuje cez rozhranie NetTcpBinging ktoré využíva serializáciu objektov a dát do binárneho formátu a následne ich posiela cez protokol TCP.

7.6 Modul Data Manager

Modul Data Manager slúži ako grafické rozhranie pre komunikáciu s so službou Data Replicator. Grafické rozhranie používa technológiu Windows Presentation Foundation. Zvolené bolo kvôli rozšíriteľnosti, jednoduchosti používania, využívaniu jazyka XAML a schopnosti tzv. „bindingu a databindingu“, teda schopnosti spojiť grafický objekt s iným grafickým objektom alebo objektom ako je DataTable a zmena v danej DataTable sa automaticky premietne do grafického objektu a zmena v grafickom objekte sa premietne do DataTable. To sa využíva primárne pri grafických objektoch mriežka (grid) a zoznam (list). Tento modul si udržiava metadata o prístupe k službe v súbore app.config ktoré je potrebné pri zmene portu na ktorom služba príjima komunikáciu editovať. Taktiež má dostupnú obmedzenú knižnicu tried ktoré používa služba DataReplicator aby dokázal prijaté data deserializovať do identických objektov ake poslala služba.

8 Test riešenia

Následujúce testy mali overiť odolnosť voči externým zásahom do funkčných predmetov využívaných navrhnutým riešením. Testy boli vykonané na systéme Microsoft Windows Server 2012 R2 a Microsoft SQL Server 2014 Developer edícii.

8.1 Zdrojové dáta nedostupné

V prípade nenájdenej alebo neexistujúcej tabuľky boli zachované replikované dáta na cieľovom systéme z poslednej úspešnej replikácie.

Riešením je sprístupniť zdrojovú tabuľku, alebo pridať tabuľku do replikácie s novým umiestnením.

8.2 Zmena zdrojovej štruktúry

Data replicator rozpoznal zmenu v zdrojovej štruktúre. Pokiaľ šlo o zmenu názvu replikovaného stlpca tak ho zaevidoval do systemovej databáze ako nový, ešte nereplikovaný. Replikácia však po prístupe k neexistujúcemu stlpcu zlyhala. Zmena maximálnej dĺžky alebo dátoveho typu stlpca malo za efekt taktiež zlyhanie replikácie. Cieľové dáta ostali neporušené.

Riešením je vrátenie zdrojovej tabuľky do pôvodneho stavu alebo zmena v definícii replikácie danej tabuľky.

8.3 Zmazanie cieľovej tabuľky

Systém vytvoril novú tabuľku podľa definícii zo systémovej databázy. Process replikácie nebol porušený, tabuľka bola vytvorená automaticky.

8.4 Zmazanie systémovej databázy a zmena záznamov v systémovej DB (pri zapnutej službe)

Keďže si služba udržuje celý obsah systémovej databázy v objekte `DataSet` ako obraz a zároveň je jediná ktorá pracuje s týmito dátami, tak obsahuje pre daný moment korektné dáta. Pri každej zmene v objekte sa prenesú zmeny aj do systémovej databáze. Počas tohto procesu je skontrolovaná existencia a prebehne prípadne vytvorenie. Následne sa aplikuje celý obsah na prázdnu alebo nekonzistentnú databázu čo ma za následok prepísanie nekorektných záznamov a doplnenie neexistujúcich.

8.5 Zmazanie záznamov v cieľovej tabuľke

Keďže proces kontroluje prítomnosť záznamov cez primárny kľuč, tak ktorýkoľvek neexistujúci záznam je nahradený pri ďalšiom behu replikácie danej tabuľky.

8.6 Pridanie záznamov v cieľovej tabuľke

Pokiaľ bol záznam pridaný a neexistuje v zdrojovej tabuľke tak ho process replikácie rozozná a následne zmaže.

Táto funkcionalita sa v budúcnosti vypne, aby bolo umožnené pridávanie záznamov novému systému.

8.7 Pridanie atribútu k cieľovej tabuľke

Pokiaľ atribút môže obsahovať hodnotu NULL alebo je automaticky generovaný na strane databázového serveru tak to vplyv na process replikácie nema. V ostatných prípadoch replikácia danej tabuľky zlyhá. Dáta ostanu neporušené (ako boli pri poslednej úspešnej replikácii)

Riešením je pridať do zdrojovej tabuľky daný atribút alebo ho nastaviť v cieľovej tabuľke ako NULL resp. automaticky generovaný.

8.8 Zmena hodnoty atribútu v zázname externým zásahom do cieľovej tabuľky.

Pokiaľ nebol zmenený primárny kľúč daného záznamu tak tento záznam ostane v nekonzistentom stave.

Riešením je daný záznam zmazať.

8.9 Zmazanie Systémovej DB (pri vypnutej službe)

Tento akt je kritickým zásahom do riešenia. Pokiaľ je služba vypnutá tak sa nedokáže automaticky vyliečiť ako pri zmazaní počas zapnutej služby.

Riešenie: Obnovenie zálohy databáze. Pokiaľ záloha neexistuje je nutné zmazať cieľové tabuľky (resp. celú databázu) a opäť pridať zdrojové tabuľky do procesu replikácie.

9 Záver

9.1 Zhrnutie

Táto Bakalárska práca obsahuje metodiku, analýzu, návrh a popis implementácie služby Data Replicator a aplikácie Data Manager ako aj popis využívaných technológii a postupov.

Prvá kapitola obsahuje zadanie a cieľ práce. V ďalšiej kapitole sú popísané vybrané technológie .Net framework. Kapitola 3 obsahuje zvolenú metodiku práce. Analýzu aktuálneho stavu v doménach bezpečnosti, štruktúry a údržby vybraného Modulu popisuje kapitola číslo štyri. Následujúca kapitola je venovaná požiadavkom zákaznika na výsledné riešenie. V 6. kapitole sú dané požiadavky analyzované a z nich sú vytvorené analytické triedy, taktiež je tu analýza dostupnosti rozhraní pre prístup k zdrojovým dátam. Samotný návrh vychádzajúci z analytických tried a popis implementácie obsahuje kapitola 7. Ďalšia kapitola je venovaná testom naimplementovaného riešenia. Literatúra a prílohy sú dostupné v kapitolach 10 resp. 11.

9.2 Diskusia

Ponúknute riešenie je naprogramované s použitím najmodernejších stabilných, podporovanných a dostupných technológií v danej oblasti a design riešenia má excelentné predpoklady pre ďalší vývoj. Služba Data Replicator zabezpečuje replikáciu zmeny v zdrojových dátach do cieľového systému spoľahlivo. Efektívným algoritmom replikácie, paralelným spracovaním a paralelným prístupom, ako k zdrojovému tak cieľovému systému, je zabezpečený maximálny možný výkon riešenia pri rešpektovani požiadavkov zákazníka a limitov použitých technológii. Keďže jednotlivé úlohy ktoré sú paralelne spracovávane replikujú vždy unikátnu tabuľku zo zoznamu aktualne spracovávaných tabuliek tak si vlákna navzájom nezamykajú tabuľky a súbory s ktorými pracujú. To je výhodne aj pre MS SQL server engine, pretože insert operácia do danej tabuľky je vždy single-thread (vždy iba jedno vlákno zapisuje, ostatné čakajú kým skončí) a teda sa vlákna neblokujú ani na úrovni SQL Servera. Výnimkou je zápis metadát o replikácii do systémovej databázy, tam však úloha mení tranzakciou maximálne jeden záznam príslušný spracovávanej tabuľke a teda zamýka tabuľku na zanedbateľnú dobu. Riešenie je limitované iba výkonom dostupných zdrojov pre operačný systém a to primárne výkonom CPU a odozvou IO operácii na diskovom poli. Grafické rozhranie Data Manager ponúka operátorovi pohodlnú správu replikácie. Požiadavok zákaznika prezerať dáta v cieľovej tabuľke bol po konzultácii zamietnutý, keďže firma Microsoft ponúka k SQL Serveru profesionálny nástroj na správu servera tzv. Management studio. Konzistentnosť a odolnosť riešenia voči externým zásahom do procesu replikácie bola otestovaná v kapitole 8. Testy boli zamerané na navodenie nekonzistentného stavu v cieľovej alebo systémovej databáze. Riešenie odolalo siedmým z deviatich

testov, kde posledné dva boli kritickým zásahom do procesu replikácie ako je napr. zmazanie systémovej databáze pri vypnutej službe Data Replicator. Systémovu databázu je preto potrebné zálohovať.
S výsledným riešením je zákazník spokojný, ponúka mu funkcionalitu a správanie ktoré požadoval. Zákazník v čase písania tejto bakalárskej práce ma riešenie nasadené už tri týždne.

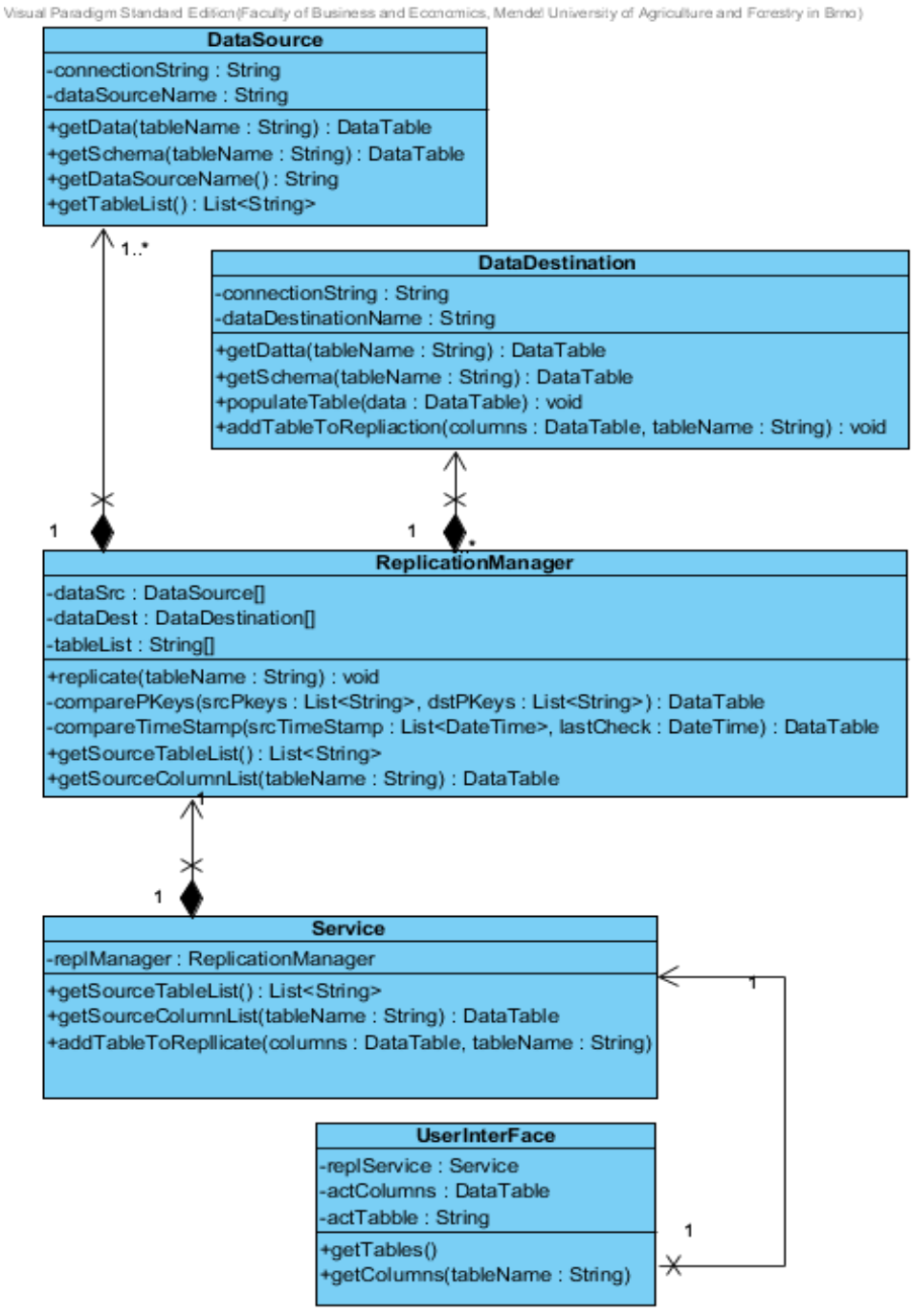
9.3 Záver

Replikácia vrstvy perzistencie daného modulu zo starého informačného systému naimplementovaného vo Visual FoxPro 9 na Microsoft SQL Server prebehla úspešne. Vytvorená služba Data Replicator a aplikácia Data Manager ponúkajú zákazníkom požadovanú funkcionalitu replikacie zmeny dát v realnom čase s využitím technológií .Net. Vďaka použitiu paralelného prístupu k vrstvám persistencie a efektívnemu algoritmu replikácie bol dosiahnutý potrebný výkon. Testy konzistencie boli splnené, zákaznik riešenie intenzívne testuje.
Cieľ bakalárskej práce je splnený.
V ďalších fázach vývoja budu v aplikácii Data Replicator viac oddelené objekty participujúce v paralelnom spracovaní dát od ostatných objektov čo zabezpečí zjednodušenie vývoja ďalších komponent. Bude vytvorená nová Windows služba zodpovedná za správu systémovej databázy ktorá bude sprácovavať požiadavky iba sériovo. Bude pridané rozhranie na správu replikácie a replikovaných tabuliek z príkazového riadku. Grafické rozhranie bude upravené pre webový server IIS. Bude pridaný e-mailový notifikačný systém.

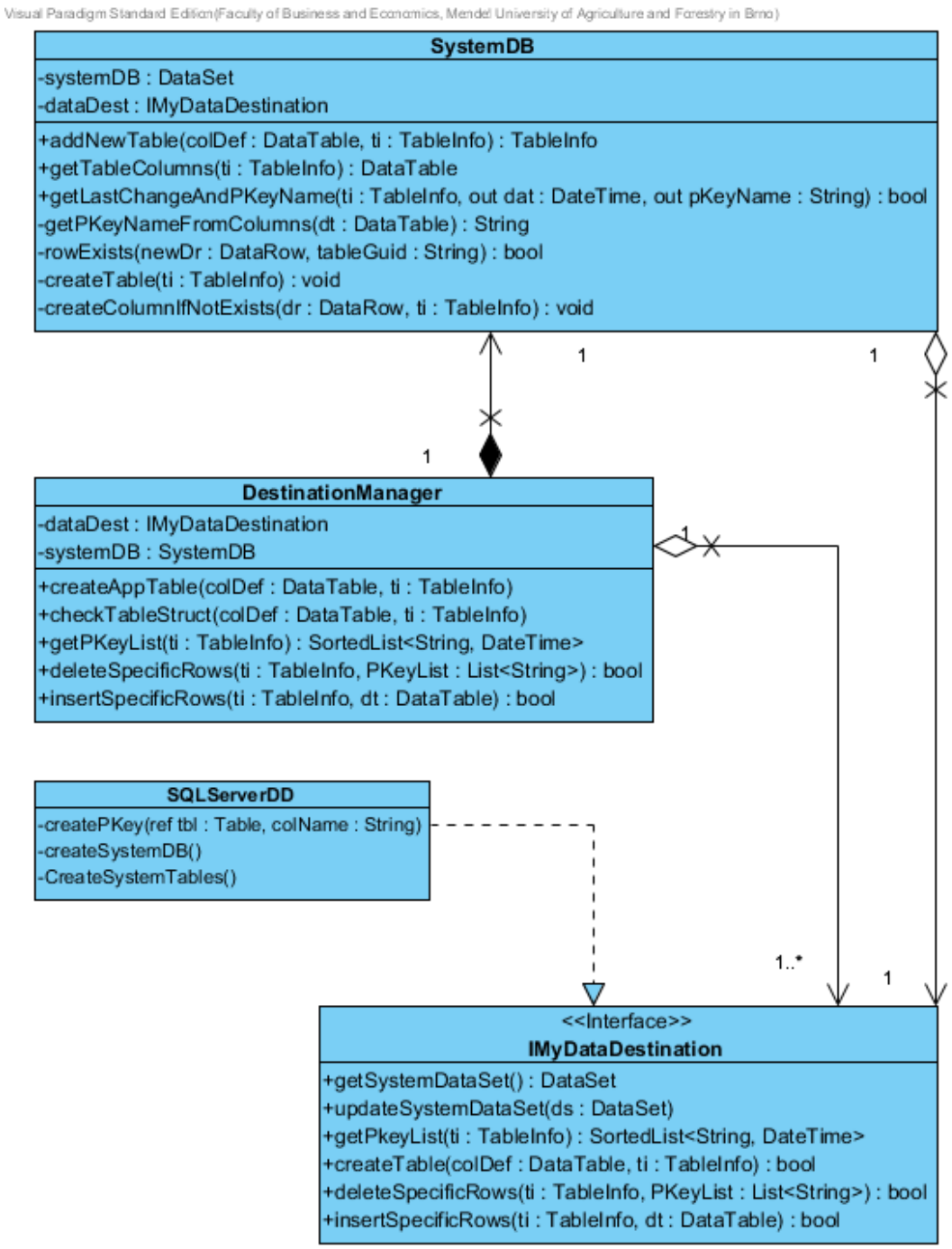
10 Literatúra

ARLOW, Jim a Ila NEUSTADT. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky*. 2., aktualiz. a dopl. vyd. Brno: Computer Press, 2007, 567 s. ISBN 978-80-251-1503-9.
ANDREW TROELSEN. *Pro C# 5.0 and the .NET 4.5 framework*. 6th ed. Berkeley, Calif.: Apress, 2012. ISBN 9781430242345.
WATSON, Ben. *C# 4.0 how-to*. Indianapolis, Ind.: Sams, c2010, xiii, 653 p. ISBN 0672330636.
LACKO, Ľuboslav. *Mistrovství v SQL Server 2012: [kompletní průvodce databázového experta]*. 1. vyd. Brno: Computer Press, 2013, 640 s. ISBN 978-80-251-3773-4.

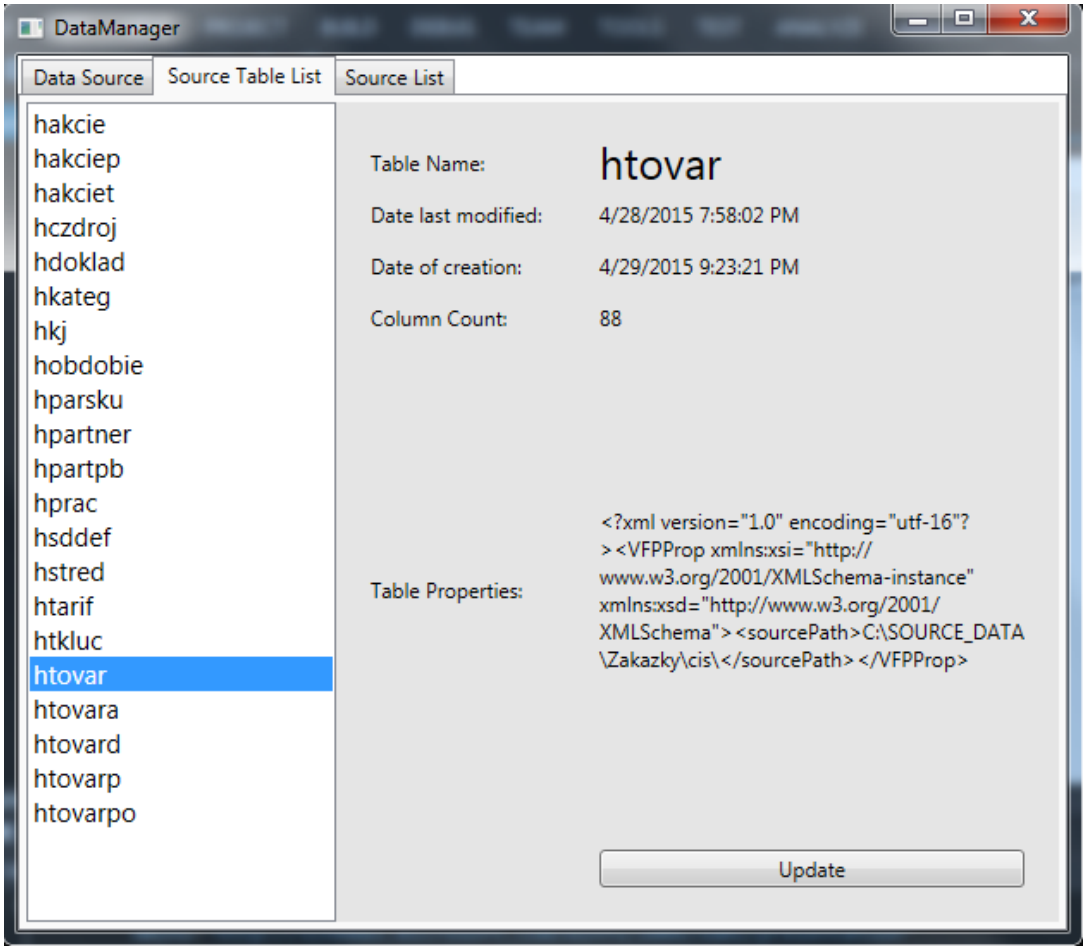
Prílohy



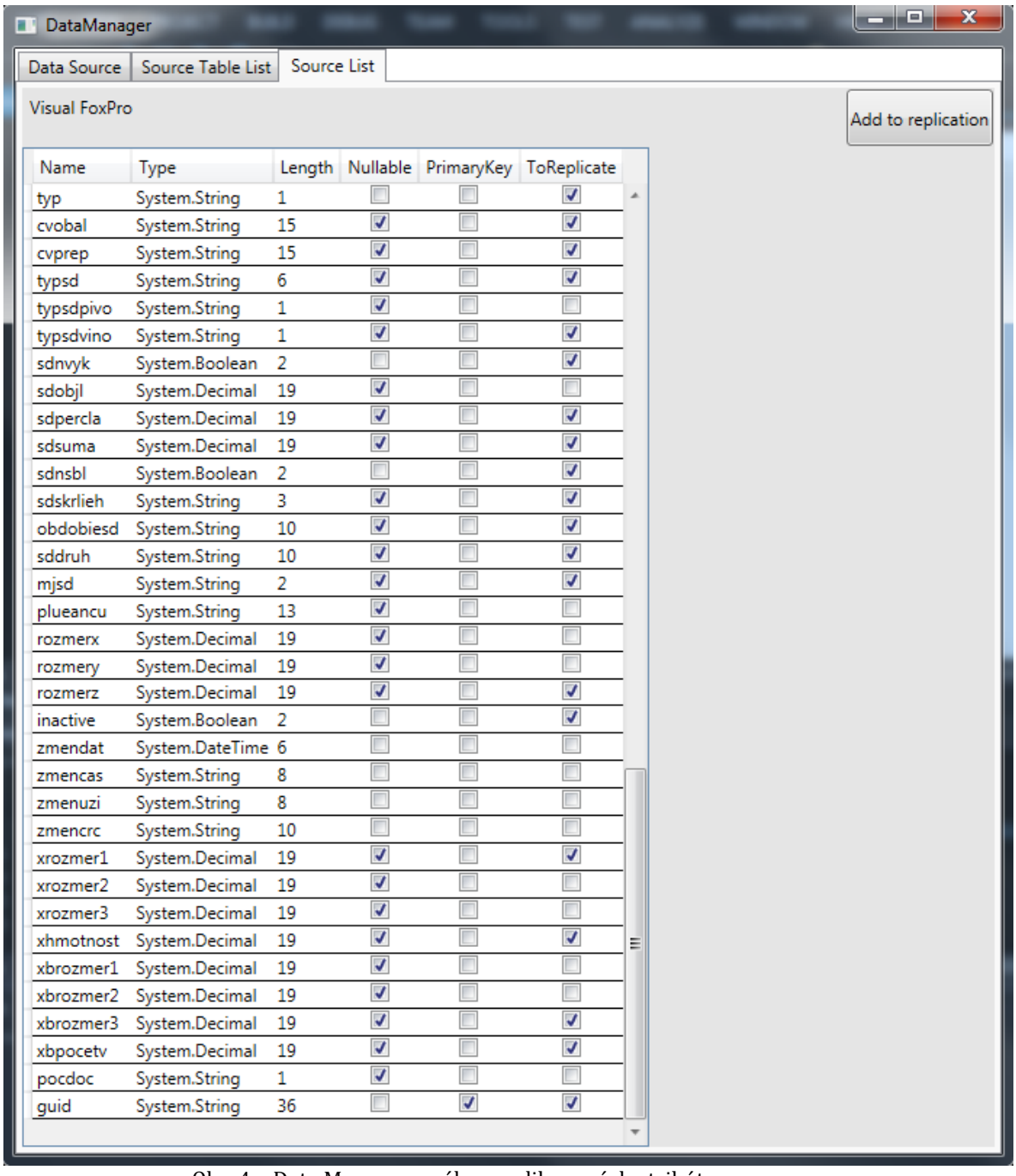
Obr. 1 Analytické triedy



Obr. 2 Návrhové triedy modulu DataDestination



Obr. 3 Data Manager – výber tabuľky zo zdrojového systému



Name	Type	Length	Nullable	PrimaryKey	ToReplicate
typ	System.String	1	☐	☐	☑
cvobal	System.String	15	☑	☐	☑
cvprep	System.String	15	☑	☐	☑
typsd	System.String	6	☑	☐	☑
typsdpivo	System.String	1	☑	☐	☐
typsdvino	System.String	1	☑	☐	☑
sdnvyk	System.Boolean	2	☐	☐	☑
sdobjl	System.Decimal	19	☑	☐	☐
sdpercla	System.Decimal	19	☑	☐	☑
sdsuma	System.Decimal	19	☑	☐	☑
sdnsbl	System.Boolean	2	☐	☐	☑
sdskrlieh	System.String	3	☑	☐	☑
obdobiesd	System.String	10	☑	☐	☑
sddruh	System.String	10	☑	☐	☑
mjsd	System.String	2	☑	☐	☑
plueancu	System.String	13	☑	☐	☐
rozmerx	System.Decimal	19	☑	☐	☐
rozmery	System.Decimal	19	☑	☐	☐
rozmerz	System.Decimal	19	☑	☐	☑
inactive	System.Boolean	2	☐	☐	☑
zmendat	System.DateTime	6	☐	☐	☐
zmencas	System.String	8	☐	☐	☐
zmenuzi	System.String	8	☐	☐	☐
zmencrc	System.String	10	☐	☐	☐
xrozmer1	System.Decimal	19	☑	☐	☑
xrozmer2	System.Decimal	19	☑	☐	☐
xrozmer3	System.Decimal	19	☑	☐	☐
xhmotnost	System.Decimal	19	☑	☐	☑
xbrozmer1	System.Decimal	19	☑	☐	☐
xbrozmer2	System.Decimal	19	☑	☐	☐
xbrozmer3	System.Decimal	19	☑	☐	☑
xbpocetv	System.Decimal	19	☑	☐	☑
pocdoc	System.String	1	☑	☐	☐
guid	System.String	36	☐	☑	☑

Obr. 4 Data Manager – výber replikovaných atribútov