VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

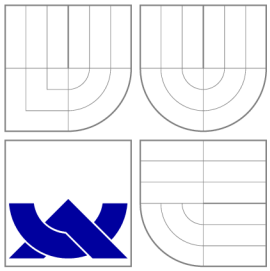
KLIENT PRO ZOBRAZOVÁNÍ OLAP KOSTEK

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

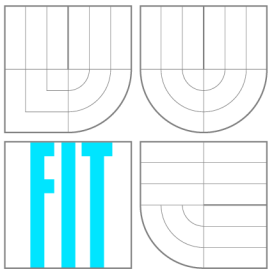
AUTOR PRÁCE
AUTHOR

Bc. JAN ZAHRADNÍK

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

KLIENT PRO ZOBRAZOVÁNÍ OLAP KOSTEK

CLIENT FOR DISPLAYING OLAP CUBES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAN ZAHRADNÍK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VLADIMÍR BARTÍK, Ph.D.

BRNO 2009

Abstrakt

Práce se zabývá tvorbou analytického nástroje pro podporu manažerského rozhodování. Cílem bylo vytvořit reportovací systém, který by umožnil jednoduchou orientaci v klíčových ukazatelích pro firemního zákazníka. Jako implementační prostředí byl použit .NET Framework 3.5 s jazykem C# a databázovým serverem Microsoft SQL 2008 včetně rozšíření Analysis Services, pro webové rozhraní pak ASP.NET.

Abstract

The purpose of this work is in analytical tool supporting manager decision making. The goal was to develop a reporting system which simplifies company customer's orientation in key performance indicators. As implementation environment has been used .NET Framework 3.5 with C# language and database server Microsoft SQL 2008 with Analysis Services extension, for web interface has been used ASP.NET.

Klíčová slova

Business Intelligence, datový sklad, OLAP, Microsoft SQL Server Analysis Services, dashboard, KPI, .NET Framework, C#, Linq2Sql, Web Parts, MVP, Enterprise Library, Unity Application Block

Keywords

Business Intelligence, data warehouse, OLAP, Microsoft SQL Server Analysis Services, dashboard, KPI, .NET Framework, C#, Linq2Sql, Web Parts, MVP, Enterprise Library, Unity Application Block

Citace

Jan Zahradník: Klient pro zobrazování OLAP kostek, diplomová práce, Brno, FIT VUT v Brně, 2009

Klient pro zobrazování OLAP kostek

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Vladimíra Bartíka. Další informace mi poskytl Ing. Ladislav Heřman za společnost Data System Solutions, s.r.o. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Zahradník
23. května 2009

Obsah

Kapitola 1

Úvod

Tato práce popisuje analytický systém, který má usnadnit přístup běžných uživatelů, manažerů, k množství dat generovaném jejich podřízenými a také prostředím, ve kterém se společnost pohybuje. Projekt je zadáním společnosti Data System Solutions, s.r.o., jejímiž největšími zákazníky jsou mezinárodní farmaceutické společnosti.

V současné době se jako nejlepší technologie pro práci s velkým objemem dat pro účely analýz a reportů jeví multidimenzionální kostky, jež jsou optimalizovány pro různé úhly pohledu. S jejich možnostmi se blíže seznámíme v obecné kapitole o technologiích, kde také nastíním možnosti vizualizace dat do podoby uživatelsky přívětivé. Rozvoj internetu i do mobilních zařízení a jeho dostupnost takřka kdekoli vytváří tlak na přístup k aplikaci za jakýchkoli podmínek.

V práci dále nalezneme podrobný popis požadavků zákazníka, jejich analýzu a technickou realizaci. Nejrozsáhlejší část práce pak popisuje analýzu jednotlivých zdrojových systémů, ze kterých bylo nutné získat užitečná data a sjednotit je tak, aby byla vzájemně slučitelná a také zautomatizovat tento proces natolik, aby vyžadoval jen minimální servisní zásahy a systém zobrazoval aktuální údaje.

Na závěr pak zhodnotím dosažené výsledky, poskytnu zpětnou uživatelskou vazbu, navrhnu zlepšení a nastíním další vývoj produktu.

Kapitola 2

Business Intelligence

Pod pojmem *Business Intelligence* (BI) se skrývá vše, co může posloužit k lepšímu pochopení chování trhu a k zlepšení obchodních strategií. Jsou to především zkušenosti a znalosti, technologie a kvalita, analýza rizik a bezpečnost a vyjádření a prezentace informací. Přestože BI je relativně novým pojmem, koncept podpory rozhodování a řízení firmy již existuje desítky let. Počátkem 80. let byly velmi populární aplikace typu *Executive Information System* (EIS), které většinou představovaly ruční kopírování klíčových hodnot z různých reportů do dashboardů, kde bylo možné vidět vše podstatné na jednom místě. Tyto systémy byly nahrazeny *systémy na podporu rozhodování* (Decision Support System, DSS), které sdružovaly data z různých zdrojů, umožňovaly porovnání aktuálních dat s historickými a také modelovat budoucí vývoj na základě určitého rozhodnutí.

BI přenesla tento koncept do všech úrovní firemní struktury, takže každý manažer vidí svoji oblast zájmu a nemusí sledovat údaje za celou společnost. Výstupy těchto systémů lze rozdělit do tří kategorií:

- **Dashboard**: Data jsou v nich velmi agregovaná, mají obvykle podobu grafického ukazatele, který dává okamžitou představu o stavu businessu. Uplatnění najde především u vedení firmy a při tvorbě strategických rozhodnutí
- **Report**: Obvykle velké detailní, zachovávají si vzhled po delší období, takže vždy najdeme tutéž informaci v různých vydáních (např. měsíční zprávy) na stejném místě. Mají obvykle podobu grafů, které známe z Excelu.
- **Analytické reporty**: Jsou dynamické, umožňují uživateli nastavit si vlastní pohled na data. Typicky umožňují zjemnění pohledu, tzv. drill-down a naopak přechod na komplexnější pohled, tzv. drill-up. Tento typ reportů je výborný pro pochopení všech aspektů současné situace, a proto je používán především analytiky.

Spousta informací důležitých pro rozhodování přichází zvenčí společnosti, informace o konkurenci, vývoj na světových trzích apod., ale také velmi mnoho údajů se nachází uvnitř firmy. Pro BI a analýzy vůbec je důležité, že většina těchto dat je číselných, anebo alespoň převoditelných na čísla, např. typ auta na indexovaný seznam.

2.1 Datový sklad

Datový sklad je obvykle klíčovým prvkem BI řešení, není to ale podmínkou. V této sekci shrnu jeho základní vlastnosti a také výhody oproti klasickým transakčním databázím. Vycházím přitom z MSDN Library [9].

2.1.1 Struktura datového skladu

Pod pojmem datový sklad (data warehouse) si většinou představíme *multidimensionální kostku*, kterou natáčíme a řežeme, abychom dostali požadovaná data. Za tímto logickým modelem se většinou skrývá standardní relační databáze, která je ovšem navržena podle jiných kritérií než v běžných aplikacích. Tento přístup se nazývá *ROLAP* (Relational Online Analytical Processing). Druhým přístupem pak je využití vlastních struktur pro ukládání těchto kostek, což se značí jako *MOLAP* (Multidimensional OLAP). Tato druhá varianta bývá zpravidla rychlejší, jelikož umožňuje serveru zahrnout do struktury již předpočítané, což ale znamená, že při načtení nových dat je nutné tuto kostku celou přepočítat.

V praxi se většinou používá hybridní řešení (HOLAP), kdy starší data obsahují velké množství agregací a předpočítaných hodnot, za to data nová, ke kterým se dohrávají aktuální hodnoty z produkčních systémů, jich obsahují méně, aby přepočítání proběhlo rychleji. Pokud např. jako nová data budeme považovat tento a minulý rok a ostatní klasifikujeme jako stará, pak shodný dotaz bude rychlejší, pokud budeme chtít data za rok 2005 než 2008. Toto nás však nemusí znepokojovat, protože obvykle je pomalost dotazu na roce 2008 vyvážena tím, že potřebujeme detailnější hodnoty a tudíž předpočítaných agregací stejně nevyužijeme.

Základním pojmem jsou *fakta*, neboli *measures*. Jsou to hodnoty, které chceme sledovat, např. počet prodaných kusů nebo celková cena. Dále musíme znát *dimenze*, které určují možný pohled na data. Jsou jimi např. čas, geografická oblast, produkt. Každá z těchto dimenzí může být hierarchická, čímž ovlivníme granularitu pohledu. Pro čas to bude sekvence rok → kvartál → měsíc → den. Je zřejmé, že při pohledu po měsících dostanu 12x více dat jako při ročním pohledu a mohu tak zjistil vývoj prodejů během roku. Běžný dotaz pro report pak může znít takto: "Zobraz mi měsíční prodeje Paralenu po krajích".

Je důležité si uvědomit, že správné navržení datového skladu (výběr fakt a dimenzí) je klíčovou událostí v celém projektu, jelikož pozdější úpravy nebývají možné. Jelikož datový sklad neobsahuje surová data, ale již nějakým způsobem upravená (agregovaná), není již třeba možné z produktu zjistit číslo šarže. Obvykle také bývá jediným zdrojem historických dat, jelikož z výkonových důvodů (velikost databáze, počet záznamů) není možné, aby produkční databáze obsahovala celou historii. Proto v úspěšných projektech musí být zastoupeni zástupci všech kategorií uživatelů, kteří přijdou se systémem do styku, aby si stanovili své požadavky, byť i jen teoretické. Chtít totiž po několikaletém provozu systému nový pohled nemusí být vůbec možné a zákazník začne být se systémem nespokojený.

Nyní když víme, jaká data chceme v datovém skladu uchovávat, je nutné navrhnout jejich strukturu. Databáze obvykle obsahuje dva typy tabulek. Prvním z nich se nazývá *tabulka faktů* (fact table), v některé literatuře taktéž označována jako faktorová tabulka. Obsahuje hodnoty jednotlivých fakt s nejmenší možnou granularitou. Různá fakta mohou být uložena ve společné tabulce v oddělených sloupcích, nebo mohou být ve vlastních tabulkách. Obvykle se v jedné tabulce nacházejí data, která se shodují v možných pohledech, např. počet prodaných kusů a utržená cena. Dále každý řádek obsahuje sloupce s hodnotami dimenzí.

Faktorové tabulky obsahují velmi mnoho záznamů (miliony a více), a proto se snažíme o minimální délku každého řádku, což má za následek menší databázi a rychlejší vyhledávání. Dosáhneme toho tak, že hodnoty dimenzí referencujeme cizími klíči do *tabulek dimenzí* (dimension table). Tyto klíče volíme celočíselné, aby jejich velikost byla co nejmenší, i když by se např. u auta nabízela SPZ jako vhodný kandidát na tento klíč (v produkční databázi tomu tak i může být).

Většina dimenzí není lineárních, ale hierarchických - např. produktová skupina obsahuje několik produktů v několika provedeních (injekce, tablety, čípky, mast) a to ještě v různých baleních. V relační databázi bychom provedli normalizaci a každou úroveň bychom dali do zvláštní tabulky. Toto schéma se v datovém skladu nazývá *sněhová vločka*. Výhodou je nízká redundance dat, což je výhodné při změnách, ale zesložiťuje to konstrukci dotazu.

Opačným extrémem je *schéma hvězdy*, které obsahuje celou dimenzi v jedné tabulce, za cenu redundance dat. Výhodou je, že nemusíme spojovat velké množství tabulek se všemi záznamy v tabulce faktů, abychom odfiltrovali ty řádky, které neodpovídají zadaným kritériím. Hvězdicové schéma nemá normalizované dimenze ani relační propojení mezi tabulkami dimenzí, proto je velmi lehce pochopitelné, ale v důsledku nenormalizovaných dimenzí je vytvoření takového modelu relativně pomalé, ale na druhé straně, tento model poskytuje vysoký "dotazovaí výkon" [7]. Běžné řešení pak bývá někde uprostřed.

2.1.2 Použití datového skladu

Datový sklad je určen pro uchovávání a analýzu numerických informací. Nachází se v něm stálá a ověřená data. Uvedu nyní několik rozdílů oproti klasickým transakčním systémům:

- Zatímco transakční databázi využíváme při běžných denních aktivitách, sklad využijeme při plánování. Příkladem může být zjištění stavu skladu a zobrazení jeho historického vývoje, z důvodu tvorby nových objednávek.
- Databáze je zaměřená na detail, naproti tomu sklad obsahuje agregovaná data. Rozdíl mezi prodejem jednoho balení léku proti chřipce a celkovým nárůstem prodeje této skupiny léčiv v zimních měsících.
- Sklad sdružuje data z mnoha aplikací, z nichž každá má vlastní databázi.
- Databáze zachycuje současnost, sklad pak vývoj v čase.
- Hodnoty v databázi se velmi rychle mění, nemusí být přesná nebo úplná. Do datového skladu se data nahrávají v pravidelných intervalech, jsou stabilní a ověřená.
- Transakční databáze musí podporovat paralelní přístup, zápis, důležitá je nejen rychlost čtení, ale i zápisu jednoduchých dat (řádků). Sklad musí být optimalizován na rychlé čtení sumarizovaných hodnot.

2.1.3 Extraction, Transformation, Load

Extraction, Transformation, Load (ETL) je velice sofistikovaný proces, kterým se plní datový sklad z primárních zdrojů dat. *Extrakce* je výběr dat z různých zdrojů, při *transformaci* jsou tato data ověřena, validována, pročištěna a zaintegrována do struktury datového skladu. V posledním kroku *loading* pak dochází k nahrání takto předzpracovaných dat do vlastního datového skladu. Kvalita informací poskytovaných datovým skladem se úzce váže na data v něm obsažená. Je proto nutné, aby byla aktuální, dostatečně vypovídající a kvalitní. V době, kdy velké množství údajů se generuje s denními, nebo dokonce hodinovými intervaly, je včerejší databáze již zastaralá a nevypovídající. Proces ETL zajišťuje, aby se tato data promítla do datového skladu včas, podle potřeb klientů. Z důvodů vysoké časové náročnosti a možné lidské chyby nelze tuto činnost vykonávat manuálně, ale snažit se o co nejvyšší stupeň automatizace. Jak bude popsáno dále, toto není vždy možné a zásah administrátora je čas od času nevyhnutelný.

Extrakce

Primárních zdrojů dat je obecně velké množství. Obvykle produkční relační databáze pracuje v národním měřítku, kdežto datový sklad slučuje informace ze všech krajin, ve kterých daná společnost rozvíjí svoji činnost. Zde máme několik zdrojů se stejným schématem, a tudíž proces čtení dat lze řešit celkem snadno. Daleko těžším případem jsou zdroje heterogenní. Jedná se o systémy běžící na odlišných platformách, databázových serverech jiných výrobců, příp. se jedná pouze o datové soubory uložené někde na síti. Pro každý typ takového zdroje je nutné mít můstek, abychom byli schopni vůbec s daným zdrojem komunikovat. Dále je potřeba jednoznačně popsat datové schéma těchto zdrojů, abychom dobře porozuměli datům, která obsahují. Je nutné neustále sledovat změny, které jsou ve strukturách těchto zdrojů prováděny, abychom získávali kvalitní výstupy. Přesto ale můžeme tento typ zdrojů považovat za jednoduše dostupný a bezpečný.

Poslední hlavní skupinou zdrojů jsou zdroje externí. Tyto zdrojová data neposkytuje samotná firma, ale získává je od třetích stran, kupuje si je. Jedná se např. o kurzovní lístek České národní banky, skladové informace jednotlivých distributorů, prodeje v jednotlivých lékárnách apod. U těchto systémů máme většinou velmi malou až žádnou možnost ovlivnit, kdy a jak se jejich struktura změní, v případech, kdy za data neplatíme, ale přistupujeme k nim volně, např. po internetu, musíme schéma empiricky odvodit, jelikož dokumentace nemusí být dostupná. Přístup k těmto zdrojům nemusí být vždy možný, údržba systému, porucha hardware. Jelikož se nejedná o vnitropodnikové systémy, nelze jejich dostupnost ovlivnit.

Transformace

Kvalitní data jsou základem jakéhokoli nástroje pro podporu rozhodování. Nekvalitní data znamenají ztrátu důvěry v takovéto řešení a systém se oprávněně přestane využívat. Za nekvalitní data můžeme považovat i případ, kdy např. jméno a příjmení zákazníka je uloženo v jediném sloupci a těžko pak pro potřeby marketingové kampaně získáme pouze křestní jméno.

Detekovat duplicitní záznamy a poté odstranit nadbytečné je sice náročné, ale většinou technicky proveditelné. V případě chybějících údajů je situace podstatně horší. Existují sice metody, jak chybějící hodnotu spočítat, aproximovat, nebo nahradit nějakou předem zvolenou konstantou, ale vždy měníme výstupy následných analýz. V případě záznamu, který je značně nekompletní, je možné uvažovat o jeho celkovém vypuštění z transformace a dalšího zpracování.

Klasickou v počítačovém světě je pak řetězcová reprezentace data a času. Dvanáctihodinový a dvacetičtyřhodinový formát, den-měsíc-rok, měsíc-den-rok a další. Na obdobné problémy narazíme u čísel, jelikož různé národy mají různé oddělovače tisíců nebo desetinných míst. Do stejné kategorie lze zařadit i reprezentaci výčtových typů, kdy např. sloupec pohlaví může obsahovat tyto hodnoty: paní, muž, 1, M, žena, 0. Všem těmto hodnotám musí proces transformace porozumět a převést na jednotnou reprezentaci používanou v datovém skladě.

Vlastní kapitolou je zpracování částek v různých měnách. Primární zdroje dat z národních systémů většinou používají tamní měnu, ale pro datový sklad potřebujeme tyto rozdíly odstranit, abychom byli schopni provádět reporty a analýzy napříč několika zeměmi a porovnávat je mezi sebou. Při návrhu datového skladu se pravděpodobně rozhodneme pro reprezentaci všech částek v jednotné světové měně, např. Eurech či amerických dolarech. Tento problém tím ale nevyřešíme. Stále potřebujeme data do společné měny převést. Použi-

jeme kurz aktuální v době transformace, či v době pořízení záznamu? Střední hodnotu stanovenou ČNB pro nákup i prodej, nebo pro každý typ operace zvlášť? Velice záleží na typu podnikání, na tom, zda firma je spíše exportér či dovozce, a proto není možné toto rozhodnutí udělat obecně pro všechny systémy, ale je nutný dialog se zákazníkem.

Loading

Po úspěšném převodu dat do podoby vhodné pro datový sklad nám zbývá tato data vložit do struktur datového skladu. Tento krok může být velmi časově náročný podle množství vkládaných dat. Při každém importu nových dat je nutné aktualizovat předpočítané agregované hodnoty, indexovat data, aby byl přístup k nim optimalizovaný a také vytvořit primární klíče, které většinou bývají umělé, jelikož záznamy v datovém skladu již neobsahují žádné přirozené klíče, umožňující jednoznačnou identifikaci. S ohledem na náročnost těchto činností je nutné zvolit interval a dobu provádění ETL procesu, kdy delší intervaly znamenají větší objemy dat během jednoho běhu, kratší pak častou aktualizaci referenčních dat. Výhodné je provádět tuto činnost v době snížené zátěže jak primárních systémů, tak samotného datového skladu, tj. v nočních hodinách či o víkendu. Pro systémy, se kterými pracují lidé na celém světě, musíme vést v patrnosti, že kdy jedna část má půlnoc, druhá právě přichází do práce.

I přes veškerou snahu tento proces automatizovat, v každé fázi se může objevit událost, kterou nejsme schopni automaticky rozhodnout, nebo vůbec předvídat. Může jí být nedostupnost datového zdroje, kde podle jeho důležitosti můžeme chybu klasifikovat jako fatální a skončit, nebo v případě doplňkového zdroje pouze zobrazit varování a pokusit se načíst data z tohoto zdroje v příštím běhu ETL procesu. Změna struktury primárního datového zdroje jistě povede také k neúspěšnému importu. Během transformace mohou nastat případy, kdy např. sloupec, ve kterém očekáváme číselnou hodnotu, obsahuje řetězec. Automatický ETL proces musí tyto chyby zaznamenat a administrátor musí rozhodnout, jak s nimi naloží.

Kapitola 3

Požadavky na systém

3.1 Eluzzion

Tento projekt má rozšířit uživatelské možnosti současného CRM systému. Eluzzion je proprietární řešení orientované na potřeby farmaceutickým firem a jejich obchodních zástupců, reprezentantů a manažerů na všech úrovních. Klíčovou funkcí tohoto systému je umožnit reprezentantům připravit se na schůzku s klientem, zjistit jeho potenciál a loajalitu k firmě. Část těchto informací putuje do systému z vnějších zdrojů, např. prodeje v dané lokalitě, telefonické kontakty firemního call-centra. Podstatnou část informací zadává reprezentant sám – strukturovaný popis návštěvy, klasifikace lékaře podle různých kritérií.

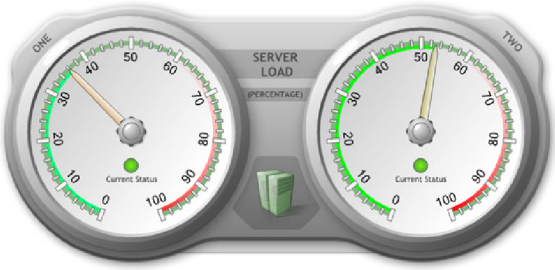
Mimo tuto hlavní funkci systém poskytuje mnoho funkcí pro usnadnění administrativních činností reprezentanta, jako je vedení peněžního deníku, evidenci rozdaných vzorků, včetně kontroly zákonných omezení ve všech zmiňovaných oblastech. Pokročilé funkce plánování umožňují reprezentantovi naplánovat si každý pracovní den efektivně, bez nutnosti dalšího vykazování činnosti nadřízenému, který má k těmto informacím přístup přímo v systému.

Z původní aplikace pro Windows se Eluzzion rozšířil na PDA, takže mohou mít medicínští reprezentanti užitečné informace opravdu vždy při sobě. Existuje také webový klient, s nímž pracují operátoři v centru a ti, kdo nemají lokální verzi funkční – ztráta notebooku, nefunkčnost hardware apod. Pro administrátory je pak připraven smart klient, který umožňuje pohodlnou editaci všech číselníků, parametrizace jednotlivých klientů a další servisní zásahy. Do kompletního výčtu aplikací skupiny Eluzzion chybí ještě reportovací a analytické nástroje. V současné době systém diponuje jen sadou statických předdefinovaných reportů, u kterých nemá uživatel možnost příliš ovlivnit zobrazovaná data.

3.2 Dashboard KPI

Vzhledem k velkému množství informací v systému Eluzzion je pro manažery na všech úrovních těžké se v těchto údajích vyznat. Aby systém získal další výhodu na trhu, je nutné jej rozšířit o reporty s grafickým uživatelským rozhraním. Cílem je poskytnout okamžitý přehled o vývoji v manažerově oblasti zájmu.

Hlavní myšlenkou je vystavět monitorovací systém obdobný těm, které se používají např. pro monitorování vytíženosti sítě, serverů apod., kdy operátor na první pohled vidí případný problém. Pokud žádný problém neobjeví, tato kontrola mu zabere pouze několik vteřin, v opačném případě musí mít možnost detailního pohledu na indikátor, který se



Obrázek 3.1: Příklad grafické komponenty [6]

nachází v červených číslech. Na detailu pak může vidět opět několik budíků s jednotlivými částmi sledovaného ukazatele, např. geografické rozdělení celorepublikového pohledu na jednotlivé kraje. Takto může v systému postupovat stále hlouběji, až se dostane na klasický tabulkový report, kde uvidí surová data, tak jak je tomu nyní.

V této kapitole uvádím pouze základní požadavky, detailnějším rozborem se budu zabývat později. Samozřejmostí jsou např. údržba a monitoring systému, správa uživatelských účtů a rolí. Zde uvedené požadavky na funkcionalitu systému mi umožní zamyslet se nyní nad dostupnými technologiemi a vybrat ty nejlepší, ať už z pohledu programátorského, finančního nebo uživatelsky přívětivého.

Kapitola 4

Dostupné technologie

4.1 Současně používané technologie

Jelikož se nejedná o nový systém, ale systém, který velkou měrou musí interagovat s existujícími aplikacemi, lépe řečeno pracovat s daty, které tyto aplikace poskytují, bude kompatibilita hrát velkou roli v hodnocení jednotlivých technologií. Dalším kritériem je pak snaha o minimalizaci nákladů využitím dostupných zdrojů – lidských, hardware, software. Lidskými zdroji rozumím především IT odborníky, starající se o bezproblémový chod serverů atd.

Nyní jako operační systém na serverech jsou použity *Microsoft Windows 2003 Server*. Uvažuje se o přechodu na verzi Microsoft Windows 2008 Server, ale v současné době neexistuje společná licence s Microsoft SQL Server 2008, která by byla cenově přijatelnější než licence na operační systém a databázový server zvlášť.

Jako aplikační prostředí je použito *Internet Information Service* (IIS) 6, s nainstalovaným .NET Frameworkem 2.0. Instalace nejnovější verze 3.5 frameworku je jen otázkou vznesení požadavku, jelikož je zdarma a jeho instalace nijak neovlivní funkčnost předchozích verzí frameworku ani ostatních součástí systému.

Databázové prostředí představuje *Microsoft SQL Server 2005 Standard edition* a vzhledem k velkému rozdílu cen se neuvažuje o pořízení Enterprise licence, která obsahuje rozšířené možnosti pro data mining a business logiku.

4.2 Databázová vrstva

Jak jsem již uvedl v předchozí kapitole, současné systémy vesměs využívají technologie Microsoftu, ale např. prodejní data se shromažďují s týdenní periodou a formátem je DBase IV. Dalším systémem, který se vymyká standardům, je SAP, který ale na druhou stranu má vlastní standardy, které jsou bohatě zdokumentovány (`www.sdn.sap.com`) a existuje široká komunita vývojářů, takže případné problémy se budou řešit snadněji.

V první fázi má systém zobrazovat pouze aktuální údaje, ale do budoucna se počítá s možností zobrazení historických dat. Ze začátku je tedy možné využít jako úložiště stávající databázovou strukturu a ušetřit tak čas do nasazení systému na produkční prostředí. Jakmile vznikne potřeba uchovávat historická data, automaticky přichází na řadu datový sklad.

Mezi jeho hlavní přednosti patři optimalizované čtení, na rozdíl od běžných databázových systémů, které musí podporovat operace vkládání, čtení i mazání na stejně kvalitní

úrovni a hlavně řešit známé problémy víceuživatelského přístupu k databázi.

Další výhodou pro náš projekt je sofistikované řešení pro ETL[1] proces. Nastavení automatického dohrávání změn do datového skladu je klíčové pro poskytování aktuálních informací. Tento proces musí být schopen vyrovnat se s většinou anomálií v datech.

Microsoft dodává svoje řešení *Analysis Services* společně s SQL serverem (verze Standard a Enterprise) [4] a tudíž firmě nevzniknou žádné náklady spojené s pořizováním softwarových licencí. Pouze v případě, že by výkon současných serverů byl pro tuto aplikaci nedostatečný, pak bude nutno uvažovat o dedikovaném stroji pro analýzy. To ale v současné době, v raných fázích projektu, nepředpokládám. Pozdější případná migrace nepředstavuje velké riziko, a pokud nebude spojena s přechodem na jinou verzi systému (operačního, databázového či aplikačního), proběhne hladce.

4.2.1 Konkurenční řešení

Velká trojka (Microsoft, Oracle, IBM) se na své vedoucí postavení v oblasti Business Intelligence dostala akvizicí dřívějších leaderů trhu. Microsoft koupil ProClarity, IBM Cognos a Oracle Hyperion. Jak ukazuje tato analýza trhu [2] z roku 2006, v současné době je jedničkou na trhu Microsoft s více jak 30% trhu. Jeho velkou výhodou je využití rozšířeného a velmi populárního Excelu, jako front-end. Koncoví uživatelé se tudíž nemusí učit ovládat novou aplikaci, pouze si rozšíří znalosti o to, jak data z datového zdroje získat. Všechny ostatní úpravy jsou pak totožné s klasickým sešitem.

Všichni velcí hráči nabízí ve svých systémech prakticky tytéž možnosti (viz. Oracle [5], IBM [3]). Důraz kladou na ETL procesy, spolupráci s různými zdroji dat, integraci s reportovacími a analytickými nástroji a především na spolupráci s jejich databázovými řešeními a tudíž minimalizaci nákladů při výběru stejného dodavatele.

Tento důvod sehrál roli i v mém výběru a proto systém budu vyvíjet na Microsoft SQL Server Analysis Services. Rád bych využil nejnovější verzi 2008, kvůli přepracovaným vývojovým nástrojům a zlepšené výkonnosti serveru.

4.3 Prezentační vrstva

Jako nejjednodušší se jeví použití standardně dodávaného reportovacího nástroje: *Reporting Services* pro Microsoft SQL Server nebo *Oracle Business Intelligence Answers* pro Hyperion. Tyto nástroje obsahují uživatelsky přívětivý designer a umožňují koncovému uživateli, který má rámcovou představu o fungování databází, vytvořit report podle vlastních požadavků. Jsou standardní součástí databázových serverů, a tudíž jejich použití je striktně vázáno na použití s SQL serverem daného výrobce.

Pro návrh jednoduchých reportů jsou tyto nástroje velmi užitečné a značně urychlují práci, jelikož poskytují již vestavěnou prezentační vrstvu, do které se nahraje pouze definice reportu, který se má zobrazit. Jak to bývá u podobných nástrojů, složitější požadavky jsou nerealizovatelné, nebo jen s velkými obtížemi. Jako srovnání bych použil Microsoft Word a systém LaTex. Většina uživatelů používá první možnost, protože je příjemnější a jejich potřebám bohatě dostačuje. V okamžiku, kdy na jednom dokumentu pracuje více lidí, potřebujeme dodržovat typografická pravidla, používáme stále stejné vzory, zvolíme druhou možnost, protože se můžeme spolehnout, že výsledek bude vždy kvalitní i ve velmi

[1] Extract, Transform, Load – postup pro import dat z jiných systémů, více na http://en.wikipedia.org/wiki/Extract,_transform,_load

rozsáhlých projektech. Problém, který řeším je obdobný. Celý report je totiž možné sestavit v Business Intelligence Studiu, vyexportovat jako XML a tuto definici zpřístupnit na reportovacím serveru. Tento způsob je dostupnější širšímu okruhu uživatelů, většinu požadavků v něm lze vyřešit, ale pokud potřebujeme např. složitější matematické funkce, nebo propojení jednotlivých hodnot, přestává tento způsob vyhovovat.

Reporting Services od Microsoftu obsahují komponenty pro zobrazení reportu na webu (při použití technologie ASP.NET) i na Windows platformě (opět jako komponenta pro .NET Framework). Při instalaci služby na server se současně nainstaluje webová administrace, přes kterou je možné upravovat všechny parametry reportů závislé na prostředí, jako jsou přístupy k databázím, uživatelská oprávnění atd.

Možné použití této technologie přicházejí v mém případě v úvahu dvě. Přes standardní webové rozhraní Reporting Services, které ale předpokládá management uživatelů přímo v tomto nástroji (lze využít doménové účty a skupiny). Jeho největší nevýhodou je přílišná grafická strohost a nemožnost jejího přizpůsobení. Druhou, mnohem flexibilnější variantou je vytvořit si aplikaci vlastní, do které vložím komponentu *ReportViewer* podobně jako se vkládá tlačítko nebo textové pole. V tomto případě mám plnou kontrolu nad grafickou stránkou věci, mohu implementovat i ostatní požadavky na plnohodnotnou aplikaci jako je omezení přístupu, audit atd.

Způsob, který nabízí naprostou variabilitu v řešení, umožňuje využít veškerých možností datových skladů a relačních databází, je vlastní aplikace, nepoužívající Reporting Services. Velmi významnou daní za to ovšem je nutnost vytvořit všechny součásti aplikace vlastními silami. Nejvíce úsilí oproti předchozí variantě musím vynaložit v části návrhu reportu, ať po datové, či grafické stránce.

V současné verzi MS SQL Serveru 2008 jsou již standardně dodávané grafické komponenty společnosti Dundas, kterou Microsft koupil. Jelikož tato koupě proběhla těsně před uvolněním SQL serveru, jsou zakomponovány pouze základní objekty. Ostatní mají následovat spolu s prvním opravným balíčkem. Komponenty společnosti Dundas lze využít pouze v Reporting Services a ačkoli jsou stejné jako ty pro standardní ASP.NET – musí podporovat renderování HTML kódu, nastavování parametrů pro zobrazení, není možné použít je ve vlastní aplikaci. Důvod je prostý, komponenty pro ASP.NET stojí cca 1000$ (listopad 2008).

4.3.1 Silverlight

S představením *Windows Presentation Foundation* (WPF) umožnil Microsoft vyvíjet dobře vypadající aplikace pro Windows na platformě .NET. Rozvoj aplikačních možností přišel s Windows Vista a grafickým módem Aero. Hlavními výhodami je zapojení multimédií, vektorové grafiky a animací do vývoje aplikace. Oddělení kódu aplikace a grafické podoby aplikace umožnilo grafikům podílet se přímo na vývoji. Pomocí Microsoft Expression Studia mají možnost navrhnout vzhled jednotlivých obrazovek aplikace a přechodů mezi nimi. Výstupním formátem je *XAML* (čteno zaml), což je XML soubor, jehož schématu rozumí .NET aplikace od verze frameworku 3.0. Na jednotlivé akce definované v XAML (události tlačítek, pohyb myši, apod.) je pak možné navázat aplikační kód a další vývoj je potom shodný se standardními Windows aplikacemi.

Rozšířením WPF na *Windows Presentation Foundation / Everywhere* (WPF/E) zavedl Microsoft podobný princip i pro webové aplikace. Toto rozšíření se nazývá *Silverlight*, jeho implementace na Linuxu (Mono framework) pak *Moonlight*.

Největším konkurentem pro Silverlight je *Adobe Flash*, jenž je současnou jedničkou

v grafických aplikacích na webu. Jeho výhodami je dlouholetý vývoj a podpora ze strany drtivé většiny prohlížečů. Odpůrci Silverlightu dále poukazují na vysokou pořizovací cenu vývojových nástrojů, jak grafického studia Expression, nebo Visual Studia pro vývoj samotné aplikace, a také na malou kompatibilitu v prohlížečích. V současné době jsou podporovány pouze IE 6 a vyšší, Firefox a Safari. Z výčtu hlavních prohlížečů schází Konqueror a Opera, přičemž podpora pro Operu již byla ohlášena. Zde musím uvést, že je to podstatný rozdíl oproti Reporting Services, které jsou podporovány pouze v Internet Exploreru.

Silverlight bychom neměli chápat jako samostatný systém pro vývoj aplikací, ale pouze jako komponentu .NET frameworku. Při vývoji je tudíž možné použít všechny znalosti z vývoje standardních aplikací, lze jej sloučit s již existujícím kódem. Pokud tedy vyvíjíme jak tlustého, tak tenkého klienta, můžeme pak nadefinovat pouze vzhled jednotlivých aplikací v XAMLu a máme obě aplikace hotové. Bohužel definice XAMLu pro WPF a Silverlight není kompatibilní a nelze je zaměnit.

4.3.2 Web Parts

Systém web part přináší do webového prohlížeče možnosti přizpůsobení webové stránky ze strany uživatele, který má možnost poskládat si stránku z dostupných komponent, upravovat jejich parametry, stylovat je, atd. Standardní součásti ASP.NET zvládnou všechny tyto funkce, včetně perzistence takto personalizovaných stránek. Vývojáři jednotlivých komponent (web part) se pak vůbec nemusí zabývat tím, jak je která funkcionalita implementována, o vše se stará generovaný kód vně samotného objektu web party. Pro potřeby naší aplikace je tento systém výhodný především v tom, že nenutí administrátora vytvářet "statické" stránky s dashboardy, ale stačí pouze nadefinovat komponenty (KPI ukazatele) a nechat uživatele, aby si je rozmístil na stránku tak, jak to právě jemu přijde výhodné a přehledné.

Jelikož se v poslední době v podnikové sféře hodně rozmáhá využití Sharepointu jako centrálního skladiště dokumentů a informací, je dobré poznamenat, že nativně podporuje web party. Teoreticky tak lze použít komponenty užité v běžné aplikaci na portálu Sharepointu. Praxe je trochu složitější, jelikož web party pro Sharepoint dědí od jiné základní třídy a vyžaduje některé parametry navíc.

I tato technologie Microsoftu je plně podporovaná pouze v Internet Exploreru. Ostatní prohlížeče mají problémy především s editací – např. ve Firefoxu nefunguje drag'n'drop komponent mezi jednotlivými zónami stránky.

4.3.3 Grafické komponenty

Klíčovou komponentou celého systému je budík. Jednoduchá grafická komponenta, která dokáže zobrazit jedinou hodnotu, resp. její rozsah. Budíky, teploměry a jiné ukazatele jsou nosnou částí systému, jelikož reprezentují data, pro která uživatel do systému přišel. Musí tedy splňovat nejvyšší požadavky na kvalitu, srozumitelnost a grafickou vytříbenost. Na internetu můžeme najít množství grafických komponent odpovídajících našemu zadání. Mnohé dokonce obsahují přímo ve svém názvu slovo dashboardy, čímž se snaží naznačit primární oblast, pro kterou jsou určeny. Existují i řešení, která jsou dostupná zdarma, případně pod licencí GPL. Bohužel tento typ licence nemohu použít, jelikož vyvíjím proprietární aplikaci a zveřejnění jejího kódu není žádoucí. Kvalitou tyto komponenty nedosahují na řešení společností Dundas nebo Infragistics.

Firma Data Systém Solutions disponuje licencí na produkt druhé ze jmenovaných společností nazvaný *Net Advantage*, proto byla volba jednoduchá. Firma Dundas nabízí své komponenty

zdarma k SQL Serveru 2008, které jsou ale použitelné pouze v Reporting Services, a jako technologii pro tento systém jsem je zavrhl. Jelikož pro běžné webové aplikace jsou tyto komponenty zpoplatněny, lze říct, že se jedná o pouhý preview produktů této společnosti se snahou nalákat klienty na placené služby.

4.4 Web Client Software Factory

Web Client Software Factory (WCSF) představuje řešení pro základní úlohy v návrhu podnikových webových systémů. Cílem je modularita systému, což jednak umožňuje několika nezávislým týmům pracovat na různých částech aplikace a také vylepšuje bezpečnost a možnosti testování. WSCF obsahuje množství komponent nejen pro ASP.NET, ale i AJAX a také Enterprise Library, o které se zmíním později. Jelikož je tato factory produktem Microsoftu, dá se očekávat její propracovanost a vysoká kvalita. Samozřejmostí je pak podpora na MSDN (Microsoft Developer Network).

Představím hlavní výhody WCSF uváděné na stránkách MSDN zabývající se návrhovými vzory. Pro business to jsou především konzistentnost uživatelského rozhraní, čímž se snižují náklady na školení uživatelů, a také jednodušší nasazování nových verzí a aktualizací. Pro softwarové architekty, pak představuje přínos především v možnosti vytvořit částečné implementace obsahujících nejdůležitější součásti systému a odhalit tak možné chyby v návrhu a jiné problémy v raných fázích projektu, což má pozitivní dopad na jeho finanční i časovou stránku, tzv. *risk costs*. Samotným vývojářům zase zlepší orientaci v systému jeho modularita, vyřešené logování, ošetření výjimek, či správa uživatelů.

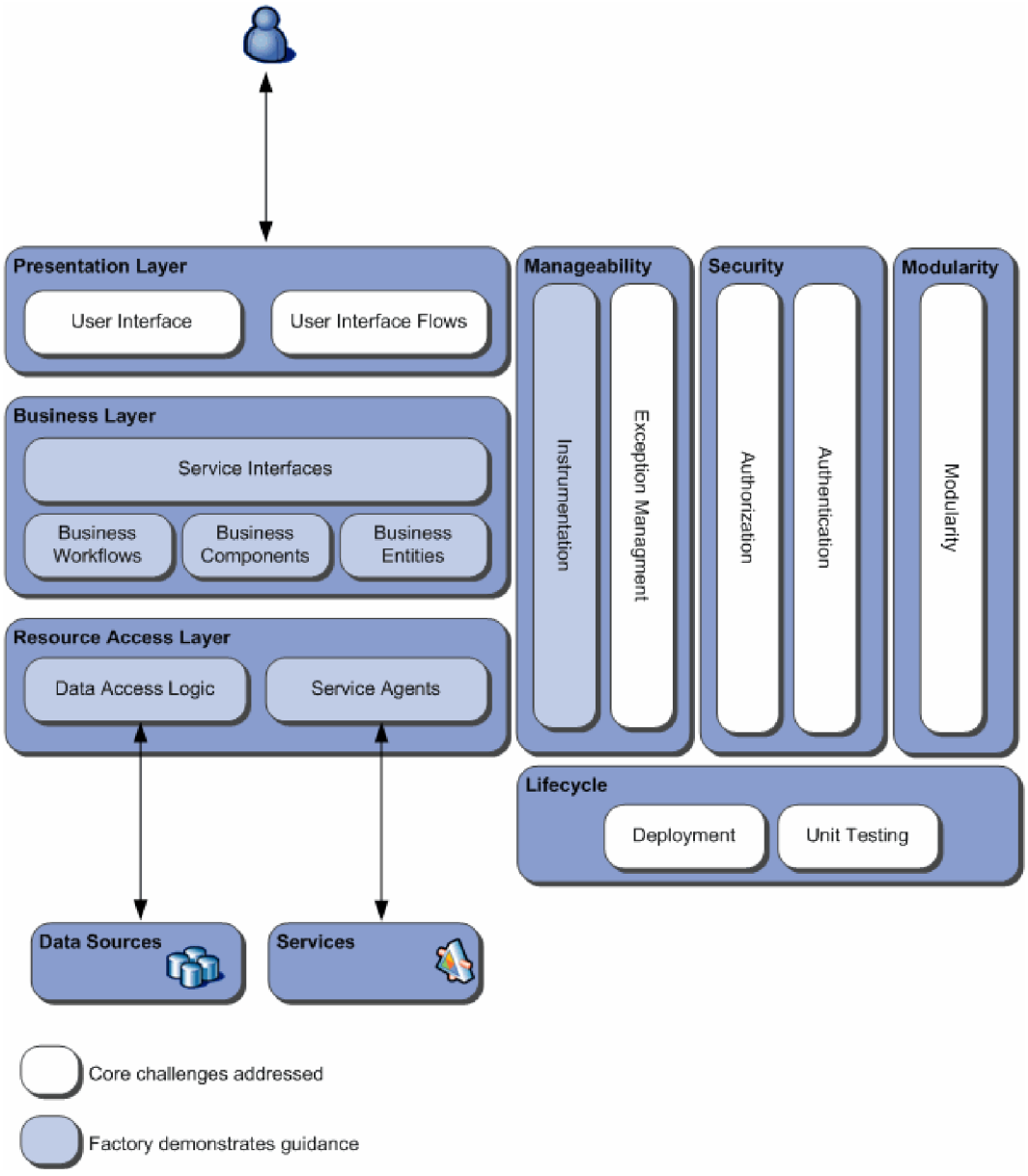
4.4.1 Enterprise Library

Podle názvu bychom asi ve světě Windows čekali dll knihovnu obsahující funkce užitečné ve velkých projektech. Ve skutečnosti se Enterprise Library skládá z několika na sobě nezávislých modulů, což z ní činí velmi praktickou pomůcku vývoje. Při vývoji systému totiž není nutné implementovat všechny dotčené oblasti, tak jak je navrhli její tvůrci, ale je možné využít pouze části, které potřebujeme.

Její poslední verze 4.1 byla představena v říjnu loňského roku a obsahuje tyto bloky[2]:

- Caching Application Block
- Cryptography Application Block
- Data Access Application Block
- Exception Handling Application Block
- Logging Application Block
- Policy Injection Application Block
- Security Application Block
- Unity Application Block

[2]Názvy bloků jsou velmi výstižné, detailní popis přesahuje možnosti tohoto textu; odkazuji proto jen na stránku MSDN, která je této problematice věnována: http://msdn.microsoft.com/en-us/library/dd203099.aspx (duben 2009)



Obrázek 4.1: Architektura aplikace využívající WCSF

- Validation Application Block

Tato knihovna není určena pouze pro webové aplikace, ale stejně tak se uplatní v aplikacích typu WinForms, Windows Presentation Foundation nebo webových službách. Jednotlivé moduly se vyznačují vysokou konfigurovatelností, a tudíž je možné značně ovlivnit jejich chování až při spuštění aplikace, příp. za běhu. Celá knihovna je navržena tak, aby bylo možno použít vlastní pluginy, doplňující aplikačně specifické funkce pro jednotlivé moduly.

Kapitola 5

Detailní rozbor systému

Každý uživatel uvidí svoji personalizovanou sadu dashboardů, které se budou odvíjet od jeho role v systému a jeho postavení v organizační struktuře společnosti. Systém musí zabránit tomu, aby uživatel mohl zobrazit data z jiných organizačních jednotek, příp. report určený pro jeho nadřízené. Směrem dolů systém omezován nebude, a tudíž manažer bude moci vidět vše, co jeho podřízení.

5.1 Vzhled a základní funkcionalita

Dashboard je celek, který bude v sobě zahrnovat jednotlivé položky, bude mu možno přiřadit společné parametry, které se pak budou propagovat do jednotlivých komponent na stránce. Tyto parametry bude mít uživatel možnost zvolit. Pokud tak neučiní, použijí se předdefinované hodnoty. Každý uživatel si bude moci sestavit dashboard podle vlastních požadavků a dále mu budou zobrazeny reporty definované administrátorem, pro jeho roli a pozici v organizační struktuře. Pro jednoduchou orientaci v systému musí každý dashboard obsahovat název a popis. Bude možné jej vyexportovat do Excelu, příp. PDF souboru.

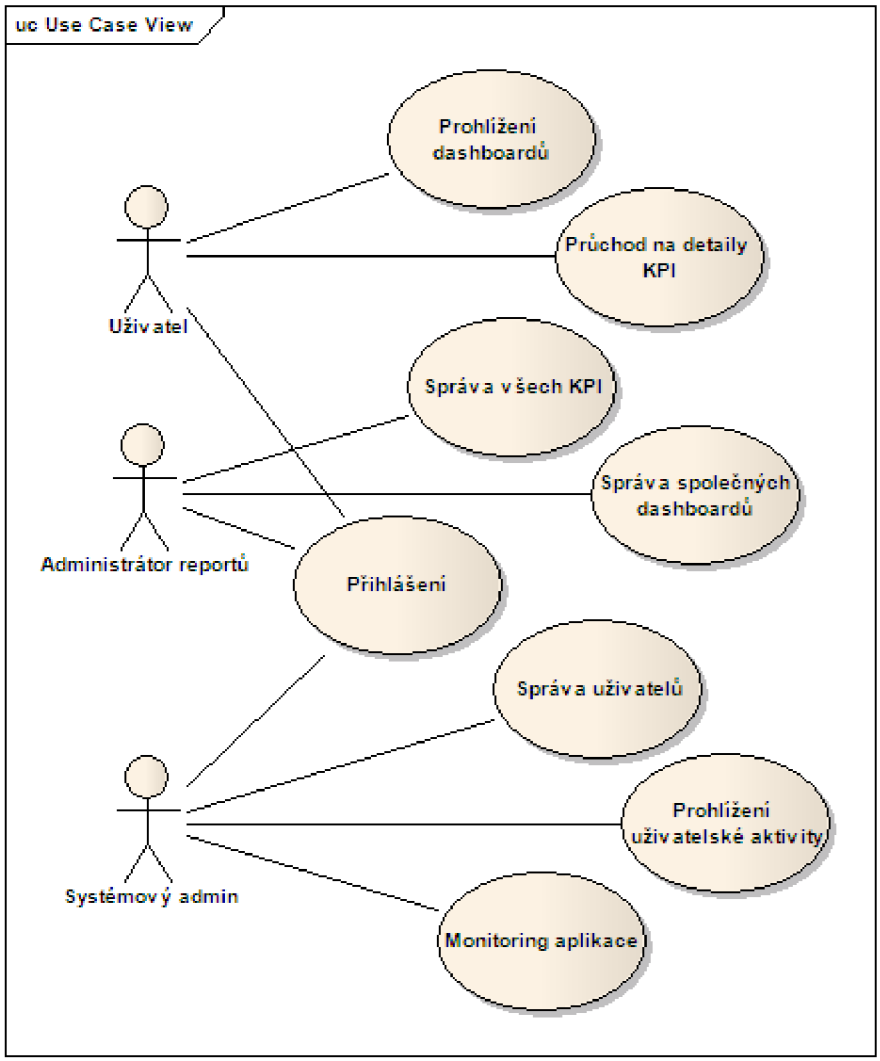
Položky dashboardu představují jednotlivé KPI[1] ukazatele, které jsou reprezentovány budíkem (viz. obrázek [3.1]), teploměrem, tabulkou, grafem. Komponenta musí obsahovat popisky hodnot a pro lepší přehlednost by měla být doplněna legendou. Parametry položek budou navázány na dashboard, ve kterém se nacházejí. V opačném případě se použije interní hodnota, buď předdefinovaná, nebo uživatelsky zvolená. Tyto uživatelské hodnoty se budou uchovávat spolu s dashboardem, tak, aby uživatel nebyl nucen při jeho příštím zobrazení ručně editovat všechny povinné hodnoty znovu.

Při kliknutí na položku se zobrazí její detail. V režimu definice těchto komponent je možné zvolit způsob zobrazení detailu:

- zobrazení stejné komponenty pro logicky menší celky (organizační jednotky, země → kraje → okresy, atd.)
- zobrazení jiného reportu
- zobrazení detailního reportu, např. tabulky, který bude nejnižší úrovní systému a nebude možné zobrazit jeho detail.

Popis jednotlivých případů použití:

[1] Key Performance Indicator – více na http://en.wikipedia.org/wiki/Key_performance_indicators



Obrázek 5.1: Use Case diagram

Případ použití: **Prohlížení dashboardů**
ID: 1
Stručný popis: Zobrazení požadované sady KPI ukazatelů
Vstupní podmínky: Uživatel musí být přihlášen do systému
Primární aktéři: Manager (uživatel)
Sekundární aktéři: Databáze
Hlavní tok: 1. Uživatel vybere modul dashboardy v aplikačním menu. 2. Systém ověří jeho uživatelská práva a nabídne mu k dispozici dostupné dashboardy. 3. Uživatel si ze seznamu vybere požadovaný dashboard.
Následné podmínky: Žádné

Případ použití: **Průchod na detail KPI**
ID: 2
Stručný popis: Zobrazení detailu jednotlivého ukazatele
Vstupní podmínky: Uživatel musí mít zobrazenu stránku s dashboardem
Primární aktéři: Manager (uživatel)
Sekundární aktéři: Databáze
Hlavní tok: 1. Uživatel je na stránce s dashboardem. 2. Při kliknutí na graf (ukazatel), který má definovaný detailní pohled dojde k přechodu na další stránku. 3. Na následující stránce se zobrazí zvolené KPI několikrát, podle typu přechodu budou jednotlivé ukazatele zobrazovat menší celky původního ukazatele. 4. Tuto činnost je možno opakovat, dokud je průchod na nižší úroveň možný.
Následné podmínky: Žádné

Případ použití: **Správa všech dashboardů**
ID: 3
Stručný popis: Administrace stránek se sadou dashboardů
Vstupní podmínky: Uživatel přihlášen do systému
Primární aktéři: Administrátor reportů (uživatel s právy na editaci dashboardů)
Sekundární aktéři: Databáze
Hlavní tok: 1. Uživatel zvolí dashboard k editaci. (a) Kliknutím na tlačítko *Nový dashboard* (b) Zvolením existujícího dashboardu a kliknutím na tlačítko *Edit* 2. Na zobrazené stránce má možnost editovat základní vlastnosti dashboardu - název, schéma zobrazení a uživatelská práva. 3. Dalším krokem je uložení změn z této části, kliknutím na tlačítko *Save*. 4. Zobrazí se stránka s dashboardem, pokud nejde o editaci bude prázdná. 5. Uživatel kliknutím na tlačítko *Katalog* zobrazí seznam dostupných ukazatelů a tyto může přidávat a následně rozmisťovat na stránce. 6. Uzavřením katalogu dojde k přepnutí do standardního módu prohlížení.
Následné podmínky: Uživatel musí mít právo na daný dashboard, aby jej mohl zobrazit

5.2 Bezpečnost aplikace

Základním požadavkem na aplikaci je zamezení přístupu neoprávněných osob. Toho bude docíleno jednak použitím zabezpečeného kanálu (https), ale především ověřením uživatelských údajů. Metody pro přihlášení budou dvě – ověření uživatele pomocí jeho doménového účtu a v případě, že tento uživatel není úspěšně autorizován, pak mu bude nabídnut standardní přihlašovací dialog. Cílem je umožnit většině uživatelů práci bez nutnosti viditelně se ověřit v systému, ale zároveň umožnit přístup k systému těm, kteří pracují např. z domova nebo ze zapůjčeného notebooku.

Všechny uživatelské činnosti budou podléhat monitorování, aby nemohlo dojít k narušení systému. Tento monitoring bude navíc sloužit k vyhodnocení využití systému, ať už pro plánování systémových zásahů do nejméně využívaných hodin, ale také pro zpětnou vazbu, které moduly jsou uživatelsky nejatraktivnější a které nejsou vůbec využívány. Tato informace pak může být použita vývojáři k efektivnějšímu budoucímu vývoji systému, ale také manažery k hodnocení zájmu o tu kterou oblast informací.

5.3 Uživatelská práva

Základem tohoto systému jsou přístupová práva. Na způsobu jakým budou vyřešena, pak záleží spousta dalších věcí, jako například personalizace dashboardů. Dalším důležitým faktorem je znovupoužitelnost tohoto modulu. Nejde jenom o využití v ostatních aplikacích firmy, ale samotná tato aplikace by se mohla v budoucnu stát pouze modulem v nějakém portálu a proto správa uživatelů musí být dostatečně obecná a rozšiřitelná o specifika dalších modulů.

5.3.1 Základní pojmy

Vzhledem k tomu, že následující pojmy chápeme intuitivně každý, ale s drobnými odchylkami, které mohou ztížit celkové pochopení problému, uvedu základní terminologii používanou v této kapitole.

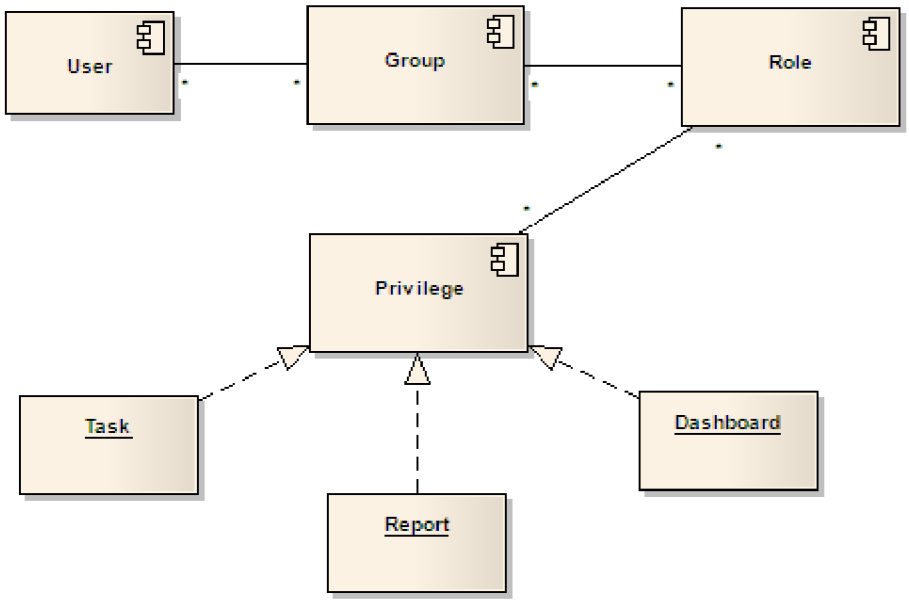
Uživatel je většinou fyzická osoba, která pracuje se systémem. Je jednoznačně identifikována loginem, přičemž nemusí platit, že by jedna osoba nemohla vlastnit více přihlašovacích údajů, tj. identifikovat se v systému různě. Stejně tak uživatelem nemusí být fyzická osoba, ale například nějaký bot, který provádí automatizovanou činnost.

Skupinou rozumíme množinu uživatelů s podobnými vlastnostmi. Skupiny nemusí být nutně disjunktní.

Role představuje skupinu logicky spojených oprávnění.

Privilegium neboli *oprávnění* je konkrétní činnost, množina dat, pro kterou se dá omezit přístup.

5.3.2 Komponenty a jejich vazby



Obrázek 5.2: Vazby mezi komponentami pro správu uživatelů

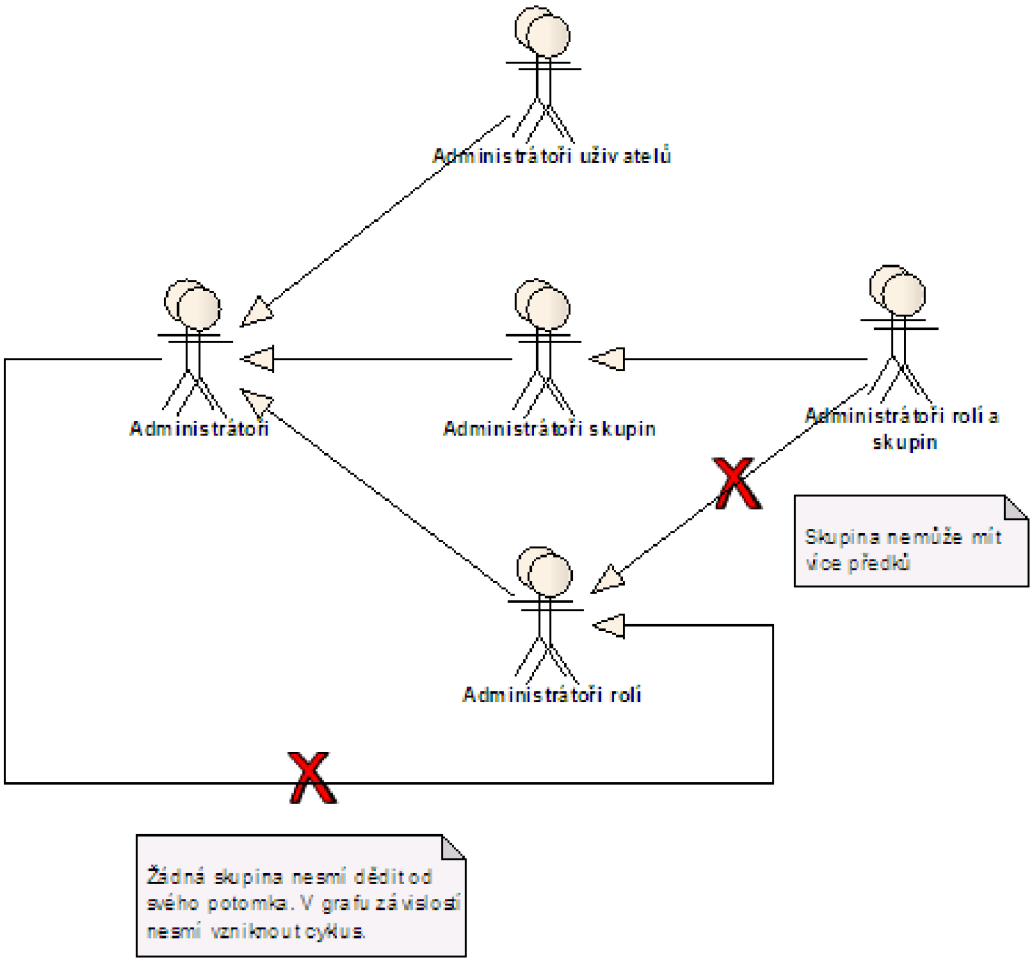
Uživatel může být obsažen ve více skupinách. Hlavní úlohou skupin je sdružovat uživatele stejných vlastností dohromady a díky tomu umožňovat jednodušší správu oprávnění.

Skupiny jsou organizovány hierarchicky. V jedné skupině může být více členů a může jí příslušet několik rolí. Role je seznam logicky svázaných privilegií. Ke každé roli je možné přiřadit více oprávnění a může ji sdílet několik skupin.

Do systému jsem nezařadil možnost přímého přiřazení role k uživateli, což se může zdát užitečné, ale značně by to komplikovalo zjišťování oprávnění i správu všech vazeb. Jako náhradu lze použít tzv. *uživatelskou skupinu*, což je skupina, ve které je obsažen pouze daný uživatel. Tato skupina se tedy nijak neliší z aplikačního pohledu, je odlišná pouze z pohledu správce systému.

Oprávnění jsou rozdělena do jednotlivých typů. Obecným typem je pouze *Task*, který je ve skutečnosti jeden konkrétní Use Case. Na schématu [5.2] jsou zobrazeny i tabulky pro oprávnění na dashboardy a reporty. Tyto už patří k modulu Dashboard a stejným způsobem by přibývaly další tabulky, pokud bychom do systému integrovali další, nyní zatím neznámý modul. Vrstvu knihoven, které spravují uživatelská oprávnění, také nemusíme upravovat – stačí použít *Extension methods* definované přímo v daném modulu.

5.3.3 Hierarchie skupin



Obrázek 5.3: Vztahy mezi skupinami

Jak je naznačeno na obrázku [5.3], skupiny musí tvořit stromovou strukturu, tj. nesmí

obsahovat žádné cykly a každý potomek musí mít pouze jednoho předka. Celá hierarchie nemusí být nutně reprezentována jedním stromem, ale může jich být více. Jinými slovy, nemusí existovat žádná skupina, do které by patřili všichni uživatelé.

Z pohledu oprávnění můžeme říci, že potomek skupiny má všechna její privilegia rozšiřitelná o specifická pro tohoto potomka.

5.3.4 Privilegia

Aby celý systém správy uživatelů fungoval dostatečně obecně a pružně, bylo potřeba zakomponovat dědičnost, kterou jsem zpracoval na úrovni skupin, a také zamezení práv. Příznak zda jde o kladné členství, bylo možné přiřadit ke každé vazbě. Vícenásobná vazba by měla za následek velmi složitý proces zjišťování oprávnění. Pro jednoduchost a přehlednost jsem tedy tento příznak dal až do posledního článku, tj. přiřazení vlastních privilegií do rolí.

Standardní způsob, jakým se negativní oprávnění chápou, je jejich významová nadřazenost kladným právům. Uživatel tedy může být v 10 rolích, které obsahují dané oprávnění, ale přesto mu bude odepřeno, pokud bude členem jediné role, která bude toto oprávnění odmítat.

5.3.5 Získání oprávnění

Díky jednoduchému návrhu je rozhodnutí, zda povolit či zamezit přístupu k danému zdroji, velmi jednoduché. Postup je následující:

1. Pro daného uživatele zjistíme seznam skupin, do kterých patří. Včetně skupin, do kterých patří automaticky, díky skupinové hierarchii.
2. Pro každou skupinu z tohoto seznamu se zjistí role, které danému uživateli patří.
3. Zjistíme, zda daná role obsahuje hledané oprávnění.
   (a) Neobsahuje-li role dané oprávnění, pokračujeme další rolí.
   (b) Při pozitivním oprávnění si tento nález poznačíme a také pokračujeme další rolí.
   (c) Pokud obsahuje negativní, okamžitě vrátíme zamítavý výsledek.
4. Do tohoto bodu se dostane pouze v případě, že nebylo nalezeno žádné záporné oprávnění. Pokud máme poznačeno alespoň jedno kladné oprávnění, právo udělíme, v opačném případě přístup zakážeme.

Pokud předpokládáme, že uživatelská práva jsou neměnná po dobu jedné session (od přihlášení po odhlášení uživatele), můžeme si seznam rolí z bodu 2 uložit do cache a při dotazech na práva provádět třetí bod. Uchování seznamu rolí může být po dobu celé session nebo jenom jednoho postbacku. Toto rozhodnu až při zátěžových testech systému, zda se zbytečně nezvýší objem uchovávaných dat. Stálost uživatelských práv je vhodné předpokládat z důvodu možné inicializace jednotlivých modulů podle práv v době přihlášení a jejich následné nesprávné funkce.

5.4 Správa dashboardů a KPI

Pro každou pozici ve firmě, organizační jednotku, musí existovat možnost předdefinovat sadu dashboardů, které uživatel uvidí. U každého z těchto dashboardů bude možnost povolit personalizaci, tj. doplnit si jej o další ukazatele a uspořádat si je podle vlastního uvážení.

Vlastní definice dashboardu bude grafická do té míry, do jaké to umožní použité technologie. Jelikož tento modul nebudou používat jenom administrátoři, ale bude dostupný s omezenou funkcionalitou pro všechny uživatele, je nutné uvažovat především o intuitivnosti používání.

Při tvorbě dashbordu bude možné definovat parametry jak globální, tak specifické pro jednotlivé komponenty. Dále bude možné zvolit grafický styl reportu, předpokládá se pouze výběrem z předem navržených.

Manažer bude schopen upravit si vzhled i obsah tak, aby v něm jednoduše a rychle nalezl všechny potřebné informace, které požaduje. Tato podmínka je základem pro spokojenost uživatelů, jejich dobrou orientaci v systému a tím pádem celkový komfort práce se systémem, který povede k jeho hojnému využívání. Dále musí být uživateli nabídnuta sada ad-hoc dashboardů, které nemusí upravovat, pokud potřebuje rychlý přehled o některé oblasti postihované systémem, nebo není technologický nadšenec a nechce studovat detailně všechny možné funkce. Editace dashboardu má být spíš třešnička na dortu pro zkušenější uživatele, kteří dokážou využít potenciál systému, aby vlastní aplikace nebyla tím úzkým místem, překážkou, v jejich pracovní činnosti.

5.4.1 Přehled základních KPI

Pro většinu uživatelských rolí budou KPI typově stejné, pouze budou zobrazovat jinou úroveň detailu. Pro medicínského reprezentanta bude viditelná pouze jeho oblast, zatímco manažer si bude moci zobrazit výsledky za všechny své podřízené a tudíž za mnohem větší oblast, nejen teritoriálně. Systém musí být natolik modulární, aby přidání nového KPI znamenalo pouze jednoduchou úpravu v datové definici a nikoli vytvoření nové verze celého systému.

Sledované oblasti pro KPI:

- prodejní výsledky: porovnání aktuálních oproti požadovaným, po jednotlivých oblastech, jejich srovnání
- vývoj prodejů: počty aktivit oproti prodejním výsledkům, srovnání oblastí
- výdaje: porovnání jednotlivých oblastí
- pokrytí: frekvence návštěv, počet navštívených osob (v cílové skupině, mimo ní)
- práce podřízených: procenta času stráveného v terénu, na seminářích, administrativou, společné návštěvy manažerů s medicínskými reprezentanty, porovnání jednotlivých manažerů, reprezentantů
- skladové zásoby: aktuální vs. požadované, po produktech, skupinách produktů

5.5 Předpokládané fáze nasazení systému

Vzhledem k požadavku zákazníka mít základní reporty přístupné co nejdříve, musím funkčnost systému v první fázi omezit na nezbytné minimum. Vše ostatní pak rozdělím do několika dalších verzí, jelikož vývoj celého systému zabere několik měsíců a je nemožné, aby klient čekal tak dlouho na většinu funkcí systému.

Fáze 1

- Přihlášení do systému
- Zobrazení statických, předdefinovaných, dashboardů
- Dashboard se bude skládat z komponent
- Stanovení KPI pro jednotlivé uživatelské typy

Fáze 2

- Vytvoření datového skladu s požadovanou strukturou
- Rozšíření seznamu zobrazovaných KPI
- Administrační rozhraní pro správu uživatelů

Fáze 3

- Možnost vytváření personalizovaných dashboardů
- Prostředí pro definování dashboardu
- Monitoring uživatelské činnosti v systému
- Export dashboardu do Excelu a PDF

Kapitola 6

Implementované řešení

6.1 Data Transfer Objects

Data Transfer Object (DTO) je návrhový vzor, který by neměl obsahovat žádnou business logiku, jen nejzákladnější funkce na validaci a kontrolu konzistence [1]. Oddělení dat od metod je výhodné při vzdálených voláních, např. webových službách, kde se snadno zajistí serializace do XML.

Na obrázku [6.1] můžeme vidět datový model vytvořený Visual Studiem jako grafická reprezentace jazyka DBML[1] pro použití v LINQ to SQL. Linq2Sql, jak bývá často zkráceně označován, představuje 1:1 mapování databázové struktury (pouze vybraných objektů) do aplikace v .NETu. Vytvoří třídy reprezentující jednotlivé tabulky a prováže je vazbami. K objektům v databázi je tak možno přistupovat pomocí standardních kolekcí, příp. psát dotazy využitím Lambda výrazů.

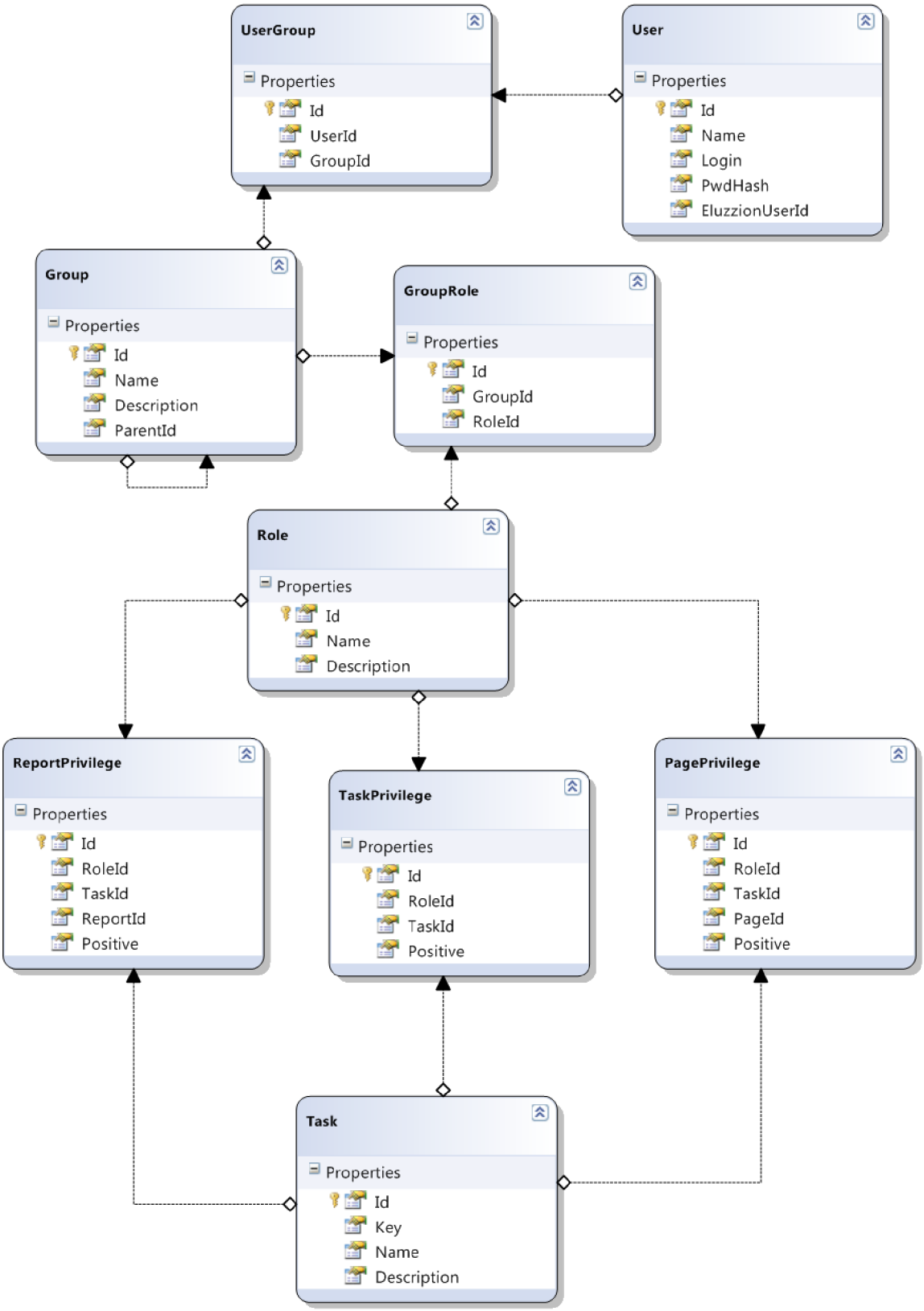
Velkou výhodou je, že veškerá tato činnost podléhá syntaktické kontrole a tudíž nedovolí napsat chybný SQL dotaz, jak je tomu v případě psaní dotazů v podobě textových řetězců. Při změně schématu je tak nutné upravit pouze DBML schéma a velkou většinu případných nutných úprav v kódu aplikace odhalí syntaktický analyzátor. Projekty využívající verzovací systémy (source control) se ovšem potýkají s problémy, že DBML definice je reprezentována jedním XML, včetně grafického rozložení designeru. Posunutí tabulky v návrhu tedy způsobí velmi výraznou změnu definičního souboru a programy na zjištění rozdílů mezi verzemi (např. diff) mají problémy.

S prvním Service Packem pro .NET Framework 3.5 byl představem Entity Framework (EF), který úroveň abstrakce nad databází posouvá ještě dále, třídy již nemusí korespondovat s databázovými tabulkami, mapování nemusí být 1:1, lze využít dědičnost, atd. Samozřejmostí je práce nad objekty, kolekcemi objektů pomocí Linqu, v tomto případě LINQ to Entities. Podle roadmap obou konceptů je zřejmé, že Microsoft považuje Linq2Sql za slepou cestu a dále bude vyvíjet pouze EF. Bohužel v době, kdy jsem vybíral technologie pro tento projekt, mi EF přišel zbytečně komplexní pro tento typ aplikace, a proto jsem zvolil přímočařejší Linq2Sql.

6.1.1 Spojení Linq2Sql a DTO

Třídy, které generuje Linq2Sql, budeme nazývat Data Access Objects (DAO). DAO nesplňují požadavky na DTO uvedené výše. Obsahují totiž funkce pro perzistenci – čtení, zápis

[1] DBML - Database Markup Language



Obrázek 6.1: Model datové vrstvy pro správu uživatelů

do databáze. Musím tedy ručně vytvořit objekty DTO, které již budou opravdu nést pouze proměnné a žádnou funkcionalitu. Toto mapování nemusím vytvářet jako 1:1, ale pokud to bude výhodné, využiji dědičnosti a sloučím některé tabulky do jediné třídy. Práce s nimi pak na business vrstvě bude jednotná. Každá DAO třída obsahuje konverzi na odpovídající třídu DTO. S DAO objekty se pracuje pouze interně na nejnižší vrstvě abstrakce – všechny vyšší vrstvy již pracují pouze s DTO objekty.

Práce s databází je v Linq2Sql řešena pomocí objektu třídy *DataContext*, přes jejíž proměnné lze přistupovat ke všem definovaným perzistentním objektům. Bohužel pro rozsáhlejší projekty je nutné mít možnost transakcí na úrovni business objektů, a tudíž musíme tento objekt zpřístupnit mimo knihovnu DAO.

6.2 Unity Application Block

Aktuální verze Unity Application Block 1.2 (Unity) byla vydána v říjnu 2008. Představuje odlehčený, rozšiřitelný, kontejner pro vkládání závislostí, který umožňuje vkládání objektů do konstruktorů, vlastností nebo volání metod. Podporuje abstrakci požadavků pro běh aplikace, které je možné specifikovat až při běhu a zjednodušit tak jejich správu. Tím, že je konfigurace komponent pozdržena do kontejneru, zvyšuje flexibilitu systému. Pro webové aplikace je přínosná možnost cachování kontejneru pro session, příp. pro celou aplikaci.

Následující příklad ukazuje registrování instancí a typů do UnityContaineru. Nejprve kontejner vytvoří, poté tuto vytvořenou instanci zaregistruje a následně zaregistruje typ DataContextu, který mají použít objekty DAO pro správu uživatelů. Jedná se pouze o registraci typu, takže při jeho použití je nutné vytvořit instanci této třídy. S registrací typu se zároveň konfigurují parametry – v tomto případě konstruktor s automatickým parametrem *connection* typu řetězec.

```
UnityContainer uc = new UnityContainer();
uc.RegisterInstance<IUnityContainer>(uc);
uc.RegisterType<UserManagementDataContext, UserManagementDataContext>
   (new ContainerControlledLifetimeManager()).Configure<InjectedMembers>().
   ConfigureInjectionFor<UserManagementDataContext>(new InjectionConstructor
   (new ResolvedParameter(typeof(string), "connection")));
```

Většina moderních aplikací je tvořena komponentním způsobem, který odděluje specifické a generické funkce, např. záznamy událostí, autorizace, správu výjimek. Díky pozdnímu provázání jednotlivých částí systému, je nejen lépe udržovatelný, ale lze jej v různých fázích vývoje lépe testovat. Hlavní výhodou je, že není nutné vkládat sdílený objekt (např. správce vyjímek) do všech objektů, které ho vyžadují, ale tyto objekty si jej mohou samy aktivně vyhledat.

6.3 Model, View, Presenter

Mnoho informačních systémů funguje tak, že načítá data z datového zdroje, které následně zobrazuje. Uživatel tato data upraví a následně je uloží zpět do datového zdroje. Mohlo by se zdát, že je proto výhodné propojit uživatelské rozhraní co možná nejvíce s databázovými operacemi, ať už kvůli zjednodušení kódu, zvýšení jeho čitelnosti, nebo zvýšení výkonu aplikace. Tento přirozený způsob ovšem naráží na fakt, že uživatelské rozhraní se mění podstatně častěji, než je tomu u databázové vrstvy [8]. V podnikových aplikacích se mezi

prací s databází a grafickou reprezentací u klienta objevuje ještě business vrstva, obsahující velké množství logiky, odrážející povahu podnikání a ne způsob práce se zdroji.

Změna uživatelského rozhraní je ještě více častá u webových aplikací, jelikož náklady na distribuci nové verze jsou nízké – není potřeba aktualizovat verze u všech klientů, ale stačí pouze jediná instance na serveru. Pokud by business logika byla úzce provázána s rozhraním, velmi pravděpodobně by se objevovalo větší množství chyb a bylo by ji nutné při každé jeho změně důkladně přetestovat. V každé aplikaci také najdeme stejná data, která se zobrazují různými způsoby – analytik potřebuje výstup do tabulky, aby mohl zjišťovat detailní závislosti, naproti tomu manažerovi postačí koláčový graf, který postihne pouze nejpodstatnější informace. Logika zpracování dat je pro oba pohledy stejná, jen výstup je jiný. Pokud by tedy byla pevně provázána s pohledem, musela by se duplikovat a v případných úpravách aplikace by jistě došlo k chybě, kdy se některý z pohledů zapomene upravit a budou se pak zobrazovat jiná (chybná) data.

Dalším důvodem pro oddělení zpracování dat od uživatelského rozhraní je jeho možná rozmanitost. Stejná data si můžeme představit v tlustém klientu pro systém Windows, jako pro mobilní zařízení typu PDA, případně dnes i mobilních telefonů, jako tenký klient na webu. Určitě by nebylo vhodné vyvíjet business logiku pro všechny tyto klientské aplikace zvlášť.

Tyto problémy řeší návrhový vzor, obecně známý jako *Model, View, Controller* (MVC). Ve webových aplikacích ovšem naráží na problém, kdy webová stránka zobrazená u klienta nedokáže sama reagovat na změnu dat na serveru. Řešením je rozhraní *Observer*, které sdružuje události a obsahuje mechanismy, jak oznámit změny v modelu ostatním částem bez toho, aniž by do nich vkládal závislosti na modelu, což je přesně to, čemu se MVC vzor chce vyhnout.

Z tohoto vychází *Model, View, Presenter* (MVP), kde Model představuje business vrstvu, View je čisté uživatelské rozhraní, řeší jeho události jako kliknutí myši apod. a Presenter reaguje na požadavky pohledu, volá metody business logiky a vrácená data formuluje do podoby očekávané pohledem.

Následující příklad ukazuje názorné použití MVP vzoru v tomto projektu:

```
public partial class EditPage : Microsoft.Practices.CompositeWeb.Web.UI.Page,
                                IEditPageView {
    private EditPageViewPresenter _presenter;
    protected void Page_Load(object sender, EventArgs e) {
        if (!IsPostBack) {
            _presenter.OnViewInitialized();
        }
        _presenter.OnViewLoaded();
    }

    [CreateNew]
    public EditPageViewPresenter Presenter {
        get { return _presenter; }
        set {
            _presenter = value;
            _presenter.View = this;
        }
    }
```

```
    ...
}
public class EditPageViewPresenter : Presenter<IEditPageView> {
    public IDashboardService Service { get; set; }
    public EditPageViewPresenter
           ([ServiceDependency] IDashboardService service) {
        Service = service;
    }
    ...
}
public interface IEditPageView { ... }
```

V příkladu si můžeme všimnout, že oproti běžné ASP.NET stránce, *EditPage* není odvozena od `System.Web.UI.Page`, ale od třídy v namespace *CompositeWeb*. Obdobná situace nastává s uživatelskými kontrolami. Dále musí implementovat rozhraní, přes které komunikuje s Presenterem. Ten je potomkem generické třídy, jejímž typem je právě toto rozhraní. Další zvláštností jsou atributy *CreateNew* a *ServiceDependency*, které zprostředkovávají pozdní vazbu implementovanou pomocí Unity Application Block, popisovanou výše.

6.4 Perzistence dashboardů

Každý uživatel má možnost přizpůsobit si jejich vzhled i obsah tak, aby vyhovoval právě jemu. To je základ pro spokojenost uživatelů, jejich dobrou orientaci v systému a tím pádem celkovou spokojenost se systémem, která povede k jeho hojnému využívání. Na druhou stranu jako vývojář musím vyřešit problém s novým uživatelem, který přijde poprvé k systému a chce okamžitě vidět nějaké výsledky. Není možné nutit jej do zdlouhavého zkoumání systému, vybírání vhodných ukazatelů, jejich rozmisťování na stránce a další nutné činnosti pro vytvoření vlastního dashboardu. Toto má být spíš třešnička na dortu pro zkušenější uživatele, kteří dokážou využít potenciál systému a vlastní aplikace tak není úzkým místem, překážkou, v jejich pracovní činnosti.

Systém musí tedy poskytovat i dashboardy, které jsou dostupné pro množinu uživatelů, v kontextu této aplikace a správy uživatelů, pro jednotlivé role. Abych znovu nevymýšlel kolo, podívám se na řešení tohoto velmi běžného problému samotným Microsoftem. V tomto případě není potřeba ani sahat po Enterprise Library, či podobných externích knihovnách, ale skrývá se uvnitř samotného frameworku.

Nazývá se *PersonalizationProvider*. Ten ovšem spolupracuje se standardním správcem uživatelů dostupným v ASP.NET a proto jej budu muset upravit pro práci s mnou definovanou strukturou uživatelů. Také data, která potřebuji uchovávat pro každý dashboard nelze serializovat do blobu standardním postupem. Schématem databázových tabulek jsem se inspiroval v oficiálním řešení, v databázi *AspNetDb*, která se vytvoří, pokud ve webovém projektu použijeme personalizaci, nebo správu uživatelů poskytovanou .NET Frameworkem.

Skládá se ze dvou tabulek, kde jedna obsahuje hromadnou parametrizaci a druhá pak parametrizaci pro uživatele. Obsahují název stránky, v mém případě je to cizí klíč do tabulky s definicemi dashboardů a sloupec pro samotná serializovaná data typu `image`, což je datový typ v T-SQL nesoucí binární data. Přestože se obě tabulky liší pouze v jednom sloupci, který nese ID uživatele, a zajisté by bylo možné je spojit do jedné a globální nastavení

reprezentovat NULL hodnotou v tomto sloupci, Microsoft se rozhodl pro řešení se dvěma tabulkami, tak jej budu respektovat.

Přístup k těmto personalizovaným datům z pohledu aplikace zajišťuje *DashboardPersonalizationProvider*, který upravuje standardní metody pro ukládání a načítání obsahu stránky. Značná část informací totiž nemusí být obsažena v blobu, protože je dostupná v relační podobě v ostatních tabulkách.

Nevýhodou blobu je jeho těžká modifikovatelnost a nemožnost jakkoli vyhledávat relevantní údaje. Pokud požadujeme administraci dashboardů administrátorem, musíme mu umožnit přístup k jejich datům a pouhé zachování, nebo zahození již neplatné uživatelské konfigurace by jistě nebylo dostatečným řešením. Klasickým příkladem je chyba v některém z reportů a nutnost opravit SQL dotaz, kterého využívá. V prvním případě by bylo nutné data všech dashboardů obsahujících tento ukazatel vymazat. Řešení, které implementuji, ponechává v blobu pouze název třídy, která představuje report a parametry tohoto reportu. Vlastní definice reportu se pak při požadavku na zobrazení dashboardu načte z databáze a sestaví tak kompletní stránku.

Pokud uživatelská definice daného dashboardu neexistuje, je použita globální. Uživatelská se nevytváří při prvním dotazu konkrétního uživatele na stránku, ale až při její modifikaci tímto uživatelem.

6.5 Web Parts

Web Party nejsou v .NET Frameworku žádnou novinkou, poprvé se objevily již ve verzi 2.0. Obsahují funkce, které usnadňují uživatelům práci s webem a vývojářům, kteří nemusí tyto funkce psát sami. Tyto funkce ovšem pro svůj běh vyžadují velké množství JavaScriptu na pozadí webové stránky a ten je problémem napříč webovými prohlížeči. Přesto jsem zvolil tuto variantu pro její možné budoucí propojení s Sharepointem a provázaností s ostatními částmi systému, např. personalizace.

WebPartManager v sobě zahrnuje několik režimů, ve kterých je možno zobrazit webovou stránku a mezi kterými je možné přecházet. Je také možné si další stavy doplnit, ale tuto možnost nepotřebuji. V tomto projektu si vystačím s režimem *browse*, který představuje klasické zobrazení pro uživatele, pouze pro čtení. Dalším je *catalog*, ve kterém se zobrazí seznam dostupných komponent (KPI ukazatelů) a ty je možné přidávat a odebírat ze stránky nebo jakkoli po stránce přemisťovat. Posledním důležitým módem je *edit*, ve kterém se kromě základních parametrů ovlivňujících grafickou podobu komponenty zobrazí také parametry daného reportu, které je možné upravovat a aktivně tak měnit obsah ukazatele.

6.5.1 Template

Katalog web part je umí přiřazovat pouze do předem definovaných zón. Aby byla zajištěna variabilita dashbordů, je nutné měnit pozice těchto zón. Nejpřímějším řešením by bylo vytvořit pro každé schéma stránky speciální ASPX stránku, která by toto schéma jasně definovala. Ovšem při výběru dashboardu z menu, nebo při ukládání jeho personalizované verze do databáze narazíme na problémy. Taktéž znovupoužitelnost je v tomto případě nízká, jelikož každá stránka obsahuje část informací stejnou. Způsob, který jsem použil, oba tyto problémy eliminuje. Do stránky jsem vložil user control, který podle typu dashboardu rozhodne, kterou instanci třídy (rozložení zón) vyrenderuje do HTML kódu. Drobnou nevýhodou je, že podstatná část kódu stránky se skrývá v code-behindu a tudíž je

ztížena orientace v tom, co vlastně odchází klientovi a například stylování pomocí CSS se dělá obtížně.

Zóna nemusí obsahovat pouze jednu komponentu, jeden report. Další komponenty se řadí pod sebe. Pro mé účely tedy stačí definovat několik málo šablon, každou s jiným počtem sloupců. Zatím nejpraktičtěji se jeví šablona o 3 sloupcích, a proto ji v nabídkách zobrazuji jako default.

6.6 Editace reportů

Report je atomická položka v systému z pohledu uživatele. Ten již jeho editaci nemůže provádět sám, ale musí se obrátit na admistrátora dashboardů, který má znalosti o vnitřní struktuře systému, o databázových schématech a o vazbách daných business logikou. Pro manažera je report produktem, který mu tento systém nabízí. Má možnost si jej lehce upravit, ale pouze předem definovanými parametry, které specifikoval administrátor při jeho vytváření.

U reportů tedy budou dva typy administrace – nastavování parametrů, které se provádí změnou pohledu na dashboard do režimu edit a výběrem příslušné komponenty (web party), a celková správa dostupná pouze administrátorům s odbornějšími znalostmi. Tento druhý režim musí umožňovat pojmenování reportu, zvolení grafické komponenty, která jej bude reprezentovat, výběr databázového spojení – systém je vytvořen obecně, dokáže pracovat s velkým množstvím datových zdrojů, není omezen pouze na jednu databázi, či datový sklad. Typy podporovaných spojení lze podle potřeb libovolně rozšířit. V současné době aplikace podporuje práci s SQL Serverem společnosti Microsoft, SQL Server Analysis Services a OleDb pro obecný přístup k datům. Vlastní dotaz pak musí být zkonstruován pro zvolený typ spojení. Během definice reportu je možné tento dotaz v systému odzkoušet, zda je platný. Parametry, které jsou vyčíslitelné v době tohoto testu, se převedou na správné hodnoty, za ostatní se dosadí NULL. Pokud vyčíslení dotazu selže, uživatel obdrží obsah výjimky typu `SqlException`, která mu musí být dostatečným vodítkem pro korekci chyby. Žádné vlastní parsování a kontrolu dotazu aplikace neprovádí, jelikož by se v žádném případě nemohla dostat kvalitativně na úroveň samotného systému řízení báze dat v SQL Serveru.

Kontrola se provádí pouze syntaktická dotazem na schéma prováděného příkazu, aby testováním dotazů při vytváření nebyl zbytečně vytěžován server a také relevantnost získaných údajů by byla stejně velmi malá, jelikož některé parametry mohou obsahovat nedefinované hodnoty.

S definicí dotazu úzce souvisí parametry, se kterými tento dotaz bude pracovat. Každý parametr musí mít název, což je text zobrazovaný uživateli při jeho editaci, klíč, který se použije při sestavování dotazu a datový typ tohoto parametru. Tyto hodnoty jsou povinné, jako volitelnou pak lze zadat hodnotu tohoto parametru. Tato bude použita, pokud si uživatel hodnotu sám nezmění. Jako hodnotu nelze použít výrazy, jelikož by mohlo dojít snadno k SQL injection útoku na systém a proto jsou povoleny pouze konstanty. Pokud uživatel vybere jako datový typ `UserDefined`, má možnost použít jednu ze systémových konstant. Ze zkušenosti z obdobného systému vím, že pro potřeby reportingu je dostačující znát identifikátor přihlášeného uživatele. Z něho pak lze vše odvodit: příslušnost k organizační jednotce, teritoriální strukturu, promované produkty, nákladová střediska atd.

V mém případě musím přidat ještě parametry, které budou sloužit pro drill-down. Budou jimi organizační jednotka, okres a časem přibude i perioda, časová jednotka, za kterou je daný report kalkulován. V současné době ale primární zdroje dat neobsahují tuto dimenzi jednotnou, vývojový tým se shoduje, že je to velká slabina celého systému Eluzzion a již

se začalo s analýzou možných řešení. Proto počkám s její implementací až do fáze, kdy bude zřejmé, jakým směrem se klíčový systém zdrojových dat bude ubírat. Pro nastínění problému jenom uvedu, že každý modul tohoto, v současné době hodně rozsáhlého, systému obsahuje vlastní pojetí period a jejich provázanosti. Například prodejní data fungují po týdnech, které ale nemusí nutně korespondovat s kalendářními, peněžní deník zas pracuje v každé zemi jinak podle místní legislativy, taktéž prodejní kampaně se liší podle jednotlivých organizačních jednotek a podle toho, který produkt propaguje.

Dalším nezbytným krokem je nastavení práv pro vytvořený report. Nejen specifikování skupin, které jej mohou editovat, ale také výběr manažerů / uživatelů, kteří budou mít možnost přidat si report do vlastních dashboardů, příp. mu upravovat parametry. Funkcionalita i vzhled této komponenty je totožný s editací práv pro celé dashboardy. V případě, že nedefinujeme roli, která může report prohlížet, neumožníme uživatelům jeho přidání do dashboardu a tudíž naše práce při jeho vytváření bude nevyužitá. Jelikož si reporty s sebou nenesou informaci o tom, kdo je vytvořil, hrozí v případě, že nedefinujeme žádného administrátora, vytvoření "zombie", která bude ležet v systému, bez možnosti ji nějak změnit.

Nabízí se celkem jednoduché řešení a to vytvořením ve správě uživatelů tzv. uživatelské role, kde každému uživateli bude příslušet jedna role tohoto typu a to pouze jemu. Pro zkrácení času do nasazení systému na produkční prostředí, nebudu tuto funkci implementovat, povedu v patrnosti toto možné riziko a po nějaké době běhu zjistím, jaké množství těchto zombie v systému vzniklo a pokud bude velké, provedu její implementaci.

6.7 Grafické komponenty

Klíčovou komponentou celého systému je samotný ukazatel. Primitivní grafická komponenta, neobsahující žádnou logiku, jen reprezentující data, kvůli kterým budou uživatelé používat tento systém. Z dostupných komponent byly vybrány ty od firmy Infragistics, nacházejících se v produktu nazvaném NetAdvantage.

Z obrovského balíku komponent Infragistics budu potřebovat jenom jedinou: *UltraWebGauge*. Tato třída nepředstavuje jeden konkrétní ukazatel, ale je spíše nástrojem, jak takový ukazatel definovat a ona, podle této definice, zajistí jeho správné vykreslení. Způsob vykreslování zaručuje 100% použitelnost ve všech (grafických) webových prohlížečích. Na serveru se totiž vygeneruje obrázek, reprezentující definovaný ukazatel a ke klientovi se přenese stejně jako například logo firmy v záhlaví stránky. Nevýhodou tohoto přístupu je, že do zvoleného adresáře na serveru generuje tyto dočasné soubory. Standardně jsou umístěny ve složce s webovou aplikací, je však možné její umístění změnit, například do systémového adresáře temp. Nemusíme pak řešit množící se počet souborů a jejich mazání – tuto činnost obstará operační systém sám při běžné údržbě.

Společnost Infragistics má na svém webu 101 ukázkových budíků a teploměrů, včetně jejich definičních XML souborů. Tyto definice se dají přímo použít v kódu ASPX stránky, bohužel pro kontroly, které neobsahují žádný HTML kód, ale jsou reprezentovány pouze třídou, je využít nelze. Je nutno převést XML definici do standardního kódu v C#. Jako všechny komponenty v tomto balíku, i UltraWebGauge obsahuje obrovské množství parametrů k nastavení. Bez takto kvalitních příkladů, by bylo takřka nemožné vytvořit nějakou dobře vypadající komponentu.

V těchto příkladech jsem čerpal inspiraci a vzorové budíky a teploměry, které aplikace v současné době obsahuje, jsou hlavně na demonstraci možností systému. Pro reálné prostředí bude nutná konzultace se zákazníkem, který si navrhne svoje ukazatele. Samozřejmě se nečeká, že by přišel s kompletním grafickým návrhem, spíše drobné úpravy ve stá-

vajících, jako např. přidání druhého indikátoru do stejné komponenty, čímž se rozšíří na ukazatel rozsahu typu min-max, nebo vložením loga své firmy dovnitř komponenty.

Každý jeden ukazatel je reprezentován svojí třídou, kterou administrátor dashboardů musí zvolit při definici KPI. Podle jejího výběru, pak musí přizpůsobit i SQL či MDX dotaz generující jeho obsah. Většina ukazatelů zobrazuje pouze jedinou hodnotu, lze tedy použít skalární databázový dotaz. Pro ostatní se využije DataReader, který je schopen vrátit více hodnot. Jelikož všechny tyto hodnoty jsou očekávány v jediném řádku v různých sloupcích, je možné dovolit si zrychlit dotaz specifikací exekučního parametru `CommandBehaviour`, `SingleRow`, díky němuž lze dosáhnout optimalizace prováděného dotazu.

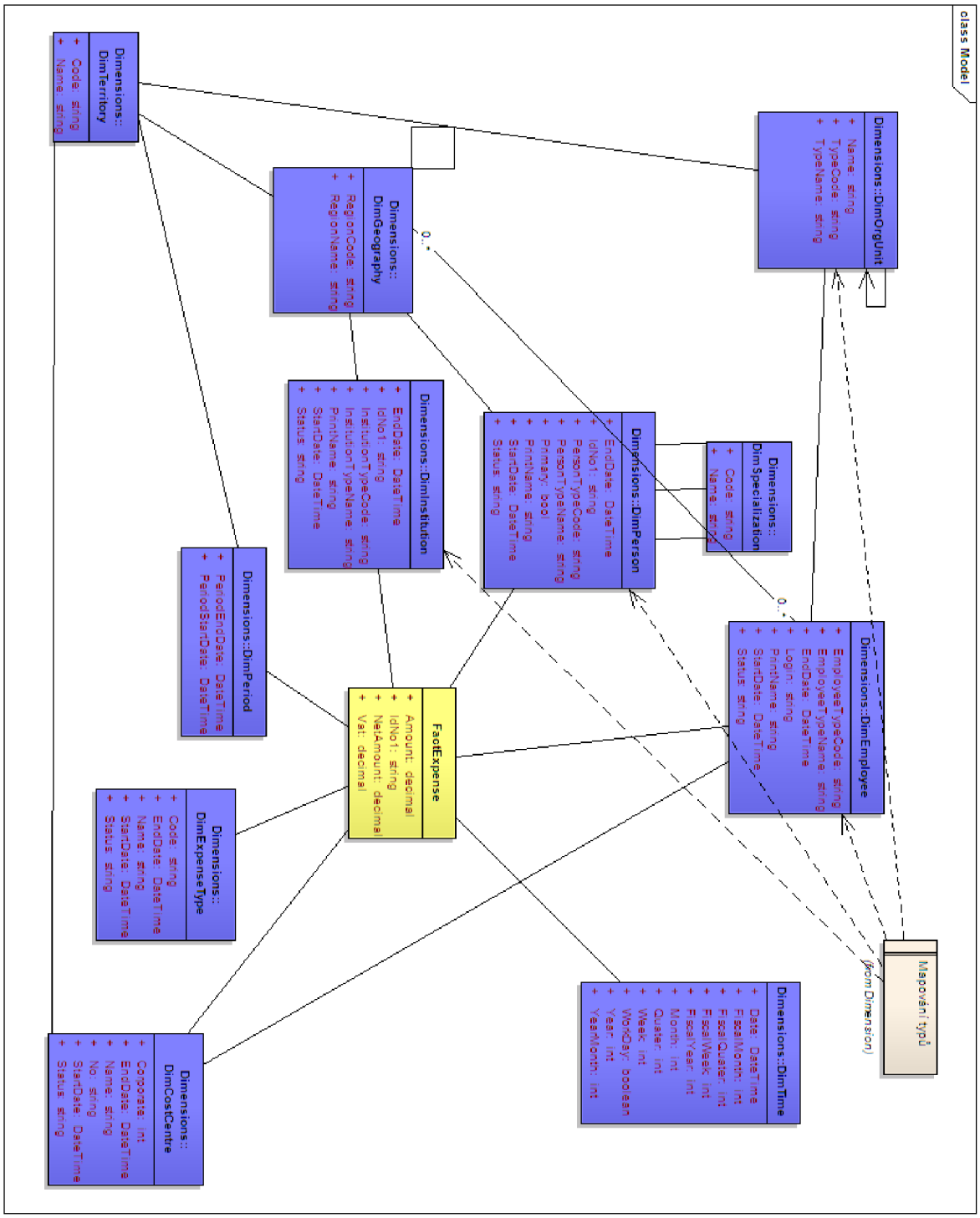
6.8 Datové kostky

Výběr vhodných dat je taktéž nedílnou součástí systému. Toto rozhodnutí je ryze manažerské a s vývojem vlastní aplikace příliš nesouvisí, resp. vývoj systému by neměl být závislý na volbě dat. Datové kostky, které v této sekci popíši, jsou vytvořeny z transakční databáze CRM systému Eluzzion, jelikož tento zdroj, jakožto dominující produkt společnosti, bude jednotný pro všechny klienty tohoto analytického systému. Ostatní zdroje již závisí od velikosti firmy a jejích dalších aktivit, jestli jsou například informace o produktech získávány ze SAPu, či jiného korporátního systému, zda bude systém propojen s informacemi z logistiky, nebo jaké externí zdroje informací si daná firma, daný zákazník, kupuje.

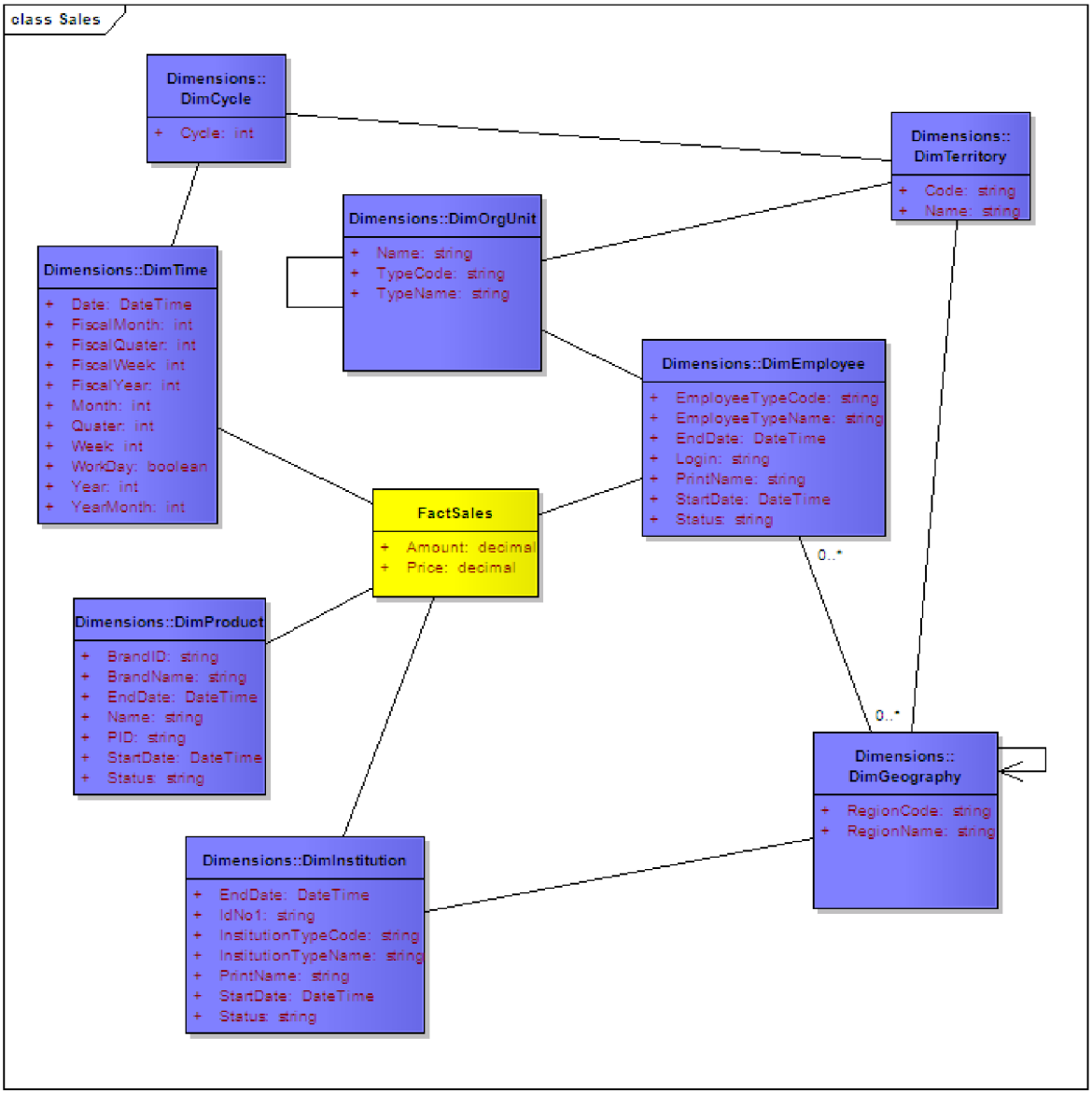
Systém Eluzzion je modulární a každý klient si může vybrat moduly, které bude používat, které si koupí. Z tohoto důvodu jsem zvolil možnost několika menších kostek, než jedné velké centrální. Jelikož jsem si na tomto projektu ověřil, že navrhovat strukturu datového skladu vyžaduje nemalé zkušenosti, bylo pro mě přijatelnější, začít s kostkami méně komplexními. Kostka, která bude agregovat veškeré informace, je nezbytná pro možnost dotazů napříč moduly, např. provázanost výdajů a počtu aktivit, nebo návaznost na prodejní data.

První z uvedených kostek na obrázku [6.2] obsahuje informace o výdajích spojených s činností reprezentantů, převážně prezentací produktů. Jednotlivé výdaje jsou rozdělené do nákladových středisek (*CostCentre*). Každý výdaj je svázán s reprezentantem, obecně zaměstnancem (*Employee*), který jej vykázal. Kromě servisních položek, jako je třeba poplatek za parkování, je většina výdajů vázána na lékaře (*Person*), příp. lékárnu, instituci (*Institution*), se kterou se pojí. Je tak možné zjišťovat průměrnou cenu jedné návštěvy u lékaře, a snažit se ji optimalizovat. Poslední klíčovou informací o výdaji je čas; datum, kdy byl výdaj uskutečněn (*Time*). Jelikož z účetního hlediska je nutné výdaje vést v peněžním deníku, který se periodicky uzavírá a kontroluje účetní, je záznam také vázán na tuto periodu (*Period*). Jelikož je toto období dáno zákony jednotlivých zemí, obsahuje datová kostka vazbu na toto správní území (*Territory*).

Druhou kostkou na schématu [6.3] jsou výsledky prodejů. Na rozdíl od předchozí kostky, kam data přibývají denně, jelikož zdrojem je vnitrofiremní systém, prodejní data jsou kupována od lékáren a v současné době se tak děje na měsíční bázi a proto je po stránce údržby podstatně méně náročná, ETL proces je možné spouštět pouze jednou za měsíc a šetřit tak systémové prostředky na serveru. Jádrem kostky je tabulka faktů, která je, stejně jako u předchozí kostky, zvýrazněna žlutou barvou. Všechny dimenze mají barvu fialovou. Stručně řečeno, tabulka faktů obsahuje informace kdy, kde, co, kolik a za kolik se prodalo. Tyto informace zprostředkovávají dimenze produktů (*Products*), lékáren (*Institution*) a času (*Time*). Instituce se nachází v některém z okresů, z čehož je možné zjistit, který medicínský reprezentant působí v dané oblasti a přiřadit prodeje k jeho osobě (*Employee*) a následně hodnotit jeho dosažené výsledky.



Obrázek 6.2: Schéma kostky pro modul Peněžní deník



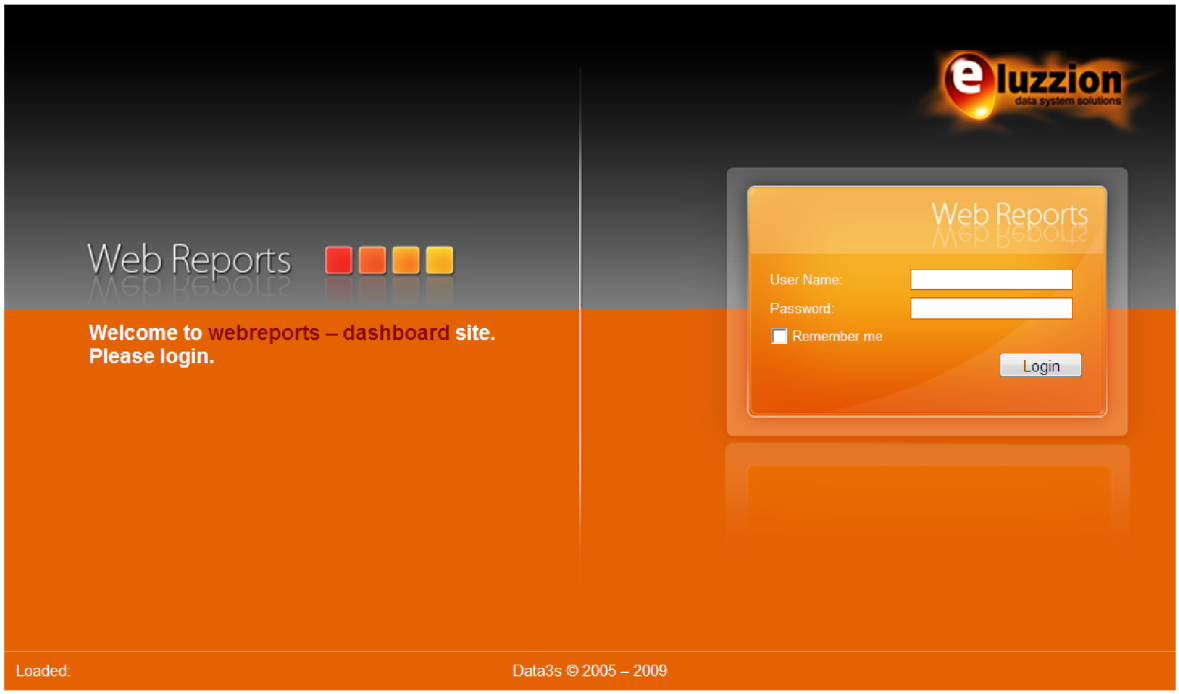
Obrázek 6.3: Schéma kostky pro modul Prodejní data

Kapitola 7

Uživatelská příručka

Klasická uživatelská příručka se jistě do této práce nehodí a svým rozsahem by ji určitě přesahovala. Uvedu tady proto jen základní předpokládané schéma činnosti manažera. Cílem je spíše jen demonstrovat faktickou realizaci projektu, jednoduchost a intuitivnost ovládání, než naučit potenciálního uživatele plnohodnotně tento systém ovládat.

První s čím se uživatel setká snad u každého systému, je přihlašovací dialog. Na obrázku [7.1] můžeme vidět vstupní obrazovku mé aplikace. Její černo-oranžová barevná kombinace vychází ze stylu ostatních aplikací skupiny Eluzzion, který koresponduje s firemními barvami. Jelikož jeden z větších klientů využívá pro Eluzzion svůj proprietární název i grafiku, není toto grafické téma fixní, ale je možné změnou jednoho parametru v nastavení aplikace mezi nimi přepínat. V současné době jsou dostupné právě tyto dvě varianty.

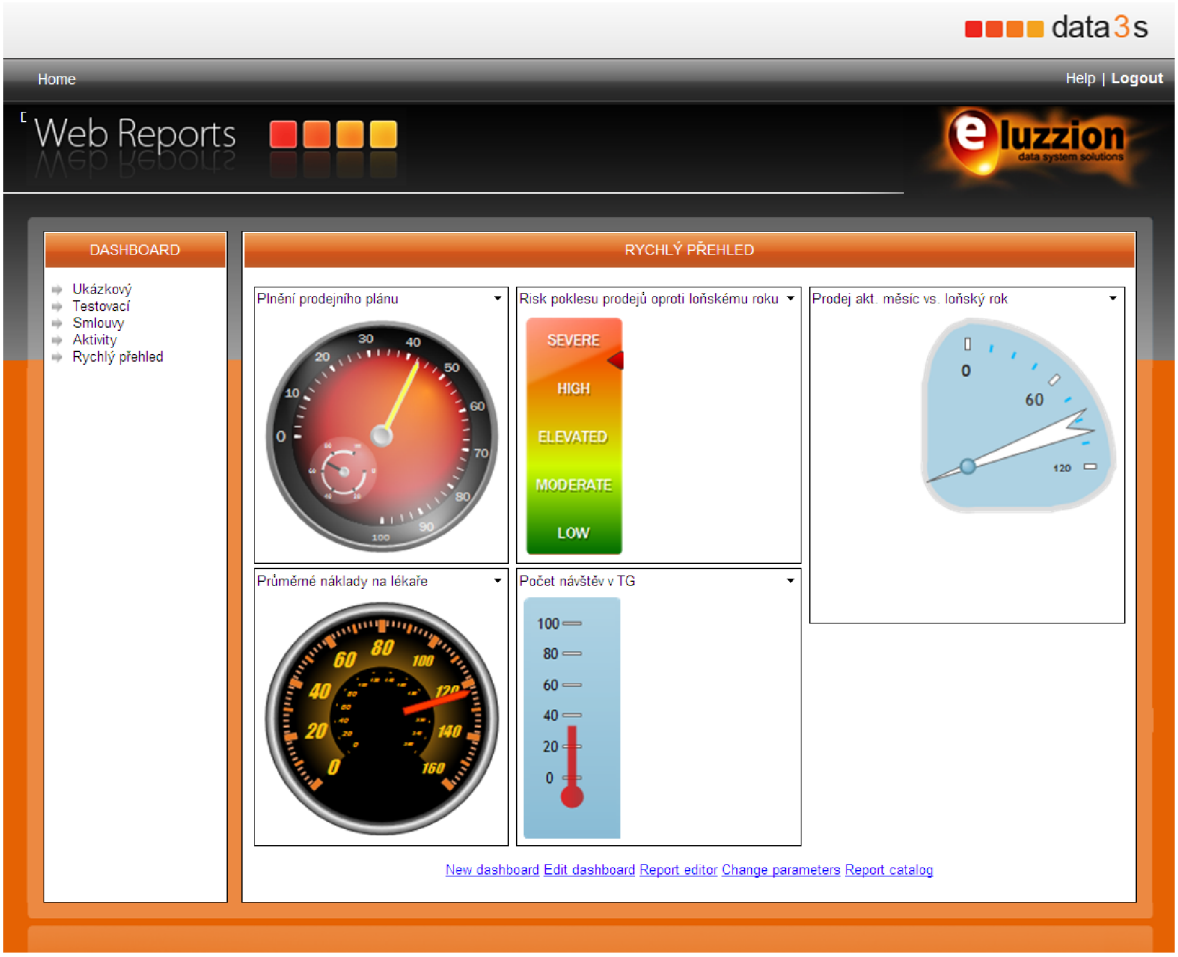


Obrázek 7.1: Přihlašovací stránka do systému

Po úspěšném přihlášení se zobrazí uvítací obrazovka systému, nikoli dashboardů. Pro vstup do tohoto modulu je nezbytné v záhlaví aplikace zvolit Dashboardy. V současné době

jsou zde pouze dvě volby – Home a Dashboard, ale do budoucna je počítáno s integrací dalších aplikací, jako jsou standardní reporty, e-Learning, Eluzzion web client apod. Teprve po vstupu do modulu dashboardů se zobrazí uvítací stránka pro dashboardy. Původně jsem zvažoval automatické zobrazení prvního nabízeného dashboardu, ale kvůli obavám z možného poklesu výkonu pro načtení úvodní stránky – počítaly by se totiž hned všechny ukazatele, a v případě, že uživatel chce zobrazit jiný report, představovalo by to pro něj velké zdržení. Proto i tento modul obsahuje svoji statickou úvodní stránku, která však již na levém boku obsahuje seznam dostupných dashboardů, které může uživatel vybrat podle svého uvážení.

Zvolením dashboardu z postranního menu dojde k vygenerování stránky, podle definice tohoto dashboardu a jeho personalizovaných informací. Příklad výsledku můžeme vidět na dalším obrázku [7.2]. Tento konkrétní dashboard obsahuje tři zóny a tudíž je možné v něm vidět tři ukazatele na řádku. Běžný uživatel bez možnosti jakkoli tento dashboard modifikovat již nevidí spodní menu nabízené pod dashboardem a jeho práce s tímto dashboardem skončila. Může samozřejmě procházet na detaily jednotlivých ukazatelů pomocí funkce drill-down.

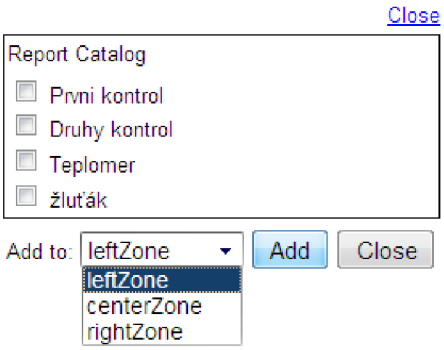


Obrázek 7.2: Vzorový dashboard

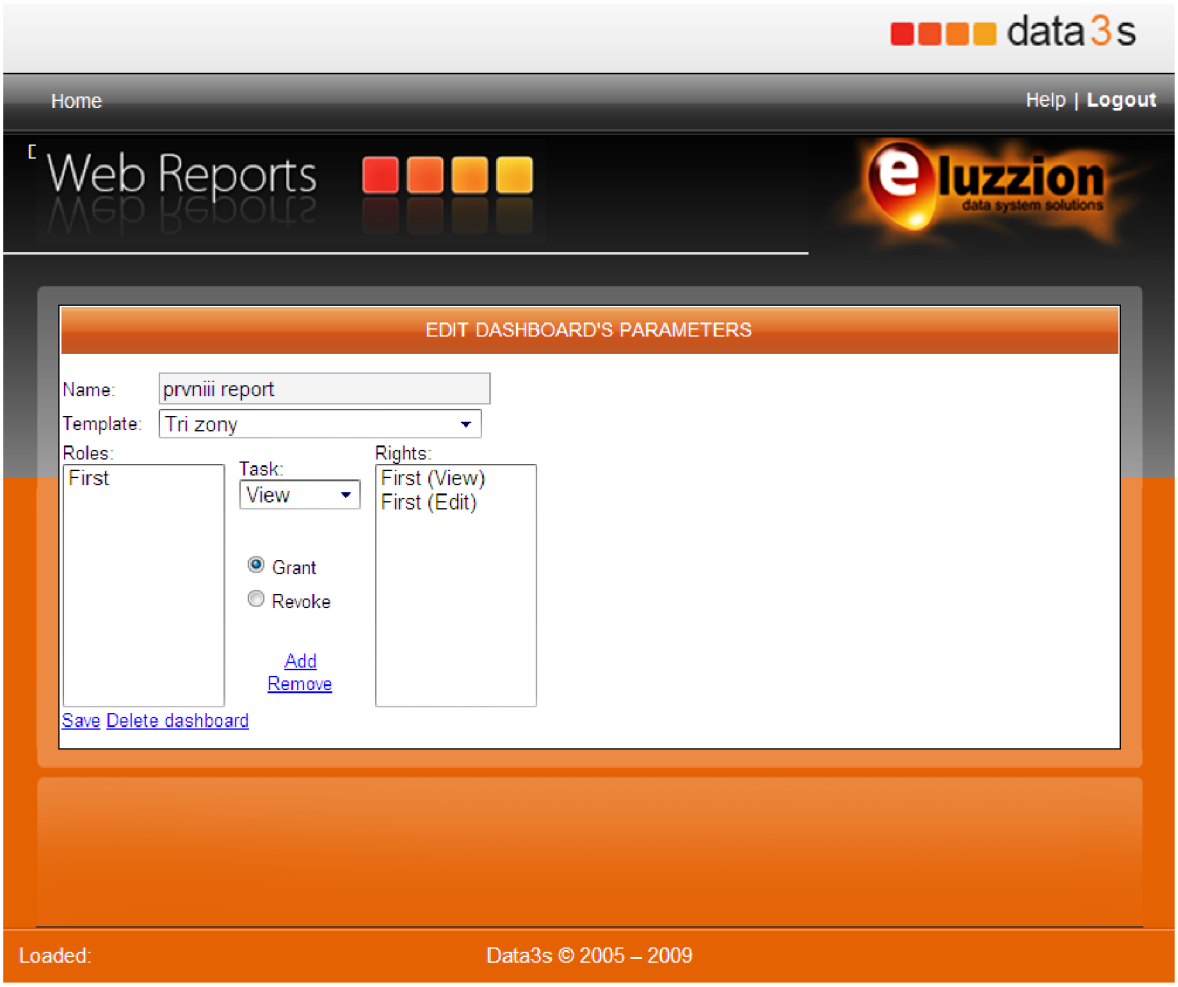
Na závěr se podíváme na editační stránku dashboardu, která je dostupná pokročilejším uživatelům s vyššími oprávněními. Je zachycena na obrázku [7.4]. Jak jsem uvedl dříve,

je možno měnit název zobrazovaný v bočním menu, šablonu, která obsahuje jednotlivé zóny pro vlastní reporty a uživatelská práva. Po výběru jedné či několika rolí v levém seznamu, zvolení požadované činnosti (úlohy) a výběru, zda bude oprávnění přiznáno či odmítnuto, můžeme pravidlo přidat do pravého seznamu. Stejně tak výběrem jednoho nebo několika pravidel z pravého seznamu a následným kliknutím na tlačítko Remove, lze tato pravidla odebrat. Tyto změny v uživatelských oprávněních se provádějí okamžitě, není nutné potvrzovat je tlačítkem Save na stránce.

Editace dashboardu pokračuje již na stránce jeho samotného, přepnutím do režimu katalog, ve kterém si zvolíme komponenty (reporty), které bude zobrazovat. Ukázka komponenty katalogu je na obrázku [7.3]. Je možné zaškrtnout jeden nebo několik reportů, zvolit zónu, do které se mají přidat a stisknout tlačítko Add. Odebírání reportu se děje v kontextovém menu každé jedné web party volbou Close.



Obrázek 7.3: Výběr reportů z katalogu a přidávání do zón stránky



Obrázek 7.4: Editace dashboardu

Kapitola 8

Závěr

Diplomová práce je nejrozsáhlejším projektem, který jsem měl možnost realizovat za celou dobu svého studia. Mohu směle prohlásit, že tento projekt je důstojným zakončením mého pětiletého studia na Fakultě informačních technologií. Zhodnotil jsem v něm znalosti ze všech předmětů magisterského studia, snad kromě těch z oblasti hardware.

Velkou motivací pro mě také byla možnost hovořit s lidmi z reálného prostředí, se svými budoucími zákazníky, ověřit si schopnost komunikace architekt / vývojář a manažer, neznalý technických možností, hovořící jiným jazykem, který je zaměřen pouze na výstup, na přínos systému pro něj samotného. Tuto jedinou věc, tak potřebnou pro tento systém mi vysoká škola nedala a musel jsem se jí naučit přímo na ostrém případu, žádném školním modelovém příkladu. Až delší zkušenosti uživatelů se systémem a jejich spokojenost ukážou, nakolik jsem porozuměl jejich požadavkům.

Celé řešení systému jsem pojal jako výbornou možnost seznámit se s novými technologiemi pro platformu .Net a Microsoft SQL Server. Snažil jsem se použít doporučované postupy a návrhové vzory pro rozsáhlé aplikace, i když v některých případech jde možná částečně o příliš robustní řešení. Za velký přínos považuji seznámení se s Enterprise Library, Unity Application Block a návrhovým vzorem Model, View, Presenter. Jelikož moje předchozí projekty byly vždy ve starších verzích .Net Frameworku, taktéž samotný Linq byl pro mě v tomto projektu novinkou.

Z důvodů ekonomické krize se projekt potýkal s problémy ze strany financování a obavy o budoucí zákazníky. Původní plán hovořil o 4 vývojářích a s tímto faktem byla i tato diplomová práce zadávána. Bohužel nakonec jsem byl nucen projekt řešit sám a začátek projektu byl také o půl roku odložen, takže řešení je funkční, splňuje požadavky zadání, ale taktéž vím, že v systému jsou rezervy a bude potřeba ve vývoji i nadále pokračovat. Snažit se tento produkt ještě více přizpůsobit potřebám zákazníků, rozšířit paletu dostupných grafických komponent, zapracovat na pohodlné práci i pro administrátory, zajistit jim větší možnosti monitorování a rozšířit parametrizaci systému, aby se co možná redukovala závislost na vývojářích, což je jedna z klíčových strategií firmy Data System Solutions.

Systém se nyní nachází ve fázi, kdy je možné jej hrdě prezentovat zákazníkům, umožňuje prohlížení dashboardů i jejich editaci, obsahuje prostředí pro definici reportů. Správa uživatelů byla definována s ohledem na ostatní systémy, a proto nebyla ve výsledku zahrnuta do tohoto systému, ale je součástí administrativního smart-klienta nazvaného AdminTool. Z plánovaného rozfázování vývoje jsou hotové první dvě a ze třetí chybí dodělat monitorování činnosti uživatelů v systému a exportování dashboardů do externích formátů, jako je Excel nebo PDF. Ani jednu z těchto chybějících funkcí nepovažuji za kritickou. Monitoring by poskytoval relevantní informace až po nějaké době používání systému, kdy si

uživatelé najdou svoje zaběhnuté schéma vykonávaných činností a nebudou pouze zkoušet, co systém zvládá. Export lze, i bez speciální funkce, provést PrintScreenem, či převod do PDF pak tiskem na některou z tiskáren typu PDFFactory apod.

Celkově musím tento projekt hodnotit velmi pozitivně. Nejen z hlediska nabytých znalostí a zkušeností, ale také proto, že se jedná o reálný systém, který dojde svého použití v praxi a mé úsilí nepřijde vniveč, ale bude odměněno množstvím spokojených uživatelů. Teoretická část také zevrubně popisuje základní praktiky a vzory společnosti Microsoft, a jelikož její vývojářské webové stránky jsou dostupné pouze v angličtině, umožní i méně zdatným studentům čerpat mé poznatky pro svoje vlastní projekty.

Literatura

[1] Microsoft patterns & practices Developer Center.
http://msdn.microsoft.com/en-us/library/ms978717.aspx, cit. leden 2009.

[2] Business Application Research Center: OLAP market share analysis.
http://olapreport.com/market.htm, cit. prosinec 2008.

[3] IBM Cognos 8 BI.
http://www.cognos.com/products/cognos8businessintelligence/index.html,
cit. prosinec 2008.

[4] MSDN SQL Server Developer Center.
http://msdn.microsoft.com/en-us/library/cc645993.aspx, cit. prosinec 2008.

[5] Oracle Business Intelligence Standard Edition One Whitepaper.
http://www.oracle.com/appserver/business-intelligence/docs/
oracle-bi-se1-whitepaper.pdf, cit. prosinec 2008.

[6] Dundas: Data Visualization.
http://www.dundas.com/Gallery/Gauge/NET/index.aspx?Img=Circular%20Gauge25,
cit. prosinec 2008.

[7] Lacko, L.: *Business Intelligence v SQL Serveru 2005*. Computer Press, 2006,
ISBN 80-251-1110-5.

[8] Library, M.: Model-View-Controller.
http://msdn.microsoft.com/en-us/library/ms978748.aspx, cit. květen 2009.

[9] Library, M.: Creating a Data Warehouse.
http://msdn.microsoft.com/en-us/library/aa905991(SQL.80).aspx, cit. leden
2009.