VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

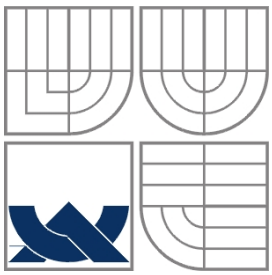
KLIENT PRO ZOBRAZOVÁNÍ OLAP KOSTEK

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

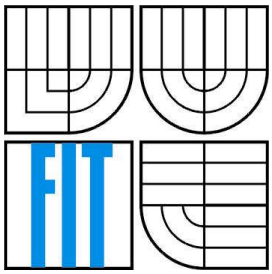
AUTOR PRÁCE
AUTHOR

BC. LUKÁŠ PODSEDNÍK

BRNO 2010



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. LUKÁŠ PODSEDNÍK

VEDOUCÍ PRÁCE
SUPERVISOR

ING. VLADIMÍR BARTÍK, PH.D.

BRNO 2010

Abstrakt

V úvodu práce popisuje základy a využití datových skladů a OLAP technik a operací využívaných nad datovými sklady. V návaznosti je popsán jeden z komerčních OLAP klientů. Na základě vlastností tohoto produktu je vytvořena analýza požadavků na klienta pro zobrazování OLAP kostek – je vybrána funkčnost, která bude v klientu implementována. Na základě analýzy požadavků je vypracován návrh struktury aplikace a jsou vytvořeny UML diagramy, které jsou součástí tohoto návrhu. Pro návrh se vybere nejlepší řešení ze srovnaných knihoven, frameworků a vývojových prostředí. Následuje kapitola o implementaci a nástrojích v ní použitých. V závěru se práce zabývá zhodnocením dosažených výsledků a dalšími možnostmi vylepšení.

Abstract

At the beginning, the project describes basics and utilization of data warehousing and OLAP techniques and operations used within the data warehouses. Then follows a description of one of the commercial OLAP client – based on the features of this product the requirement analysis of the freeware OLAP cube client displayer is desribed – choosing the functionality to be implemented in the client. Using the requirement analysis the structural design of the application (including UML diagrams) is made. The best solution from compared libraries, frameworks and development environments is chosen for the design. Next chapter is about implementation and tools and frameworks used in implemetation. At the end the thesis clasifies the reached results and options for further improvement.

Klíčová slova

Online Analytical Processing (OLAP), datový sklad, klient, datová kostka, Eclipse IDE, plug-in, MySQL, databáze, třívrstvá architektura, Spring Remoting.

Keywords

Online Analytical Processing (OLAP), data warehouse, client, data cube, Eclipse IDE, plug-in, MySQL, database, three-layer architecture, Spring Remoting.

Citace

Lukáš Podsedník: Klient pro zobrazování OLAP kostek, diplomová práce, Brno, FIT VUT v Brně, rok 2010

Klient pro zobrazování OLAP kostek

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Vladimíra Bartíka, Ph.D.
Další informace mi poskytli Doc. Ing. Jaroslav Zendulka CSc. a další v předmětech IDS, PDB, AIS a ZZN a v podpůrné literatuře k nim.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Lukáš Podsedník
26.5.2010

Poděkování

Děkuji Ing. Vladimíru Bartíkovi, Ph.D. za vedení a konzultace při zpracovávání této práce. Dále děkuji Doc. Ing. Jaroslavu Zendulkovi, CSc. a jeho kolegům za pedagogické vedení v množství kurzů, které mě na vypracování této práce připravily. Zvláštní poděkování také patří bývalým kolegům z firmy OKsystem s.r.o., s kterými pravidelně konzultuji problémy vývoje aplikací v jazyku Java.
V neposlední řadě děkuji své manželce Hance za podporu a povzbuzování během mé práce a studia.

Obsah

1 Úvod

V dnešní době jsou informační technologie pro podnikání velmi důležité. Společnosti investují do nových produktů tohoto oboru ve víře, že tím získají výhodu oproti konkurenci. Pokud chce firma pracovat úsporněji, efektivněji a rychleji, pravděpodobně investuje do nového hardware vybavení, nebo si objedná nový informační systém.

Co ale když chce podnikatelský subjekt vědět něco více, než co ví ostatní? K tomu obyčejný informační systém nestačí. Klasický OLTP systém zaměstnancům a zákazníkům pomůže se uložením, modifikací a znovuprohlížením dat. Je výhodný pro každodenní přidávání, modifikaci, prohlížení a mazání dat. Pokud chce podnikatel vědět více než ostatní, potřebuje analyzovat data (např. ze svého informačního systému) a na základě této analýzy provést úsudek o obchodních trendech či výsledcích. K tomu, aby to dokázal, vznikly datové sklady a OLAP technologie.

Cílem této diplomové práce je popsat výše uvedené technologie a navrhnout a implementovat software, který umožňuje zobrazování OLAP dat – klienta pro zobrazování OLAP kostek.

1.1 Obsah jednotlivých kapitol

V následující kapitole se budeme zabývat definicí výše uvedených pojmů (datové sklady a OLAP), řekneme si něco o jejich využití, o tom, jak jsou data v OLAP databázi uložena a jaké dotazy se nad nimi obvykle provádí.

V třetí kapitole objasníme, jakým způsobem se OLAP data zobrazují. Uvedu příklad jednoho komerčního OLAP klienta a popíši práci s ním. To poslouží jako základ pro další kapitolu – ta bude začínat analýzou požadavků na klienta pro zobrazování OLAP kostek. Poté tato kapitola obsahuje návrh struktury aplikace a popis možných prostředků, které by šly využít, jejich výhody a nevýhody.

Pátá kapitola se zabývá implementací OLAP klienta. V úvodu popisuje použitelné prostředky a komponenty k vývoji a dále popisuje vlastní implementaci a nasazení.

Poslední kapitolou je krátký závěr, který shrnuje poznatky získané zpracováním předchozích kapitol a přináší zhodnocení přínosu práce a možnost dalšího rozšíření implementovaného klienta.

2 Datové sklady a OLAP technologie

Podívejme se nyní blíže na technologie, které slouží k analýze dat. V následujících podkapitolách vysvětlíme jednotlivé pojmy a popíšeme základní principy, způsoby prezentace dat a jejich využití ke strategickým rozhodnutím.

2.1 Datové sklady

Chceme-li data nějakým způsobem třídit, musíme najít nějaká kritéria, podle kterých data do jednotlivých tříd rozdělujeme. Obvykle se jedná o atributy, které jednak považujeme za důležité a které také data z tabulek určitým způsobem spojují. K tomu nám velmi dobře poslouží multidimenzionální model, kde jednotlivé dimenze reprezentují zmíněné atributy. Datový sklad potom zobecňuje a sjednocuje data v multidimenzionálním prostoru [han06].

Datové sklady jsou úložiště dat nezávislá na běžných každodenních úložištích typického firemního informačního systému. Definice datového skladu se různí. Abychom zdůraznili vlastnosti datového skladu, použijeme tuto definici: Datový sklad je subjektově orientovaná, integrovaná, časově variantní a stálá kolekce dat, která poskytuje podporu v procesu strategického rozhodování managementu [inm02].

Každá společnost je orientovaná na jiné subjekty – např. zákazník, produkt, objednávka, dodavatel apod. Datový sklad je orientován na tyto subjekty místo každodenních operací, na které jsou orientovány OLTP systémy. Nejdůležitější vlastností je integrovanost. Znamená to, že data obvykle nezískáváme pouze z jednoho zdroje: zdrojů může být větší množství a datový sklad získáme integrováním těchto zdrojů do jednoho úložiště. To můžeme plnit jednak z OLTP databází, ale také ze souborů (TXT, CSV, XML), front zpráv (např. JMS) a dalších. Další vlastností je neměnnost. Data se do datového skladu se obvykle jednorázově načtou a potom se už nemění. Místo toho, když jsou načítána data do datového skladu, jsou načítána v tzv. snímcích (angl. snapshot) – ve statickém formátu. Když současně nastane změna, zapíše se nový snímek, při tom se však v datovém skladu uchová historie dat [inm02]. Poslední, časově variantní, vlastnost datového skladu znamená, že každý záznam je v datovém skladu označen časovou známkou, datem modifikace a podobně. Čas je veličina, která hraje v úložištích tohoto typu klíčovou roli.

2.2 Využití datových skladů

Mnoho organizací používá informace z datových skladů k podpoře rozhodnutí činit aktivity jako [han06]:

- Zvyšování zaměření na zákazníka, což zahrnuje analýzu nákupních zvyků (šablon) zákazníka (jako nákupní preference, čas nákupu, rozpočtové cykly a chuť utrácet.
- Přemísťování produktů a řízení portfolia produktů porovnáváním objemu prodejů podle kvartálu, podle roku nebo podle zeměpisných regionů tak, aby se jemně vyladila produkční strategie.
- Analýza operací s hledání zdrojů zisku.
- Řízení vztahů se zákazníky, provádění změn prostředí a vyčíslování aktiv společnosti.

Velký přínos datových skladů je také v samotné integraci heterogenních databází. Běžné společnosti mají typicky množství velmi rozsáhlých heterogenních databází, které je potřeba čas od času sjednotit do jednoho úložiště. Některé databázové systémy používají k integraci tzv. *query-driven approach*. Znamená to, že nad jednotlivými heterogenními databázemi je abstraktní vrstva, jíž zadáváme databázové dotazy, které jsou potom touto vrstvou překládány jednotlivým databázím a jejich sjednocený výsledek je nám potom zobrazen. Tento způsob je poměrně neefektivní pro časté dotazy (uvažme, že při provádění každého dotazu se tento musí překládat pro několik různých databází). Naproti tomu datové sklady používají tzv. *update-driven approach*. Nová data se jednorázově integrují, předzpracují a transformují do požadovaného formátu. Datový sklad sice neobsahuje aktuální data, ale čas potřebný k integraci se potom ušetří při zpracovávání dotazů uživatele. Navíc je v úložišti obsaženo množství dat z historie, s kterými se dá pracovat.

2.3 OLAP

Klasické relační databáze a OLTP (on-line transaction processing) se používají ke každodennímu běhu obchodu. Obvykle se do nich velmi často ukládají jednotlivé položky. Ve středu zájmu jsou instance jedné entity – např. objednávka, zboží, rezervace, položka apod. OLTP systémy jsou orientované na zákazníka. Naproti tomu OLAP (on-line analytical processing) systémy jsou orientovány na trh [pon01]. Tyto systémy se také někdy nazývají systémy pro podporu rozhodování (decision support systems) – nemají totiž za úkol zajišťovat každodenní běh obchodu, ale pouze umožňovat pozorování, jak se obchod vyvíjí, a potom pomoci přijímat strategická rozhodnutí, aby se obchod v nějakém smyslu vylepšil. Zatímco cílem OLTP je zpravidla dostat data do databáze, cílem OLAP dostat z databáze strategické informace.

V literatuře [pon01] jsou rozdíly mezi OLAP a OLTP shrnuty tabulkou (viz tabulka 2.1). OLAP systémy jsou zde nazývány informační (*informational*) a OLTP systémy jako operační (*operational*). Kvůli větší přesnosti se ale budeme raději držet původních názvů.

	OLTP	OLAP
Obsah dat	aktuální hodnoty	archivované, odvozené a sumarizované hodnoty
Struktura dat	optimalizované pro transakce	optimalizované pro komplexní dotazy
Frekvence přístupu	vysoká	střední až malá
Typ přístupu	čtení, změna, zápis	pouze čtení
Použití	predikovatelné, opakovatelné	Ad hoc, náhodné, heuristické
Čas odezvy	< sekundy	několik sekund až minut
Počet uživatelů	vysoký	relativně nízký

Tabulka 2.1: Srovnání OLTP a OLAP systémů. Převzato z [pon01].

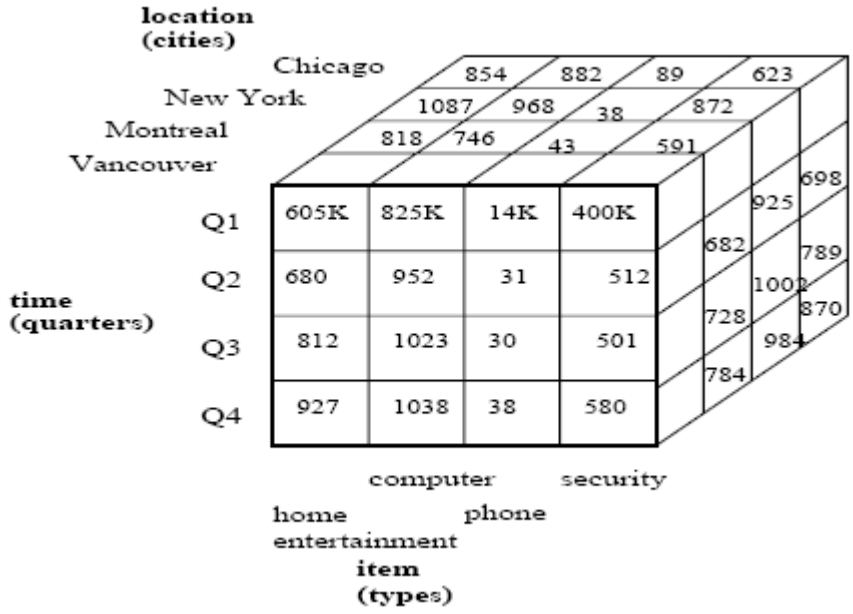
Podíváme-li se ještě jednou na tabulku 2.1, vidíme mezi těmito systémy množství rodzílů. Aby jedny operace nebrzdily druhé a aby mohl být datový sklad optimalizován na OLAP operace, je vhodné je oddělit. Hlavním důvodem je vysoký výkon obou systémů.

2.4 Multidimenzionální datový model

Tato kapitola popíše základní princip uložení dat v datovém skladu – multidimenzionální datový model. V příští kapitole potom popíšeme možnosti různých schémat uložení dat.

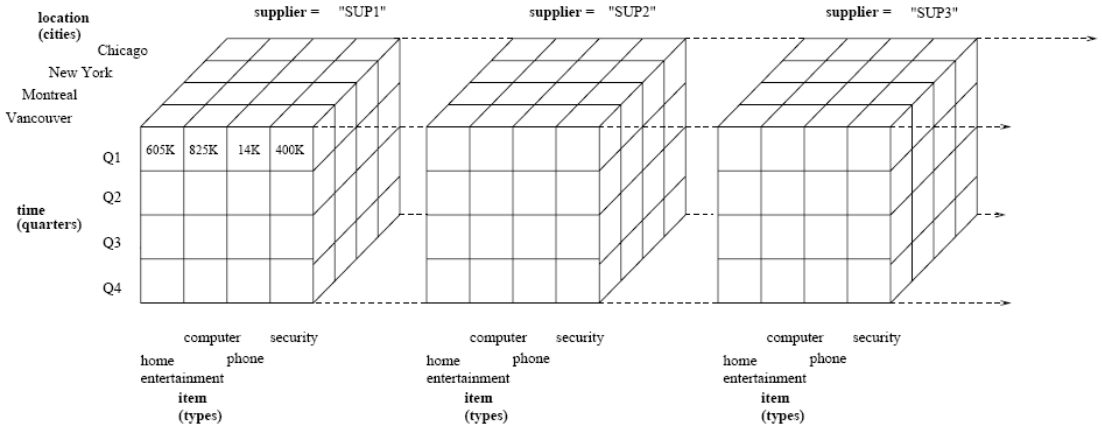
Obvykle se datový sklad zaměřuje na nějaké základní téma. Jak jsme si řekli v předchozí kapitole, OLAP systémy jsou zaměřené na trh. Základním tématem tedy může být prodej. Uvažme nyní atributy jako čas, místo prodeje a typ zboží. Dále existují obecně spojité veličiny, které se týkají prodeje, jedna z nich je např. částka, která byla na určitých místech za dané zboží v určitém čase utracena. Abychom uložili takovéto údaje, poslouží nám nejlépe multidimenzionální model.

Dimenzemi nazveme atributy jako místo prodeje, typ zboží a čas. Fakty nazveme spojité veličiny popsané v předchozím odstavci. V tomto modelu se snažíme najít souvislost mezi fakty a dimenzemi. Multidimenzionální datový model používá datovou kostku, která umožňuje zobrazovat data v několika (nikoli nutně třech) dimenzích a je definována právě tabulkou dimenzí a tabulkou faktů. Tento způsob uložení se užívá v relačních databázích, existují samozřejmě i jiné možnosti uložení.



Obrázek 2.1: Příklad datové kostky, převzato z [han06].

Příklad převzatý z literatury [han06] je popsán obrázkem 2.1 – uvažujeme dimenze čas, místo a položka (typ) a fakt objem prodeje v dolarech (tisících dolarů). Faktů může být samozřejmě více, přehlednější je ale zobrazit je jednotlivě. Datová kostka je také někdy nazývána kuboid. Máme-li více než tři dimenze, můžeme jí zakreslit pomocí sítě kuboidů. Výsledný obrázek potom vypadá jako datové kostky (nebo tabulky) ve vnější mřížce, která reprezentuje ostatní dimenze. Příklad čtyřdimenzionálního kuboidu můžeme vidět na obrázku 2.2. V případě použití kostky (3-D) a mřížky (2-D) lze tak ve zobrazení dosáhnout až pěti dimenzí. Kuboid, který obsahuje všechny dimenze na nejnižší úrovni sumarizace nazýváme základní kuboid (base cuboid). Naopak kuboid na nejvyšší úrovni sumarizace (0-D, tj. jeden bod) je nazýván nejvyšší kuboid (apex cuboid) – obsahuje jednu buňku, tedy sumarizovaný údaj o všech položkách.



Obrázek 2.2: Čtyřdimenzionální datová kostka reprezentující data o prodeji podle dimenzí čas, položka, lokalita a dodavatel. Převzato z [han06].

Existují i jiné možnosti pro reprezentaci datové kostky než krychle a tabulka (mřížka), které jsou popsány v dostupné literatuře [han06].

2.5 Způsoby uložení dat v datovém skladu

Pro uložení dat v datovém skladu existuje několik schémat. Všechna obsahují tabulky faktů a tabulky dimenzí. Omezíme se pouze na slovní popis, více k daným schématům lze vyhledat v literatuře [han06].

- **schéma hvězdy** – Na jednu tabulku faktů jsou navázány tabulky dimenzí. Tabulka faktů obsahuje jednak cizí klíče do tabulek dimenzí a jednak hodnoty faktů (již zmíněné obecně spojité veličiny). Tabulky dimenzí potom obsahují kromě primárního klíče atributy související s dimenzí. Např. tabulka faktů ProdejníMísto obsahuje atributy ulice, město, kraj, stát a kontinent. Podle těchto atributů je potom jednoduché provádět operace roll-up a drill-down, které budou zmíněné později. Podrobnější informace ke schématu hvězdy lze také nalézt v literatuře [pon01].
- **schéma sněhové vločky** – Toto schéma je velice podobné schématu hvězdy, liší se pouze normalizací tabulek dimenzí. Např. máme-li v jednom městě více poboček, je vhodné zavést tabulku měst, která bude obsahovat kromě primárního klíče atributy kraj a stát. Jinak by se tyto informace v tabulce faktů ProdejníMísto opakovaly. Avšak normalizujeme-li tabulky, klesá výkon při načítání dat a OLAP operacích, protože musí docházet ke spojení tabulek (join). Proto je vhodné zvolit kompromis a normalizovat pouze tabulky, které obsahují opravdu velké množství záznamů. Potom dojde k úspoře paměťového prostoru, která vyváží zmíněné zpomalení výkonu.
- **schéma souhvězdí** – Někdy je potřeba uložit více tabulek faktů. Zejména analyzujeme-li více témat zaráz (např. prodej a doručení). Výsledné schéma potom vypadá jako spojení schémat hvězdy, přičemž různé tabulky faktů mohou (a typicky mají) mít vazbu i na stejné tabulky dimenzí.

Tato schémata se používají při uložení dat v relační databázi. Pro uložení dat v jiných typech databází se používají jiné modely, kterými se ale nebudeme zabývat, protože s tvorbou OLAP klienta tohoto typu nesouvisí.

Míra v datové kostce je numerická funkce, která může být vyhodnocena v každém bodě datové kostky. Měrná hodnota je vypočítána pro daný bod agregací dat korespondující příslušným dimenzím definujícím daný bod [han06]. Tyto míry mohou být klasifikovány do třech kategorií podle toho, jaká agregační funkce je použita:

- **distributivní** – Tuto funkci lze spočítat distributivním výpočtem. Pokud uvážíme rozdělení dat do n souborů dat a pro každý soubor vypočteme touto distributivní funkcí měrnou hodnotu a spočítáme-li z těchto dílčích výsledků pomocí též funkce funkční

hodnotu pro všechna data, měla by dát stejný výsledek, jako pro agregaci všech položek naráz. Např. funkce count(), která spočítá počet položek v souboru dat, této specifikaci vyhovuje, protože součet počtů prvků v n souborech je roven počtu prvků v jejich sjednocení.

- **algebraická** – Jedná se o algebraickou funkci s M parametry (kde M < MAX; MAX je maximální počet argumentů této funkce), kterou lze vypočíst algebraickými operacemi. Každý z vstupních parametrů lze vypočíst distributivní agregační funkcí. Např. funkce aritmetického průměru avg() lze vypočíst jako sum()/count(), kde distributivní agregační funkce sum() počítá součet všech argumentů a distributivní agregační funkce count() počítá počet prvků v souboru dat.
- **holistická** – Tato funkce nelze vypočíst algebraickou funkcí s konstantním počtem argumentů. Typickým příkladem je medián nebo modus.

Na dimenze datové kostky se používá konceptuální hierarchie, která umožňuje podle dimenze zjemňovat (nebo naopak sjednocovat) body v datové kostce. Některé konceptuální hierarchie jsou zřejmé už z databáze – jak je tomu například u místa prodeje, kde lze seskupovat podle kontinentu, státu, kraje, města a ulice. Pokud budeme body krychle dělit v tomto pořadí, zjistíme, že dochází ke zjemňování. Tuto konceptuální hierarchii lze zakreslit do grafu, viz např. literatura [han06]. Některé konceptuální hierarchie můžeme získat diskretizací, resp. určením rozsahu hodnot (např. rozdělení zaměstnanců do platových tříd). Existují konceptuální hierarchie, na kterých není definována úplná relace uspořádání ve smyslu konceptuální hierarchie – typicky to bývá na časové dimenzi, pokud uvažujeme rozdělení dat na rok, čtvrtletí, měsíc, týden, den. Měsíc ani čtvrtletí nemůžeme na týden zjemnit, protože může nastat případ týdne, který spadá do dvou měsíců, potažmo čtvtletí.

2.6 OLAP operace

V této kapitole definujeme formálněji některé z operací popsaných v předchozí kapitole.

Operace **roll-up** se posouvá v konceptuální hierarchii dimenze o úroveň výše. V příkladě míst prodeje se např. posuneme od města na kraj. Říkáme, že se provede agregace na datové kostce – na data se aplikuje agregační funkce, kterou jsme již dříve popsali.

Operace **drill-down** provede opak – posune se v hierarchii o úroveň níže a dojde ke zjemnění. Posuneme se tedy např. z města na ulici a zobrazí se údaje prodeje pro určité místo prodeje (pokud předpokládáme, že místo prodeje je pro ulici unikátní).

Operace **slice** vznikne stanovením určité podmínky pro dimenzi. Vybíráme určitý typ míst prodeje, který nás zajímá – např. pro město Brno. Představíme-li si data jako 3-D krychli, kde místo prodeje je jedna z dimenzí, dojde k uříznutí (vyříznutí) plátku krychle, který nás zajímá a s kterým dále pracujeme (to odpovídá významu slova slice, které se používá např. při krájení salámu na plátky). Slovo **dice** se používá např. při krájení masa na kostky – a to je také to, co operace dice

s datovou kostkou provádí – selekci podle několika dimenzí. Jako výsledek potom dostaneme podkostku požadovaných vlastností. Jako příklad lze uvést selekce prodejů, kde je místo prodeje Brno (první dimenze), během druhého kvartálu roku 2009 (druhá, časová dimenze).

Operace **pivot** (**rotate**) provede prohození dimenzí kostky. Zobrazená data se při této operaci nemění, dochází pouze k jiné prezentaci kostky. Dále existují ještě další operace **drill-across** a **drill-through**, jejichž podrobný popis lze najít v literatuře [pon01].

2.7 Návrh architektury datových skladů

Při návrhu architektury musíme mít na paměti, co náš zákazník potřebuje (obchodní požadavky), jaké datové zdroje má k dispozici, jak bychom na základě předchozích dvou bodů navrhli datový sklad (tabulky faktů a dimenzí) a s nimi spjaté obchodní dotazy, jejichž odpověď by zákazník chtěl z datového skladu zjistit. To definuje pohledy, ze kterých bychom se jako návrháři měli na datový sklad podívat.

Pomocí **pohledu shora dolů** vybereme relavantní informace, které potřebujeme pro datový sklad. Tyto informace souvisí právě s obchodními potřebami zákazníka. Při **pohledu na datový zdroj** definujeme informace, které jsou zpracovávány informačnímy systémy dané společnosti a vybereme ty, které mají (podle předchozího bodu) nějaký užitek. Jsou to obvykle databázové tabulky a jsou obvykle modelovány pomocí ER-modelu nebo jiných technik. V některých společnostech jsou však data ukládána a modelována i jinou formou.

Při **pohledu na datový sklad** jako takový nadefinujeme tabulky faktů a dimenzí – to už je konkrétní databázové schéma datového skladu. Nesmíme zapomenout přidat informaci o zdroji, datu a čase pořízení dat, což je v datovém skladu důležitá informace, poněvadž čas je jednou s dimenzí. Podle potřeb také přidáme předkalkulované součty a počty položek. **Pohled na obchodní dotazy** potom ukazuje data z perspektivy koncového uživatele.

Návrh datových skladů se provádí třemi způsoby: shora dolů, zdola nahoru nebo kombinací. První z nich se používá při dobré znalosti používané technologie a obchodních problémů, které mají být vyřešeny, a začíná celkovým návrhem a plánováním. Naopak postup zdola nahoru začíná experimentováním a prototypy a je užitečný při raném stádiu obchodního modelování a technologického vývoje. Dovoluje to organizaci za menší náklady vyhodnotit výhody této technologie před tím, než je učiněno nějaké zásadní rozhodnutí. V kombinovaném způsobu může společnost učinit návrh pomocí postupu shora dolů, přičemž implementace je vedena opačnou cestou – zdola nahoru.

Z pohledu softwarového inženýrství je se vývoj datového skladu skládá z plánování, analýzou požadavků, analýzou problémů, návrhem datového skladu, integrací dat a testováním a nakonec nasazením datového skladu. Tento postup je známý metodou vývoje vodopád. Pokud se tento postup provádí inkrementálně a opakuje se, mluvíme o spirálové metodě.

Proces návrhu datového skladu se skládá z volby obchodního procesu, který se bude analyzovat (např. zásilky, prodeje, objednávky), z volby jednotky obchodního procesu (např. jednotlivé transakce, denní výpisy z účtu apod.), z volby dimenzí (např. čas, položka, zákazník, prodejce) a z volby měrných jednotek, ktere budou spjaty s každým záznamem v tabulce faktů.

Architektury moderních datových skladů jsou většinou třívrstvé. Spodní vrstvu tvoří databázový server datového skladu obvykle reprezentovaný relačním databázovým systémem. Jaké technologie se použijí pro extrakci dat z této vrstvy závisí na zvoleném implementačním prostředí pro střední vrstvu. Pro jazyk Java jsou to například ODBC/JDBC nebo objektově relační mapování pomocí JPA/Hibernate. Střední vrstvu tvoří OLAP server, a to buď ROLAP (relační OLAP), který mapuje OLAP operace na standardní relační operace, nebo MOLAP (multidimenzionální), který přímo implementuje multidimenzionální data a operace.

Vrchní vrstvu tvoří OLAP klient, který je hlavním předmětem tohoto projektu. Obsahuje nástroje pro OLAP dotazy a výstupní sestavy, případně některé nástroje pro dolování. Z pohledu architektury existují tři základní modely datových skladů – podnikový datový sklad, datový trh a virtuální datový sklad, které jsou popsány v literatuře [han06].

ROLAP servery používají k uložení dat relační systém pro řízení báze dat. Tento server potom tvoří styčnou vrstvu mezi OLAP klientem pro zobrazování a relačním databázovým systémem. Naproti tomu MOLAP server používá úložiště dat postavené na datových polích a podporuje proto přímý pohled na struktury datových kostek. Tato metoda dovoluje rychlé indexování předkalkulovaných dat, ale nastává tu problém s efektivitou využití paměťového prostoru pro tzv. řídká data. V takovémto případě se musí prozkoumat možnosti komprese dat. Dále existují servery HOLAP (kombinace dvou předešlých) a speciální SQL servery – relační databázové servery, které poskytují dotazy nad datovými kostkami.

2.8 Implementace datového skladu

Implementovaný datový sklad musí zpracovávat a vracet výledky na OLAP dotazy v řádu sekund. Zároveň víme, že datové sklady obsahují velmi vysoké množství dat. Proto musí implementované řešení disponovat efektivním výpočtem datových kostek, indexováním OLAP dat, efektivním zpracováním OLAP dotazů, úložištěm metadat a dalšími nástroji.

Efektivním výpočtem datových kostek myslíme efektivní agregaci. V SQL se popisuje pomocí klauzule group by, která seskupí záznamy podle požadovaného sloupce. První možností je vypočítání součtu všech hodnot měrných jednotek kostky pro všechny kombinace dimenzí. Tato metoda má při větším počtu dimenzí velké prostorové nároky, proto se v praxi nepoužívá. Další možností je tzv. částečná materializace, kde se vypočítávají datové kostky pouze pro vybrané podmnožiny dimenzí. Kritéria pro výběr těchto podmnožin jsou frekvence jednotlivých dotazů, přístupová cena, cena těchto výpočtů a cena jejich opětovného přepočítání v případě změny dat v datovém skladu. Metoda

vícecestné agregace polí používá optimalizační techniky jako seřazení (či hašování) pro dimenzi a předpočítané agregace pro ROLAP, nebo rozdělení pole na části a výpočet agregací během přístupu k jednotlivým hodnotám v záznamech (výsledkem je snížení počtu přístupů k jednotlivým buňkám) pro MOLAP. Optimalizační techniky pro MOLAP jsou dokonalejší, proto je MOLAP rychlejší než ROLAP. Ten je vhodnější pro menší počet dimenzí.

V OLAP systémech se používá bitmapové a spojovací indexování. První z nich se používá pro zrychlení vyhledávání v kostkách, druhé pro zrychlení operace spojení (join). Obě metody indexování jsou popsány v literatuře [han06].

Na základě předpočetených pohledů (kostek) a indexových struktur popsaných v předchozích dvou odstavcích by mělo zpracování dotazu vypadat následovně:

- Určení, které operace se mají učinit s předpočtenými pohledy (views) a překlad této OLAP operace do SQL dotazu.
- Určení, na které materializované kuboidy se tyto operace mají použít (Identifikují se všechny pohledy, které mohou být využity pro zpracování dotazu).

Metadata jsou data popisující data. V tomto případě mluvíme o popisování objektů v datovém skladu. Tato data hrají úplně odlišnou roli, než data v datovém skladu, a jsou uložena persistentně – tj. na disku. Úložiště metadat by mělo obsahovat:

- popis struktury datového skladu – schéma datového skladu, pohledy, dimenze, konceptuální hierarchie a definice odvozených dat a umístění a obsah datového trhu.
- operační metadata – historie dat (historie dat a transformace na ně aplikované), aktuálnost dat (aktivní, archivovaná, vyčištěná) a monitorovací informace (statistika použití datového skladu, chybové výstupy atd.)
- algoritmy použité pro sumarizaci
- popis mapování z operačního prostředí (např. informační systém společnosti) na datový sklad
- data spojená s výkonem systému
- obchodní metadata – obsahují obchodní termíny a definice, informace o vlastnictví dat a platební podmínky.

Dále datový sklad používá back-end nástroje pro načtení a aktualizaci dat ve své databázi. To zahrnuje nástroje na extrakci, čištění, transformaci a načtení (třídění, sumarizace, počítání pohledů atd.) dat.

3 Business Intelligence Development Studio

Firma Microsoft poskytuje podporu OLAP operací pomocí svého produktu Microsoft SQL Server. MS SQL Server umožňuje vytvoření datového skladu včetně všech operací popsaných v minulých kapitolách. Tento OLAP server Microsoftu má rozhraní, kterým poskytuje připojeným klientům operace jako definici a vytvoření datového skladu, nahrání dat do datového skladu a OLAP operace (roll-up, drill-down atd.).

V této kratší kapitole si popíšeme nástroje Microsoft Visual Studia jako klienta pro definici a vytvoření datového skladu a provádění a vizualizaci OLAP operací. Nástroj pro práci s OLAP kostkami se jmenuje Business Intelligence Development Studio. Jedná se o Microsoft Visual Studio 2008 s dalšími přidanými typy projektů, které jsou specifické pro business intelligence MS SQL Serveru:

- Analysis Services – tento typ projektu se používá pro plnění datového skladu, ale zejména pro OLAP analýzu nad datovou kostkou.
- Integration Services – tento typ projektu provádí integraci dat do datového skladu, ale také jeho definici a vytvoření.
- Reporting Services

3.1 Vytvoření datového skladu

Podle druhé kapitoly nejdříve navrhneme datový sklad. Musíme mít představu o tom, odkud budeme data čerpat, jak bude vypadat schéma datového skladu (tabulky faktů a dimenzí) a jaké budou konceptuální hierarchie v tabulkách dimenzí.

Nejdříve napíšeme skript (v tomto případě skript pro MS SQL Server), který nám vygeneruje schéma datového skladu. Vytvoříme tabulky faktů a tabulky dimenzí. V každém případě vytvoříme tabulku časové dimenze v závislosti na tom, po jakých časových úsecích chceme umožnit analytikovi časovou dimenzi zobrazovat. Můžeme zavést sloupce jako rok, kvartál, měsíc, den, ale také staletí, padesát let, deset let, týden, nebo hodina. Musíme brát v úvahu časové členění v dané společnosti. Např. týká-li se náš datový sklad personalistiky a postupujeme-li podle českého zákoníku práce, budeme členit na rok, kvartál, měsíc, týden, den, protože vyrovnávání pracovních hodin může probíhat podle zákoníku práce po týdnech.

Na internetu je možné nalézt příklady skriptů, které časovou dimenzi generují. Lze ji vytvořit cyklem ve vložené proceduře. Pokud bychom nepoužívali MS SQL Server a náš databázový systém nepodporuje cykly, bylo by nutné vypsat všechny řádky v jazyku SQL, což by bylo časově náročné. V takovém případě je ale možné použít skriptovací jazyk, který SQL skript vygeneruje.

Důvod, proč se tabulka časové dimenze vytváří, je rychlý přístup k datům. Samozřejmě by bylo možné časovou dimenzi při každém přístupu počítat, bylo by to ale časově náročné, pokud by se tak dělo pro každý řádek v tabulce faktů.

Vytvoření projektu pro integrační služby

V Microsoft Business Intelligence Development Studio (BIDS) vytvoříme nejdříve projekt pro integrační služby (Integration Services project). V grafickém editoru si pro každou dimenzi vytvoříme tzv. Data Flow task. U každého Data Flow task je nutné definovat jeho průběh.

Jednoduchý Data Flow task může vypadat takto: Na začátku tohoto úkolu mám definovaný nějaký datový zdroj, u kterého stanovím, jakým způsobem se k datům dostanu – typicky SQL příkazy select a join. Druhá komponenta je komponenta výstupu, u které definuji, o jakou dimenzi (tabulku datového skladu) se jedná. Mezi první a druhou komponentou je natažena hrana datového toku.

Dále je možné vytvořit složitější Data Flow task, který např. spojí data z databáze a ze souboru MS Office Excel pomocí komponenty Merge. Komponenta Merge vyžaduje na vstupu seřazené hodnoty, proto je potřeba použít komponentu Sort. Komponenta Sort umí také vymazat řádky s duplicitní hodnotou atributu (atributů) definovaného pro seřazení, což se hodí také za komponentou Merge (databázové řádky a řádky v MS Excel mohou být duplicitní). Na konci tohoto úkolu je opět komponenta výstupu.

V BIDS je možné definovat i složitější Data Flow tasks, předchozí dva příklady nám však pro přehled a vytvoření požadavků pro OLAP klienta postačí.

Nakonec je možné pomocí příkazu Run spustit jednotlivé úkoly. Pokud je vše nadefinováno v pořádku, dojde k naplnění tabulek v datovém skladu.

3.2 OLAP analýza

Dále vytvoříme projekt pro analytické služby (Analysis Services project). Jak jsme si vysvětlovali v úvodu této kapitoly, tento typ projektu slouží už přímo k OLAP analýze.

V části Solution Explorer se objeví struktura projektu ve stromu s jednotlivými částmi. Jako první se zaměřím na část Data Sources, kde definuji datové zdroje. Musím definovat, odkud budu data datového skladu načítat. Zvolíme umístění a název databáze, kterou budu používat. Dále musíme zvolit tabulky datového zdroje, které budou vstupem pro OLAP analýzu. Tyto tabulky jsme vytvořili v předchozí podkapitole pomocí integračních služeb.

Bohužel BIDS z nějakého důvodu není schopen rozeznat cizí klíče mezi tabulkami, proto je nutné vše dodefinovat. Obecně je dobré zkontrolovat, zda struktura databáze datového skladu koresponduje našim představám a popřípadě ji upravit. Typicky se jedná o schéma hvězdy nebo sněhové vločky, jak bylo popsáno v druhé kapitole.

Dále v Solution Explorer vyberu část projektu pro datové kostky a vytvořím novou datovou kostku. BIDS se snaží automaticky odhalit, co je tabulka faktů a co je tabulka dimenzí. Někdy je nutné tento odhad upravit podle našich požadavků. Dále je nutné v průvodci dodefinovat, která tabulka je tabulka časové dimenze. U časové dimenze musím definovat, které sloupce odpovídají jednotkám času definovaným v BIDS. Důvod proto je ten, že systém vytvoří automaticky konceptuální hierarchii časové dimenze.

U ostatních dimenzí musíme hierarchie dodefinovat. To provedeme přetažením položek do části okna definující hierarchii. Máme-li všechny hierarchie definovány, vybereme kostku v Solution Explorer a provedeme nahrání dat do kostky. Až potom teprve dojde k fyzickému vytvoření datové kostky.

Pod záložkou pro hierarchii dimenzí v podzáložce Browser je možné zobrazit tyto hierarchie. To se nejedná ještě o zobrazování dat z datové kostky, ale pouze nadefinované hierarchie, která ani nemusí obsahovat žádné položky z tabulky faktů.

Zajímavější je záložka Browser pod datovou kostkou. Zde je už možné provádět OLAP analýzu. Do pracovní části okna je možné přetahovat (drag and drop) fakta a dimenze. Datová kostka se vizualizuje v tabulce. Typicky se dimenze přetahují do záhlaví tabulky (ve smyslu řádků i sloupců) a fakta doprostřed tabulky. Z faktů vybereme metriku, která nás zajímá a přetáhneme ji doprostřed tabulky.

Do záhlaví nemusíme přetáhnout celou dimenzi, ale její část – tím provedeme operaci slice. Jednotlivé položky dimenzí je možné v záhlaví tabulky expandovat kliknutím – tabulka se potom zvětší – jedná se o OLAP operaci drill-down. Naopak při opětovném kliknutí a provedení kontrakce se jedná o operaci roll-up. Operace dice provedeme nejsnázeji definicí podmínky pro výběr dimenzí v horní části pracovní plochy.

Jako klient pro prohlížení OLAP kostek lze použít také Microsoft Office Excel. Ten má výhodu v možnosti vygenerování grafů, které lze při OLAP analýze využít. Touto aplikací se lze připojit na již existující datovou kostku na MS SQL Serveru.

3.3 Shrnutí

Cílem této kapitoly nebylo vytvořit přesný postup, jak lze v BIDS vytvořit datovou kostku a provádět na ní OLAP analýzu, ale spíše nastínit danou problematiku v kontextu OLAP klientů a popsat jedno konkrétní řešení. Proto se zde nevyskytují obrázky, což je běžné u průvodců (tutorials). Poznatky získané v této kapitole nám potom poslouží v příští kapitole při definici a analýze požadavků pro OLAP klienta, který je předmětem tohoto diplomového projektu.

4 Analýza a návrh OLAP klienta

V předchozích kapitolách jsme popsali teoretický základ, který k vytvoření datového skladu potřebujeme (zejména průběh návrhu datového skladu, který musíme promyslet manuálně) a nástroj, který je možné použít pro vytvoření, správu a používání datového skladu. V této kapitole popíšeme návrh podobného, ale jednoduššího, nástroje pro OLAP operace nad datovým skladem.

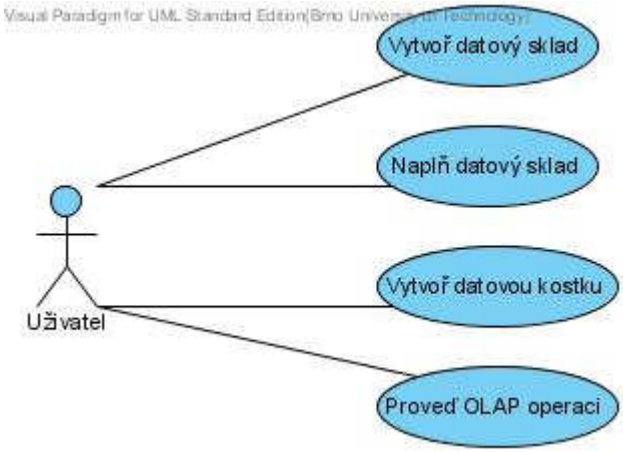
4.1 Specifikace a analýza požadavků

Kapitola 3 ukazuje základní vlastnosti OLAP klienta Business Intelligence Development Studio od firmy Microsoft. Vezmeme ji jako základ pro to, jaké funkce by jednoduchý OLAP klient měl poskytovat.

V OLAP klientu by mělo být možné:

- Vytvořit datový sklad – měla by stačit možnost spuštění SQL skriptu, který vytvoří tabulky datového skladu. Uživatel potom musí systému nějakým způsobem dát vědět, které tabulky jsou tabulky faktů a tabulky dimenzí. Systém by měl znát vazby mezi těmito tabulkami.
- Naplnit datový sklad – stačí možnost spuštění skriptu (select, join), který naplní tabulku (tabulky) faktů.
- Vytvořit datovou kostku – dojde k vytvoření datové kostky v paměti, kterou je možné vhodným způsobem vizualizovat – nejjednodušším způsobem je tabulka či matice tabulek v případě více dimenzí.
- Provádět OLAP operace – v rámci vizualizace interaktivní provádění OLAP operací roll-up, drill-down, slice/dice, případně pivot (rotate).

Následující diagram případů užití na obrázku 4.1 ukazuje předchozí 4 body jako případy užití. Tabulka 4.1 potom ukazuje detail případu užití „Vytvoř datový sklad“.



Obrázek 4.1: Diagram případů užití

ID:	1
Název:	**Vytvoř datový sklad**
Autor:	Lukáš Podsedník
Primární aktéři:	Uživatel
Sekundární aktéři:	
Vstupní podmínky:	Žádné
Následné podmínky:	Datový sklad je vytvořen a uložen.
Akce pro spuštění:	Viz hlavní tok.
Hlavní tok:	1. Uživatel vybere volbu File>New>Project… V následujícím dialogu vybere typ projektu Data Warehouse, pojmenuje projekt a dokončí vytvoření projektu. 2. V části okna Project Explorer se objeví nově vytvořený projekt. 3. Zákazník zobrazí dialog vlastnosti projektu, vybere vlastnost Main Data Source, kde definuje datový zdroj, kde bude datový sklad uložen. Po vyplnění stiskne OK. 4. V podsložce projektu Data Sources přibude tento datový zdroj. 5. Uživatel klikne pravým tlačítkem myši na podsložku Scripts, vybere New Script. Objeví se standardní dialog pro nový soubor, kde může skript pojmenovat. Poté klikne pro dokončení na Finish. 6. Systém zobrazí v části Editor nový SQL skript, který může uživatel ukládat a modifikovat. 7. Po dokončení modifikace skriptu klikne uživatel pravým tlačítkem na hlavní datový zdroj a vybere volbu Run Script… 8. Objeví se dialog pro výběr skriptu. Standardně se zobrazí všechny skripty dostupné v projektu. Uživatel vybere nově vytvořený skript. Alternativně může uživatel vybrat externí skript tlačítkem External SQL Script… Poté uživatel klikne na tlačítko Run. 9. Dojde ke spuštění SQL skriptu. Výsledek skriptu je vypsán do části okna Console.
Alternativní toky:	
Výjimky:	Storno Chybně zadané údaje Chyba systému
Frekvence:	Občas
Speciální požadavky:	Žádné

Tabulka 4.1: Detail případu užití Vytvoř datový sklad. Výjimky z případu užití pro zřejmost neuvádíme.

Vidíme, že případ užití je poměrně komplikovaný. Takto by bylo možné popsat i ostatní případy užití, ale rozsáhlá analýza požadavků není předmětem této práce. Princip fungování aplikace vychází z principů práce s prostředím Eclipse IDE (či Eclipse RCP), které je zamýšleno jako platforma pro OLAP klienta. Více informací o Eclipse IDE/Eclipse RCP je možné nalézt na www.eclipse.org nebo v literatuře [mca08].

Zadání projektu se také zmiňuje, že klient by měl používat databázi MySQL a celá aplikace musí mít (minimálně) třívrstvou architekturu.

4.2 Návrh

Požadavek na třívrstvou architekturu znamená, že kromě klienta je potřebné také implementovat jednoduchý OLAP server. Tento server bude spadat do kategorie ROLAP (Relační OLAP), protože úložiště datového skladu je relační databáze MySQL.

Cílem je vytvořit co nejjednodušší a nejpoužitelnější projekt. Proto budeme používat technologie, které jsou volně dostupné. Pro ROLAP server lze použít aplikační kontejner Tomcat a Spring Framework, který bude jednak tvořit rozhraní mezi OLAP klientem a ROLAP serverem, ale také obhospodařovat instance jednotlivých OLAP služeb. Jako rozhraní se může použít Spring Remote Procedure Call (Spring RPC) nebo další možnosti Spring Framework pro vzdálenou komunikaci (Spring Remoting, např. HTTP invoker).

ROLAP server bude poskytovat rozhraní pro:

- Ověření připojení k databázi.
- Spuštění libovolného SQL skriptu na MySQL databázi.
- Provedení SQL dotazu na databázi a vrácení výsledku.
- Nahrání dat do datového skladu z jiného datového úložiště.
- Vytvoření datové kostky podle zadaných parametrů (vrácení objektu datové kostky).
- Provedení OLAP operace drill-down, roll-up, slice/dice na libovolné datové kostce.

OLAP klient bude těchto služeb využívat. Jeho funkcionalita včetně jednoho případu užití byly popsány v předchozí kapitole.

4.3 Volba technologií a vývojového prostředí

Na tomto místě bych rád objasnil, které technologie budou v projektu použity včetně důvodů, proč jsou použity právě ty. Pro implementaci byl zvolen jazyk Java. Je tomu tak zejména z důvodu jeho platformové nezávislosti a existenci nepřeberného množství knihoven pro práci s databázemi, na tvorbu informačních systémů, webových aplikací a klientů.

Jako vývojové prostředí bylo zvoleno Eclipse IDE – prostředí, v kterém má autor této práce nejvíce praxe a zkušeností a které považuje za v dnešní době nejlepší možné pro vývoj aplikací v programovacím jazyku Java díky jeho relativně nízkým paměťovým nárokům a intuitivnímu ovládání.

Stejné prostředí/platforma bylo zvoleno pro implementaci OLAP klienta. Eclipse nejenom že nabízí komfortní vývojové prostředí, ale je zde také možnost stavět na Eclipse Rich Client Platform (Eclipse RCP) vlastního bohatého klienta, či vytvořit plug-in přímo do Eclipse IDE: Plug-in, který

bude poskytovat funkce OLAP klienta. Jako platforma pro implementaci klienta byla též zvažována NetBeans Platform, či čistý Java Swing/Java SWT. První z nich má – dle názoru autora – ne příliš přehledné API a příliš vysoké paměťové nároky, při výběru druhé možnosti by bylo nutné implementovat celé grafické rozhraní bez možnosti využití předpřipravených komponent pro vývojové prostředí. Jak jsme viděli v kapitole 3, BIDS využívá také služeb vývojového prostředí, a to MS Visual Studia. Není proto překvapením, že OLAP klient vznikne rozšířením nějakého stávajícího vývojového prostředí nebo platformy.

V předchozí kapitole je též zmínka o tom, že pro ROLAP server bude použit volně dostupný aplikační kontejner Tomcat, namísto více nebo méně komerčně dostupným aplikačním serverům jako WebLogic, JBoss a dalších. Jedinou výhodu, kterou by tyto aplikační servery přinesly, je lepší podpora zasílání zpráv, jako např. JMS (např. služby JPA (Java Persistence API) v případě OLAP technologií nejsou potřeba).

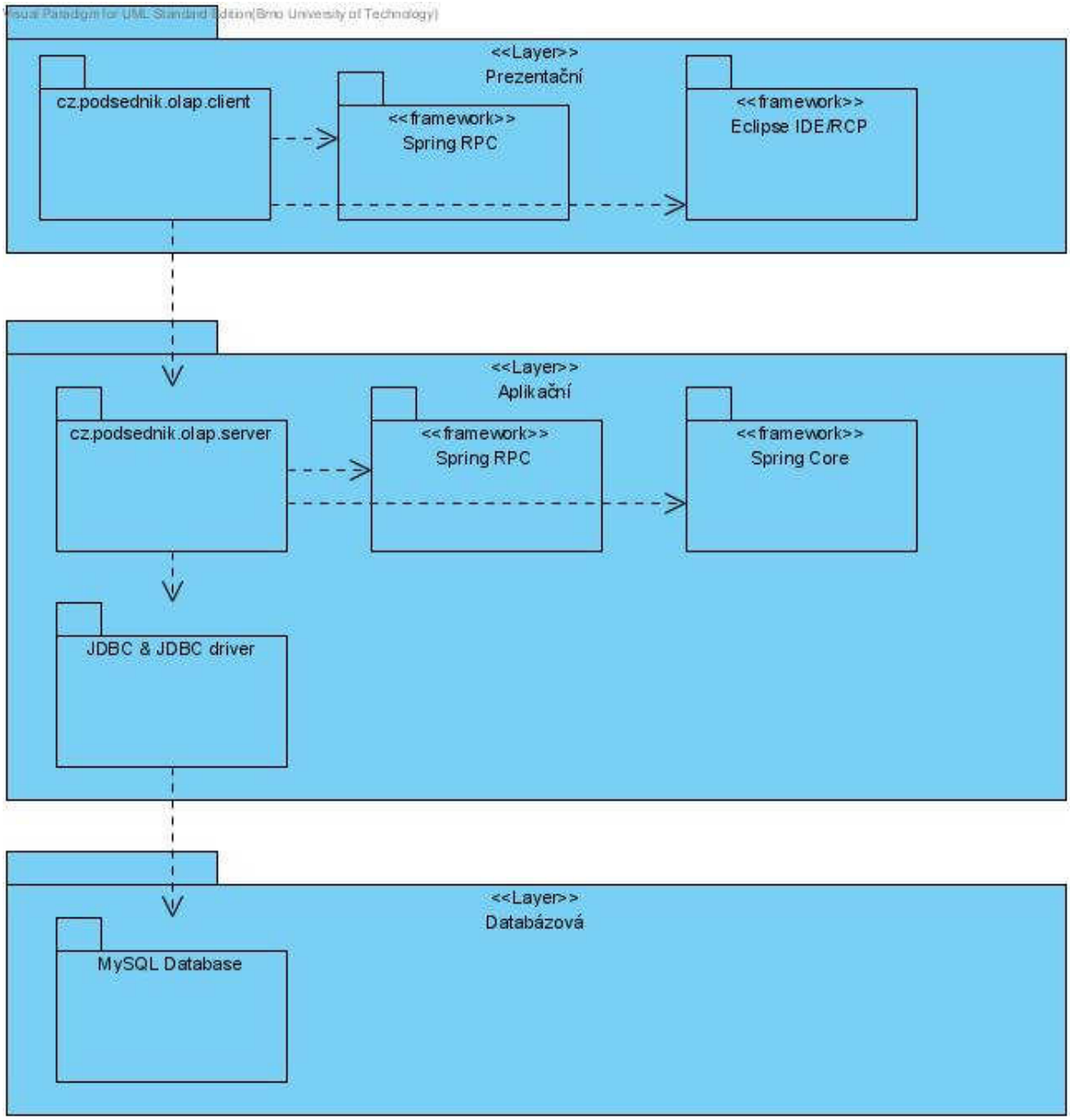
Jako databázi použijeme MySQL, což je zřejmé přímo ze zadání. Jako rozhraní mezi databází a ROLAP serverem použime klasické JDBC příkazy. Je tomu tak proto, že bude z větší části potřeba spouštět spíše SQL skripty, než operace nad entitami. Jména a strukturu entit v době vzniku této aplikace nebudeme znát: tabulky datového skladu si bude uživatel vytvářet sám. Z toho plyne také abscence využití nástrojů na objektově relační mapování jako Hibernate, či jiné implementace JPA.

Jak bylo psáno v minulé kapitole, pro ROLAP server použijeme Spring Framework, který se bude starat o správu instancí jednotlivých služeb serveru a také o komunikaci mezi OLAP klientem a serverem.

4.4 Architektura

Představme si případ, kdy uživatel vyvolá operaci drill-down např. kliknutím (expandováním) položky dimenze v tabulce. OLAP klient má v paměti uložen model datové kostky. Dojde k pokusu o expanzi datové kostky. V případě, že nejsou informace uloženy lokálně v datové kostce, je potřeba provést volání ROLAP serveru, aby uskutečnil na podkostce operaci drill-down. Klient pomocí Spring-RPC (nebo jiný nástroj, např. Spring Remoting) zavolá službu na aplikačním (ROLAP) serveru. Server OLAP operaci přeloží do SQL syntaxe a odešle ji pomocí JDBC MySQL databázovému systému. Ten serveru odpoví na tento SQL dotaz. V případě chybějících funkcí v MySQL je nutné některé operace provést přímo na aplikačním serveru. Po jejich provedení se vytvořená odpověď na operaci drill-down vrátí přes synchronní rozhraní (Spring-RPC) zpět klientovi. Ten potom výslednou datovou kostku zobrazí.

Tento příklad ilustruje architekturu a závislosti v aplikaci. V tomto případě nemluvíme pouze o OLAP klientu, ale celému OLAP systému a jednoduchém datovém skladu. Následující diagram balíčků na obrázku 4.2 ilustruje architekturu systému.



Obrázek 4.2: Diagram balíčků popisující architekturu systému.

5 Implementace

Tato kapitola se bude zabývat popisem použitých nástrojů a knihoven, a to včetně návodů a příkladů jejich použití. Dále se zmíním o projektu Eclipse Business Inteligence Reporting Tools (Eclipse BIRT), jehož komponenty budou při implementaci použity. Následuje kapitola o vlastní implementaci aplikací – OLAP serveru a OLAP klienta. V závěru jsou potom tyto aplikace představeny jako hotový produkt.

5.1 Databáze

Jak již bylo zmíněno výše, bude použita databáze MySQL, která je volně dostupná jak pro soukromé, tak pro komerční použití. Podle kapitoly 2 bude tato databáze obsahovat tabulky faktů a tabulky dimenzí podle vytvořeného uživatelem navrženého datového skladu.

S databází bude komunikovat OLAP server. Tato komponenta bude také implementována v rámci tohoto projektu. Rozhraní mezi databází a OLAP serverem musí být řešeno nějakou formou JDBC (Java Database Conectivity) a jazykem SQL. Objektově relační mapování není možné, protože databáze může mít obecně jakýkoli obsah (jakékoliv tabulky faktů a dimenzí). Nelze tudíž vytvořit třídy a jejich instance, na které bychom mohli mapovat řádky datového skladu. Tyto třídy by nemohly být vytvořeny za běhu aplikace (programovací jazyk Java to neumožňuje). V následující kapitole popíši použití JDBC ke komunikaci s databází a nástrojů Spring frameworku, které toto použití zjednodušují.

5.1.1 Spring framework a JDBC

Nejjednodušší možností je použití klasického JDBC rozhraní, které poskytuje knihovny jazyka Java. Následující fragment kódu ukazuje použití JDBC:

```
Connection connection = null;
String url = "jdbc:mysql://localhost:3306/";
String database = "myDatabase";
String driver = "com.mysql.jdbc.Driver";
String username = "username";
String password = "passwd";
try {
  Class.forName(driver).newInstance();
  connection = DriverManager.getConnection(url + database, username,
password);
  // SQL statements and operations
```

```
  connection.close();
} catch (Exception e) {
  e.printStackTrace();
}
```

Jakkoliv je práce v JDBC standardní, vyžaduje příliš redundantního opakovaného kódu. U klasického JDBC musí programátor zaštítit spoustu rutinní práce, která se neustále opakuje: definice JDBC URL databáze, jméno databáze, uživatelské jméno a heslo, vytvoření spojení, vytvoření JDBC dotazu pomocí SQL dotazu, extrakce výsledku, zavření spojení a další. Prověřme tedy další možnosti, jak práci s JDBC zjednodušit. Spring framework poskytuje dobrou podporu pro JDBC. Prvním prvkem je datový zdroj (Data Source), který lze definovat v rámci aplikačního kontextu frameworku Spring. Ilustruje to následující fragment kódu v mírné modifikaci převzatý z [wab08].

```
<bean id="dataSource"
  class="org.apache.commons.dbcp.BasicDataSource">
  <property name="driverClassName" value="com.mysql.jdbc.Driver" />
  <property name="url"
  value="jdbc:mysql://localhost:3306/myDatabase" />
  <property name="username" value="username" />
  <property name="password" value="passwd" />
  <property name="initialSize" value="5" />
  <property name="maxActive" value="10" />
</bean>
```

Tato varianta počítá s mechanismem „pooling“ – tj. uchovávání instancí datových zdrojů pro další použití. Využívá knihovny Commons DBCP, nejedná se tedy přímo o Spring. Je možné také použít jednoduchou Spring implementaci [wab08]:

```
<bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="driverClassName" value="com.mysql.jdbc.Driver" />
  <property name="url" value="jdbc:mysql://localhost:3306/myDatabase" />
  <property name="username" value="username" />
  <property name="password" value="passwd" />
</bean>
```

Tento datový zdroj ale nepodporuje výše popsaný connection pooling. Je na místě zvážit tuto implementaci pro OLAP operace, kde lze mnohdy spočítat počet spojení na prstech jedné ruky. Pro běžné OLTP databáze je ale vhodná první varianta, která uchovává připojení k databázi pro další použití. Základní datový zdroj pro OLAP server je tedy vyřešen následujícím způsobem:

```
<bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
      <property name="driverClassName">
            <value>${jdbc.driverClassName}</value>
```

```
        </property>
        <property name="url">
              <value>${jdbc.url}</value>
        </property>
        <property name="username">
              <value>${jdbc.username}</value>
        </property>
        <property name="password">
              <value>${jdbc.password}</value>
        </property>
```

Proměnné uzavřené ve svorkách ${} jsou definovány v .properties souboru, na který aplikační konext odkazuje. Pro datový zdroj se definuje JDBC URL, uživatelské jméno, heslo a třída ovladače, jak je obvyklé. Tento datový zdroj potom můžeme použít kdekoliv v aplikaci.

V souvislosti s datovými zdroji řešíme ale další problém: při vytváření datového skladu musíme nějak odlišit existující datové sklady v databázi. To lze vyřešit:

- Odlišným jménem tabulek – systém nedovolí uživateli vytvořit tabulku faktů/dimenzí se stejným jménem, které již v databázi existuje. Tento problém by se musel řešit programově. Je použití jedné databáze, a tedy jednoho datového zdroje bez nutnosti přepínání mezi databázemi.
- Odlišným jménem databáze – pro každý nový datový sklad se vytvoří nová databáze se stejným umístěním. Je zde nutné vyřešit nutnost přepínání mezi databázemi. Jednou možností řešení je použití více datových zdrojů, nebo modifikace existujícího datového zdroje. Ten už potom nelze definovat kompletně v konfiguračním souboru frameworku Spring. Instance DriverManagerDataSource se musí vytvořit za běhu v aplikaci.
- Odlišným jménem a umístěním databáze – nejnáročnější způsob, lze ovšem vyřešit také výše uvedeným způsobem.

Za běhu aplikace lze vytvořit datový zdroj např. takzvanou lazy inicializací následujícím způsobem:

```
public class DataSourceFactory {
  private DriverManagerDataSource ds;
  public DriverManagerDataSource getDataSource(${init_params}) {
    if (this.ds == null) {
      DriverManagerDataSource ds = new DriverManagerDataSource();
      // inicializace
    }
    return this.ds;
  }
}
```

Instance třídy DataSourceFactory se vytvoří standardním způsobem v rámci konfiguračního souboru frameworku Spring. Výhoda je v tom, že instance třídy datového zdroje se vytvoří pouze jednou v rámci jednoho sezení (session). Jedná se ve své podstatě o modifikovaný návrhový vzor Singleton. Singleton ovšem vytváří jednu instanci pro všechny uživatele na jednom JVM (Java Virtual Machine). To by ale byla chyba, protože k aplikačnímu serveru se může připojit více vláken, a každé vlákno jistě používá jinou databázi. Proto DataSourceFactory nemá metodu getInstance(), která vrací tutéž třídu, nýbrž metodu getDataSource(), jež vrací instanci datového zdroje. Instance DataSourceFactory je administrována frameworkem Spring a existuje právě jedna pro jedno vláknu na serveru – což je přesně to, o co usilujeme. Dále v této kapitole popíši ještě další možnosti, jak přepínat mezi databázemi a tedy jednotlivými datovými sklady.

Spring framework poskytuje nesčetné množství knihoven a služeb. Už jsem popisoval podporu JDBC, kterou na následujících řádcích ještě dále rozvedu. Poté se budu zabývat podporou transakcí ve frameworku Spring.

Spring v JDBC dále přichází s několika zjednodušeními. První z nich je použití JDBC šablony (JDBC template). Tato šablona umožňuje jednodušší práci při dotazech a aktualizaci databáze. Pro OLAP server definujeme JDBC šablonu následujícím způsobem:

```
<bean id="jdbcTemplate"
      class="org.springframework.jdbc.core.JdbcTemplate">
      <property name="dataSource" ref="dataSource" />
</bean>
```

Vidíme, že se opět jedná o Spring bean, které nastavujeme vlastnost datového zdroje odkazem na již definovaný datový zdroj (viz výše). Existují i další možnosti volby konkrétní JDBC šablony (viz literatura [wab08]). Použití šablony je potom omezeno na jeden příkaz, jak je patrné z následujícího příkladu, který přidává řádek do tabulky DATA_WAREHOUSES:

```
jdbcTemplate.update("insert into data_warehouses values (?, ?)",
  new Object[] {warehouse.getName(), warehouse.getFactTable().getName()});
```

Toto zjednodušení je již dostatečné pro použití v OLAP serveru. Bohužel nemůžeme využí pro OLAP podporu objektově relační mapování, neboť nevíme, jak budou databázové objekty vypadat – relační tabulky totiž vytváří sám uživatel.

S využitím dědičnosti lze dále použít třídu JdbcDaoSuppport jako nadtřídu pro služby přistupující k databázi. JdbcTemplate je potom integrovaná do služby přímo a lze ji vyvolat voláním getJdbcTemplate(). Tento způsob je ale vhodnější pro návrhový vzor DAO četně používaného v OLTP systémech, proto ho nebudeme používat.

5.1.2 Transakční zprazování

Pro OLAP server ale potřebujeme ještě jednu důležitou vlastnost: nutnost přepínání mezi databázemi. Každý datový sklad totiž tvoří jiná databáze. Možností by bylo vytvořit datový zdroj pro každou databázi. To by se však muselo vytvářet manuálně v programovém kódu, nikoli v konfiguračním souboru aplikačního kontextu. Nevíme totiž dopředu, jak se daná databáze bude jmenovat. V minulé kapitole jsme již popsali nevýhody tohoto způsobu přepínání. Na následujících řádcích popíši další možné řešení, které bylo nakonec pro implementaci zvoleno.

Vhodnější možností řešení tohoto problému je použití jazyka SQL. Chceme-li změnit databázi v rámci jednoho připojení, použijeme SQL příkaz USE database_name, který provede změnu. Zavoláme-li ho však pomocí metody execute() z JDBC šablony, změna databáze se provede pouze v jednom volání. Při dalším volání JDBC šablony je databáze původní, definová v datovém zdroji. Jak je to možné?

Klasická JDBC šablona totiž provádí jeden příkaz v rámci jedné SQL transakce. Vytvoří transakci, provede příkaz a následně provede COMMIT (potvrzení) nebo ROLL-BACK (návrat do původního stavu), a tím ukončí transakci. Spustíme-li tedy příkaz USE pro změnu databáze, v dalším JDBC dotazu už tato změna není patrná – datový zdroj má totiž definovanou JDBC URL adresu, ve které je skryto také jméno databáze.

Řešením je nespouštět každý příkaz JDBC šablony v jedné transakci, ale seskupovat více příkazů do jedné transakce. Toto výchozí chování JDBC šablony se změní použitím takzvaného Transaction Manager. Spring framework jich definuje vícero, pro účel OLAP serveru se hodí nejvíce DataSourceTransactionManager, který odkazuje na datový zdroj. Jeho definici provedeme opět jako Spring Bean:

```
<bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
   <property name="dataSource" ref="dataSource"/>
</bean>
```

Odkazujeme přitom na výše popsaný datový zdroj. Další možnosti transaction managers pro jiné způsoby ukládání než JDBC viz literatura [wab08]. Vytvořením této Spring bean však konfigurace nekončí. Musíme nějakým způsobem definovat, které operace budou probíhat v rámci jedné transakce. K tomu existují dva způsoby.

První z nich, programový způsob, definuje transakci v programovém kódu. Programátor vytvoří instanci TransactionTemplate, které zadá daný transaction manager do kontruktoru jako parametr. TransactionTemplate má potom metodu execute, které se zadá jako parametr např. instance třídy TransactionCallbackWithoutResult, která musí definovat metodu doInTransactionWithoutResult(). V ní se už volá JDBC šablona klasickým způsobem. Tato možnost je poměrně složitá, proto jsem se jí při implementaci OLAP klienta snažil vyhnout.

Druhou možností je deklarativní přístup. Spočívá v definici tříd a jejich metod, které se mají vykonávat v jedné transakci. Je to přesně to, co je potřeba: jedna metoda (např. vytvoření datového skladu nebo operace drill-down) je vykonávána v rámci jedné transakce. Metoda se z hlediska výsledku v databázi buď vykoná kompletně, nebo se nevykoná vůbec. Co je ještě lepší, při tomto způsobu konfigurace transakcí nemusíme vůbec zasahovat do kódu. Pouze „zabalíme“ určitou Spring bean do jiné Spring bean. Následující fragment kódu shrnuje, jak lze tuto konfiguraci uskutečnit:

```
<bean id="DataWarehouseService"
  class="org.springframework.transaction.interceptor
   .TransactionProxyFactoryBean">
  <property name="transactionManager" ref="transactionManager"/>
  <property name="target" ref="DataWarehouseServiceTarget"/>
  <property name="proxyTargetClass" value="true"/>
  <property name="transactionAttributes">
    <props>
      <prop key="*">PROPAGATION_REQUIRED,-Exception</prop>
      <prop key="get*">PROPAGATION_SUPPORTS</prop>
      <prop key="set*">PROPAGATION_SUPPORTS</prop>
    </props>
  </property>
</bean>

<bean id="DataWarehouseServiceTarget"
class="cz.podsednik.olap.server.service.impl.DataWarehouseServiceImpl">
  <property name="jdbcTemplate" ref="jdbcTemplate" />
</bean>
```

Princim „zabalení“ spočívá v definici Spring bean se stejným jménem, jaké měla původní Spring bean služby. Té přidáme ke jménu příponu „target“ (můžeme však její jméno změnit jakkoli). Na „zabalenou“ službu potom odkážeme ve vlastnosti „target“. Nová DataWarehouseService je instance TransactionProxyFactoryBean, která se chová jako původní služba. Jediný rozdíl spočívá v tom, že jsou její metody v rámci jedné transakce. Vlastnost „transactionAttributes“ definuje šablony názvů metod, které je třeba vykonávat v rámci jedné transakce. Výše uvedená definice zajišťuje jednotransakční zpracování všech metod kromě metod začínající na „get“ a „set“, tedy metod, které nastavují jednotlivé proměnné třídy.

Tím definice transakce končí a metody probíhají v rámci jedné transakce. Databázi lze tedy v rámci jedné transakce změnit. Při další transakci se pracuje opět na původní databázi OLAP serveru.

5.2 OLAP server

Aplikační server je střední vrstvou navrhované aplikace. Navenek bude vystupovat jako tzv. ROLAP server – relační OLAP. Jak bylo navrženo v kapitole 4, bude se jednat o WAR archiv běžící v aplikačním kontejneru Tomcat. WAR archiv ukrývá webovou aplikaci. Ta se konfiguruje souborem web.xml – tato konfigurace určuje mapování Java servletů na URL webové aplikace. Následující fragment kódu popisuje konfiguraci, kterou je potřeba provést pro načtení aplikačního kontextu frameworku Spring:

```
<web-app>
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/applicationContext.xml</param-value>
    </context-param>

    <servlet>
        <servlet-name>context</servlet-name>
        <servlet-class>
              org.springframework.web.context.ContextLoaderServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet>
        <servlet-name>olap-server</servlet-name>
        <servlet-class>
              org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <load-on-startup>3</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>olap-server</servlet-name>
        <url-pattern>/http/*</url-pattern>
    </servlet-mapping>
</web-app>
```

V první části se definuje soubor s aplikačním kontextem – základní konfigurační soubor frameworku Spring, ve kterém se nastavují instance jednotlivých Spring Beans – objektů, jejichž instance Spring spravuje.

Servlet jménem „context“ se stará o načtení konfiguračního souboru Springu pro každý definovaný servlet instance DispatcherServlet. Jméno tohoto souboru je podle konvence olap-server-

servlet.xml. Servlet „olap-server“ je potom mapován na veškeré základy adresy URL, které končí na /http/*.

Dále je potřeba definovat aplikační kontext frameworku Spring. Pro základy a více detailů ohledně jeho definice odkazuji na literaturu [wab08].

V následujících řádcích popíši serverovou implementaci vytvoření datového skladu a provádění OLAP operací.

5.2.1 Vytvoření datového skladu

O administraci datových skladů se na OLAP serveru stará služba, která je implementací rozhraní DataWarehouseService. Vytvoření datového skladu probíhá ve dvou fázích: nejdříve se vytvoří záznamy v tabulkách DATA_WAREHOUSES a DIM_TABLES, které spadají do databáze OLAP serveru.

Potom OLAP server vytvoří databázi pro datový sklad, vygeneruje skript podle konfigurace nového datového skladu a vytvoří databázové schéma datového skladu. Generování SQL skriptu pro definici schématu není triviální a probíhá programově podle objektu DataWarehouse, který přišel spolu s požadavkem z OLAP klienta.

K programovému generování je použit v jazyku Java standardní nástroj StringBuilder, který vytváří řetězec postupným přidáváním podřetězců na konec. Generuje se nejdříve příkaz CREATE TABLE pro všechny tabulky dimenzí, poté CREATE TABLE pro tabulku faktů. Toto pořadí je nutné kvůli vazbě cizích klíčů z tabulky faktů do tabulek dimenzí. Během generování dochází také k různé validaci pro případ, že z OLAP klienta přišla neúplná nebo chybná data. Tato událost se ošetří vyvoláním výjimky s odpovídající zprávou. Výjimka je odchycena v hlavní metodě, kde se její zpráva zadá do výsledku operace Result, který je vrácen klientovi. Klient potom zobrazí chybovou hlášku, která je uživateli srozumitelná.

5.2.2 Implementace OLAP operací

Volání OLAP operací přijímá služba OlapService přítomná na OLAP serveru. OLAP server je zodpovědný za tyto OLAP operace:

- roll-up – posunutí zobrazených dat o úroveň výše v rámci konceptuální hierarchie dimenze
- drill-down – posunutí zobrazených dat o úroveň níže v rámci konceptuální hierarchie dimenze
- pivot (rotate) – otočení dimenzí – změna pohledu na datovou kostku

Služba neposkytuje zvláštní rozhraní pro všechny tyto operace, roll-up a drill-down zpracovává souhrně metodou updateCube(). To, zda se má provést operace roll-up nebo drill-down se udává v datovém modelu ve třídě Dimension. Z této vlastnosti OLAP serveru plyne, že OLAP operace je

možné vzájemně kombinovat, a na jedno volání je server schopen provést OLAP operací vícero. Klient ovšem v základní verzi této výhody nevyužívá.

OLAP server umožňuje OLAP operace pro datové kostky s neomezeným počtem dimenzí pomocí propracovaného generování SQL dotazů. OLAP klient této výhody také zatím nevyužívá, avšak do budoucna je s vyšším počtem dimenzí počítáno, proto je tato funkcionalita v serverové části implementována.

Operace pivot (rotate) je implementována ve zvláštní metodě. Pop přehození dimenzí však využívá však také interní metodu updateInternal() společnou s roll-up a drill-down. V ní je skryto vše podstatné pro změnu datové kostky. Aktualizace datové kostky probíhá ve třech fázích.

V první fázi se inicializují položky dimenzí. Jde o položky, které se v OLAP klientovi zobrazují v záhlavích tabulky. Tedy pro dimenzi čas například leden, únor, březen, atd. v případě, že je vybrán v konceptuální hierarchii měsíc. Tyto hodnoty se získají jednodušším generovaným SQL příkazem SELECT z příslušné tabulky dimenzí.

V druhé fázi dochází k permutacím všech těchto položek dimenzí, které se ukládají pro pozdější použití. Permutace má velikost počtu dimenzí. Následovně ve třetí fázi je potřeba vypočíst hodnoty pro jednotlivé permutace. Hodnota se získá dotazem SQL SELECT s použitím funkce SUM nebo AVG (podle zvolené strategie výpočtu v OLAP klientovi). Generování tohoto SQL příkazu je poměrně složité a opět probíhá s využitím nástroje StringBuilder popsaného v předchozí kapitole. Vygenerované SQL dotazy se postupně spouští a data se ukládají do Hash tabulky, která je připravená v objektovém modelu datové kostky. Klíčem v tabulce je adresa, která je vypočtena z položek dimenzí. Více o výpočtu adres do tabulky viz kapitola 5.6.2. Výsledná datová kostka je potom pomocí Spring HTTP Remoting předána zpět OLAP klientovi, který ji zobrazí.

5.3 Spring Remoting

Jak již bylo popsáno v návrhu, jako rozhraní mezi klientskou a serverovou stranou bude použita jedna z možností podpory Spring Remoting. Nejdříve se blíže podívejme na požadavky, které musí rozhraní splňovat:

- nejedná se o webového klienta – nepřipadají v úvahu rozhraní, která se běžně používají pro webové prohlížeče (např. Struts 2, WebWork, Spring MVC apod.)
- musí se jednat o vhodnou interakci objektů na klientu a na serveru fungující na základě definovaného rozhraní
- musí být možné zasílání objektů
- mělo by fungovat co možná nejjednodušeji – téměř jak kdyby byly obě části na jednom stroji

Poslední požadavek vylučuje webové služby (tzv. Web Services), jejichž konfigurace je poměrně náročná. Uživatel (programátor, klient) se musí zabývat (v kódu) URL webové služby a

další konfigurací. Nicméně i toto řešení je reálné použít. Jako příklad můžeme jmenovat framework AXIS, který využívá SOAP protokolu pro zasílání dat ve formátu XML. Je zde ale nutný tzv. Marshalling a Unmarshalling objektů pomocí knihovny JAXB – jedná se o překlad objektů na XML entity a zpět. K tomu je dále potřeba vytvořit definici XML schématu, což je poměrně náročné. Toto řešení je platformově nezávislé – ve smyslu platformy jako programovacího jazyka. Požadavek na webovou službu není problém zaslat i z programů v jazyku Cobol apod. My tuto možnost ale příliš nevyužijeme.

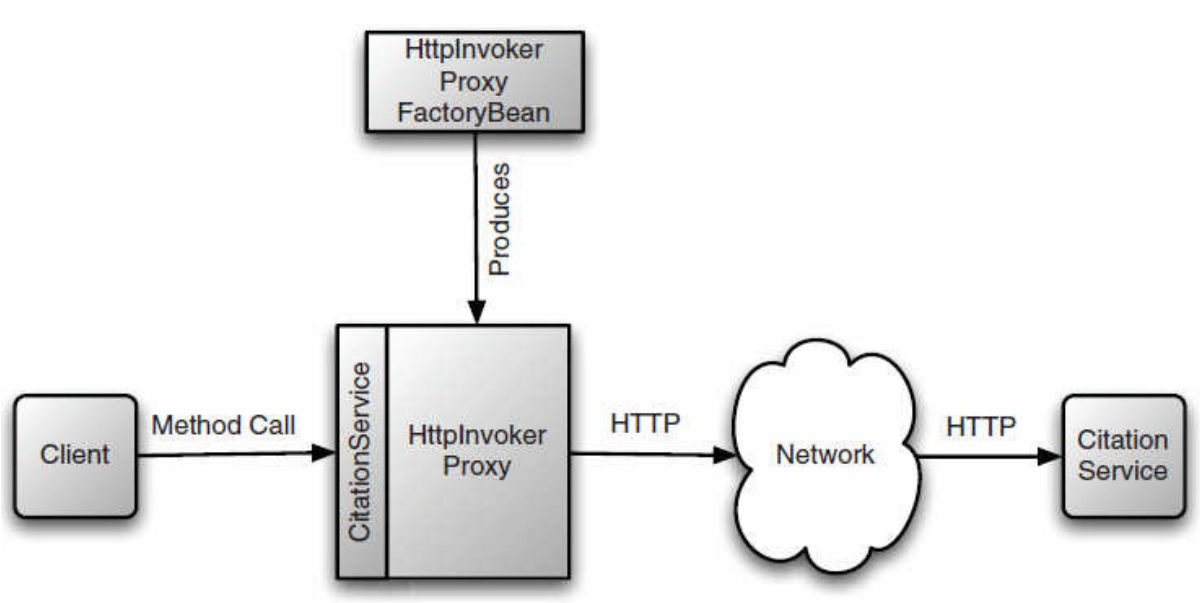
Spring naproti tomu poskytuje množství služeb pro vzdálený přístup [wab08]:

- Remote Method Invocation (RMI) – přístup/vystavování služeb postavených na jazykových knihovnách Java, když nehrají roli síťová omezení (např. firewall)
- Hessian, Burlap – přístup/vystavování služeb postavených na jazykových knihovnách Java, když hrají roli síťová omezení
- HTTP Invoker – přístup/vystavování služeb postavených na frameworku Spring, když hrají roli síťová omezení
- JAX-RPC – přístup/vystavování platformově nezávislých webových služeb postavených na protokolu SOAP

Poslední možnost je podobná frameworku AXIS popsaného v předchozím odstavci. Proto ji nevyužijeme – platformovou nezávislost v našem případě nepotřebujeme. RMI je pro nás nepoužitelné, jelikož používá obecně libovolné síťové porty, a je tedy neprůchodné přes síťová omezení, jako např. firewall. RMI také podporuje pouze primitivní datové typy – neumožňuje tedy zasílání objektů.

Jak Hessian a Burlap, tak Spring HTTP Invoker používají k transportu protokol HTTP, jsou tedy vhodné pro systémy s ochrannými síťovými prvky. Rozdíl mezi nimi je pouze v metodě serializace objektů. My vybereme pro implementaci Spring HTTP Invoker, který využívá standardní Java serializaci objektů.

Způsob zpřístupnění služby klientovi u Spring HTTP Invoker ilustruje následující obrázek:



Obrázek 5.1: Architektura zpřístupnění služby klientovi (převzato z [wab08]).

HttpInvokerProxyFactoryBean vytvoří proxy objekt pro vzdálené volání protokolem postaveným na HTTP. Tento proxy objekt se na klientské straně tváří jako instance služby.

Příklad konfigurace vystavení služby demonstruje následující fragment kódu:

```
<bean name="/SQLQuery" class="org.springframework.remoting.httpinvoker
      .HttpInvokerServiceExporter">
  <property name="service">
    <ref bean="SQLQueryService"/>
  </property>
  <property name="serviceInterface">
    <value>cz.podsednik.olap.service.SQLQueryService</value>
  </property>
</bean>
```

Ke konfiguraci není potřeba moc komentářů. Jméno Spring bean udává suffix adresy URL, na které je služba vystavena. Atribut „service“ udává referenci na instanci služby, která je popsána jinde v aplikačním kontextu. Atribut „serviceInterface“ je odkaz na rozhraní služby, které musí být definováno. Odkazovaná SQLQueryService v předchozím atributu musí toto rozhraní implementovat.

Pro správnou vzájemnou komunikaci klienta a serveru je potřeba také nakonfigurovat klienta, a to také aplikačním kontextem. Tento aplikační kontext obsahuje následující definici:

```
<bean id="SQLQueryService" class="org.springframework.remoting.httpinvoker
    .HttpInvokerProxyFactoryBean">
  <property name="serviceUrl">
    <value>http://localhost:8080/olap-server/http/SQLQuery</value>
  </property>
  <property name="serviceInterface">
    <value>cz.podsednik.olap.service.SQLQueryService</value>
```

```
  </property>
</bean>
```

Instance SQLQueryService potom vznikne v aplikačním kontextu. Atribut „serviceUrl“ udává URL adresu služby vystavené na aplikačním serveru, atribut „serviceInterface“ potom rozhraní služby, jak tomu bylo také v serverové konfiguraci.

Klientská aplikace ale v našem případě nebude webová aplikace – musíme tedy vyřešit jiný způsob inicializace a načtení aplikačního kontextu. Ilustruje to následující příklad:

```
ApplicationContext ctx = new FileSystemXmlApplicationContext(
        "./src/main/resources/applicationContext.xml");
SQLQueryService sqlService = (SQLQueryService)
        ctx.getBean("SQLQueryService");
```

Načteme soubor s aplikačním kontextem a z nakonfigurovaného aplikačního kontextu vrátíme instanci služby. Tento způsob vracení služeb bude použit v OLAP klientu. Vidíme, že způsob vzájemné interakce není nijak složitý a splňuje požadavky uvedené v začátku této kapitoly. Spring HTTP Invoker je vhodný způsob komunikace mezi OLAP klientem a OLAP serverem.

5.4 Eclipse Plug-ins

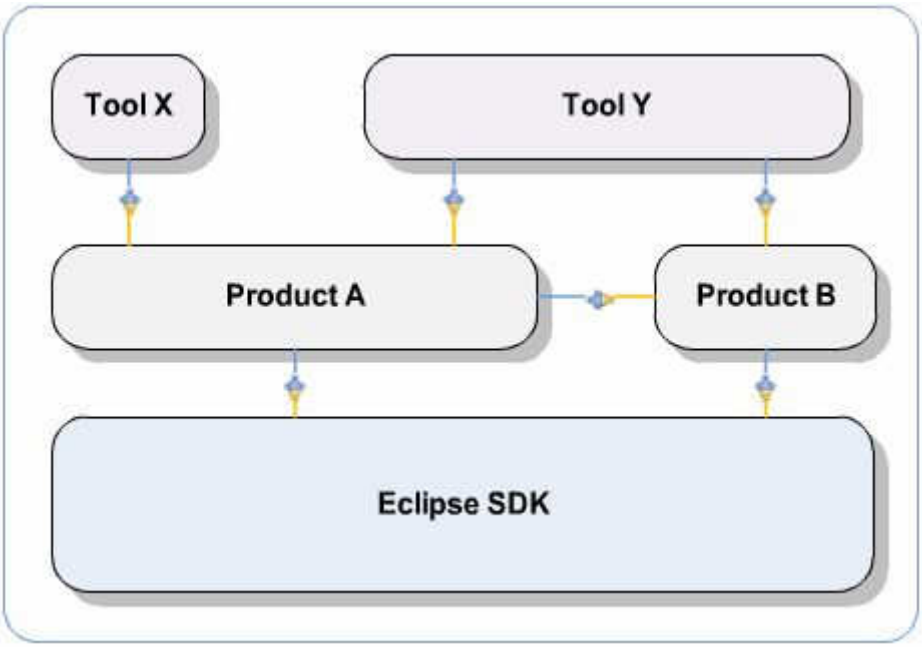
V této kapitole popíši detaily ohledně použité technologie zásuvných modulů do vývojového prostředí Eclipse. Začneme systematicky nejdříve popisem prostředí Eclipse, jeho architektury a pak přejdeme k vlastnímu vývoji zásuvných modulů a RCP aplikací v prostředí Eclipse.

5.4.1 Vývojové prostředí Eclipse IDE

Vývojové prostředí Eclipse zná asi každý vývojář v Jazyku Java. I když je na akademické půdě známější prostředí NetBeans od společnosti Sun, osobně jsem se setkal v množství pracovních pohovorů, které jsem zatím absolvoval, pouze s požadavky na prostředí Eclipse IDE. O rozdílech těchto prostředí by se dalo polemizovat, to ovšem není předmětem této diplomové práce.

Eclipse je vývojové prostředí vyprodukované Eclipse Foundation, což je open-source komunita vývojářů. Projekt Eclipse byl původně vyvinut společností IBM v listopadu 2001 původně jako vývojové prostředí pro Java vývojáře. Postupem času se na tuto platformu začalo přidávat více a více projektů z oblastí statických a dynamických programovacích jazyků (C/C++, PHP aj.), frameworků pro bohaté a tenké klienty a servery, modelovací a business reporting nástroje, podpora mobilních zařízení aj.

Eclipse není velký jednolitý program, ale sestává z množství na sebe navázaných zásuvných modulů (plug-ins). Ilustruje to dobře obrázek 5.2.[eps08]. Nad Eclipse SDK (Standard Development Kit) existují stovky a tisíce modulů, které rozšiřují jeho funkcionalitu. Modul není nic jiného, než další program v Jazyku Java.

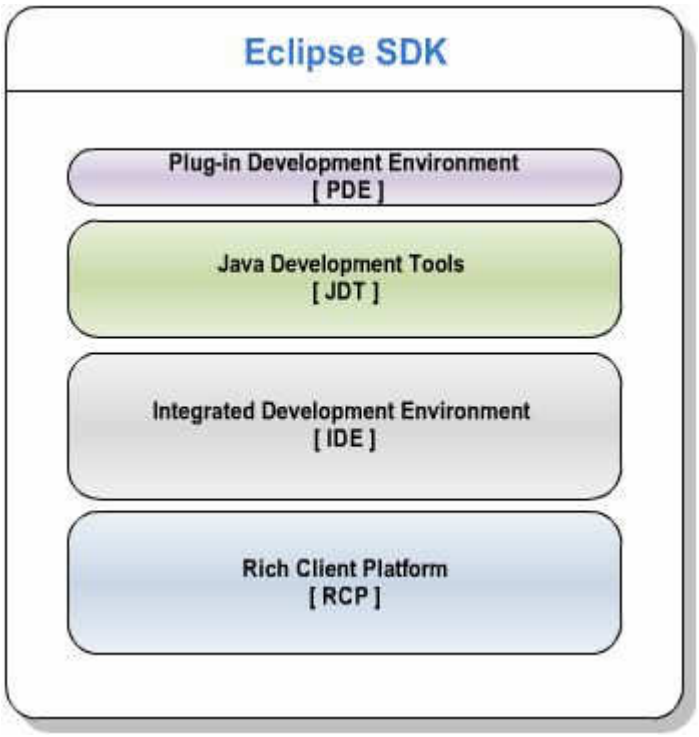


Obrázek 5.2: Modulární členění Eclipse IDE [eps08]

Modul může být závislý pouze na Eclipse SDK nebo i na některých již existujících modulech. moduly jsou dynamicky načítány prostředím Eclipse kdykoliv jsou potřeba. Ve formě modulů se vytváří všechna rozšíření Eclipse (produkty, nástroje). Tyto produkty a nástroje vytváří jednak Eclipse Foundation a jednak komerční společnosti i soukromí vývojáři. Vývoj Eclipse Plug-ins je možný díky Plug-in Development Environment (PDE) – vývojové prostředí pro tvorbu zásuvných modulů do Eclipse.

Prostředí Eclipse SDK, které toto všechno umožňuje se skládá z několika vrstev (viz obrázek 5.3 [eps08]). Spodní vrstvu tvoří Eclipse Rich Client Platform – platforma, nad kterou běží jak vývojové prostředí Eclipse, tak i RCP aplikace, kterými se budeme zabývat v příští kapitole. Další vrstvu tvoří vlastní vývojové prostředí. Vezmeme-li např. klasické Eclipse IDE pro Java vývojáře, jedná se o téměř všechno, co vidíme, kromě toho, co souvisí s programovacím jazykem Java – základní položky v menu, vytváření projektů v kategorii General, nastavení apod.

Nad vrstvou IDE je vrstva Java Development Tools (JDT). Tato vrstva už je specifická pro vývojáře v jazyku Java. Balíčky např. pro vývojáře C/C++ jsou odlišné. Plug-in Development Environment (PDE) je potom nadstavba JDT používaná právě pro vývoj modulů do projektu Eclipse.



Obrázek 5.3: Architektura Eclipse SDK [eps08]

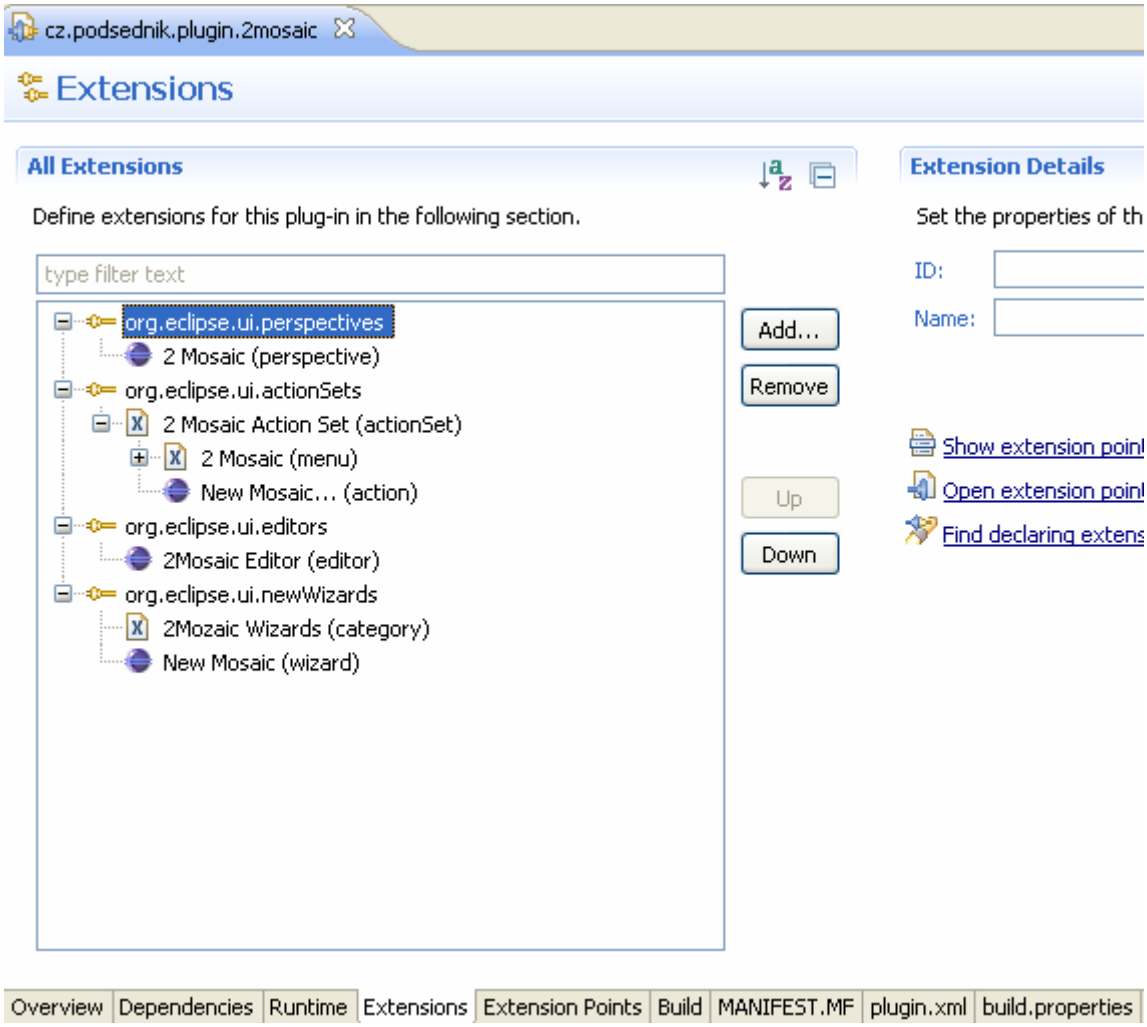
Samozřejmě všechny části Eclipse SDK (RCP, IDE, JDT a PDE) jsou opět tvořeny množstvím zásuvných modulů.

5.4.2 Vývoj zásuvných modulů v Eclipse

Jak jsme už zjistili v minulé kapitole, veškerá funkcionalita (kromě jádra Eclipse) je rozdělena do modulů. Modul (plug-in) je malá jednotka, která může být vyvíjena odděleně od ostatního kódu [eps08]. Modul se typicky distribuuje v JAR archivech, do kterého se zabalí veškerý jeho obsah: přeložené zdrojové soubory, obrázky, další zdrojové soubory (např. XML), ale také popis svého rozhraní a co všechno přináší (položky v menu, akce, pohledy, editory, perspektivy).

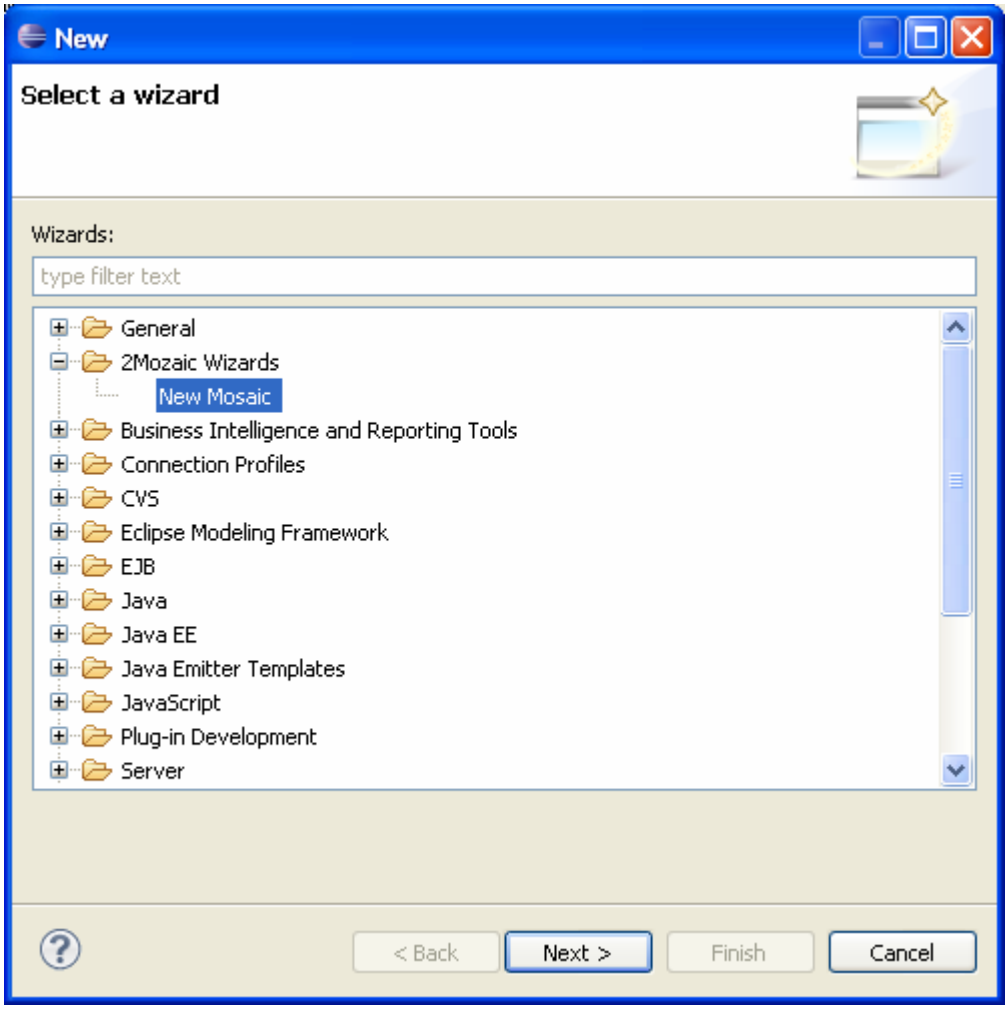
Právě tento popis rozhraní je velmi důležitý a odlišuje zásuvné moduly od ostatních Java aplikací a knihoven. Je definován v souborech plugin.xml a MANIFEST.MF. Zejména první z nich je důležitý: definují se v něm rozšíření (extensions) a body rozšíření (extension points). Vzájemná závislost zásuvných modulů je potom realizována přes ně. Z objektově orientovaného programování známe jiný vztah: dědičnost. U dědičnosti, dědí-li třída B vlastnosti třídy A, pak ji jistým způsobem rozšiřuje – přidává atributy nebo metody (případně je modifikuje). Podobný vztah jako u dědičnosti je u rozšíření a bodů rozšíření. Modul A poskytuje jistý bod rozšíření, zatímco modul B ho rozšiřuje. Chceme-li tedy poskytnout budoucím uživatelům nějaké rozšíření našeho zásuvného modulu, vytvoříme bod rozšíření. Chceme-li rozšířit existující zásuvný modul, vytvoříme rozšíření. Pojďme si tato rozšíření přiblížit na příkladu.

Vytváříme plug-in pro mozaikový editor. Do tohoto editoru vstoupíme přepnutím do speciální perspektivy. Dále bude tento editor sestávat z položky v menu, akce, kterou tato položka vyvolá, vlastního mozaikového editoru a průvodce vytvořením nové mozaiky.



Obrázek 5.4: Editace rozšíření

Eclipse zásuvné moduly mají naštěstí svůj vlastní editor, nemusíme tedy editovat soubory plugin.xml a MANIFEST.MF ručně. V našem příkladu jsou podstatná rozšíření. První z nich se týká vytvoření již zmíněné perspektivy (org.eclipse.ui.perspectives). Dále rozšiřujeme bod rozšíření org.eclipse.ui.actionSets, který obsahuje menu a akci s ním spojenou. Editor mozaiky se vytvoří rozšířením bodu rozšíření org.eclipse.ui.editors a konečně průvodce vytvořením mozaiky vytvoříme rozšířením org.eclipse.ui.newWizards. Tento bod rozšíření se týká průvodců pro vytvoření nového objektu – souboru nebo projektu. Výběr všech průvodců se vyvolá, klikneme-li pravým tlačítkem do pohledu Package Explorer a vybereme-li New->Other…, jak to ilustruje obrázek 5.5. V příkladu jsme vytvořili kategorii 2Mosaic Wizards, do které jsme zařadili našeho průvodce. Výsledek je vidět právě na obrázku 5.5. Podobným způsobem budeme vytvářet průvodce pro nové položky v aplikaci OLAP klienta.



Obrázek 5.5: Výběr průvodců pro nový projekt nebo soubor

Definováním rozšíření práce na modulu nekončí, ale spíše začíná. Definovali jsme nyní rozšíření, které je třeba implementovat. Eclipse za nás vytvoří základní potřebné třídy, my do nich musíme ale doplnit to, jak budou komponenty vypadat a co mají dělat.

Uveďme nyní přehled hlavních grafických komponent, kterých jsme se dotkli, a dalších:

- **Menu –** jedná se o položku v menu, s kterou je spjata nějaká akce
- **Tlačítko panelu nástrojů (Toolbar button)** – tuto položku netřeba vysvětlovat, taktéž s ní může být spjata nějaká akce
- **Akce (Action)** – toto není přímo grafická komponenta, ale logická komponenta, jejíž chování uživatel implementuje. Typicky se jedná o provedení nějaké business logiky, případně inicializace a zobrazení další komponenty.
- **Pohled (View)** – pohled je komponenta zobrazující cokoliv, co se neukládá do souboru. Pohled je např. Package Explorer, který zobrazuje projekty, jejich balíčky a zdrojové soubory, nebo také Console, která zobrazuje konzoli spuštěné aplikace ve vývojovém prostředí. Jeho použití je široké a jedná se jednu ze dvou základních

jednotek pracovní plochy (workbench), tj. oblasti pod hlavním menu a panelem nástrojů.

- **Editor (Editor)** – druhá ze základních jednotek pracovní plochy. Tato komponenta typicky zpracovává nějaký soubor a umožňuje jeho uložení. Obecně lze však v jednom editoru upravovat více souborů najednou – například právě v již zmíněném editoru zásuvných modulů, kde se najednou modifikují soubory plugin.xml, MANIFEST.MF a další soubory, které souvisí s konfigurací zásuvného modulu. Je však potřeba podotknout, že tento případ je spíše výjimkou – typické příklady editorů jsou spíše Java Editor (editor pro editaci Java tříd), XML editor (pro editaci XML souborů), text editor (pro editaci obyčejných textových souborů) a mnoho dalších.

V této kapitole jsme si nastínili základy vývoje zásuvných modulů v Eclipse, které je třeba znát před tím, než se pustíme vůbec do jejich implementace. Další zmínka vývoji zásuvných modulů je v kapitole 5.6, která se bude zabývat implementací konkrétního zásuvného modulu pro OLAP klienta. Další informace o Eclipse modulech lze čerpat z průvodce [eps08] a literatury [clr06].

5.4.3 Zásuvný modul nebo RCP aplikace?

V kapitole o návrhu jsme zmiňovali, že OLAP klient bude sestaven buď jako zásuvný modul vývojového prostředí Eclipse (Eclipse plug-in) jako rozšíření stávajícího prostředí, nebo jako samostatná RCP aplikace. Nyní je vhodné popsat si rozdíl mezi oběma možnostmi.V kapitole 5.4.1 na obrázku 5.3 je zobrazeno členění Eclipse SDK do vrstev. Spodní vrstvu tvoří RCP aplikace. Na RCP platformě běží tedy celé vývojové prostředí Eclipse. Může na ní také běžet náš zásuvný modul bez jakýchkoli dalších zásuvných modulů, na kterých není závislý. Celá RCP aplikace bude tedy obsahovat pouze vyvíjený modul a ty moduly, které jsou deklarovány v editoru modulů v záložce Extensions (viz obrázek 5.4) nebo v záložce Dependencies (o této konfiguraci jsme se ještě dříve nezmiňovali. Jedná se o definici všech modulů, na kterých je náš modul závislý, ale nerozšiřuje je. Je to tedy další forma závislosti.).

Taková aplikace, pokud je prázdná, zobrazí pouze prázdné okno RCP aplikace. Naproti tomu rozšiřujeme-li existující vývojové prostředí Eclipse, objeví se nám při zkušebním spuštění tatáž obrazovka s vývojovým prostředím obohacená o náš modul.

Rozdíl mezi RCP aplikací a klasickým modulem je také v jejich distribuci: RCP aplikaci bychom chtěli distribuovat jako JAR archiv s celou fungující aplikací, zásuvný modul můžeme distribuovat jako JAR archiv bez RCP platformy a modulů, na kterých je závislý. Tento JAR archiv se po stažení (ať už automatickém přes Eclipse či manuálním z webu) nakopíruje do adresáře plugins v domovském adresáři vývojového prostředí a funguje při dalším spuštění (pokud jsou ovšem přítomny moduly, na kterých je náš plug-in závislý (deklarované v záložkách Dependencies a Extensions editoru modulů).

Výhodou RCP aplikace je tedy její nezávislost na Eclipse IDE. Naproti výhodou obyčejného zásuvného modulu je jeho následující rozšiřitelnost a malá velikost při distribuci. Jeho uživatel může také během používání OLAP klienta jako zásuvného modulu používat další nástroje, které projekt Eclipse poskytuje. To je jeden z důvodů, proč jsem se rozhodl pro variantu zásuvného modulu místo RCP aplikace při implementaci OLAP klienta. Další důvod je také návaznost na produkt BIRT, který popíši v další kapitole.

Z hlediska vývoje jsou si RCP aplikace i zásuvný modul velmi podobné: oba začínají průvodcem pro vytvoření nového zásuvného modulu, v kterém se možnost „Is an RCP application“ nastaví na příslušnou hodnotu ano nebo ne. I následující vývoj pokračuje podobně, vytváří se stejné komponenty a v obou případech je možná závislost na dalších zásuvných modulech.

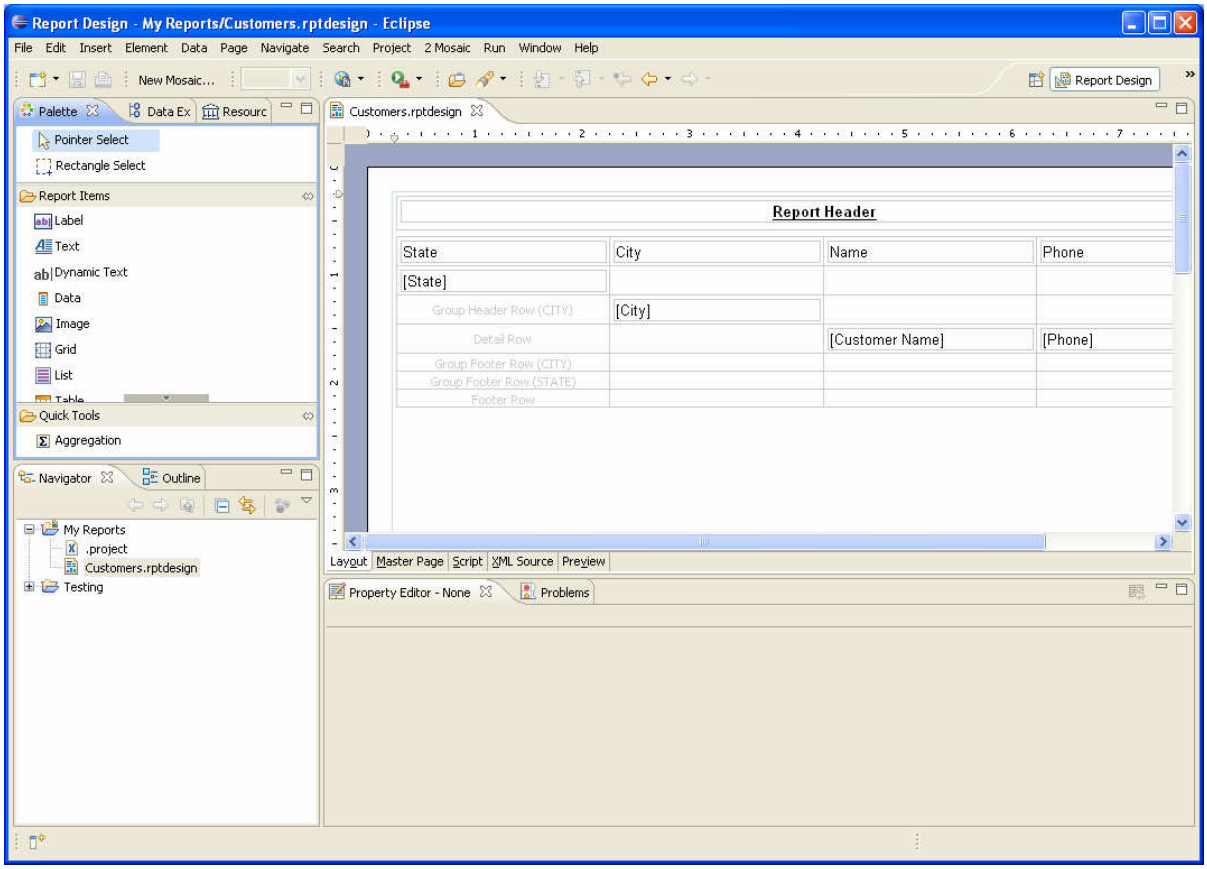
5.5 Business Inteligence Reporting Tools

Business Inteligence Reporting Tools (zkráceně BIRT) je další z open-source projektů vyvíjený Eclipse Foundation na platformě Eclipse. Jeho účelem je tvorba tiskových sestav (reports) z databází. Ačkoliv se jedná o produkt s odlišným účelem, než je OLAP klient, daly by se jeho komponenty využít jako základ pro implementaci OLAP klienta. V této kapitole popíši vlastnosti prostředí BIRT a možnosti použití jeho komponent.

BIRT sestává ze dvou hlavních částí: BIRT Designer pro editaci reportů a BIRT Engine, který je možné integrovat s aplikačním serverem a zobrazovat na něm vytvořené tiskové sestavy. Klíčovou komponentou, která nás bude zajímat, je BIRT Designer, které obsahuje prvky grafického uživatelského rozhraní v OLAP klientovi. BIRT Engine by šel určitě také rozšířit na OLAP server, jednodušší bude ale jistě implementovat OLAP server tzv. „na zelené louce“, protože BIRT Engine byl vytvořen za jiným účelem, než je vytváření datových skladů a zobrazování dat v nich.

5.5.1 BIRT Designer

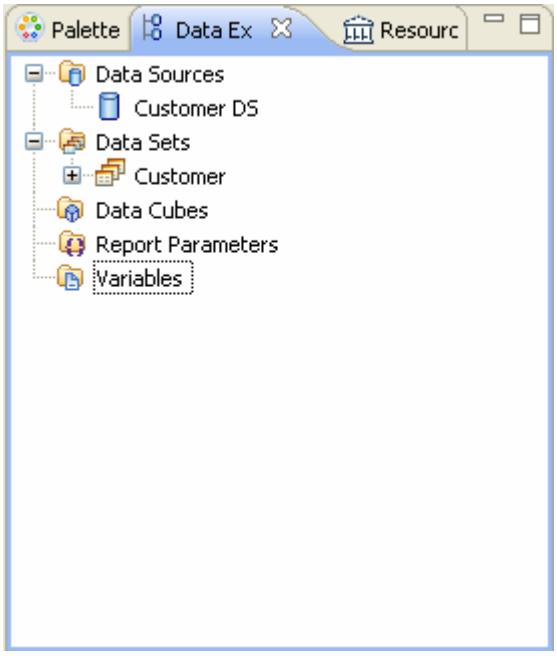
Základem BIRT Designer je perspektiva „Report Design“. Ta poskytuje potřebné pohledy a editor pro editaci tiskové sestavy. Podívejme se nyní na perspektivu Report Design blíže – Obrázek 5.6 ji zobrazuje.



Obrázek 5.6: Report Design perspektiva v BIRT Designer

Vlevo nahoře se nachází tři různé pohledy (views):

- Palette – tento pohled slouží jako panel komponent pro návrh tiskové sestavy. Komponentu umístíme do editu sestavy přetažením. Je spíše specifický pro tiskové sestavy, proto ho při implementaci OLAP klienta nevyužijeme.
- Data Explorer – tento pohled se zabývá zobrazením datových komponent a bude použitelný i pro OLAP klienta. Jeho obsah vidíme na obrázku 5.7: jedná se o strom datových komponent jako datové zdroje (připojení k databázím), Data Sets (prakticky se jedná o uložené SQL SELECT příkazy, tedy něco jako pohledy v databázi), Data Cubes (datové kostky – jedná se ovšem o zobrazení datových kostek v reportu a přímé spojení s databází, nikoli přes OLAP server – pro OLAP klienta je nutné značně přepracovat), parametry reportu a proměnné (pro OLAP klienta nevyužitelné).
- Resource Explorer – jednoduchý pohled pro zobrazení konfiguračních souborů s reporty ve workspace, který se bude jistě hodit i pro zobrazení konfiguračních souborů pro datové sklady kostky.



Obrázek 5.7: Data Explorer View

V perspektivě (obrázek 5.6) se dále nachází vlevo dole pohledy Navigator (zobrazení projektů a souborů v nich) a Outline, běžné pohledy známe z Java perspektivy, které také využijeme. Pohled Outline shrnuje projekt reportu a zobrazuje všechny datové zdroje, pohledy, kostky a další parametry reportu.

Ve spodní části se ještě nachází pohled Property Editor, který upravuje některé parametry reportu (v OLAP klientovi využitelné pro volbu parametrů) a pohled Problems známý z Java perspektivy.

V hlavní (editorové) části je potom vícestránkový editor reportů, který není pro OLAP klienta využitelný. Místo toho zde bude podobný editor, který bude na jedné stránce zobrazovat OLAP kostku a na další stránce bude možnost editovat přímo konfigurační soubor, jak je tomu i v tomto případě. Více informací o užívání nástroje BIRT je možné získat na jeho domovské stránce: http://www.eclipse.org/birt/

5.5.2 Prvky využitelné při implementaci

Většina využitelných prvků již byla popsána v minulé kapitole. Nyní se budeme zabývat jejich detailním popisem a konkrétními možnostmi využití. Pro prohlížení zdrojových souborů BIRTu je nutné si všechny jeho projekty stáhnout z jeho CVS repository – sestává ze 106-ti podstatných projektů (kromě testovacích projektů a příkladů), což zabere značnou dobu. Tento úkon včetně jeho kompilace a sestavení je popsán na jeho domovské stránce (viz výše uvedený odkaz).

Nejdříve se zabývejme využitelnými pohledy. BIRT pohledy jsou definovány v modulu org.eclipse.birt.report.designer.ui.views. První zmíněný využitelný pohled je Data Explorer. Třída s jeho definicí se jmenuje DataView. V případě, že není vybraný žádný report, je tento pohled

prázdný, jinak se v něm zobrazí DataViewTreeViewerPage se stromem parametrů týkajících se dat. Další zmíněný pohled je Ressource Explorer, který popisuje třída LibraryExplorerView. Vytvoříme podobný pohled pro konfigurační soubory OLAP klienta.

Standardní pohledy Navigator a Outline je možné přidat do OLAP perspektivy pod jejich ID, které získáme z konstant IPageLayout.ID_RES_NAV a IPageLayout.ID_OUTLINE. Standardní pohled Problems má ID IPageLayout.ID_PROBLEM_VIEW a potenciálně použitelný pohled BIRTu Property Editor se skrývá pod Java třídou AttributeView.

Každá perspektiva má svoji třídu implementující rozhraní IPerspectiveFactory, která udává základní chování perspektivy a rozložení pohledů a editorů v ní. U Report Design perspektivy se tato třída jmenuje ReportPerspective a můžeme se jí inspirovat při rozložení pohledů a editorů v OLAP perspektivě.

5.6 OLAP klient

Tato kapitola popisuje vlastní implementaci OLAP klienta s využitím komponent nástroje BIRT, které byly popsány v minulých kapitolách. Implementace klienta je rozdělena na části: kontrukce OLAP perspektivy, nástroje pro návrh datového skladu, pohled pro zobrazení OLAP kostek, OLAP server a jeho rozhraní a nakonec sestavení aplikace a její deployment. Implementace řešila samozřejmě více úkolů, ale tyto patří k těm nejzajímavějším.

5.6.1 Vytváření OLAP perspektivy

V prvním kroku vytvoříme OLAP perspektivu a pohledy v ní. Jak je již popsáno v kapitole 5.4.2, perspektiva se vytváří rozšířením bodu rozšíření org.eclipse.ui.perspectives. Vytvoříme tedy novou perspektivu OLAP Perspective a vytvoříme třídu OlapPerspectiveFactory, která bude řídit rozložení komponent v perspektivě. Umístíme do ní zatím existující pohledy Navigator a Outline.

Dále vytvoříme pohledy Data Explorer a Resource Explorer specifické pro OLAP projekty. Před tím je ale potřeba vytvořit nový typ projektu, který nazveme OLAP Project. Pro nový typ projektu je potřeba průvodce rozšiřující bod rozšíření org.eclipse.ui.newWizards popsaný v kapitole 5.4.2. Další průvodce bude potřeba na vytvoření nového OLAP souboru. Podobně jako v BIRT se bude jednat o soubor sdružující veškerou konfiguraci k projektu – datové zdroje, pohledy, datový sklad, OLAP kostka.

Po vytvoření průvodce přistoupíme k implementaci editoru, což bude vícestránkový editor. Na první stránce bude zdrojový XML soubor popisující konfiguraci OLAP klienta. Jeho součásti jsou popsány v následující kapitole.

Pro konverzi XML konfiguračního souboru do datového modelu a zpět jsem použil technologii JAXB (Java API for XML Binding). V jazyku Java je manipulace s XML souborem poměrně náročná. Používají se zejména dva přístupy – SAX, což je jednoprůchodový syntaktický analyzátor, a

DOM, který zpracovává XML soubor objektově. Oba tyto přístupy jsou poměrně náročné na použití, nehledě na to, že každý element (ať už v tom či onom přístupu) je nutné zpracovávat zvlášť, což je extrémně programátorsky náročné.

JAXB naproti tomu umožňuje automatický převod XML souboru do doménového modelu a zpět. Java třídy tvořící doménový model musí mít speciální anotace, které signalizují, s kterým XML elementem jsou svázány. Naštěstí Java poskytuje další nástroj, XJC. XJC umožňuje vygenerovat Java třídy doménového modelu včetně potřebných anotací nad atributy tříd z XML schématu. Toto schéma je popsáno v následující kapitole, jeho detailní podoba je potom v příloze 5. Pro více informací o nástroji JAXB odkazuji na následující webovou stránku průvodce JAXB:

http://java.sun.com/webservices/docs/1.6/tutorial/doc/JAXBUsing.html

Na dalších stránkách OLAP editoru jsou zobrazeny jednotlivé OLAP kostky definované v konfiguraci. Protože je zobrazování řešenou tabulkou, jsou v této verzi OLAP klienta povoleny pouze dvojdimenzionální datové kostky. Datový model je ale navržen obecně (viz následující kapitola). Více informací o zobrazování OLAP kostek poskytuje kapitola 5.6.4.

5.6.2 Model

V souvislosti s tvorbou editoru je už potřebné vytvořit třídy pro uložení dat a určitých nastavení editoru, datových zdrojů, pohledů, datového skladu a kostek. Tato nastavení se potom budou ukládat do XML souboru s nastavením. Ten má určitý, přesně definovaný, formát.

Definujme nyní informace, které je potřeba uložit. Lze vycházet také z formátu XML souboru pro uložení nastavení prostředí BIRT (BIRT má také datové zdroje a datové pohledy). Nastiňme tedy nyní hierarchii XML souboru :

olap-design – kořenový objekt

- **property** – jakákoli vlastnost, datový typ je ovšem omezen na String (řetězec). Řetězce property se klíčují taktéž řetězcem.
- **data-sources** – množina datových zdrojů.
  - **data-source** – reprezentuje datový zdroj. Obsahuje informace jako URL datového zdroje, jméno databáze, uživatelské jméno, heslo, jméno datového zdroje aj.
- **data-sets** – množina datových pohledů.
  - **data-set** – jednotlivý pohled. Obsahuje jméno, SQL SELECT příkaz, odkaz na datový zdroj, případně část dat datového pohledu pro zobrazení. Může také obsahovat jména sloupců a aliasů.
- **data-warehouses** – množina datových skladů. Obecně se předpokládá, že datový sklad bude pouze jeden.

data-warehouse – datový sklad. Obsahuje jméno datového skladu a databáze, která ho jednoznačně na ROLAP serveru identifikuje. Obsahuje také další podelementy.

fact-table – jméno tabulky faktů a její sloupce. U sloupců jsou odkazy do určité tabulky dimenzí. V případě, že tento atribut není definován, bere se v potaz, že se jedná o určitou míru, která se má objevovat při zobrazení datové kostky.

dimension-tables – množina tabulek dimenzí.

dimension-table – tabulka dimenzí. Tento element obsahuje jméno tabulky dimenzí a její sloupce včetně označení primárního klíče. Také v sobě obsahuje skrytou konceptuální hierarchii vzájemnými odkazy mezi sloupci.

commands – příkazy sloužící k naplnění datového skladu

command – příkaz sloužící k naplnění datového skladu. Obsahuje fragment příkazu insert a odkaz na datový pohled (data set).

data-cubes – množina datových kostek.

data-cube – datová kostka.

dimensions – množina dimenzí.

dimension – symbolizuje jednu dimenzi v datové kostce. Můžeme si ji představit jako momentální zobrazení záhlaví jedné dimenze v tabulce (uvažujeme-li dvě dimenze). Obsahuje seznam položek **item**, které vyjadřují hlavičky v záhlaví (např. čas = rok 2009 pro dimenzi čas). Tyto položky také obsahují část adresy do Hash tabulky se zobrazenými daty.

cached-data – momentálně zobrazené informace obsažené v datové kostce. Tvoří je klíčovaná Hash tabulka, kde klíčem je vypočtená adresa agregovaného faktu.

Předchozí popis detailně ilustruje většinu podstatných tříd a podtříd užitých coby dočasného (Java třídy) i trvalého (XML konfigurační soubor) úložiště napříč celou aplikací. Zvlášť podstatná je třída (element) data-cube/DataCube, která reprezentuje datovou kostku. Tato třída bude odesílána na ROLAP server a zpět, kdykoliv uživatel zadá požadavek pro expanzi řádků (operace drill-down) nebo operaci roll-up. ROLAP server potom provede příslušnou operaci a vrátí její výsledek.

Můžeme uvažovat o dvou možnostech zobrazování OLAP operací:

- hierarchické – hlavičku zobrazovací tabulky datové kostky v klientovi tvoří strom, kde každý uzel je položka dimenze na různé úrovni abstrakce. Uživatel potom může expandovat jednotlivé uzly a analyzovat jejich detail. Při operaci drill-down tedy OLAP server nevrátí lineárně všechny řádky tabulky faktů podle konceptuální hierarchie níže, ale pouze ty, které jsou potomkem expandovaného uzlu dimenze. Nevýhodou tohoto přístupu je nemožnost expandovat všechny uzly najednou (zvýšení/snížení úrovně abstrakce pro všechny položky dimenze na stejné úrovni abstrakce). Pokud by byla tato možnost programově zařízena, je zde nutné počítat všechny uzly-potomky zvlášť – nelze vrátit nižší úroveň na jeden příkaz SQL SELECT. Dále je tato možnost poměrně náročná na zobrazení (viz níže).
- lineární – hlavičku zobrazovací tabulky datové kostky v klientovi tvoří prosté položky. Operace roll-up a drill-down se provádi pro všechny položky dimenze – celá datová kostka se na určité dimenzi posune o úroveň abstrakce níže nebo výše. Výhodou tohoto řešení je snadnější implementace, snadnější zobrazení a efektivnější (a rychlejší) práce s databází (tedy rychlejší odezva OLAP serveru).

Pro implementaci jsem vybral druhou možnost, neboť se domnívám, že výhody u ní převažují nad nevýhodami. Hierarchické zobrazování OLAP operací umožňuje více možností zobrazení uživateli, je ale – jak již bylo řečeno – náročné jak na zobrazení, tak pochopení uživatelem. Tabulka datové kostky by musela obsahovat v hlavičkách strom, což by jí činilo nepřehlednou. V případě rozšíření OLAP klienta o zobrazování tří a vícedimenzionálních datových kostek by byla hierarchická možnost zobrazení téměř neřešitelná a pro uživatele naprosto nečitelná.

O hierarchickém zobrazení jsem nicméně nadále uvažoval a definoval jsem XML schéma uvedené v příloze 5 (olapdesign.xsd). Dříve v příloze 5 je uvedeno také XML schéma doménového modelu pro linární zobrazení. Porovnejme nyní obě možnosti.

Z velké části se obě schémata neliší, liší se pouze reprezentace datové kostky. Zatímco v hierarchickém zpracování obsahuje element typu Dimension kořenový uzel stromu položek dimenzí, v lineárním zpracování obsahuje pouze lineární seznam položek (items). U lineárního zobrazení je informace o požadované operaci uložena přímo v typu Dimension, u hierarchického zobrazení musí být tato informace uložena přímo v uzlu (Node), protože různé uzly mohou být různě expandovány. Společné mají oba přístupy uložení dat v Hash tabulce. Hash tabulka vlastně simuluje pole, kde jsou indexy řetězce tvořené následujícím způsobem:

```
dim_table|dim_col1_val|dim_col2_val|...|dim_coln_val
```

`dim_table` je název tabulky dimenzí (důležité je rozlišit, o jakou dimenzi se v indexu jedná) a `dim_colx_val` jsou hodnoty sloupců v tabulce dimenzí, číslo x označuje stupeň sloupce v konceptuální hierarchii dimenze. Celkový index (adresa) do Hash tabulky se potom tvoří složením klíčů jednotlivých dimenzí následujícím způsobem:

```
index_dim_1;index_dim2;...
```

Oddělovačem je v tomto případě středník. Vidíme, že indexování v Hash tabulce s daty silně připomíná indexování vícerozměrného pole. Hash tabulka je nejobecnější reprezentace tohoto pole, protože může mít obecně n dimenzí, aniž by se musel měnit její datový typ. Jak bylo popsáno v minulé kapitole, k převodu XML reprezentace z editoru do datového modelu a zpět je použita technologie JAXB. A právě při kombinace XML a JAXB je uložení do vlastního datového typu Hash tabulky tím nejjednodušším řešením řešením. Následující příklad ilustruje uložení Hash tabulky v XML souboru:

```
<cachedData>
  <entry key="time|2009|10;branch|USA|Los Angeles;">1800</entry>
  <entry key="time|2009|10;branch|USA|San Francisco;">1500</entry>
  <entry key="time|2009|11;branch|USA|Los Angeles;">1200</entry>
  <entry key="time|2009|11;branch|USA|San Francisco;">400</entry>
</cachedData>
```

5.6.3 Návrh datového skladu

Datový sklad vytvoříme nejjednodušším možným způsobem: pomocí konfiguračního souboru, jehož schéma jsme v předchozí kapitole definovali. Ten se pak přeloží na OLAP serveru na SQL skript, který datový sklad vytváří.

V budoucnu se samozřejmě může OLAP klient rozšířit o editor datového skladu (např. vizuální), pro potřeby jednoduchého OLAP klienta ale postačí tato možnost.

Pomocí XML konfigurace uživatel zadává zejména tyto parametry:

- název datového skladu – název datového skladu je shodný s názvem databáze, v které bude datový skladu uložen. Tento název musí tvořit výhradně písmena (velká nebo malá).
- tabulka faktů – její název, názvy sloupců a datové typy. Je také nutný údaj o primárním klíči a záznamy o odkazech do tabulek dimenzí, které se použijí jako cizí klíče.
- tabulky dimenzí – výčet tabulek dimenzí, který opět obsahuje název, názvy sloupců, datové typy a údaj o primárním klíči. Protože je podporováno pouze schéma hvězdy, není možné do těchto tabulek uvádět údaje o cizích klíčích.

Uživatel může kdykoli změnit nastavení datového skladu. Pokud změní název, OLAP klient zobrazí potvrzovací dialog, zda vytvořit nový datový sklad. Při změně jména se totiž systém nemůže rozpoznat, zda uživatel zamýšlel sklad přejmenovat, nebo jedná-li se o konfiguraci skladu nového.

V případě změny nastavení datového skladu se stejným jménem je nutné datový sklad smazat a znovu vytvořit. V takovém případě uživatel ovšem přijde o všechna data v datovém skladu uložená, na což ho OLAP klient náležitě upozorní. Před vytvořením skladu dochází k validaci. Zjišťuje se, zda už takový datový sklad neexistuje (potom je uživateli nabídnuto jeho smazání a znovuvytvoření) a za

je název datového skladu tvořen pouze písmeny. Chceme-li se připojit k již existujícímu skladu, musíme zadat přesnou jeho konfiguraci – musí sedět jeho jméno a jména tabulek faktů a dimenzí. V případě, že se jména tabulek neshodují, nabídne klient smazání a znovuvytvoření daného datového skladu.

Mazání datového skladu probíhá ve dvou fázích. V první fázi se vymažou jeho údaje v databázi OLAP serveru (údaje v tabulkách DATA_WAREHOUSES a DIM_TABLES) a v druhé fázi dojde ke zrušení databáze datového skladu (příkaz DROP DATABASE).

Nový datový sklad se vytvoří opačným postupem: OLAP server vloží řádek nejprve do tabulky DATA_WAREHOUSES (tabulka obsahuje jméno datového skladu a jméno tabulky faktů) a potom do tabulky DIM_TABLES (záznamy o tabulkách dimenzí vázané na datový sklad). Potom OLAP server vytvoří novou stejnojmenou databázi pro datový sklad.

Dále dojde ke konstrukci SQL skriptu na vytvoření databáze podle zadaných parametrů. Tyto parametry se na klientovi nejdříve přeloží z XML formátu do objektového modelu jazyka Java pomocí nástroje JAXB popsaného v kapitole 5.6.1. a potom se odešlou na OLAP server pomocí technologie Spring HTTP Remoting. Framework Spring zavolá serverovou službu a ta provede výše popsanou transformaci do SQL skriptu, který se spustí pomocí technologie JDBC. Tato technologie a její podpora ve Spring framework je podrobně popsána v kapitole 5.1.1. Více o transformaci do SQL skriptu viz kapitola 5.2.1.

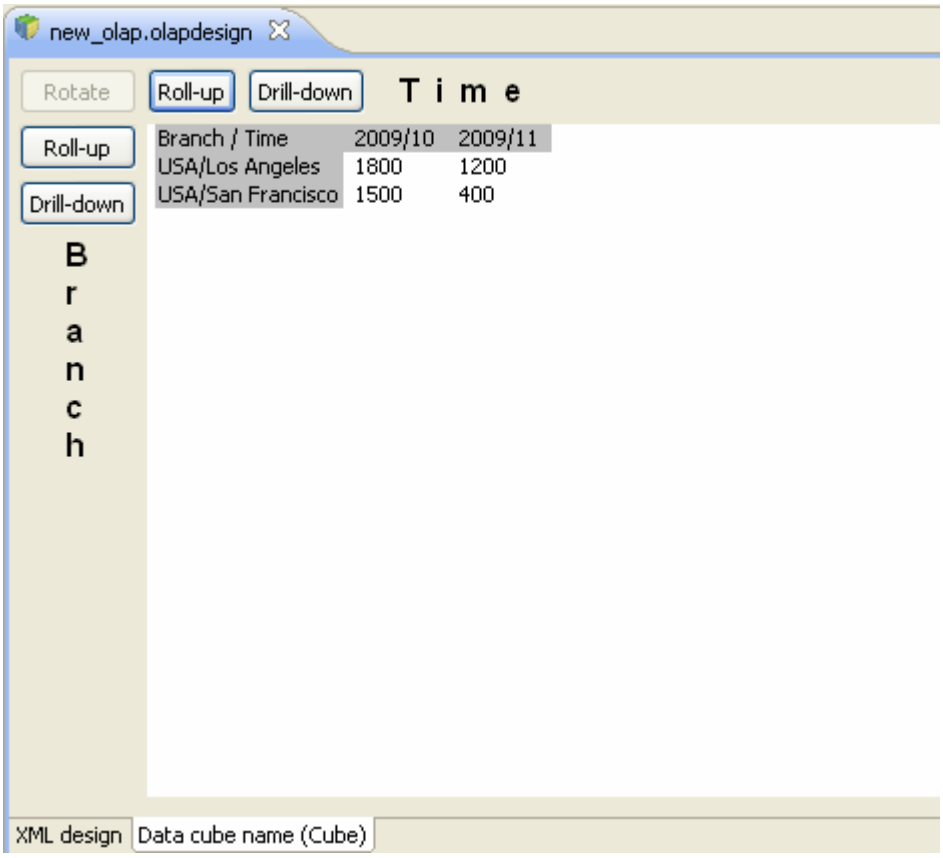
5.6.4 Zobrazování OLAP kostek

Hlavní funkcí OLAP klienta je zobrazování OLAP datových kostek. Pro toto zobrazení jsem vybral tabulkovu reprezentaci. Pro každou OLAP kostku definovanou v konfiguračním souboru se do OLAP editoru přidá stránka, která ji zobrazuje. Jednoduchou OLAP kostku postavenou na databázi z literatury [han06] můžeme vidět na obrázku 5.8.

Zvolené dimenze jsou čas a pobočka – jejich popis je uveden jak v záhlaví tabulky, tak na osách každé dimenze. Dostupná jsou také tlačítka pro OLAP operace – roll-up a drill-down pro každou dimenzi. V levém horním rohu je také tlačítko pro operaci Pivot (rotate) sloužící k otočení datové kostky (prohození dimenzí).

Stránky OLAP kostek se dynamicky přidávají. Kdykoliv překlikneme do XML konfigurace a přidáme další kostku, vytvoří se její stránka včetně tabulkového zobrazení. Tato operace se aktivuje opět překliknutím na stránku datových kostek, nebo uložením XML souboru.

OLAP operace vykonává OLAP server. Při kliknutí na tlačítko OLAP operace se vyvolá akce, která pomocí rozhraní Spring HTTP Remoting odešle aktuální datovou kostku na OLAP server s požadavkem na provedení dané operace. Služba na OLAP serveru vrátí zpět datovou kostku s výsledkem. O serverové straně operací se zmiňuje kapitola 5.2.2.



Obrázek 5.8: Jednoduchá datová kostka

5.6.5 OLAP server a jeho rozhraní

V kapitole 5.3 jsme si popsali technologii Spring Remoting, která slouží jako rozhraní mezi OLAP klientem a OLAP serverem. Princip spočívá ve vytvoření samostatného Java projektu pro rozhraní, který nazveme olap-interface. Tento projekt obsahuje všechny sdílené třídy a rozhraní. Jedná se především o třídy modelu aplikace (všechny objekty, které se posílají z klienta na server a zpět) a rozhraní služeb poskytovaných OLAP serverem.

Na tomto projektu rozhraní je závislý jak projekt klienta, tak projekt OLAP serveru. Podle rozhraní vytvoří Spring HTTP Remonting na klientovi proxy instanci služby, která se chová, jak kdyby byla umístěna přímo na klientovi. Tento způsob vzdálené komunikace je nesmírně elegantní, neboť schovává kompletně před programátorem síťovou komunikaci. Ta je nakonfigurována v XML souboru aplikačního kontextu, jak bylo popsáno v kapitole 5.3. Na serveru potom framework skutečnou instanci dané služby zavolá, získá její výsledek, který pošle zpět na klienta. Tam výsledek vrátí výše popsaná proxy služba.

Instance služeb na serveru jsou také administrovány pomocí Spring Frameworku. Ten se stará o jejich vytváření a životní cyklus. Obecně platí, že služba (v terminologii frameworku Spring bean) je jediná v rámci jednoho vlákna. Jednotlivé Spring beans se podle konfigurace v souboru aplikačního kontextu spolu navzájem provážou.

V aplikačním kontextu definujeme služby, jejich třídy, identifikátory v rámci aplikačního kontextu, vlastnosti (properties), které se mají nastavit a někdy také argumenty kontruktoru pro vytvoření instance, pokud jsou potřeba. Spring framework automaticky vytvoří instance popsaných Spring beans a prováže je v případě, že jedna služba navazuje na druhou. Poradí si i s případným cyklem, který v grafu závislostí může vznikat. Této strategii se říká **Dependency Injection** a je považována za návrhový vzor. Český překlad by zněl nepěkně, anglický výraz je ale samovysvětlující: framework sám injektuje závislosti do definovaných instancí.

Rozsah vytvořené instance také Spring framework umožňuje definovat: jedna možnost již byla popsána výše – atribut scope=”singleton” definuje pouze jednu instanci v rámci jednoho vlákna. Pokud tento atribut změníme na “prototype”, bude platnost jedné instance pouze pro jeden požadavek klienta. To se hodí zejména u webových aplikací, kde jistě nechceme, aby si jeden požadavek pamatoval informace (např. stejně nazvaná pole formulářů) z druhého. Pro další požadavek se vytvoří další prázdná instance. Pro aplikaci v OLAP klientovi budou stačit Spring beans typu singleton. Ty mají podobné chování jako návrhový vzor Singleton, ovšem v rámci jednoho vlákna.

Implementace vytvoření datového skladu v rámci OLAP serveru byla již popsána v minulé kapitole a v kapitole 5.1.1. O serverové straně OLAP operací pojednává kapitola 5.2.2.

5.6.6 Sestavení aplikace a deployment

Jak již bylo rozhodnuto v kapitole 5.4.3, OLAP klient se bude distribuovat jako JAR archiv se zásuvným modulem jako rozšíření vývojového prostředí Eclipse a prostředí BIRT. V editoru zásuvného modulu tedy musíme nadefinovat všechny závislosti na BIRT tak, aby plug-in pracoval správně. Plug-in se exportuje volbou „Export the plug-in in a format suitable for deployment using the Export Wizard“ na stránce Overview v editoru daného modulu.

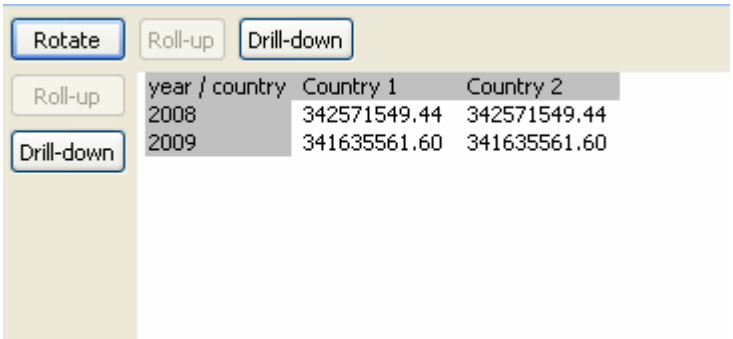
Pokud se vybere v průvodci exportem možnost exportovat do JAR archivu, stačí pro deployment tento JAR archiv nahrát do adresáře plugins instalace Eclipse a restartovat Eclipse. Tato instance Eclipse musí mít samozřejmě nainstalovány moduly BIRTu, protože je na nich tento projekt závislý. Jejich instalace je detailně popsána na http://www.eclipse.org/birt/, což je domovská stránka projektu BIRT.

5.7 Výsledný produkt

Programový výsledek této diplomové práce se skládá ze dvou produktů: OLAP klient a OLAP server. Server je distribuován ve webovém archivu WAR. Pro jeho spuštění je nutný deployment na aplikační kontejner Tomcat a připojení k databázi MySQL, jejíž konfigurace se provede souborem jdbc.properties. Více informací o spuštění serveru se dozvíte v manuálu, který je dostupný v příloze č.3. Sestavení obou projektů ze zdrojových kódu (příloha č.2) je popsáno tamtéž.

Na tomtéž místě jsou popsány úkoly k započetí práce na projektu: spuštění OLAP klienta, přepnutí do OLAP Perspective, vytvoření nového projektu typu OLAP design. Na tomto místě pouze stručně nastíníme příklad postupu jednoduché OLAP analýzy.

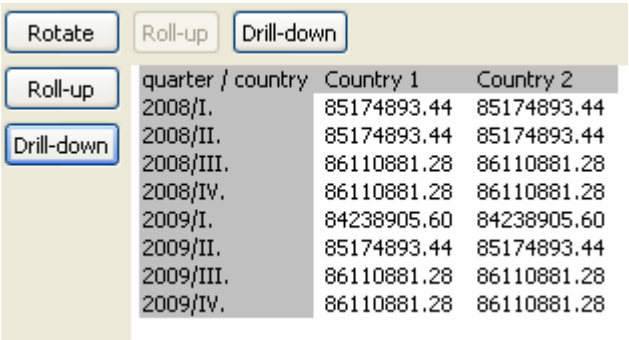
Pro analýzu jsem zvolil zkušební data, která jsem vygeneroval generátorem náhodných čísel s rovnoměrným rozložením, proto obsahují sumarizovaná data stejné nebo podobné hodnoty. Na obrázku 5.9 můžeme vidět úvodní fázi po vytvoření datové kostky.



Obrázek 5.9: Úvodní fáze OLAP analýzy,datová kostka na vysoké úrovni abstrakce

To, že program počítá korektní operace, můžeme vidět na sečtení dat z jednotlivých let. Výsledek jasně ukazuje, že obrat v přechodném roku 2008 (o jeden den více) je větší než obrat v roce 2009.

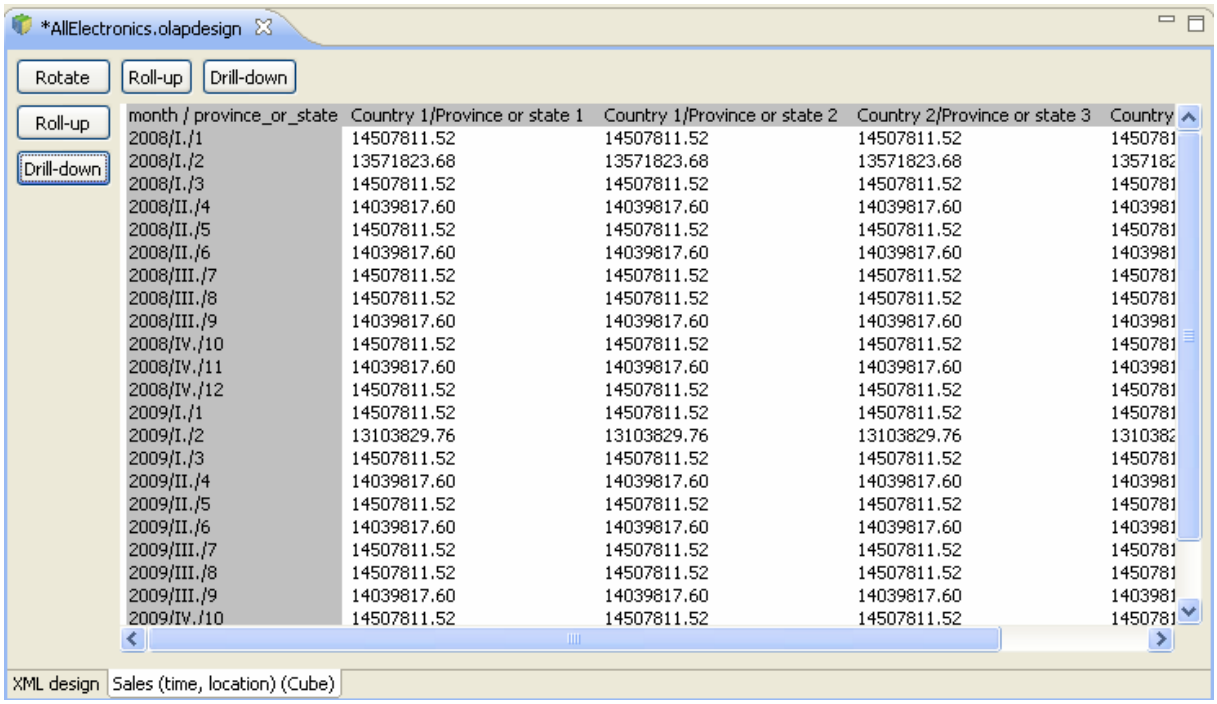
Dále provádíme další operace Roll-up a Drill-down. Pokud nemá již atribut v konceptuální hierarchii rodiče, tlačítko roll-up zešedne. Taktéž je tomu v opačném případě (abscence potomka) u drill-down. Na obrázku 5.10 vidíme, jak vypadá kostka po provedení drill-down na dimenzi time.



Obrázek 5.10: Provedení operace drill-down na dimenzi time (oproti obrázku 5.10)

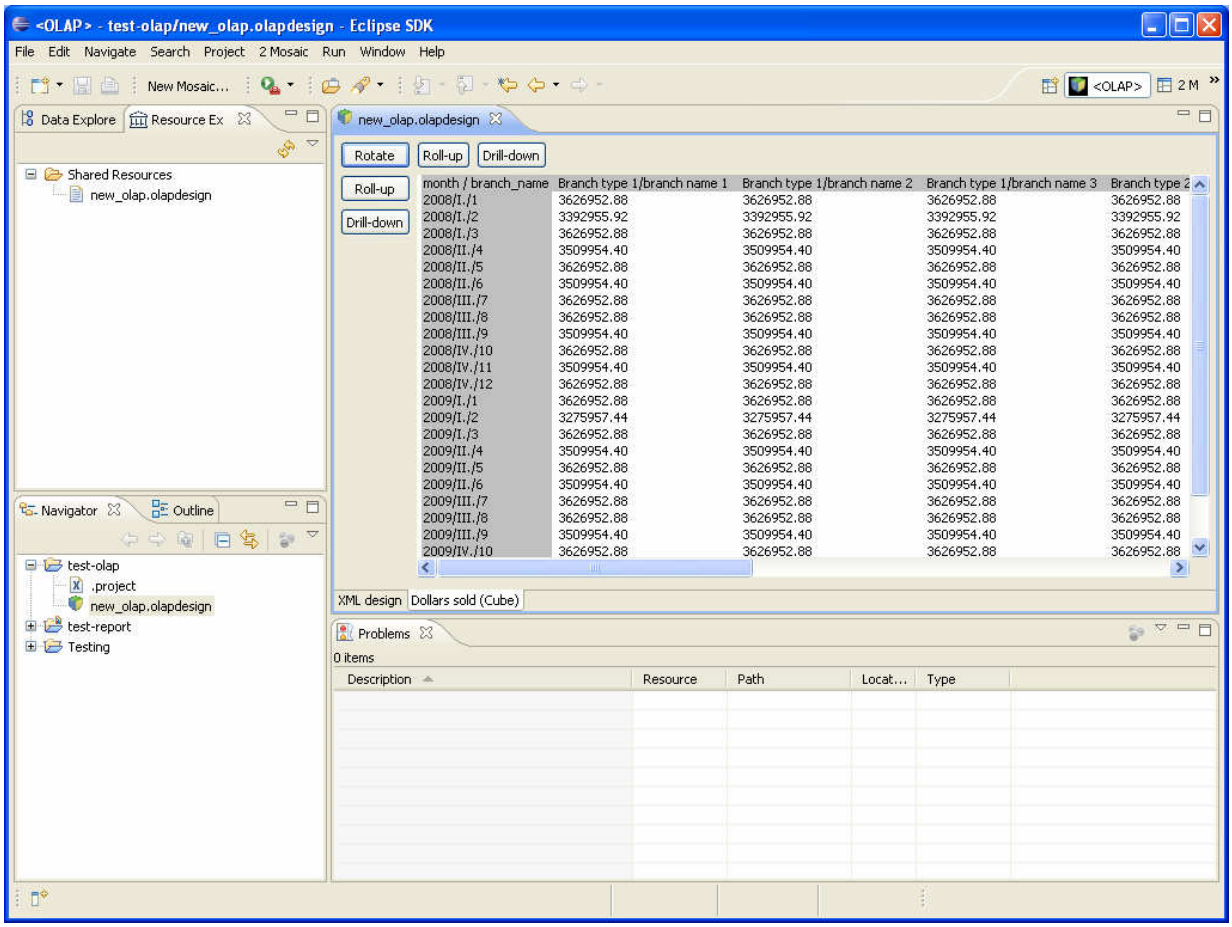
Dále ještě provedeme drill-down na dimenzi location i na dimenzi time. Výsledek můžeme vidět na obrázku 5.11. Více informací o tom, jak pracovat s OLAP klientem, se dozvíte z jeho manuálu v příloze č. 5.

Nyní se ještě na obrázku 5.12 jednou podíváme na OLAP klienta při OLAP analýze a popíšeme hlavní části grafického uživatelského rozhraní.



month / province_or_state	Country 1/Province or state 1	Country 1/Province or state 2	Country 2/Province or state 3	Country
2008/I./1	14507811.52	14507811.52	14507811.52	1450781
2008/I./2	13571823.68	13571823.68	13571823.68	1357182
2008/I./3	14507811.52	14507811.52	14507811.52	1450781
2008/II./4	14039817.60	14039817.60	14039817.60	1403981
2008/II./5	14507811.52	14507811.52	14507811.52	1450781
2008/II./6	14039817.60	14039817.60	14039817.60	1403981
2008/III./7	14507811.52	14507811.52	14507811.52	1450781
2008/III./8	14507811.52	14507811.52	14507811.52	1450781
2008/III./9	14039817.60	14039817.60	14039817.60	1403981
2008/IV./10	14507811.52	14507811.52	14507811.52	1450781
2008/IV./11	14039817.60	14039817.60	14039817.60	1403981
2008/IV./12	14507811.52	14507811.52	14507811.52	1450781
2009/I./1	14507811.52	14507811.52	14507811.52	1450781
2009/I./2	13103829.76	13103829.76	13103829.76	1310382
2009/I./3	14507811.52	14507811.52	14507811.52	1450781
2009/II./4	14039817.60	14039817.60	14039817.60	1403981
2009/II./5	14507811.52	14507811.52	14507811.52	1450781
2009/II./6	14039817.60	14039817.60	14039817.60	1403981
2009/III./7	14507811.52	14507811.52	14507811.52	1450781
2009/III./8	14507811.52	14507811.52	14507811.52	1450781
2009/III./9	14039817.60	14039817.60	14039817.60	1403981
2009/IV./10	14507811.52	14507811.52	14507811.52	1450781

5.11: Provedení drill-down na obou dimenzích oproti obrázku 5.10



month / branch_name	Branch type 1/branch name 1	Branch type 1/branch name 2	Branch type 1/branch name 3	Branch type 2
2008/I./1	3626952.88	3626952.88	3626952.88	3626952.88
2008/I./2	3392955.92	3392955.92	3392955.92	3392955.92
2008/I./3	3626952.88	3626952.88	3626952.88	3626952.88
2008/II./4	3509954.40	3509954.40	3509954.40	3509954.40
2008/II./5	3626952.88	3626952.88	3626952.88	3626952.88
2008/II./6	3509954.40	3509954.40	3509954.40	3509954.40
2008/III./7	3626952.88	3626952.88	3626952.88	3626952.88
2008/III./8	3626952.88	3626952.88	3626952.88	3626952.88
2008/III./9	3509954.40	3509954.40	3509954.40	3509954.40
2008/IV./10	3626952.88	3626952.88	3626952.88	3626952.88
2008/IV./11	3509954.40	3509954.40	3509954.40	3509954.40
2008/IV./12	3626952.88	3626952.88	3626952.88	3626952.88
2009/I./1	3626952.88	3626952.88	3626952.88	3626952.88
2009/I./2	3275957.44	3275957.44	3275957.44	3275957.44
2009/I./3	3626952.88	3626952.88	3626952.88	3626952.88
2009/II./4	3509954.40	3509954.40	3509954.40	3509954.40
2009/II./5	3626952.88	3626952.88	3626952.88	3626952.88
2009/II./6	3509954.40	3509954.40	3509954.40	3509954.40
2009/III./7	3626952.88	3626952.88	3626952.88	3626952.88
2009/III./8	3626952.88	3626952.88	3626952.88	3626952.88
2009/III./9	3509954.40	3509954.40	3509954.40	3509954.40
2009/IV./10	3626952.88	3626952.88	3626952.88	3626952.88

Obrázek 5.12: OLAP klient při OLAP analýze

Hlavní částí klienta je OLAP editor, kde se jednak konfiguruje soubor s konfiguračními parametry, a jednak zobrazují datové kostky. Uživatel může provádět operace roll-up, drill-down a pivot (rotate) na zobrazených dimenzích.

Jeden konfigurační soubor může obsahovat více datových skladů i více datových kostek. Všechna zobrazená data jsou průběžně ukládána do konfiguračního souboru, aby mohl uživatel při opětovném spuštění začít s prací přesně z místa, kde OLAP analýzu skončil při předchozím použití.

Jedná se o první verzi OLAP klienta pod Eclipse platformou, u které je možné provést množství rozšíření. Zde jsou náměty na některá z nich.

- Implementace zobrazování tří a vícedimenzionálních datových kostek. Klient zatím podporuje pouze jedno- a dvojdimenzionální kostky. OLAP server již vícedimenzionální funkcionalitu poskytuje.
- Export dat z OLAP analýzy do různých formátů (a např. do prostředí BIRT).
- Zavedení administrace uživatelů na OLAP serveru, přihlašování k OLAP klientovi.
- Naplnění datového skladu - např. pomocí integrace s projektem BIRT, nebo prostým databázovým skriptem vyvolaného z prostředí Eclipse a zaslaného na OLAP server.
- Změna MySQL Database Engine na InnoDB na Infobright - povede ke zvýšení rychlosti odezvy datového skladu.

6 Závěr

V této diplomové práci se podařilo stručně popsat teorii problematiky datových skladů a OLAP technologií. Tyto poznatky jsou nutné nejen k implementaci klienta pro zobrazování OLAP kostek, ale také k samotnému ovládání OLAP klientů. Značná část rozhodování při vytváření datového skladu totiž spočívá na analytikovi/vývojáři, nikoli na vývojovém nástroji.

Dále se podařilo stručně popsat nástroj BIDS od firmy Microsoft jako jeden z příkladů klienta pro zpracování OLAP kostek. Na základě těchto poznatků byla sestavena specifikace požadavků pro volně dostupného OLAP klienta, který byl implementován v další fázi tohoto projektu. V kapitole 4 je zpracován přehled dostupných technologií a výběr těch z nich, které jsou nejvhodnější k implementaci.

Další kapitola se potom zabývá přímo implementací, a je sepsána přibližně tak, jak chronologicky implementace postupovala. Obsahuje pojednání o databázové a serverové části a jejich rozhraní. Dále detailně popisuje další komponenty, které byly použity k implementaci celé aplikace. V části OLAP serveru to byly především Spring framework a jeho podpora pro JDBC a transakce a mezi klientem a serverem je popsáno rozrhraní Spring HTTP Remoting. V klientské části je popsáno použití Eclipse SDK pro tvorbu zásuvných modulů (plug-ins) a RCP aplikací, dále pak projekt BIRT, jehož některé komponenty byly při tvorbě OLAP klienta využity.

Po implementační kapitole následuje stručné představení OLAP klienta a jeho vlastností. Pomocí OLAP klienta lze vytvořit datový sklad a zobrazovat data uložené v něm pomocí editoru pro zobrazování OLAP kostek, které lze rovněž definovat. Naplnění datového skladu z jiných datových zdrojů zatím není implementováno, a může být námětem pro další rozšíření klienta.

Výsledek této práce je možné použít k vytvoření a administraci jednoduchého datového skladu a provádění OLAP operací, jejichž výstup bude vstupem pro OLAP analýzu prováděnou uživatelem. Následně lze projekt zdokonalovat a přidávat další pokročilejší funkce pro uživatele z reálného obchodního prostředí jako vhodnou alternativu ke komerčním projektům. Produkt lze také používat pro výuku uživatelů-začátečníků a jejich seznámení s datovými sklady a OLAP technologiemi.

Přínos této diplomové práce vidím v seznámení se s technologiemi, které se běžně nepoužívají k vývoji klasických informačních systémů v jazyku Java. Využití Spring JDBC namísto objektově-relačního mapování, Spring Remoting namísto běžné HTTP komunikace přes webový prohlížeč a konečně použití bohatého klienta tvořeného platformou Eclipse namísto webového prohlížeče přinese vývojáři globálnější pohled na širokou oblast Java technologií a zkušenosti, kterých může využít při implementaci netradičních úkolů v oblasti informačních systémů i jinde. V neposlední řadě je také přínosem nahlédnutí do OLAP technologií a datových skladů, a to i do jejich implementace.

Literatura

[han06] Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Second Edition. Elsevier Inc., 2006, 770 stran, ISBN 1-55860-901-3.

[inm02] Inmon, W.H.: Building the Data Warehouse, 3rd edition, John Wiley & Sons, 2002.

[pon01] Ponniah, P.: Data Warehousing Fundamentals. John Wiley & Sons, 2001, 516 stran, ISBN 0-471-41254-6.

[mca08] McAffer, J., Lemieux, J.-M.: Eclipse Rich Client Platform: Designing, Coding and Packaging Java Applications. Addison Wesley Professional, 2008, 552 stran.

[clr06] Clayberg, E., Rubel, D.: Eclipse: Building Commercial-Quality Plug-ins, Second Edition, Addison Wesley Professional, 2006, 864 stran.

[eps08] Eclipsepluginsite.com community: Introduction to Eclipse Plugin Development [online], 2008 [cit 2010-05-03]. Dostupné na WWW: http://www.eclipsepluginsite.com

[wab08] Walls, C., Breidenbach, R.: Spring in Action, Second Edition, Manning Publications Co., 2008, 730 stran, ISBN 1-933988-13-4.

Seznam příloh

Příloha 1. Disk CD – obsahuje veškerý předepsaný obsah dle zadání (viz další body).
Příloha 2. Zdrojové texty – na CD.
Příloha 3. Manuál – návod ke kompilaci a instalaci – na CD.
Příloha 4. Spustitelný program pro OLAP server, OLAP klient a knihovna olap-interface – na CD.
Příloha 5. Odkazované zdrojové texty.
Příloha 6. Databázové skripty - vytvoření DB pro OLAP server, naplnění databáze z příkladu - CD.

Příloha 5. – Odkazované zdrojové texty

olapdesign.xsd – použité XML schéma pro lineární OLAP operace

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:jxb="http://java.sun.com/xml/ns/jaxb"
            jxb:version="2.0">

<xsd:element name="olap-design" type="OlapDesign"/>

<xsd:complexType name="OlapDesign">
  <xsd:sequence>
    <xsd:element name="properties" type="MapType" />
    <xsd:element name="dataSources" type="DataSource"
maxOccurs="unbounded" />
    <xsd:element name="dataSets" type="DataSet" maxOccurs="unbounded" />
    <xsd:element name="dataWarehouses" type="DataWarehouse"
maxOccurs="unbounded" />
    <xsd:element name="dataCubes" type="DataCube" maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="MapType">
  <xsd:sequence>
    <xsd:element name="entry" maxOccurs="unbounded" type="EntryType">
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="EntryType">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attribute name="key" type="xsd:string" />
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>


<xsd:complexType name="DataSource">
  <xsd:sequence>
    <xsd:element name="url" type="xsd:string" />
    <xsd:element name="database" type="xsd:string" />
    <xsd:element name="username" type="xsd:string" />
    <xsd:element name="password" type="xsd:string" />
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="DataSet">
  <xsd:sequence>
    <xsd:element name="dataSourceRef" type="xsd:string" />
    <xsd:element name="query" type="xsd:string" />
    <xsd:element name="columns" type="Column" maxOccurs="unbounded" />
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="Column">
```

```
  <xsd:sequence>
      <xsd:element name="alias" type="xsd:string" />
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="DataWarehouse">
  <xsd:sequence>
    <xsd:element name="factTable" type="FactTable" />
    <xsd:element name="dimensionTables" type="DimensionTable"
maxOccurs="unbounded" />
    <xsd:element name="commands" type="Command" maxOccurs="unbounded" />
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="DimensionTable">
  <xsd:sequence>
    <xsd:element name="column" type="DimensionColumn" minOccurs="1"
maxOccurs="unbounded" />
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="FactTable">
  <xsd:sequence>
    <xsd:element name="column" type="FactColumn" minOccurs="1"
maxOccurs="unbounded" />
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="DimensionColumn">
  <xsd:attribute name="name" type="xsd:string" use="required" />
  <xsd:attribute name="primary" type="xsd:boolean" default="false" />
  <xsd:attribute name="datatype" type="xsd:string" use="required" />
  <xsd:attribute name="parent" type="xsd:string" use="optional" />
</xsd:complexType>

<xsd:complexType name="FactColumn">
      <xsd:attribute name="name" type="xsd:string" />
      <xsd:attribute name="datatype" type="xsd:string" />
      <xsd:attribute name="primary" type="xsd:boolean" default="false" />
      <xsd:attribute name="isFact" type="xsd:boolean" default="false" />
      <xsd:attribute name="dimensionRef" type="xsd:string" use="optional"
/>
</xsd:complexType>

<xsd:complexType name="Command">
  <xsd:sequence>
      <xsd:element name="statement" type="xsd:string" />
      <xsd:element name="dataSetRef" type="xsd:string" />
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="DataCube">
  <xsd:sequence>
      <xsd:element name="dataWarehouseRef" type="xsd:string" />
      <xsd:element name="factTableColRef" type="xsd:string" />
      <xsd:element name="countStrategy" type="xsd:string" />
```

```
      <xsd:element name="dimensions" type="Dimension" minOccurs="1"
maxOccurs="2" />
      <xsd:element name="cachedData" type="MapType" />
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="Dimension">
  <xsd:sequence>
    <xsd:element name="item" maxOccurs="unbounded" type="Item" />
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" />
  <xsd:attribute name="ref" type="xsd:string" />
  <!-- Which operation to perform (client->server) -->
  <xsd:attribute name="drill-down" type="xsd:boolean" default="false" />
  <xsd:attribute name="roll-up" type="xsd:boolean" default="false" />
  <!-- Is it a terminate item in hierarchy? (Drill-down will not be
available) -->
  <xsd:attribute name="terminate" type="xsd:boolean" default="false" />
  <!-- Is it a root item in hierarchy? (Roll-up will not be available) -->
  <xsd:attribute name="root" type="xsd:boolean" default="false" />
</xsd:complexType>

<xsd:complexType name="Item">
  <xsd:sequence>
    <!-- Name of the dimension item in the table header -->
      <xsd:element name="headerName" type="xsd:string" />
      <!-- Addres of the node in hash table -->
      <!-- Address contains:
dim_table|dim_col1_val|dim_col2_val|...|dim_coln_val -->
      <xsd:element name="address" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>

</xsd:schema>
```

olapdesign.xsd – XML schéma pro hierarchické OLAP operace

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:jxb="http://java.sun.com/xml/ns/jaxb"
            jxb:version="2.0">

<xsd:element name="olap-design" type="OlapDesign"/>

<xsd:complexType name="OlapDesign">
  <xsd:sequence>
    <xsd:element name="properties" type="MapType" />
    <xsd:element name="dataSources" type="DataSource"
maxOccurs="unbounded" />
    <xsd:element name="dataSets" type="DataSet" maxOccurs="unbounded" />
    <xsd:element name="dataWarehouses" type="DataWarehouse"
maxOccurs="unbounded" />
    <xsd:element name="dataCubes" type="DataCube" maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="MapType">
  <xsd:sequence>
    <xsd:element name="entry" maxOccurs="unbounded" type="EntryType">
    </xsd:element>
```

```
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="EntryType">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attribute name="key" type="xsd:string" />
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="DataSource">
  <xsd:sequence>
    <xsd:element name="url" type="xsd:string" />
    <xsd:element name="database" type="xsd:string" />
    <xsd:element name="username" type="xsd:string" />
    <xsd:element name="password" type="xsd:string" />
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="DataSet">
  <xsd:sequence>
    <xsd:element name="dataSourceRef" type="xsd:string" />
    <xsd:element name="query" type="xsd:string" />
    <xsd:element name="columns" type="Column" maxOccurs="unbounded" />
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="Column">
  <xsd:sequence>
      <xsd:element name="alias" type="xsd:string" />
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="DataWarehouse">
  <xsd:sequence>
    <xsd:element name="factTable" type="FactTable" />
    <xsd:element name="dimensionTables" type="DimensionTable"
maxOccurs="unbounded" />
    <xsd:element name="commands" type="Command" maxOccurs="unbounded" />
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="DimensionTable">
  <xsd:sequence>
    <xsd:element name="column" type="DimensionColumn" minOccurs="1"
maxOccurs="unbounded" />
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="FactTable">
  <xsd:sequence>
    <xsd:element name="column" type="FactColumn" minOccurs="1"
maxOccurs="unbounded" />
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" />
```

```
</xsd:complexType>

<xsd:complexType name="DimensionColumn">
  <xsd:attribute name="name" type="xsd:string" use="required" />
  <xsd:attribute name="primary" type="xsd:boolean" default="false" />
  <xsd:attribute name="datatype" type="xsd:string" use="required" />
  <xsd:attribute name="parent" type="xsd:string" use="optional" />
</xsd:complexType>

      <xsd:complexType name="FactColumn">
            <xsd:attribute name="name" type="xsd:string" />
            <xsd:attribute name="datatype" type="xsd:string" />
            <xsd:attribute name="isFact" type="xsd:boolean" />
            <xsd:attribute name="dimensionRef" type="xsd:string" />
      </xsd:complexType>

      <xsd:complexType name="Command">
  <xsd:sequence>
      <xsd:element name="statement" type="xsd:string" />
      <xsd:element name="dataSetRef" type="xsd:string" />
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="DataCube">
  <xsd:sequence>
      <xsd:element name="dataWarehouseRef" type="xsd:string" />
      <xsd:element name="factTableColRef" type="xsd:string" />
      <xsd:element name="countStrategy" type="xsd:string" />
      <xsd:element name="dimensions" type="Dimension" minOccurs="1"
maxOccurs="2" />
      <xsd:element name="cachedData" type="MapType" />
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="Dimension">
  <xsd:sequence>
    <xsd:element name="rootNode" type="Node" />
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" />
  <xsd:attribute name="ref" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="Node">
  <xsd:sequence>
    <!-- Name of the dimension node in the table header -->
      <xsd:element name="headerName" type="xsd:string" />
      <!-- Addres of the node in hash table -->
      <!-- Address contains:
dim_table|dim_col1_val|dim_col2_val|...|dim_coln_val -->
      <xsd:element name="address" type="xsd:string" />
      <xsd:element name="child-node" type="Node" maxOccurs="unbounded" />
  </xsd:sequence>
  <!-- Which operation to perform (client->server) -->
  <xsd:attribute name="drill-down" type="xsd:boolean" default="false" />
  <xsd:attribute name="roll-up" type="xsd:boolean" default="false" />
  <!-- Is it a terminate node? (Drill-down will not be available) -->
  <xsd:attribute name="terminate" type="xsd:boolean" default="false" />
</xsd:complexType>
</xsd:schema>
```