VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

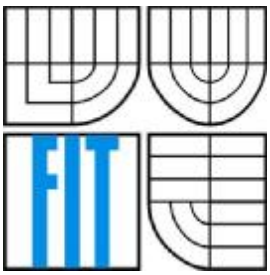
GRAFICKÝ PODSYSTÉM V PROSTŘEDÍ INTERNETOVÉHO PROHLÍŽEČE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. Petr Vlach
AUTHOR

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

GRAFICKÝ PODSYSTÉM V PROSTŘEDÍ INTERNETOVÉHO PROHLÍŽEČE

GRAPHS SUBSYSTEM IN INTERNET BROWSER ENVIRONMENT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. Petr Vlach
AUTHOR

VEDOUCÍ PRÁCE — Hruška Tomáš, prof. Ing., CSc.
SUPERVISOR

BRNO 2008

Grafický podsystém v prostředí internetového prohlížeče

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením
prof. Ing. Tomáše Hrušky, CSc.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jméno Příjmení
Datum

Poděkování

Děkuji prof. Ing. Tomáši Hruškovi, CSc. za odborné vedení mé diplomové práce.

Abstrakt

Tato diplomová práce se skládá ze dvou částí. První části se zabývá existujícími komerčními a nekomerčními systémy pro zobrazení OLAP dat pomocí kontingenčních tabulek a grafů a jejich srovnáním.

Zjištěné poznatky o jednotlivých systémech jsou použity pro implementaci grafického subsystému pro prostředí internetových prohlížečů. Je kladen velký důraz na přívětivé uživatelské prostředí a jeho ovládání.

Klíčová slova

databáze, datové sklady, okamžité analytické zpracování dat (OLAP), kontingenční tabulka, kontingenční graf, Flash, JavaScript, ActionScript, vizualizace, Cube presentation model (CPM)

Abstract

This master’s thesis, divided into two sections, compares in part one existing (non-)commercial systems for OLAP presentation using contingency table or graph. The main focus is put on a graph.

Results received from my observations in part one are used for implementing a graphic subsystem within internet browser's environment. User friendly interface and good arrangement of displayed data are the most important tasks.

Keywords

Database, data warehouse, on-line analytical processing (OLAP), contingency table, contingency chart, Flash, JavaScript, ActionScript, visualization, Cube presentation model (CPM)

Obsah

1 Úvod

S rozvojem firem zakládajících svou činnost na obchodování s nejrůznějším zbožím se začali objevovat požadavky na analýzu trhu. Nejlepší prostředky pro řešení těchto požadavků nabízí obor informační technologie (IT). Bylo potřeba vytvořit systém, který bude uchovávat velké množství dat a v relativně krátkém čase bude tato data prezentovat ve srozumitelné podobě. S návazností na tyto požadavky vznikla technologie ukládání dat „on-line analytical processing“ (OLAP), v češtině „okamžité analytické zpracování dat“. Tento systém uchovávání dat je primárně zaměřen na ukládání dat ve velkém množství a zároveň na provádění složitých a komplexních dotazů nad těmito daty v relativně krátkém čase.

K tomu bylo zapotřebí vybudovat systém se způsoby zobrazení těchto dat. Primárně se data v OLAP systémech zobrazují jako datová kostka, kde na každé ose kostky je kategorizující údaj a v bloku kostky je hodnota, kterou hledáme. Bylo nutností tyto data zobrazovat jasně a přehledně, a proto se začaly vyvíjet vizualizační metody a nástroje pro zobrazení multidimensionálních dat. V dnešní době je nejznámější zobrazení pomocí kontingenčních tabulek nebo formou grafů. Existují i jiná zobrazení jako je *Table Lens* nebo *Fisheye Table*.

V mé práci se budu zabývat hlavně zobrazením ve formě grafu a budu se snažit vytvořit efektivnější zobrazení podle již exitujících řešení a to jak po stránce interakce s uživatelem tak i po stránce přehlednosti zobrazených dat. Zaměřím se hlavně na možnosti zobrazení různých úrovní agregace v jedné ploše grafu.

V druhé kapitole se budu snažit nastínit teoretický základ nutný pro ostatní kapitoly mé diplomové práce. Budu zde popisovat základní myšlenku databázových systémů a konkrétně OLAP systém. Dále popíšu některé dnes již existující metody pro vizualizaci multidimensionálních dat. A také ukážu způsoby vytváření Flashových animací pomocí programovacího jazyka ActionScript. Pokračovat budu vysvětlením způsobů, jakým se může Flashová animace umístit na webovou stránku a jak může komunikovat s uživatelem pomocí JavaScriptu.

V třetí kapitole postupně popíšu mnou nalezené komerční a nekomerční řešení zobrazování dat z OLAP systémů a v závěru kapitoly je všechny porovnám.

Kapitola čtvrtá bude zaměřena na návrh mého grafického podsystému s ukázkou objektového modelu mé knihovny a popíšu vstupní a výstupní interface.

V poslední kapitole se budu zabývat samotnou implementací této knihovny a zdůrazním některé zajímavé části. Popíšu i externí knihovny, které jsem k implementaci použil.

2 Teoretický základ

2.1 Databázové systémy

Databázové systémy umožňují uchovávat data ve strukturách, která jsou nezávislá na aplikační vrstvě.

2.1.1 Relační databáze

Relační databáze se řadí mezi databáze typu „on-line transaction processing“ (OLTP). Jsou založeny na relačním modelu dat, na relační algebře a data jsou uspořádávána do tabulek (relací). Každá tabulka má definovány přípustné operace, které se nad ní mohou provádět. Jazykem pro práci s těmito daty je obvykle Structured Query Language (SQL).

Základními konstrukcemi jazyka SQL jsou příkazy pro vytváření struktur (CREATE), vkládání dat do tabulek (INSERT), upravování dat v tabulkách (UPDATE), mazání dat (DELETE) a nejpoužívanější a také nejkomplexnější je dotaz pro výběr dat (SELECT).

Relační databáze dále umožňují vytvářet objekty vyšší, než jsou tabulky a to pohledy (VIEW). Pohled je vlastně dotaz SELECT, ale vystupuje jako tabulka. Slouží pro agregaci a zpřístupňování jen těch dat, která se mají zobrazit.

Dalšími konstrukcemi jsou uložené procedury a funkce, které se provádí na databázové úrovni. Posledním speciálním případem je databázová spoušť (trigger), která se vyvolá pokaždé, když se v databázi něco mění. Triggry slouží pro předzpracování dat.

Nejčastěji používaným návrhovým jazykem pro relační databáze je Entitně-relační (ER) diagram. Každá tabulka je zde zanesena samostatně a má definované jednotlivé sloupce a jejich typy. Mezi tabulkami se dají zakreslovat návaznosti jednotlivých dat.

2.1.2 OLAP a datové sklady

Datové sklady jsou komplexní data uložená ve struktuře, která umožňuje efektivní analýzu dat. Do datových skladů se data neukládají přímo, ale za pomoci primárních informačních systémů, které tyto data ukládají do svých databází. Z těchto databází se data dále transformují a nahrávají do struktury datových skladů[1][14].

V literatuře bývá často s datovými sklady spojován pojem „on-line analytical processing“ (OLAP).

V roce 1993 Dr. E. F. Codd poprvé představil termín OLAP a zároveň zformuloval 12 základních pravidel pro OLAP[7].

1. Multidimensionální konceptuální model: OLAP by měl být vytvořen z multidimensionálního modelu, který má splňovat všechny požadavky uživatele. Většinou se jako tento model používá datová kostka.

2. Transparentnost: data v OLAP musí být přehledné a jasné. Proto musí být vstupní data heterogenní. Tuto vlastnost docílíme pomoci ETL procesu.

3. Dostupnost: OLAP by měl být schopen přistupovat kdykoliv k datům, která jsou potřebná pro analýzu a to nezávisle na zdroji dat.

4. Stabilní výkonnost: systém by měl mít stále stejnou výkonnost a to nezávisle na struktuře a počtu dat uložených v databázi.

5. Architektura klient/server: systém OLAP musí být založen na architektuře klient/server.

6. Generická dimensionalita: každá dimense musí mít shodnou strukturu i operační schopnost

7. Dynamická manipulace s řídkými maticemi: systém OLAP musí umět přizpůsobit své fyzické schéma na konkrétní analytický model, při zachování požadovaného výkonu.

8. Podpora více uživatelů: systém OLAP musí být schopen vyřizovat požadavky více uživatelů v jeden okamžik.

9. Neomezené operace mezi dimensemi: při jakémkoliv dotazu musí OLAP umět rozpoznat dimense a provést výpočty i napříč těmito dimensemi.

10. Intuitivní manipulace s daty: rozhraní k OLAP musí být intuitivně ovladatelné a musí uživatelům poskytovat přehledně získaná data.

11. Flexibilní výstupy: výstupní data musí byt flexibilně přizpůsobitelná pro různé způsoby pohledu.

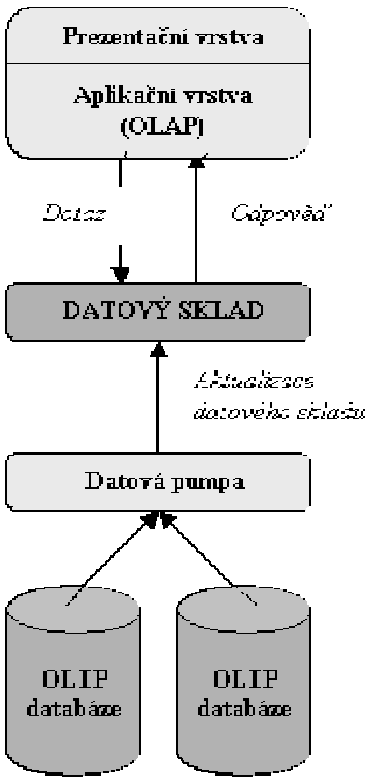
12. Neomezené dimense a úrovně agregace: systém OLAP by neměl omezovat počty možných ukládaných dimensí a ani úrovně agregace v těchto dimensích.

Datové sklady mají třívrstvou architekturu (viz Obrázek 2-1)[2]:

a) Spodní – tato vrstva obsahuje databázový server a je to základní úložiště dat, ze kterého se data dále distribuují.

b) Prostřední – v této vrstvě je obsažen samotný OLAP server. OLAP server může být implementován buď jako relační OLAP model (ROLAP), což je OLAP systém založený na relační databázi, nebo jako multidimensionální OLAP (MOLAP). Pojmy MOLAP a ROLAP popíšu blíže později.

c) Vrchní – je vrstva prezentační a je to výstup, který se zobrazuje klientovi. Jsou zde nástroje pro dotazování nad OLAP a také pro dolování dat.



Obrázek 2-1. Třívrstvá architektura datových skladů

Datový sklad je plněn procesem nazývaným ETL (extraction – transformation - load)[2]. Jak již název napovídá, je rozdělen na tři fáze.

a) extraction – získání dat z jednotlivých primárních zdrojů dat.
b) transformation – každý primární datový zdroj může obsahovat jiné modely uložení dat a některé data nemusí byt implicitně zadaná, proto se musí data transformovat do podoby přijatelné pro datový sklad.
c) load – v poslední fázi se takto upravená data nahrají do datového skladu.

V současné době jsou OLTP systémy nejčastěji implementovány jako relační databáze. Objektově-orientované technologie se často používají pouze v návrhu a implementaci aplikačního rozhraní. Proto jsou systémy objektově orientovaných databází postaveny na relačních databázích a dodává se k nim pouze rozhraní.

V OLAP systémech není relační technologie tak často používána. OLAP může být implementován jako ROLAP, tedy OLAP založený na relační databázi, nebo MOLAP, což je OLAP založený na multidimensionální technologii, který se v dnešní době začíná čím dál tím více prosazovat.

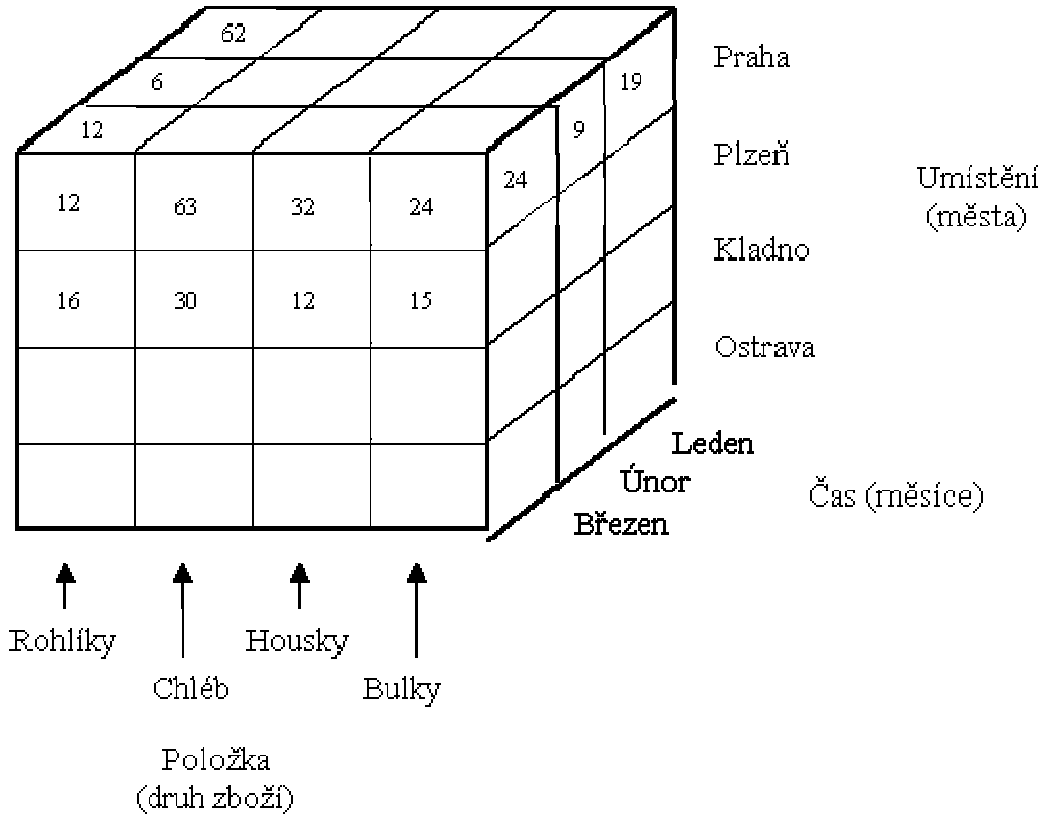
a) Nejčastěji se však používá technologie, spojující obě již zmíněné v jednu a to HOLAP. Jde o hybridní technologii, kdy většina dat je uložena v relační databázi a některá často používaná data jsou uložena v multidimensionálních datech[2].MOLAP – u tohoto systému se data ukládají pomalu, ale zato se rychleji čtou. Je to způsobeno tím, že data ukládáme přímo do multidimensionální podoby, což je velmi zatěžující, ale můžeme je rychleji získat. Systémy tohoto typu jsou určené pro malé a střední objemy dat.
b) ROLAP – data jsou zde ukládána do relačních databází. Data se ukládají do agregačních tabulek, ze kterých se poté čtou. Ukládání dat u tohoto způsobu je rychlé, ale čtení probíhá pomalu. Tento systém je určen pro velký objem dat, ale méně časté dotazy.
c) HOLAP – Tento systém stejně jako ROLAP ukládá data do relačních databází, ale následná agregační data již ukládá jako multidimensionální. HOLAP má díky tomuto způsobu uložení dat rychlý zápis i rychlé čtení.

2.1.2.1 Datové kostky

Datové kostky jsou založeny na multidimensionálním pohledu na data. Každá dimense zobrazuje rozdílné kategorie pohledu na data. Jedna kostka nemusí mít pouze 3 dimense, jak si kostku obyčejně představujeme, ale může mít i *n* dimensí[2].

Každá dimense může být potom ještě hierarchicky dělena, tedy na nejnižší úrovni hierarchie je pohled na data nejdetailnější a na úrovních vyšších jsou data agregována.

Míry (fakta) jsou data, která hledáme. Uvedeme si příklad takové kostky. Budeme hledat míru prodeje výrobků v různých lokalitách v jednotlivých měsících. Z tohoto dotazu nám vznikne kostka, která má jako dimenze výrobky, lokality a čas v měsících a míry jsou pak jejich prodejnosti. Názorně to můžete vidět na Obrázek 2-2.



Obrázek 2-2. Datová kostka

2.1.2.2 Operace v OLAP systémech

Nad OLAP systémy existují čtyři základní operace, které umožňují provádět úpravy a změny v zobrazení OLAP kostky[2].

a) Drill-down – tato operace umožňuje zvyšovat úroveň detailu pohledu na data.
b) Roll-up – pomocí této operace se úroveň detailu pohledu na data snižuje.
c) Pivoting – operace otáčející datovou kostkou, mění úhel pohledu na data.
d) Slicing – díky této operaci je umožněno vybrat hodnoty, na které se chcete v dané dimensi dívat.
e) Dicing – tato operace provádí to samé, co předchozí operace, jen s tím rozdílem, že umožňuje nastavit filtr pro více dimensí.

2.1.3 Vizualizace multidimensionálních dat

2.1.3.1 Vizualizace

Vizualizace je způsob jak zobrazit informace v přehledné formě pro rychlé a snadné pochopení. Jako hlavní nástroje vizualizace se užívají tabulky a grafy, které umí snadněji popsat danou situaci.

2.1.3.2 Vizualizace dat v OLAP systémech

Bylo vyvinuto již mnoho vizualizačních systému, které dokážou zobrazit velké množství dat na malé ploše. Bohužel tyto systémy zůstaly jen na papíře, nebo se nikdy neuvedly do provozu. Proto se v dnešní době snaží vývojáři vytvořit systém, který bude přehledně zobrazovat získaná data a zároveň bude i příjemný na používání.

Nejčastější implementace vizualizačních systémů je velké množství různých tabulek a grafů.

Kontingenční tabulka

Kontingenční tabulka zobrazuje multidimensionální data v přehledné mřížce, kde sloupce i řádky udávají kategorie a jejich dané úrovně agregace. Míry jsou pak průsečíky jednotlivých kategorií.
(viz Obrázek 2-3)

Nad kontingenční tabulkou se dají provádět všechny druhy operací, které u OLAP systémů existují.

- *Drill/down* – pomocí tlačítka plus, které je v řádku nebo sloupci
- *Roll/up* – pomocí tlačítka mínus, které je v řádku nebo sloupci
- *Pivot, slice, dice* – tyto operace se provádí většinou externím rozhraním

Kontingenční tabulka má ve svém základním zobrazení velký nedostatek v omezujícím počtu zobrazených dat na obrazovce počítače, proto se začaly vymýšlet nové způsoby zobrazení.

Součet z value	Popisky sloupců				
Popisky řádků	Dilly	Banglades	Singapur	Honkong	Celkový součet
13.1.2007		**480**	**640**	**640**	**1760**
Alloe		480			480
Hedvábí				240	240
Rýže				120	120
Kukuřice			640	280	920
14.1.2007		**420**	**420**	**840**	**1680**
Alloe				310	310
Hedvábí		190		420	610
Rýže			420	110	530
Kukuřice		230			230
15.1.2007	**50**	**540**	**590**	**710**	**1890**
Alloe	50	540		710	1300
Hedvábí			420		420
Kukuřice			170		170
16.1.2007		**620**	**480**	**500**	**1600**
Alloe			340	100	440
Hedvábí		60	140		200
Rýže		260			260
Kukuřice		300		400	700
Celkový součet	**50**	**2060**	**2130**	**2690**	**6930**

Obrázek 2-3. Kontingenční tabulka

Tabulka *Table Lens*

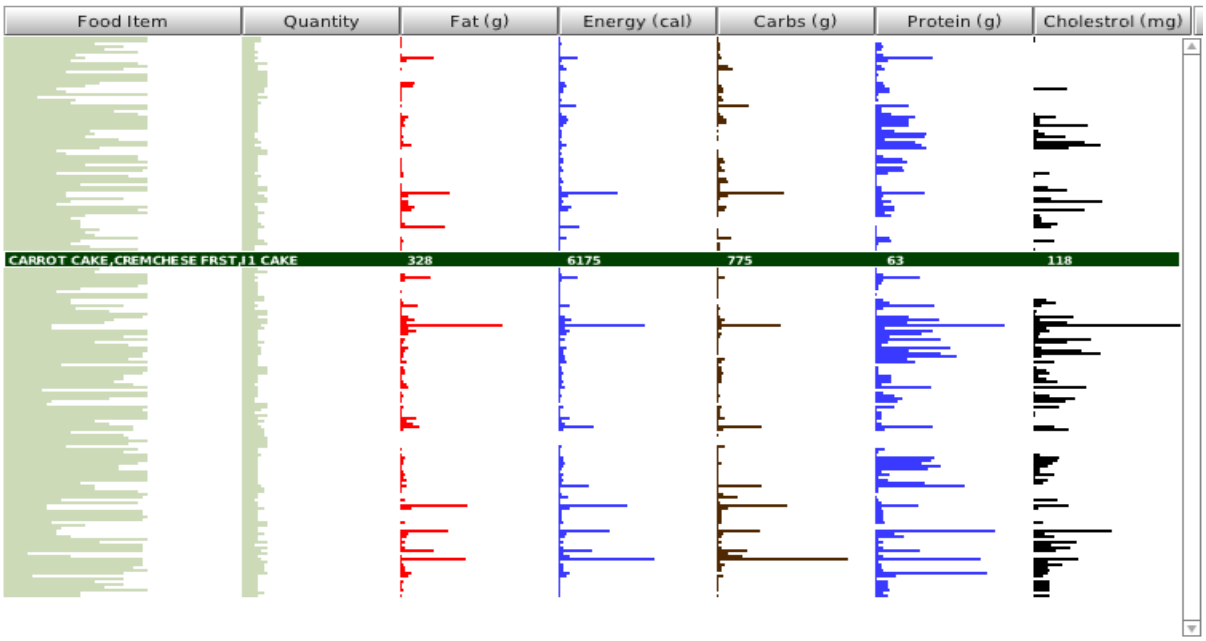
Tabulka *Table Lens*[7][12] umožňuje zobrazit velké množství dat na ploše monitoru a to tak, že se data v každém řádku pro každý sloupec zobrazí jen jako pruh o velmi malé šířce. Po kliknutí na tento řádek se v daném místě řádek rozšíří a v něm jsou data zapsána již textově a čitelně. (viz Obrázek 2-4)

Table Lens se skládá z těchto základních struktur:

- **Osy (*Axis*) –** model má dvě osy, tvořící sloupce a řádky
- **2D prostor (*2D space*) –** je to prostor sestavený z kartézského prostoru.
- **Funkce stupně zájmu (*Degree of Interest Function - DOI*) –** funkce, která mapuje každý bod z osy na hodnotu, která představuje daný stupeň zájmu. Každá osa má jiné DOI.
- **Funkce přenosu (*Transfer function*) –** tato funkce je podobná DOI, jen s tím rozdílem, že se osy nemapují na hodnoty, ale na pixely.

V případě textových polí je délka pruhů odpovídající poměrové délce textu na dané velikosti písma. V případě dat kvantitativních je to pak poměrový počet dat.

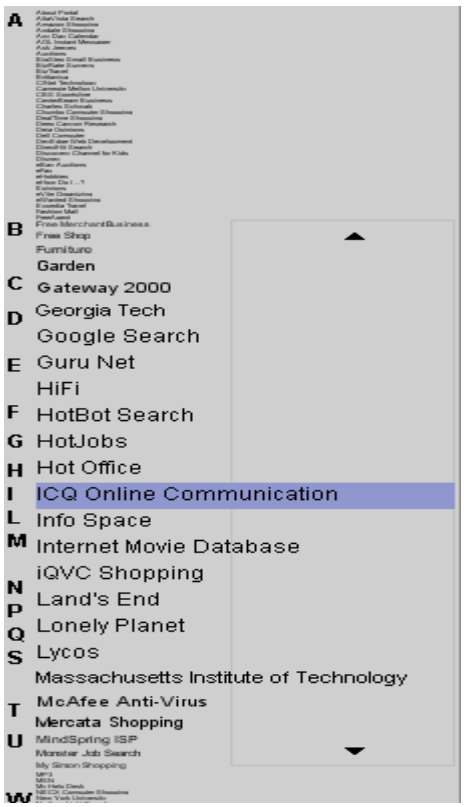
Hlavní myšlenka *Table Lens* spočívá v tom, že ne všechny buňky jsou si významově rovné.



Obrázek 2-4. Table Lens

Fysheye table

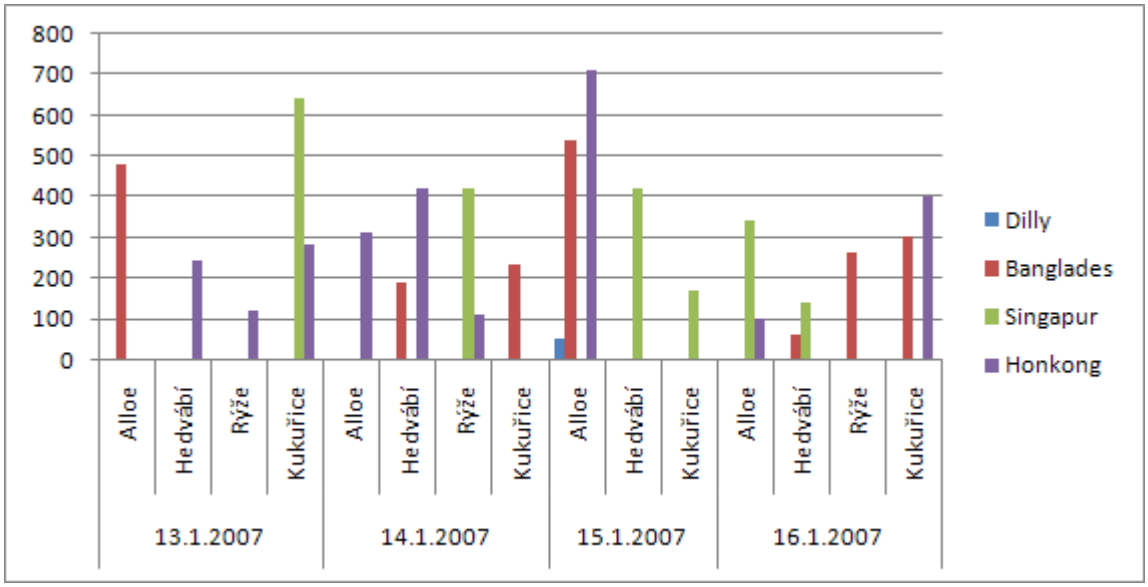
Tento způsob zobrazování vychází z předchozího principu, ale umožňuje zobrazení okolí zkoumaného řádku a to tak, že čím dále jsou řádky od středu prohlížení, tím více jsou zmenšené[7][11]. Velikost oblasti se poté dá měnit pomocí šipek nahoře a dole. Jak takové zobrazení vypadá, můžete vidět na obrázku (Obrázek 2-5), kde data jsou zobrazena ve formě menu.



Obrázek 2-5. Fysheye menu

Sloupcový graf

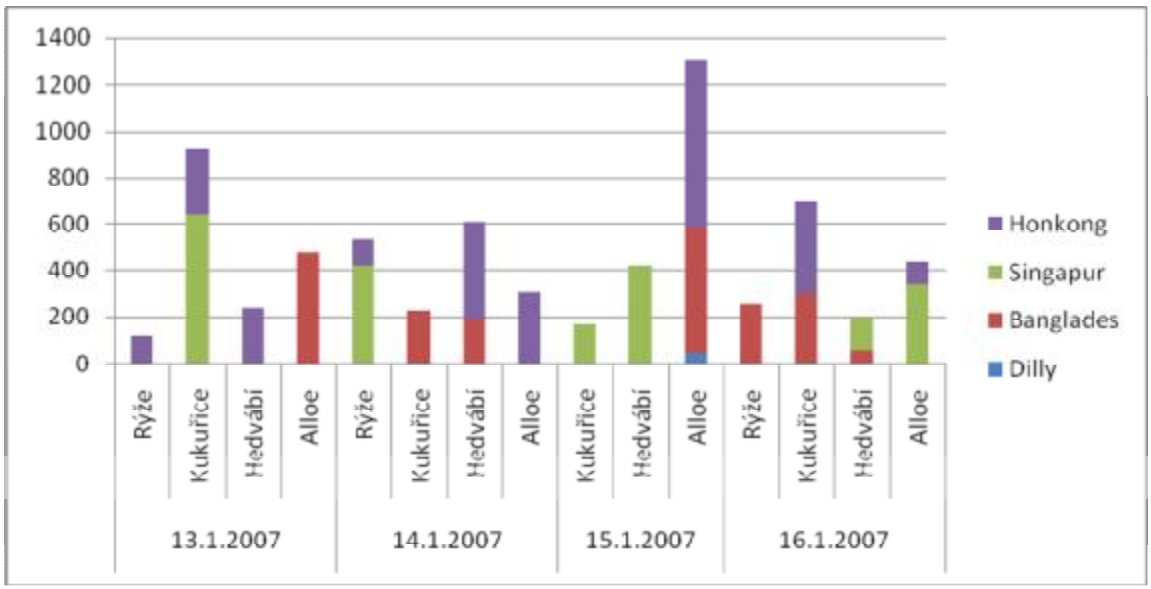
Ve sloupcovém grafu se míra zobrazuje jako osa *y*, každý sloupec reprezentuje jednu kategorii a zbytek zobrazených kategorií je na ose *x*, které jsou řazeny hierarchicky. (např. Obrázek 2-6)



Obrázek 2-6. Sloupcový graf

Sloupcový graf skládaný

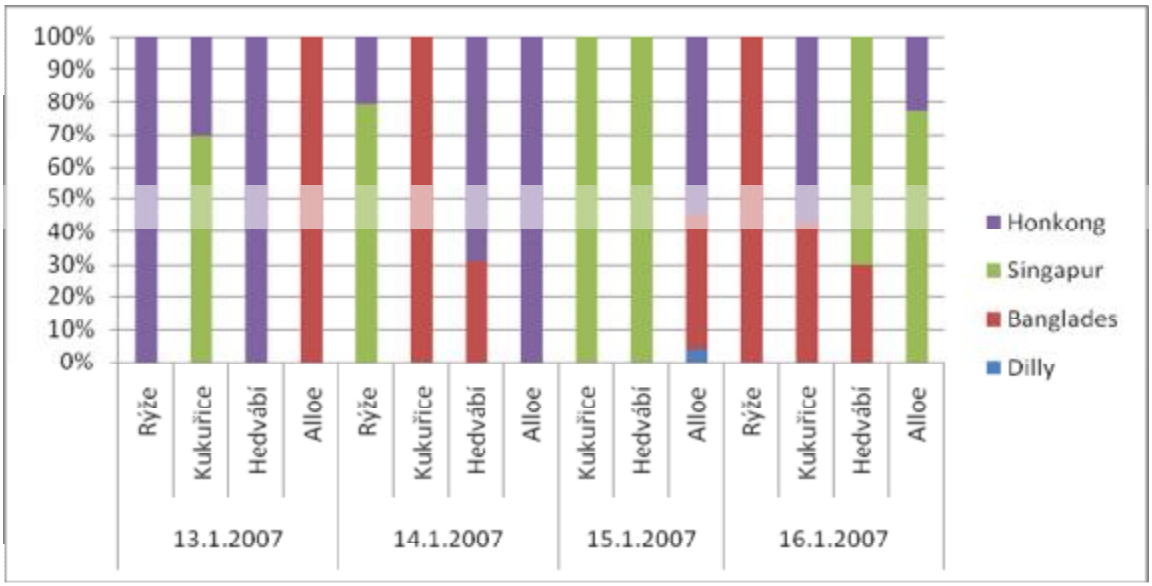
Tento typ je oproti předchozímu rozdílný tím, že sloupce v jedné kategorii jsou spojeny v jeden sloupec s jejich součtem a barevným vyznačením, které udává, kolik jednotlivé kategorie ve sloupcích zabírají (viz Obrázek 2-7).



Obrázek 2-7. Sloupcový skládaný graf

Sloupcový 100% graf skládaný

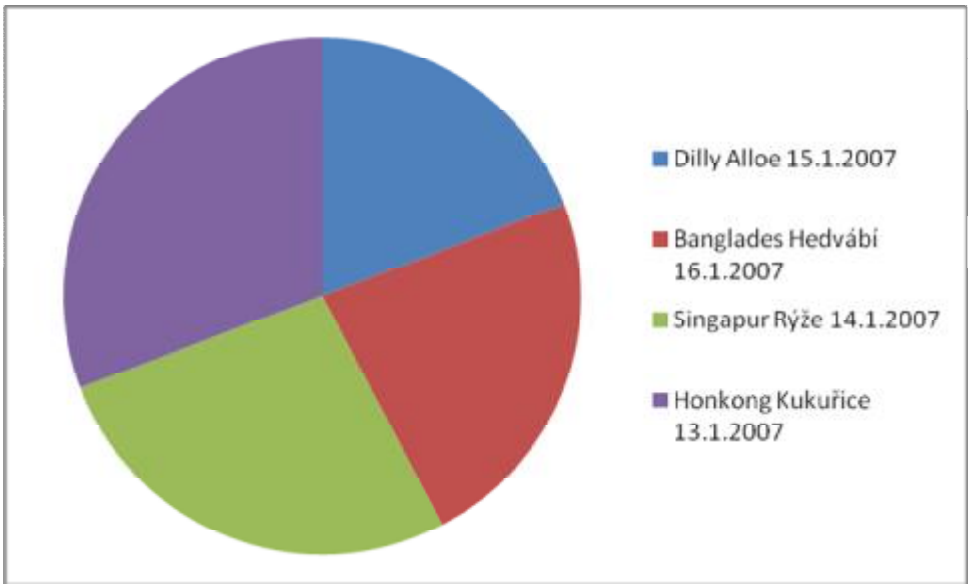
Tento graf je podobný předešlému, jen všechny sloupce jsou stejně vysoké a v každém sloupci jsou barevně odlišené jednotlivé kategorie, které ukazují procentuální zastoupení jednotlivé kategorie v dané dimensi. (viz Obrázek 2-8)



Obrázek 2-8. Sloupcoví 100% skládaný graf

Koláčový graf

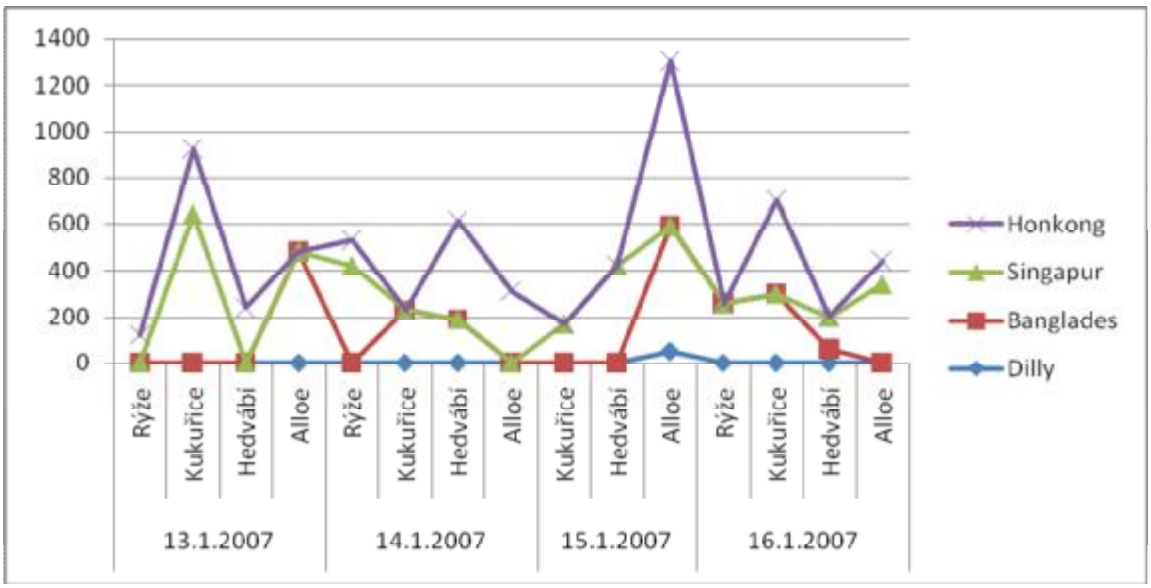
Tento typ grafu zobrazuje data v kruhových výsečích, kde každá výseč je jedna kategorie, kde její velikost je procentuální zobrazení její velikosti z celkového součtu všech kategorií.(viz Obrázek 2-9)



Obrázek 2-9. Koláčový graf

Bodový graf

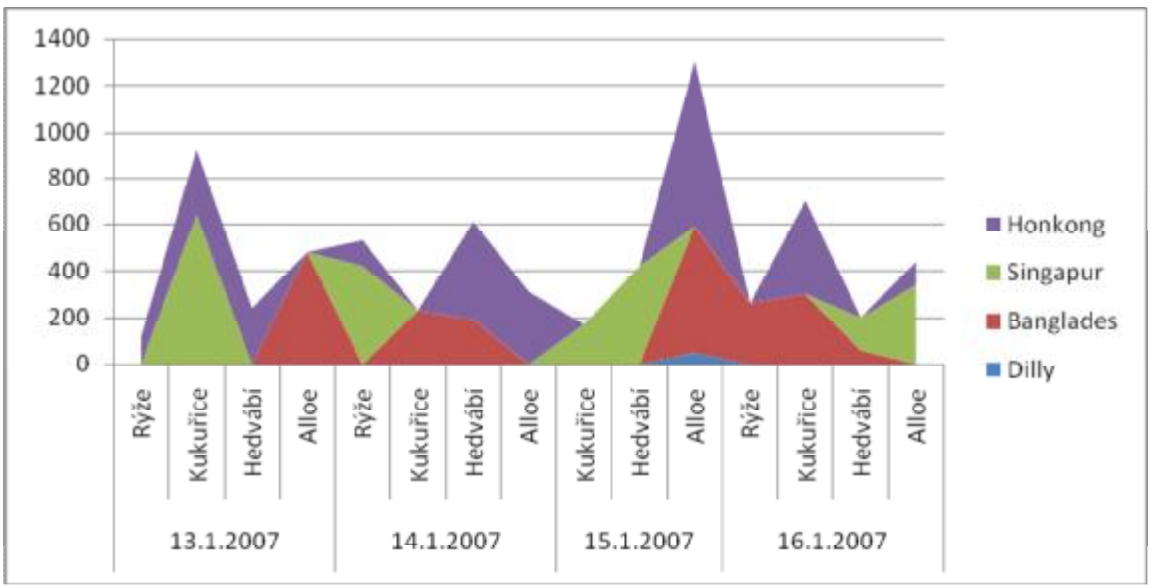
Zde je to podobné jako u sloupcového grafu. Na ose *x* jsou kategorie a na ose *y* míry, jen místo sloupců jsou body, které mohou být propojeny. (viz Obrázek 2-10)



Obrázek 2-10. Bodový graf

Plošný graf

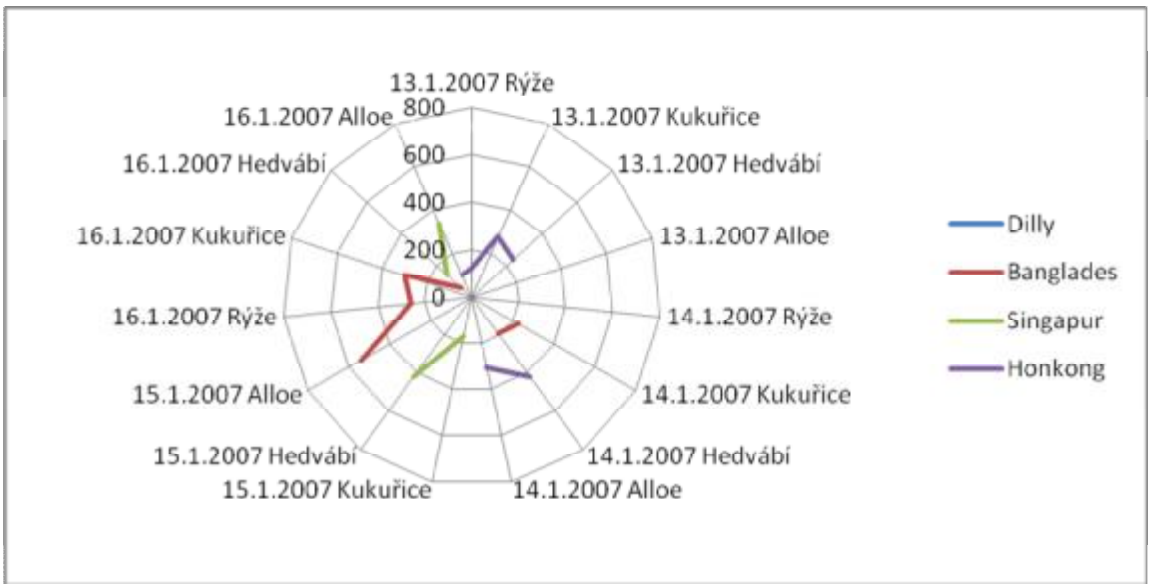
Je stejný s předchozím grafem, jen vyplňuje plochy pod spojenými body. (viz Obrázek 2-11)



Obrázek 2-11. Plošný graf

Radarový graf

Radarový graf je podobný grafu bodovému, ale jednotlivé hodnoty se vnášejí do kruhu.
(viz Obrázek 2-12)



Obrázek 2-12. Radarový graf

Další grafy

Existuje ještě celá řada grafů, které bych zde mohl zmínit, ale to přesahuje rámec mé práce.

2.2 Vizualizační modely OLAP systémů

Vizualizace je v oblasti vývoje OLAP systému hlavním tématem. Vývojoví pracovníci se snaží vytvářet systémy, které by dokázaly na malou plochu zobrazit velké množství dat tak, aby koncový uživatel měl přehled, jaká data jsme ze systému získali. Taková vizualizace je mnohdy velmi náročná a proto začaly vznikat vizualizační modely, které vytvářejí pozadí pro snadnější vizualizaci těchto dat. Vytvářejí logickou i prezentační vrstvu pro získávání dat z OLAP databází a snaží se je ve srozumitelné formě prezentovat.

I přesto, že je vizualizace tak zásadní část OLAP systému, byl vyvinut jen malý počet takových systému. Dnes jsou nejvíce známy tři systémy. Pravděpodobně nejznámější z nich, je systém vytvořený v oblasti komerčního vývoje. Jedná se o systém OLEDB od společnosti Microsoft. Tento model, díky silnému strukturovanému jazyku, může provádět dotaz nad velkým počtem kostek a může vytvářet složité reporty. Model OLEDB také klade velký důraz na samotnou prezentaci dat.

Dalším modelem, který se velmi málo používá je model páskový (*Tape model*). Tento model byl vyvinut na univerzitní půdě. V modelu je velké množství pásek, které představují jednotlivé dimense a jednotlivé buňky (*tracks*) na páskách její hodnoty.

Nejmladším modelem je CPM (*Cube presentation model*). Stejně jako předchozí modely i tento byl vyvinut na akademické půdě a v dnešní době se do něho vkládají největší naděje.

2.2.1 OLEDB

OLEDB (*Object Linking and Embedding, Database*)[8][14] není technologie primárně určená pro prezentaci OLAP dat, ale byla vytvořena jako nástupce technologie ODBC (*Open Databse Conectivity*). OLEDB oproti ODBC nebyla omezena jen na relační zdroje dat, ale díky své architektuře je schopna jako zdroj dat přijímat jak relační tak i nerelační zdroje dat, například jsou to poštovní úložiště, souborové systémy nebo grafická a textová data pro web.

OLEDB komponenty jsou složeny ze tří základních prvků.

- **poskytovatelů dat (data providers)** - zobrazují data. Většinou jde o ovladač nebo jiný program, který poskytuje OLEDB rozhraní pro komunikaci. Překládá data z databáze, na kterou je navázaný, do jazyku srozumitelného pro konzumenty.
- **konzumentů dat (data consumers)** - používají data. Konzument používá data od poskytovatelů k zobrazení koncovému uživateli.
- **servisních komponent (service components)** - přenáší data a provádějí s nimi procesy. Je to poskytovatel a konzument dohromady. Získá data od jiných poskytovatelů, upraví je a poskytne dále.

Díky tomuto způsobu jsou jednotlivé komponenty samostatně prodejné, a tím se usnadňuje práce jak vývojářům, tak i obchodníkům.

2.2.1.1 Architektura OLEDB

Díky svému členění funkcionality klasických databázi do logických celků umožňuje OLEDB relativně jednoduchý přístup k datům. Vývojoví pracovníci mohou kdykoliv jednotlivé logické celky upravit, či nahradit jinými. Jednotlivé data providery jdou nahradit providery jinými. Celý systém je založen na komponentní architektuře. Na komponenty se odkazuje pomocí ukazatelů a je zachována velká optimalizace.

Datové sklady jsou zobrazovány pomocí COM rozhraní. Díky tomu mohou být komponenty postaveny nad vším a mohou zobrazovat velké a robustní systémy.

Jelikož je tato architektura velmi otevřená, není problém vytvořit libovolného konzumenta. Tento konzument pak může spolupracovat s libovolným poskytovatelem. Vyměníme-li například za běhu databázi z MSSQL na Oracle, konzument to nepozná a bude stalé fungovat stejně.

2.2.1.2 OLEDB pro OLAP

V roce 1998 bylo OLEDB vylepšeno pro komunikaci s OLAP databází a bylo mu dán název OLEDB pro OLAP. Toto rozšíření přineslo možnost pracovat s datovými kostkami a dotazy nad nimi, díky novému jazyku založenému na bázi SQL, který byl pojmenován jako *Multidimensional Expression*

(MDX). Toto rozšíření umožnilo v rámci OLAP poskytovat velké vizualizační možnosti pro programátory a pro konečné uživatele. Díky architektuře OLEDB zůstala zachována možnost připojit se na libovolný zdroj dat.

2.2.2 Cube Presentation Model

Tento model se skládá ze dvou vrstev. Je to vrstva logická a prezentační. Na logické vrstvě se udržuje specifikace multidimensionálních kostek a prezentační úroveň potom vytváří prezentaci daných dat v 2D obraze. Díky tomuto rozdělení můžeme programově oddělit strukturu a data v logické vrstvě od vizuálního zobrazení těchto dat. Prezentační vrstva jde poté kdykoliv vyměnit za jinou bez potřeby upravovat logickou vrstvu[7][9].

2.2.2.1 Logická vrstva

Logická vrstva vytváří konstrukci datových kostek. Skládá se ze základních konstrukčních prvků, které dohromady utvářejí celou strukturu[7][9]. Jsou jimi:

- **Dimense –** je definována jako mřížka úrovní dimense *(L,* $\mathbf{p}$*)*, kde $\mathbf{p}$ je částečné uspořádání definované na úrovní L.

- **Funkce přechodu –** jsou to párové funkce $anc_{L_1}^{L_2}$, definovány jako každý pár L_1 a L_2 uspořádané podle L_1 $\mathbf{p}$ L_2. Funkce $anc_{L_1}^{L_2}$ mapuje každý prvek z dimense L_1 na dimensi L_2.

- **Detailní datové množiny –** je to množina $S = [L_1,\ldots,L_n,A_1,\ldots,A_m]$, kde L jsou jednotlivé úrovně dimense a A jsou jejich primární klíče. Tyto množiny modelují tabulky faktu na nejnižší úrovni agregace.

- **Výběrové podmínky φ –** tyto podmínky vylepšují funkce přechodu, pomocí nichž se mohou definovat podmínky, za kterých budou dvě úrovně propojeny. Můžeme zde použít operátory $\wedge,\vee,\neg$.

- **Primární datové kostky c –** je to
  $c=(DS^0,\varphi,[L_1,\ldots,L_n,M_1,\ldots,M_m],[agg_1(M_{01}),\ldots,agg_m(M_{0m})])$ kde:
  - DS^0 – jsou detailní datové množiny
  - Φ – jsou detailní výběrové podmínky
  - $M_1,\ldots,M_m$ - jsou míry
  - L – jsou jednotlivé úrovně dimense
  - Agg – jsou agregační funkce

  Primární datové kostky jsou úložiště dat.

- **Sekundární výběrové podmínky a kostky –** vyskytují se zde sekundární výběrové podmínky a kostky, které jsou založeny na primární kostce. Jednotlivé podmínky nadefinují, co chceme mít v sekundární kostce a to pak do této kostky vložíme.

Díky tomuto způsobu uložení primárních a sekundárních kostek nám vznikají výhody definovat si v sekundárních kostkách nové atributy a funkce.

2.2.2.2 Prezentační vrstva

Prezentační vrstva poskytuje pohled na data v 2D pohledu pomocí základních entit, které jsou[7][9]:

- **Body (*Points)* -** body na osách, které odpovídají klasickému vnímaní bodů v matematice. Každý bod je charakterizován pomocí příslušných podmínek na určité úrovni dimense. Jelikož je možné zobrazit několik dimensí na jedné prezentační vrstvě, tak může bod nabývat i významu z několika dimensí.

- **Osy (*Axis*) -** osa je v CPM chápána jako množina bodů. Osy dělíme do dvou typů a to na osu *neviditelnou* a osu *obsahovou.* Neviditelná osa určuje, kde se mají množiny dat nacházet. Osa obsahová určuje složitějším způsobem umístění jednotlivých kostek a jejich agregaci.

- **Multikostka (*Multicube*) –** multikostka je definována nad multidimensionálním prostorem, který zahrnuje skupinu os v jednom pohledu. Také by se dalo říci, že je definována nad daty v nejspodnější úrovni agregace. Na této kostce se poté provádí filtrování a agregace dat před samotnou prezentací uživateli. Multikostky se také mohou objevit v mapování multidimensionálního prostoru na nejnižší úroveň agregace.

- **2D-plát (*slice*) –** 2D plát je chápan jako vrstva vybraných dat, které se mohou zobrazit uživateli. Na první pohled se může zdát, že 2D-plát můžeme definovat jen nad dříve popsanými dvěma osami, ale jelikož i bod dokáže zobrazit data z více dimensí, tak 2D-plát umí zobrazit data v několika dimensích. Proto můžeme plát definovat jako množinu bodů skládající se z 2D dimensí.

- **Páska (*Tape*) –** pásku můžeme v CPM chápat jako řádek nebo sloupce v 2D-plátu, která je rovnoběžná s jednou z os. Každá páska poté obsahuje hodnoty míry pro dané dimense. Jelikož body můžou být významově brány z více dimensí, tak i páska může být chápána pro více dimensí.

- **Křížení (*Cross join*) –** chápeme ho jako křížení pásek dohromady. Jde o pásky, jejichž osy nejsou rovnoběžné a v jistém bodě se protínají. V takovém protnutí vznikne buňka, kterou označíme za protnutí dvou a více dimensí.

- **Obsahová funkce (*Content function*) –** obsahové funkce mají za úkol mapovat jednotlivé míry na multikostky se všemi podmínkami a v daném pořadí.

2.2.2.3 Příklad CPM

Abychom více přiblížili tento model, ukážeme si na jednoduchém příkladu celou funkčnost, podle [7][9]. V tomto příkladu budeme zobrazovat prodejnost produktů za určitý čas. Definujeme si kostku prodejnosti (*SalesCube)*, ve které budou tyto dimense: produkty (*Products)*, prodejce *(Salesman)*, čas *(Time)* a oblast *(Geography)*. Některé z dimensí mohou mít i několik úrovní agregace. Nastavíme omezení tak, že na dimensi čas nastavíme omezení pouze na rok 1991 a dimensi produkty budeme vždy uvažovat bez agregace a tudíž všechny hodnoty (viz Obrázek 2-13).

Year = 1991 Product = ALL			Venk USA USA_N		USA_S	Japan	Netz USA USA_N		USA_S	Japan
			Seattle	Boston			Seattle	Boston		
		Size(city)								
R1	Qtr1	Jan								
		Feb	C1		C2	C3		C4	C5	C6
		Mar								
R2	Qtr2									
R3	Qtr3									
R4	Qtr4	Jan								
		Feb								
		Mar								

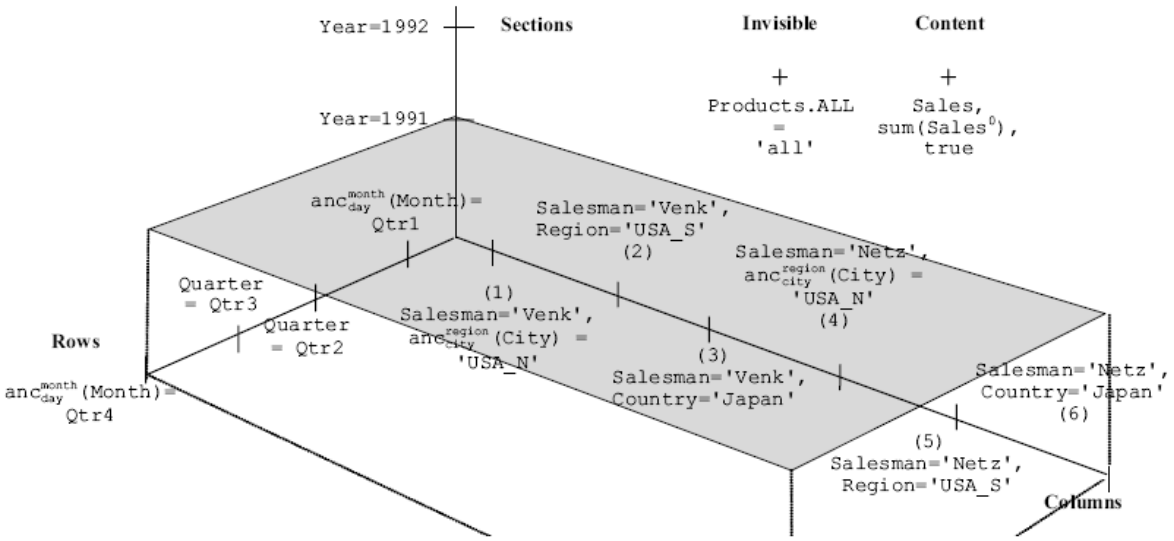
Obrázek 2-13. Příklad OLAP kostky

Kdykoliv chceme zobrazit data ve 2D obraze, která jsou více než jednodimensionální, potřebujeme získat křížení všech dimensí v daných osách. Ukážeme si to na dimensi prodejce, kterou pro ukázku omezíme jen na dva prodejce (*Venk,Netz*), dimensi oblasti na sloupcových osách a dimensi času na řádkové ose. U dimense produktů existuje několik úrovní agregace (stát (*Country*), region a město (*City*), …) a stejně i u dimense čas (čtvrtletí (*Quarter*), měsíc (*Month*), …).

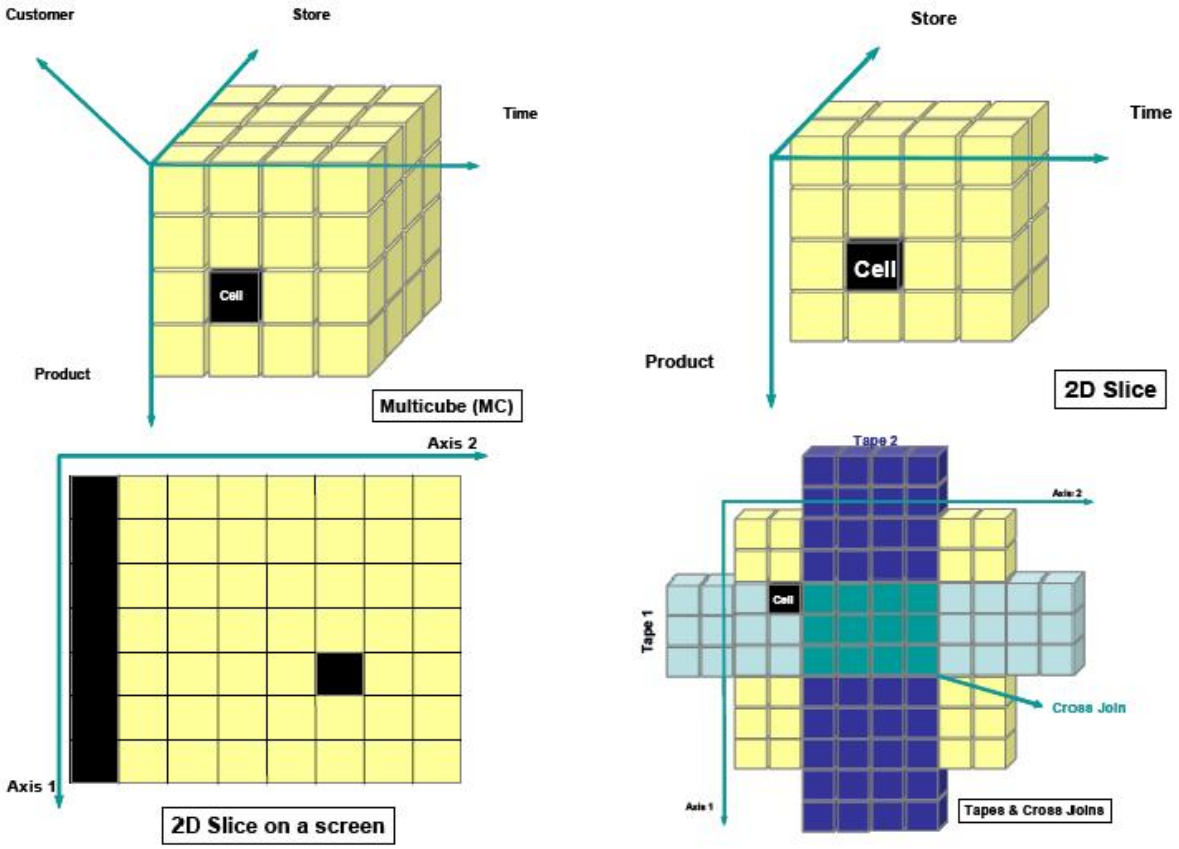
Budeme-li mít zadán následující dotaz:

SELECT CROSSJOIN({Venk,Netz},{USA_N.Children,USA_S,Japan}) ON COLUMNS
{Qtr1.CHILDREN,Qtr2,Qtr3,Qtr4.CHILDREN} ON ROWS
FROM SalesCube
WHERE (Sales,[1991],Products.ALL)

Tento dotaz představuje v CPM 2D-plát, který je definován čtyřmi horizontálními páskami označenými jako R1-R4 a šesti vodorovnými paskami označenými C1-C6. Horizontální pásky mají jednoduchou prezentaci a to časovou dimensi a jsou udávány ve čtvrtletích nebo měsících. U pásek vertikálních je to složitější. Představují kombinaci dimensí prodejce (*Venk,Netz*) s dimensí oblast (stát (*Country*), region a město (*City*)). Dvě dimense, které do daného zobrazení zasahují, jsou skryté, ale potřebujeme je také uvést. Jsou jimi dimense rok (*Year=1991*) a dimense produkt (*Product=all*). (viz Obrázek 2-14, Obrázek 2-15)



Obrázek 2-14. 2D-plát k příkladu OLAP kostky



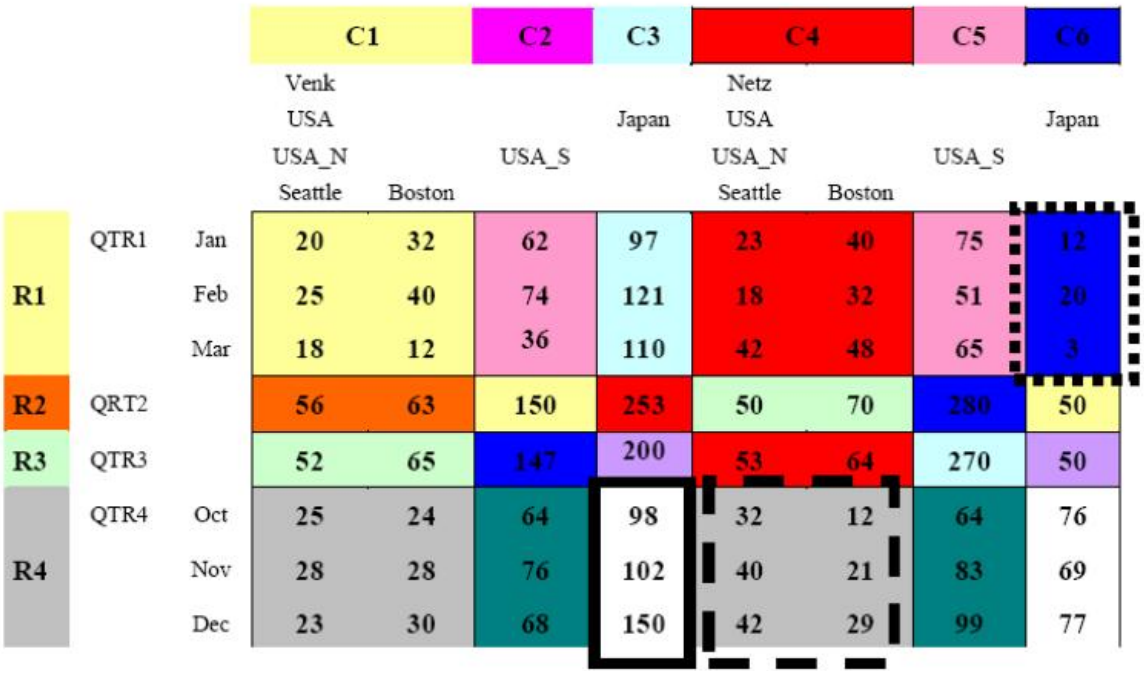
Obrázek 2-15. Model CPM se zobrazením multikostek, 2D-plátů a jejich křížení.

2.2.2.4 Vizualizace CPM

CPM je velmi dobře vytvořen pro datové spojení s OLAP systémem a proto už není problém získat data z OLAP. Největším problémem zůstává samotná vizualizace.

Jako příklad vizualizace CPM si ukážeme ve formě *Table Lens* (TL). CPM se na TL namapuje velmi snadno. Osy CPM se namapují na osy TL. 2D-plát CPM se zobrazí jako 2D-plát TL. Díky vysoké generičnosti CPM není problém poskytnout pro DOI dostatek informací a uživatel často volí i "okno zájmu", které vybírá část z 2D-plátu.

Hlavním problémem tohoto modelu je, jak uživateli co nejjednodušeji a automaticky poskytnout potřebné „okno zájmu". Musí se umět přizpůsobit požadavkům uživatele a zobrazovat data vždy ve srozumitelné podobě. Uživatel může zadávat různé požadavky na filtrování nebo řazení a okno zájmu se musí umět podle nich upravit. Také se musí přizpůsobit požadavkům na agregaci samotných dat. Proto se vyvíjejí různé algoritmy, které toto provádějí. (viz Obrázek 2-16)



Obrázek 2-16. vizualizace CPM pomoci Table Lens, kde jsou barevně a vizuálně vyznačená všechny křížení a hodnoty

2.3 Vizualizační nástroje na webových stránkách

2.3.1 HTML a webové prohlížeče

V dnešní době je pravděpodobně nejrozšířenější forma prezentování dat konečnému uživateli šířením dat po internetu.

Všem těmto vymoženostem ale předcházel dlouhý vývoj. Na začátku to bylo jen pár propojených počítačů, které si navzájem vyměňovaly omezené množství dat. Ale postupem času vyrostla internetová síť do celosvětového měřítka s miliony připojenými počítači najednou. Každý chtěl z internetu dostat co nejvíce informací a začali se objevovat první prostředky pro šíření informací po internetu. Zpočátku to byli jen malé programy poskytující informace v textové podobě. Kdo chtěl tyto informace získat, si musel příslušný program stáhnout a připojit se k serveru, který je poskytoval. Těmto programům se začalo říkat prohlížeče (*Browser*).

Lidé ale potřebovali data z internetu dostávat v srozumitelnější podobě a tak se snahy vývojářů obrátily na vizuální stránku. Začali se vytvářet prohlížeče s grafickou podobou a jednotlivá data na internet se začali propojovat pomocí tzv. hypertextových odkazů. Hypertextový odkaz byla adresa na jiné místo v internetu, pomocí něhož se dalo jednoduše mezi daty procházet.

U prohlížečů se rozšířil značkovací jazyk HTML, který umožňoval vytvářet soubory s informacemi v univerzální podobě. Každý webový prohlížeč zapojil do svého kódu vykreslovací jádro, které daný kód v HTML zobrazil. To otevřelo obrovskou možnost data zobrazovat.

V dnešní době existuje velká spousta prohlížečů, které mají svá vlastní vykreslovací jádra. K těm nejznámějším patří:

- **Internet Explorer –** Je to prohlížeč od firmy Microsoft a jako jeden z mála je komerčně zaměřený. Je velmi rozšířeny hlavně na platformě Windows.
- **Firefox –** Tento prohlížeč je jedním z nejvýznamnějších alternativních prohlížečů. Má v sobě zabudované jádro Gecko. Prohlížeč Firefox existuje na všech platformách.
- **Opera –** Prohlížeč Opera je stejně jako prohlížeč Firefox poskytován zdarma a jede taktéž na svém vlastním jádře.

2.3.2 HTML

Jelikož se značkovací jazyk HTML velmi šířil, začali si jej i různé prohlížeče upravovat do své podoby. To vedlo k potížím vytvořit dokument, který bude ve všech prohlížečích vypadat stejně. Proto zavedlo konsorcium W3C normu na HTML jazyk.

HTML jazyk se skládá z párových a nepárových značek a entit. Každá značka je ohraničena znakem „<“ na začátku a znakem „>” na konci. Značky mají své jméno a mohou obsahovat různé atributy, které určují podobu a jejich chování. Párové značky mají svůj koncový ekvivalent, který začíná znaky „</“ a je pojmenovaný stejně jako značka začínající. Příklad takové značky může vypadat takto:

* ahoj *

Nepárové značky se zapisuji se znakem „/“ před koncovým znakem. Například:

*
*

Jazyk HTML obsahuje velké množství těchto značek a širokou paletu nastavitelných atributů. Každý atribut potom utváří vzhled dané značky. Značky se mohou do sebe navzájem zanořovat, ale nesmí vznikat křížení úrovní. Některé značky mají jen význam ve formátování písma v daném místě, jiné zase utvářejí strukturu dané stránky.

Pro dynamičnost webových stránek byly vytvořeny značky hypertextových odkazů, pomocí níž prohlížeč vyvolává přechod na jiný dokument. Později se vytvořily i značky formulářových prvků, jako jsou různá tlačítka a výběrové a zadávací boxy. Data z těchto formulářových prvků se mohou pomocí metod POST nebo GET zaslat na server, kde se dále zpracovávají.

Kvůli snazšímu vytváření vzhledu HTML stránek se vytvořil i jazyk CSS, který dokáže pomocí stylů upravit stránku do vzhledné grafické podoby.

Existují jazyky, které na straně serveru vytvářejí HTML stránky přímo za běhu. Díky tomuto způsobu generování HTML se z jinak statických stránek začaly stávat stránky dynamické. Stránky reagují na podněty uživatelů a umožňují vyšší přehlednost prezentovaných dat. V dnešní době patří k nejvýznamnějším programovacím jazykům na serverové straně jazyky PHP, ASP.NET, JSP a ASP.

2.4 JavaScript a AJAX

JavaScript je skriptovací jazyk, pomocí něhož jsme schopni rozhýbat statickou stránku. Je možné pomocí něho vytvářet různé animace, reagovat na podněty od uživatele nebo zobrazovat hodiny v reálném času. JavaScript závisí na verzi prohlížeče. Každý prohlížeč má vlastní implementaci JavaScriptu, což vede k psaní nepřehledných kódů, které se snaží o překlenutí problému mezi verzemi a zajištění stejné funkčnosti pod všemi prohlížeči.

Díky názvu JavaScript by se mohlo zdát, že se bude jednat o jazyk podobný programovacímu jazyku Java. To je ale zavádějící. JavaScript se původně jmenoval LiveScript, ale byl vydáván ve

stejnou dobu jako programovací jazyk Java od firmy Sun a proto byl v rámci marketingového tahu přejmenován na JavaScript.

Tento jazyk je svou syntaxí podobný programovacímu jazyku C. Má stejné programovací struktury, ale oproti jazyku C je beztypový. Prakticky má typovost v sobě zabudovanou, ale typ se proměnné striktně neurčuje. Typ se určí až podle hodnoty, která je do něj přidělená.

Kód napsaný v JavaScriptu se nazývá skript. Skript může být volně umístěn do stránky, nebo je napsán v samostatném souboru s příponou „*.js“ a ze stránky se z něj vytvoří jen odkaz. Skript se umisťuje do stránky pomocí značek „<script>“, které mají další nastavitelné atributy typu skriptu nebo externího souboru. Příklad jak se takový skript vkládá do stránky, je zde:

```
<html>
     <head>
     ...
       <script language="JavaScript" type="text/javascript">
       .. javascript tělo skriptu ..
       </script>
     ...
     </head>
     <body>
     ..tělo dokumentu..
       <script language="JavaScript" type="text/javascript">
       .. javascript tělo skriptu..
       </script>
     ..tělo dokumentu..
     </body>
</html>
```

V hlavičce stránky se většinou uvádějí odkazy na externí JavaScripty a v těle se pak provádí skript zapsaný přímo do stránky.

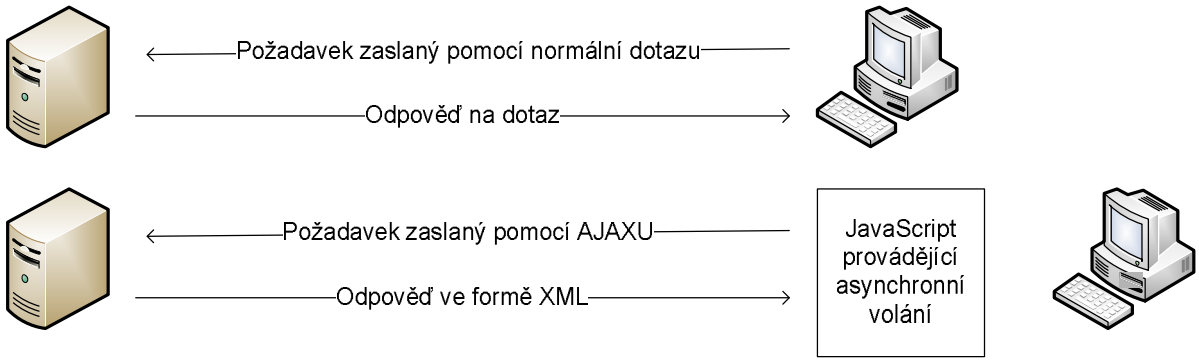
JavaScript umí reagovat na různé události od uživatele. Například zmáčkne-li uživatel tlačítko myši nad nějakým prvkem ve stránce nebo jen prostě nad tím prvkem myší přejede, tak se spustí funkce, které jsou v JavaScriptu na tyto událost navázané. Události se zapisují buďto přímo jako atribut k prvku:

```
<input type=”button” value=”test” onclick=”alert(‘test’)”>,
```

nebo přímo pomocí speciálních funkcí JavaScriptu, kde v IE to je funkce *attachEvent* a v Mozile a Opeře to je funkce *addEventListener*.

Jelikož JavaScript je jazyk běžící na straně klienta, který si uživatel nevědomky stáhne ve chvíli, kdy se chce na danou stránku podívat. Musí být u něho kladen velký důraz na bezpečnost. Nemá přístup do souborového systému počítače a lidé si mohou JavaScript ve svých prohlížečích vypnout.

U JavaScriptu byl dlouho velký problém v tom, že chtěl-li JavaScript oznámit nějakou skutečnost serveru, nebo chtěl-li něco od serveru získat, musela se celá stránka znovu načíst pomocí dotazu na server. To se ale vyřešilo později pomocí technologie AJAX (*Asynchronous JavaScript and XML*). Tato technologie umožnila asynchronní komunikaci se serverem pomocí dotazů posílaných na server pomocí klasického požadavku POST nebo GET a server poté odpovídal ve formě XML, který JavaScript zpracuje a získá z něho informace, které potřebuje. (viz.: Obrazek 2-17)



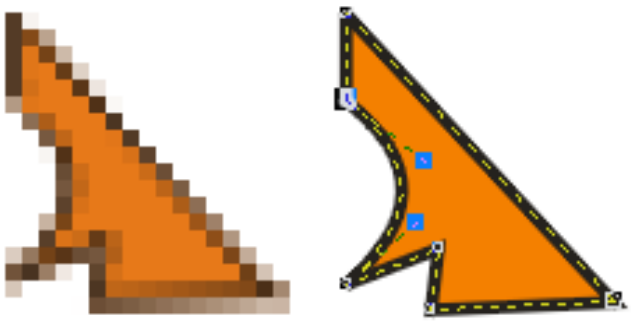
Obrazek 2-17. Schéma normálního a AJAX dotazování na server

Technologie JavaScriptu a AJAX otevřela vývojovým pracovníkům obrovské možnosti, co se týká vizualizace. Pomocí těchto dvou technologií se mohou data na internetu zobrazovat dynamicky podle požadavků uživatele. Do budoucna se uvažuje, že stránky už nebudou existovat fyzicky na serveru, ale vše se bude dynamicky generovat pomocí JavaScriptu a pomocí AJAXu získávat ze serveru potřebná data z databází.

2.5 Flashové animace

Flash je animace, kterou můžeme umístit na webové stránky nebo ji prostě pustit doma pomocí programu „*Flash player*“ [13]. Je to vlastně sled obrázků, které se postupně zobrazují za sebou a tvoří tak dojem pohybu. Podobného principu se využívá třeba v obrázcích GIF (*Graphics Interchange Format*), kde se jednotlivé obrázky vykreslují za sebou v rastrové podobě, kdy je výsledný obraz vytvářen jako popis všech bodů obrázku. Bohužel je tento způsob animace velmi náročný na velikost výsledných souborů. U Flashe je oproti tomu zaveden princip grafiky vektorové. Vektorová grafiky

zaznamenává jen informace o objektu a jeho způsobu vykreslení a ten zobrazuje podle daného měřítka. Oproti rastrové grafice má vektorová grafika velkou výhodu v tom, že ať už je obrázek jakkoliv zvětšován či zmenšován, zachovává si stále stejnou kvalitu a také je mnohdy výsledný soubor mnohem menší. Problém nastává ve chvíli, kdy se daný obraz nedá jednoduchým způsobem popsat.



Obrázek 2-18. Rozdíl mezi rastrovou a vektorovou grafikou

Flashové animace můžeme vytvářet ve vývojových nástrojích k tomu určených. Dříve to bylo vývojové studio od firmy Macromedia, ale později se už používalo jen studio od firmy Adobe. Na trhu existuje celá řada takovýchto nástrojů a mají své výhody a nevýhody[13].

Do animace můžeme vkládat různé obrazy i texty. Celá animace je rozdělena do vrstev a můžeme každou vrstvu upravovat odděleně od těch ostatních. Abychom mohli vytvořit pohyb obrazu, má Flash v sobě zabudovanou podporu „*framu*“. Každý frame je jeden snímek animace. Díky vrstvám pak můžeme každou vrstvu animovat odděleně od těch ostatních.

Po vytvoření celé animace se program zkompiluje do binárního souboru „*.swf“. Výsledný zkompilovaný soubor se však už nedá dále upravovat a proto je potřeba stále udržovat zdrojové soubory.

Flash se dá umístit do webové stránky pomocí značky „<object>“ nebo „<embed>“. Bohužel se zde opět objevuje problém s jednotlivými prohlížeči, kde každý prohlížeč bere v úvahu jinou značku a v každém prohlížeči se také jinak nastavuje. Proto vznikly způsoby, jak animace správně do stránek umisťovat [13].

```
<OBJECT classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
        codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/s
      wflash.cab#version=6,0,40,0"
        WIDTH="550"
        HEIGHT="400"
        id="myMovieName">
        <PARAM NAME=movie VALUE="myFlashMovie.swf">
        <PARAM NAME=quality VALUE=high>
```

```
        <PARAM NAME=bgcolor VALUE=#FFFFFF>
        <EMBED src="/support/flash/ts/documents/myFlashMovie.swf"
            quality=high
            bgcolor=#FFFFFF
            WIDTH="550"
            HEIGHT="400"
            NAME="myMovieName"
            ALIGN=""
            TYPE="application/x-shockwave-flash"
            PLUGINSPAGE="http://www.macromedia.com/go/getflashplayer">
        </EMBED>
</OBJECT>
```

Na daném příkladu lze vidět způsob, jak vložit animaci do stránky tak, aby ji všechny prohlížeče zobrazovaly stejně. U značky „*object*“ je povinné zadat atribut „*classid*“, který určuje identifikační číslo assembly programu, který by měl s daným „*objectem*“ umět pracovat. Značka „*codebase*“ poté popisuje místo, kde se dá interpret stáhnout, pokud není nalezen na lokálním počítači. Obdobný atribut se nachází i u „*embed*“ a jmenuje se „PLUGINSPAGE“. Určení toho, jaká animace se má zobrazovat, se u obou značek provádí pomocí atributu „*src*“, který představuje relativní nebo absolutní cestu k dané animaci.

Historie Flashe sahá až do roku 1994. Tenkrát se ale ještě nejmenoval Flash, ale „SmartSketch. Nevyhovoval ale požadavkům na rychlý animační program a tak se neuchytil. V té době se ale objevil první webový prohlížeč, který podporoval zásuvné moduly. A tak se objevuje firma Macromedia, která daný projekt koupila a vytvořila první verzi Macromedia Flash 1.0.

Od té doby prošel Flash dlouhým vývojem, kde jeho vývoj převzala firma Adobe a ta ve vývoji pokračuje dodnes. Flash má i podporu skriptování pomocí jazyka ActionScript.

2.6 ActionScript

S prvními verzemi Flashe se objevila potřeba interaktivně řídit animaci, kvůli tomu vznikl programovací jazyk ActionScript [13][15]. V té době se k podobnému účelu na webu hojně používal JavaScript, proto se ActionScript vydal v jeho stopách. Nejdříve šlo jenom o jednoduché příkazy k zastavení animace, přesunu na snímek apod. V dnešní době je ve verzi 3.0 a obsahuje řadu pokročilých vlastností, jako objektové programování. Je ho možné jednoduše používat k vytváření her, interaktivních prezentací a webů.

Pomocí ActionScriptu se dají ze statických animací udělat animace dynamické, reagující na podněty od uživatele. ActionScript pracuje se základním objektem „_root“. Tento objekt je vlastně

animace samotná. Do tohoto objektu můžeme přidávat další objekty nazvané „MovieClip“. Tyto objekty jsou vlastně už samotné vykreslené obrazce, kterým jen určíme tvar a chování.

Jelikož je ActionScript jazyk používaný ve Flashových animacích umísťovaných nejčastěji na internetové stránky, bylo zapotřebí vymyslet propojení mezi ActionScriptem a JavaScriptem. Zpočátku to byla jen jednostranná komunikace od ActionScriptu k JavaScriptu pomoci funkce „*GetUrl*“. Tato funkce nedělala nic jiného, než že vyvolala na dané stránce url, které jí bylo zadáno a jelikož do url se může zapisovat i JavaScript, tak nebyl problém volat JavaScriptové funkce.

To bohužel nestačilo a tak se ve verzi 2.0 objevil objekt „*ExternalInterface*“. Pomocí tohoto objektu šlo volat z ActionScriptu JavaScriptové funkce pomocí funkce „*call*“. Do této funkce se jako první argument zadá název JavaScriptové funkce a další argumenty jsou už jednotlivé parametry, které se dané funkci posílají.

Komunikace od Javascriptu k Actionscriptu je o něco složitější. Na úrovni Actionscriptu se musí nejdříve daná funkce vytvořit a definovat její atributy. Poté se na začátku spuštění animace provede funkce „*addCallback*“. Tato funkce má jako vstupní parametr název, pod kterým se bude funkce volat, a jako třetí vstupní parametr má již vytvořenou funkci. Z JavaScriptu už poté stačí jen získat správný objekt Flashové animace a zavolat na tento objekt funkci, která je stejně pojmenována jako byl první parametr, který jsme napsali do funkce „*addCallback*“. Asi nejsložitější na celém tomto způsobu je najít ve stránce správný objekt, se kterým daný prohlížeč umí komunikovat.

Flash se dnes používá na tvorbu interaktivních webů, animací a i třeba her (viz.: Obrázek 2-19). Je to velmi silný vizualizační nástroj. Umožňuje dynamické změny pohledu bez potřeby znovunačtení stránky. Jako každý nástroj má své výhody a nevýhody. Vytvoříme-li třeba celou webovou prezentaci pomocí Flashe a konečný uživatel poté nebude mít danou verzi Flash Playeru, nebo nebude mít dostatečný výkon, tak se Flashová animace ani nezobrazí.

Obrázek 2-19. Příklad hry vytvořené pomocí Flash

2.6.1 Další možnosti vizualizace na webu

Pro webové stránky se postupem času objevily i další nástroje pro umožnění větší interaktivnosti webu. Nejznámější je třeba „*JavaApplet*“, který je založen na objektově orientovaném jazyku Java od firmy Sun. Tento objekt umožňuje na webu vytvářet formuláře a různé komponenty.

Dnes asi nejvíce diskutovaný vizualizační nástroj pro webové prezentace je technologie „Silverlight“. Tuto technologii uvedla nedávno společnost Microsoft. Je to technologie založená na JavaScriptu. Uživatel si na svůj počítač prakticky nainstaluje jen JavaScriptové knihovny. Server pak už jen posílá klientovi kód zapsaný v XAML, který Silverlight přeloží do JavaScriptu.

2.6.2 Zhodnocení a budoucnost

V dnešní době už pomalu upadá počáteční nadšení z technologií jako je Flash, nebo JavaApplet, protože tyto objekty těžkopádně komunikují se zbytkem webové stránky. Proto se dnes největší naděje vkládají do Silverlightu a JavaScriptu. Díky AJAX technologii vznikl i pojem WEB 2.0, který vlastně představuje webovou stránku, která komunikuje se serverem bez potřeby znovu načítat stránku. Objevil se i pojem AJAX 2.0, který má fungovat tak, že celá stránka bude generována

pomocí JavaScriptu, který si klient stáhne a JavaScript poté bude pomocí AJAX komunikovat s webovými službami a získávat data z databáze.

Asi nejzajímavějším projektem na webu je projekt „EyeOs“, který simuluje operační systém přímo na webových stránkách.

3 Srovnání některých řešení zobrazování OLAP dat v grafech

V této kapitole se budu zabývat srovnáním některých komerčních i nekomerčních řešení zobrazování OLAP dat formou tabulky a grafu. Zaměřil jsem se hlavně na řešení webová a to jsou komerční „Dundas Chart for .NET - OLAP Services“, „FusionCharts“ a nekomerční „OpenI (Open intelligence)“. Na konec ukážu WinApp systém „Chart FX OLAP“ a jeden klasický systém od společnosti Microsoft v jejich tabulkovém procesoru „Excel“. Systémy budu hodnotit podle toho, jakou funkčnost poskytují a podle přehlednosti a celkového vzhledu.

3.1 Dundas Chart for .NET - OLAP Services

Tento systém se mi z obou zkoumaných webových řešení líbil nejvíce. Po zobrazení internetové stránky s on-line zkušební verzí, která se nachází na stránkách společnosti, se mi objevila pěkně vypadající stránka, která má přehledně rozmístěny všechny kategorie a míry v levém panelu, zbytek obrazovky už doplňuje graf [3]. (viz Obrázek 3-1)

Obrázek 3-1. Rozložení Dundas Chart

V hlavní zobrazovací ploše se dá přepínat mezi zobrazením grafovým nebo tabulkovým. V zobrazení grafovém můžeme navíc přepínat, zda chceme vidět graf sloupcový nebo koláčový.

Lépe použitelný se mi jeví graf sloupcový, jelikož v něm můžeme přehledně provádět příkazy „Roll-Up“ a „Drill-Down“ pomocí ikony plus či ikony mínus.

Míry se dají přidávat vybráním v pravém menu „Measures“, nebo pomocí přetažení z levého panelu do prostoru grafu.

Z levého panelu se dají do prostoru grafu přetahovat i kategorie a to stejným způsobem jako míry.

Pomocí nabídky v dolní části se dají dále provádět operace „Slicing“ a „Dicing“. Kategorie, které přidám do grafu, se zobrazí i v tomto panelu a v nich můžeme provádět výběr hodnot, které chceme vidět.

Operace „Pivot“ se dá provádět z horní nabídky, ve kterém lze dále provést tisk a další operace.

Celkově se mi tento systém líbil, jenom by mohl mít větší nabídku grafů. Z pohledu funkčnosti hodnotím možnost drill-down a roll-up pomocí ikony plus velmi kladně. Co se týká vzhledu a přehlednosti se nedá nic vytknout.

3.2 Fusions charts

Tento systém je zaměřen jen na grafy, neposkytuje k nim žádné uživatelské rozhraní. Pro použití musí programátor začlenit graf, který chce použít, do svého systému.

Každý graf je vytvářen jako Flashová animace a umožňuje svoje nastavení pomocí definovaného XML souboru. Má velké možnosti, co se týká interakce s uživateli.

Na výběr je mnoho 2D a 3D grafů. Nejvíce mne zaujaly grafy, ve kterých se kombinují různé typy dohromady. Například kombinace grafu sloupcového a bodového (viz Obrázek 3-2) nebo graf, ve kterém je oproti předchozímu navíc ještě i plošný [6]. Viz Obrázek 3-3

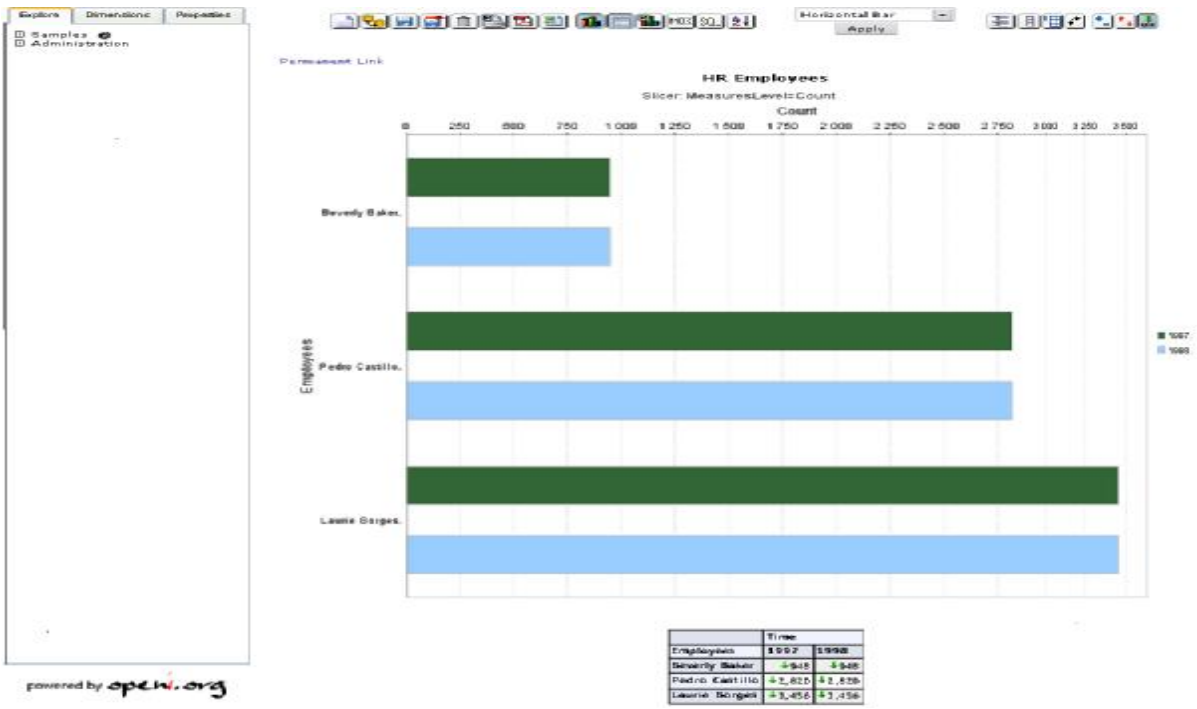
Obrázek 3-2. Kombinovaný sloupcoví a bodoví graf

Obrázek 3-3. Kombinovaný sloupcoví, bodoví a plošný graf

Celkově bych hodnotil tyto grafy jako velmi povedené, přehledné a pěkné. Po stránce funkčnosti je graf závislý na tom, jak programátor implementuje události, které graf vyvolává.

3.3 OpenI

Tento systém je jako jediný mnou zkoumaný zdarma stažitelný a ke svému běhu nepotřebuje žádné komerční nástroje. Má sice možnost připojení i na MSSQL – Analysis Services, které je komerční, ale stejně tak dobře se může připojit i na volně dostupnou databázi MySQL pomocí „Mondrian“ provideru [5].



Obrázek 3-4. OpenI

Stejně jako předchozí řešení má opět míry a kategorie v levém menu v záložce „Dimensions“. Výběr probíhá tak, že si pomocí ikonek zvolím, co má být míra a co kategorie. V hlavním panelu jsou zobrazeny dohromady graf i tabulka. Graf může mít oproti předchozímu systému více podob a

může zobrazovat i jednotlivé míry v součtu. Bohužel zde chybí jakákoliv interakce s uživatelem přímo z grafu. (viz Obrázek 3-4)

Operace „Roll-Up“ a „Drill-Down“ se dají provádět pouze v tabulce, ale projevují se i na grafu. Operaci „Pivot“ lze opět provádět z horní nabídky, kde je již zmiňované přepínání typu grafu a další možnosti zobrazovaní či tisku. Operace „Slicing“ a „Dicing“ se provádějí v levé nabídce v záložce „dimensions“.

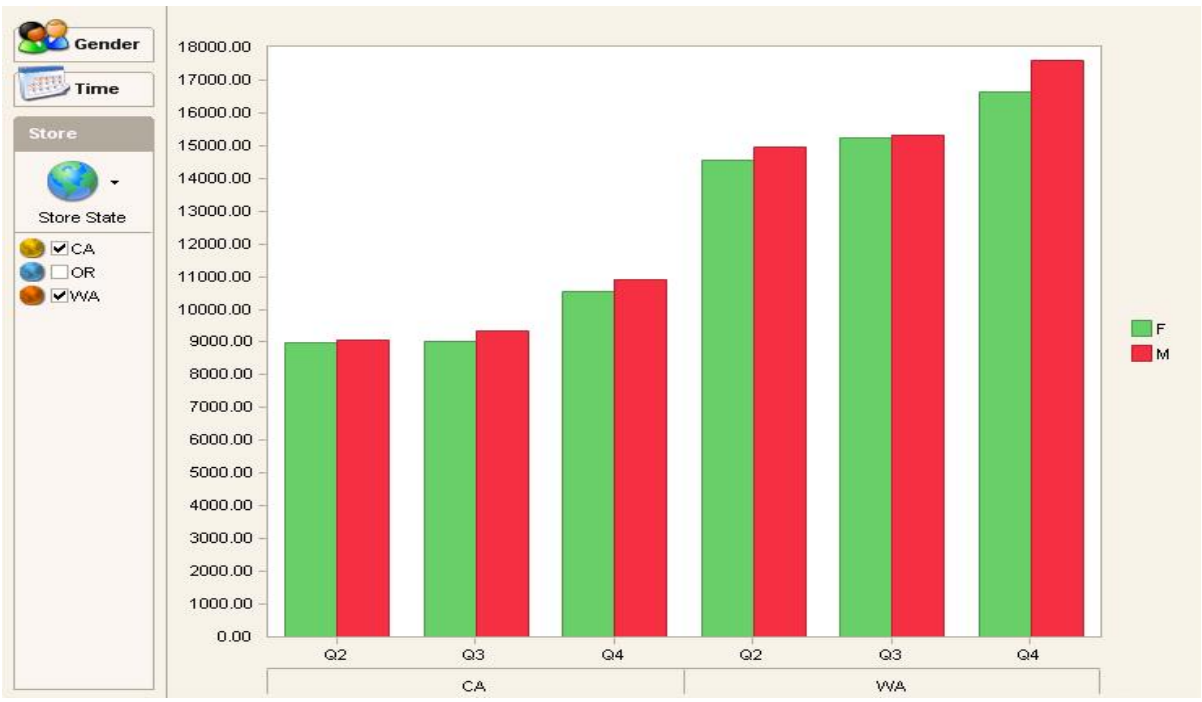
Tento systém má největší výhodu v tom, že je to open source, ale má pár nedostatků, které by bylo dobré dotáhnout do konce. Například rychlost zobrazování, či nemožnost si přímo z grafu zvolit, co chci vidět.

Proto bych tento systém po stránce funkčnosti hodnotil jako nevyhovující a po stránce přehlednosti a vzhledu jako přijatelný.

3.4 Chart FX OLAP

Tento systém je první z mnou zmíněných, který je určen pro spouštění na cílových počítačích. Je velmi strohý na ovládání a nepřehledný. Nemá žádnou možnost zobrazit data v tabulce a zobrazuje je pouze v grafu.

Je zde možnost si zvolit jen jeden typ grafu, a to sloupcový. V levém menu zobrazují jednotlivé kategorie, ze kterých je možno filtrovat („Slicing“ a „Dicing“). Systém neumožňuje provádět operace „Roll-Up“ a „Drill-Down“ a operace „Pivot“ se provádí přetahováním legend [4]. (viz Obrázek 3-5)



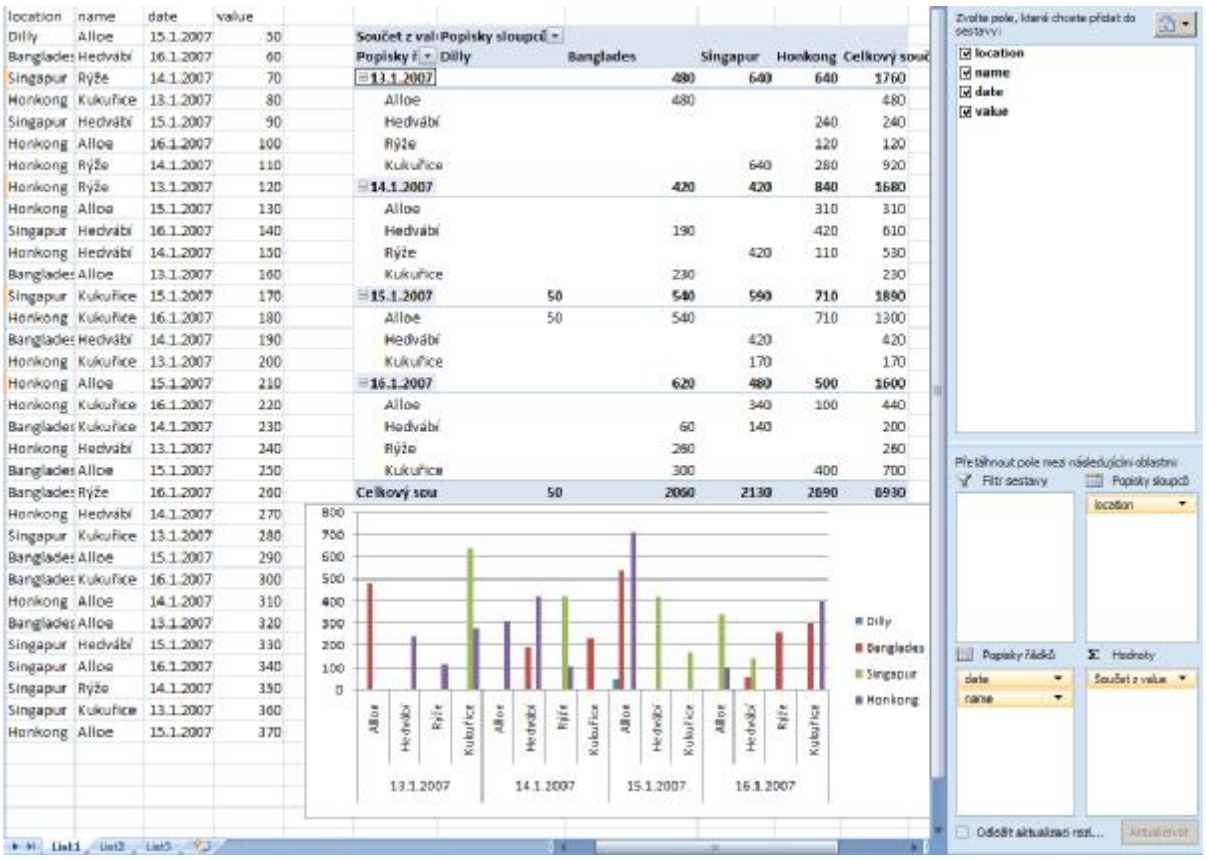
Obrázek 3-5. Chart FX OLAP

Tento systém je po všech stránkách mého hodnocení naprosto nevyhovující. Není přehledný ani dobře použitelný. Je složitý na pochopení některých funkcí. Snad jen po stránce vzhledu je přijatelný, ale vzhledem k tomu, že je to desktopová aplikace, mohl vypadat lépe.

3.5 Microsoft Excel

Co se týká windows form aplikací, je Excel asi nejpoužívanější pro zobrazování OLAP dat. Data se mohou do kontingenčních tabulek zobrazit buďto z databáze nebo zadáním hodnot přímo do Excelu a program je poté schopen sám z těchto dat vygenerovat následnou kontingenční tabulku (viz Obrázek 3-6).

V takto vygenerované kontingenční tabulce se poté dají provádět různá nastavení zobrazení. V pravé nabídce se volí, které kategorie budou jako sloupce a které jako řádky a co bude míra. Tímto způsobem se dá provádět i přehledný „Pivoting“.



Obrázek 3-6. MS Excel

V tabulce se dají potom pomocí ikonek plus a mínus schovávat různé kategorie, což jsou funkce „Roll-Up“ a „Drill-Down“. Operace „Slicing“ a „Dicing“ se provádí opět v právé nabídce v poli Filtr sestav.

Z takto vzniklé tabulky se dá vygenerovat graf. Grafů máme na výběr velké množství typů, nejpoužívanější jsou asi sloupcové a koláčové. Přehledné jsou i plošné či spojnicové. Jakákoliv změna v tabulce se projeví i v grafu a stejně tak jakákoliv změna v grafu se promítne do tabulky.

Systémy společnosti Microsoft jsou na trhu již velmi dlouho, a proto měly šanci se vyvinout dle přání uživatelů do nejlepší podoby. Proto je taky tento systém velmi pěkný na pohled a po stránce jednoduchosti, přehlednosti a funkčnosti mu není skoro co vytknout.

3.6 Srovnání zde popsaných systémů.

Systémy byly srovnávány podle tří základních kritérií a to přehlednost, jednoduchost a použitelnost. Jako nejpoužitelnější systém se jeví v internetových aplikacích Dundas Chart, a to díky svému přehlednému a jednoduchému použití.

Grafy Fusion Charts jsou také dobře použitelné a jednoduché, už jen díky velké nabídce typů a také proto, že jsou všechny dělány pomocí Flashe.

Systém OpenI se řadí mezi méně použitelné, protože byl vyvíjen jako nekomerční řešení. Obsahuje spoustu chyb a nedodělků, ale velkou výhodu je open source licence a tudíž se může i dle libosti upravovat.

U Windows aplikací je nejlépe použitelný MS Excel kvůli jeho snadné a jednoduché použitelnosti.

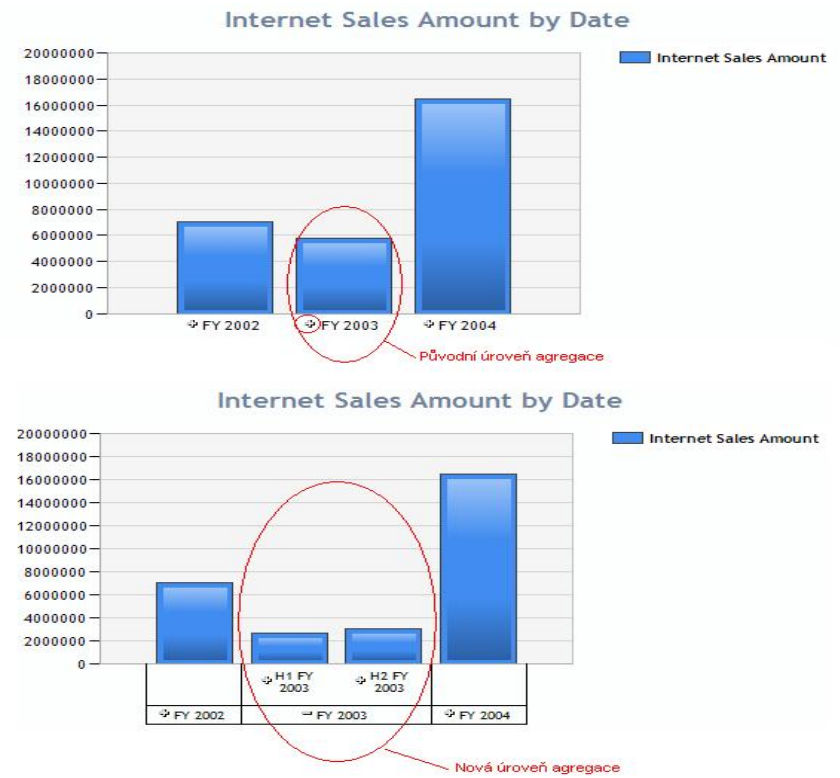
Tato srovnání jsem prováděl jen na těchto systémech, ale existuje i celá řada jiných systémů, které by se daly srovnávat a některé by možná byly i lepší než ty, co jsem popisoval.

4 Návrh knihovny a podoby grafů

Pří návrhu grafů jsem vycházel z již existujících řešení OLAP grafu, zvláště pak z řešení od firmy Dundas Chart [3] a firmy FusionChart[6]. Grafy obou firem jsem popisoval v předchozí kapitole. U firmy Dundas Chart jsem se inspiroval myšlenkou samotné interakce grafu s uživatelem a u firmy Fusion potom jejich možností jednoduchého zasazení do jakéhokoliv systému.

4.1 Návrh uživatelského rozhraní

Od firmy Dundas Chart [3] jsem převzal myšlenku možnosti zobrazovat více úrovní agregace v jednom grafu, kterou mají implementovanou pomocí tlačítek „+“ a „-“. Tyto tlačítka se nacházejí vedle názvu kategorií na ose x. Tlačítko „+“ provádí přechod z vyšší úrovně agregace do nižší a to tak, že nová úroveň agregace se objeví jako graf vnořený do původního grafu a nahradí původní kategorii. (viz: Obrázek 4-1) Pomocí tlačítka „-“ se poté dá provést opět přechod na původní úroveň agregace.



Obrázek 4-1. Ukázka přechodu mezi úrovněmi agregací

Díky tomuto způsobu lze libovolně procházet data a vidět úrovně agregace bez ztráty přehledu o zbytku dat a můžeme tak porovnávat více úrovní vedle sebe. Bohužel však tento způsob neumožňuje získávat nižší pohledy i pro jiné dimense než je dimense právě zobrazená.

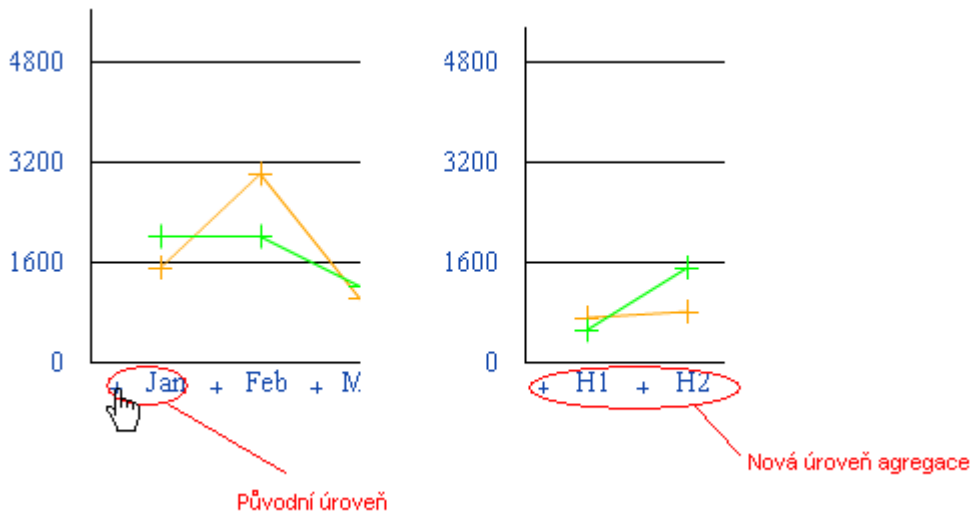
Můj návrh na uživatelské rozhraní se tedy zaměřil na více druhů grafů, a to na grafy sloupcové, plošné a bodové. Každý graf je vykreslován do souřadného systému „x, y“, ve kterém jsou na osu x vynášeny hodnoty měr jednotlivých kategorií a na ose y se poté vynáší kategorie, přičemž objekty v ploše jsou také kategorie.

Pro přechod mezi úrovněmi jsem navrhl tři způsoby popsané níže.

4.1.1 Přechod mezi úrovněmi agregací pomocí „+“ a „-“

Pod grafem budou, stejně jako u Dundas Chart, vedle názvu kategorií tlačítka „+“ a „-“. Tato tlačítka jsou navržena tak, aby poskytovala možnost průchodu mezi úrovněmi agregací. Jelikož stisk tlačítka vyvolá jen událost, která se projeví na JavaScriptové straně aplikace (vysvětlím později), můžeme provést i jiné akce než zde budu popisovat.

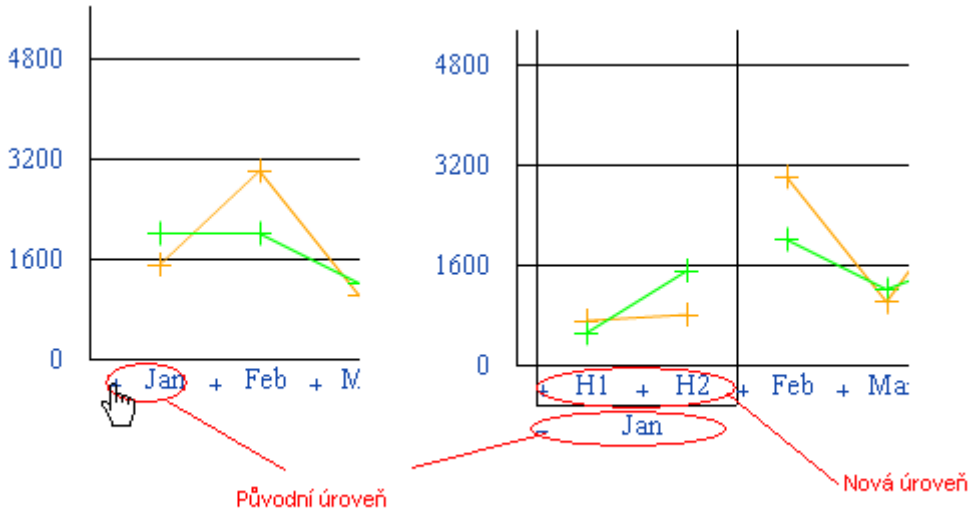
- První možnost, jak se projeví stisk tlačítka „+“ je, že se daný graf celý znovu načte a zobrazí jen hodnoty nové úrovně agregace. Zde ale ztrácíme možnost se dostat pomocí rozhraní grafu zpět na původní úroveň.



Obrázek 4-2. Zobrazení přechodu mezi úrovněmi agregací se ztrátou předchozích úrovní

- Druhou možností je princip podobný předchozímu. Graf opět zobrazí novou úroveň agregace, ale předchozí úrovně agregace jsou stále vidět jako okolí nové úrovně. Tato úroveň je v grafu oddělená svislými čárami, které z pravé a levé strany ohraničují novou oblast. V popisu kategorií je stále vidět popis předchozí úrovně agregace, ale už

zde není tlačítko „+“, ale „-“, je posunuto o pár pixelů dolů a u osy x jsou nyní nové názvy kategorii v dané úrovni.(viz: Obrázek 4-3)



Obrázek 4-3. Zobrazení přechodu mezi úrovněmi agregací bez ztráty předchozích úrovní

Pomocí tlačítka „-“ se potom u druhého způsobu může rušit daná úroveň agregace.

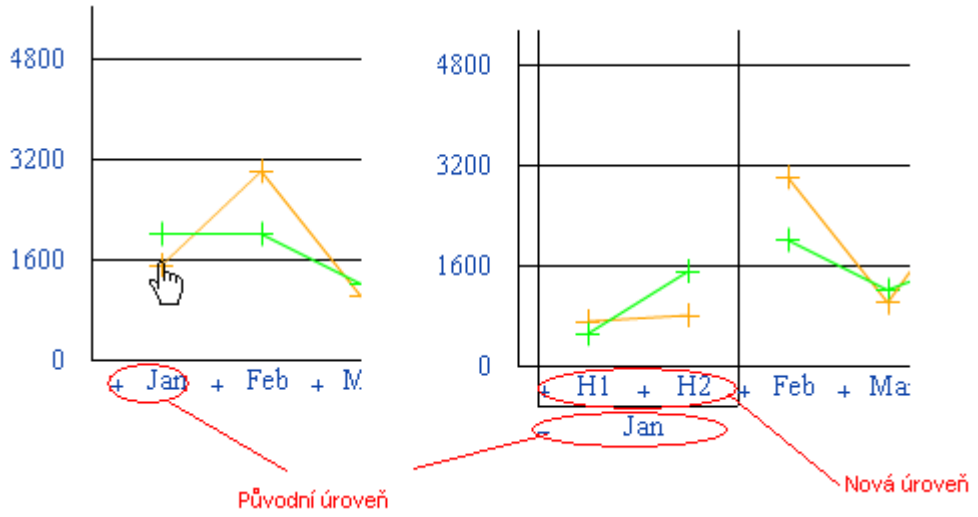
4.1.2 Přechod mezi úrovněmi agregací pomocí kliku do plochy grafu

Další možností jak provést přechod do nižší úrovně agregace je jednoduše kliknutím do plochy grafu. U sloupcového grafu to je kliknutí na libovolný sloupec, u grafu bodového to je klik na daný bod v grafu a u plošného to je klikem do plochy.

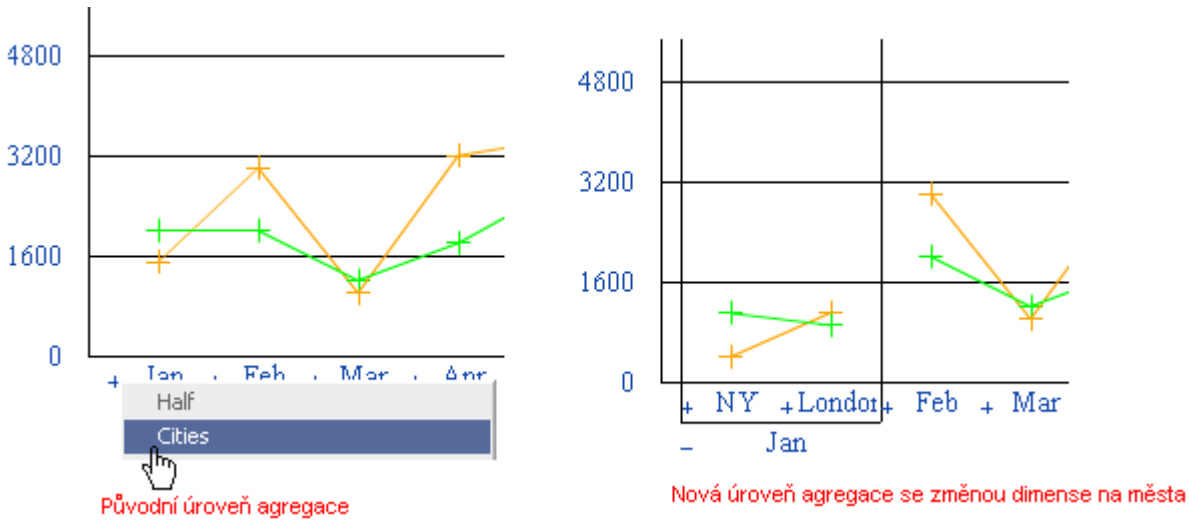
Tento způsob umožňuje jen procházet do nižších úrovní agregace, ale ne zpět. Stejně jako u předchozího způsobu potom může být implementován dvojím způsobem.

4.1.3 Přechod mezi úrovněmi pomocí nabídky volby kategorie

Toto je nejkomplexnější způsob průchodu úrovněmi a umožňuje nám definovat, jestli chceme v další úrovni stejnou kategorii nebo chceme zobrazit jinou dimensi. Princip je založený na předchozích dvou řešení a to tak, že klikneme-li pravým tlačítkem myši na „+“ nebo do plochy grafu, objeví se nabídka, ve které můžeme vybrat kategorii, kterou chceme procházet v další úrovni. Výsledné zobrazení může být stejně implementováno dvojím způsobem jako u předchozích způsobů.



Obrázek 4-4. Zobrazení přechodu mezi úrovněmi agregací pomocí kliku do oblasti grafu



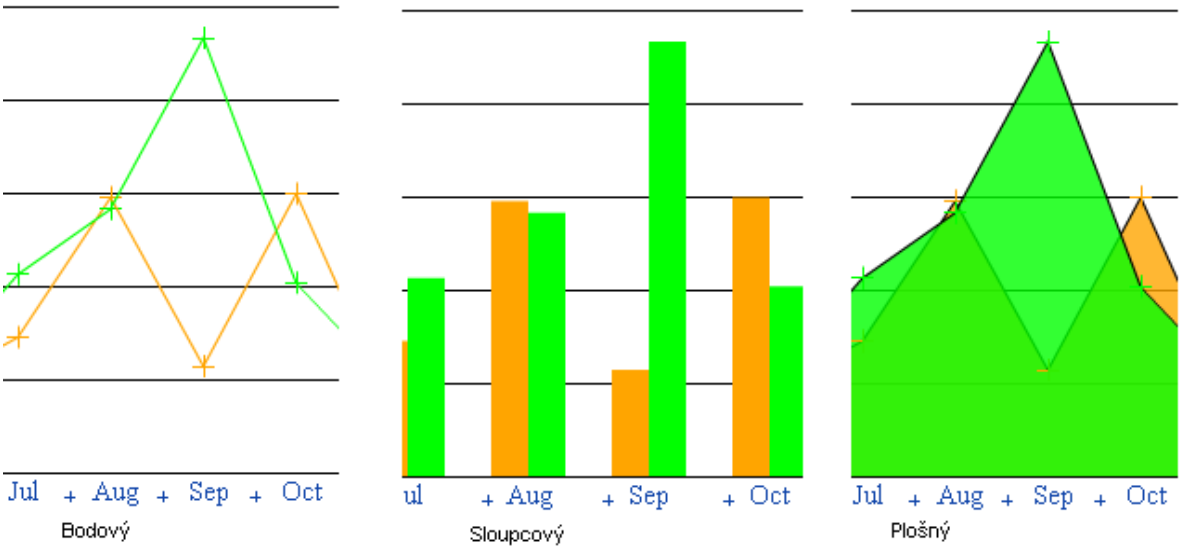
Obrázek 4-5. Zobrazení přechodu mezi úrovněmi agregací pomocí menu

4.1.4 Typy grafů a typy bodů

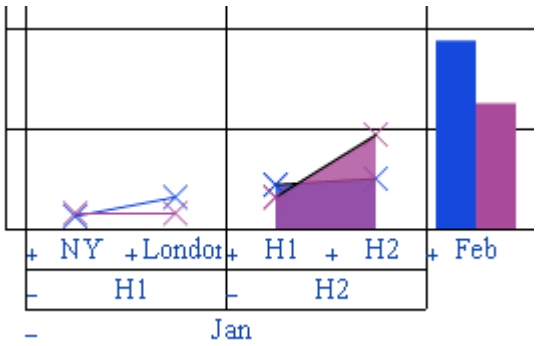
Pří návrhu jsem počítal s třemi druhy grafu a to plošný, bodový a sloupcový. (viz: Obrázek 4-6) Každý graf má na dané úrovni stejné vlastnosti, ale mezi úrovněmi už mít nemusí. V praxi to vypadá tak, že máme-li na první úrovni graf bodový, tak na úrovni druhé už to může být graf sloupcový. (viz: Obrázek 4-7)

Taktéž se může u grafu nastavit, jakým stylem se budou vykreslovat body v grafu plošném a bodovém. Nastavitelné styly jsou tři (viz: Obrázek 4-8)

- Bod – bod v ploše se vykreslí jako puntík
- Kříž – bod v ploše se vykreslí jako malý kříž
- Kříž45 – bod v grafu se vykreslí jako malý kříž natočený o 45 srupňů



Obrázek 4-6. Typy grafů



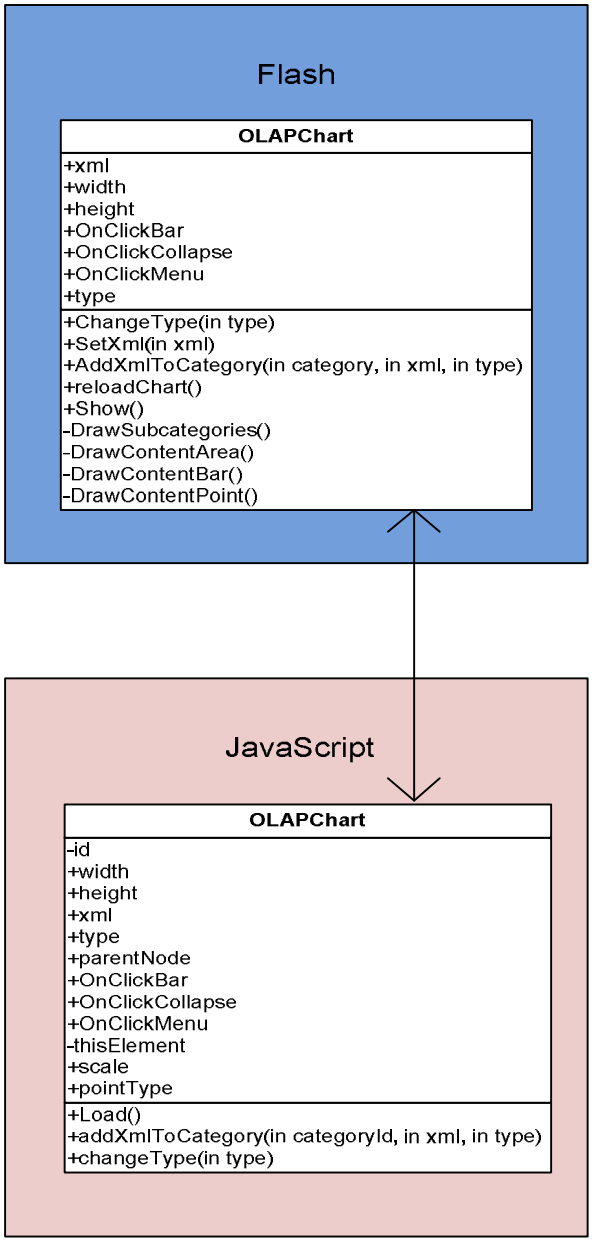
Obrázek 4-7. Všechny typy grafů v jednom grafu

Obrázek 4-8. Typy bodů

4.2 Návrh modelu tříd

Pro vývoj tohoto grafického podsystému jsem si vybral program Flash, kde pomocí ActionScriptu budu daný graf vykreslovat. Vývojář, který bude tuto komponentu chtít použít, jednoduše pošle mé komponentě na vstup XML, který bude obsahovat data grafu a ActionScript pak toto XML rozebere a zobrazí v požadovaném tvaru.

Jelikož se jedná o interaktivní komponentu, která by měla být snadno použitelná a to jak pro uživatele, tak i pro vývojáře, navrhl jsem i JavaScriptový objekt, který se stará o samotné zasazení Flashe do stránky a následnou komunikaci s Flashem. Vývojář poté jen ve stránce vytvoří daný objekt, řekne, kam ho chce umístit, a navolí vlastnosti a objekt se už sám o všechno postará. Vývojář může zadat i názvy svých JavaScriptovích funkcí ve stránce, které mají předepsaný počet a pořadí parametrů a ty jsou pak volány v případě, že byla vyvolána nějaká událost.



Obrázek 4-9. Návrh tříd grafického podsystému

Na (Obrázek 4-9) je vidět návrh tříd. Prakticky se jedná jen o třídy dvě, každá se jmenuje stejně, ale jsou oddělené prostředím. Třída v prostředí Flash obsahuje:

- XML – atribut představující XML s vstupními daty
- Width – šířka vykreslovací plochy
- Height – výška vykreslovací plochy
- OnClickBar – název JavaScriptové funkce, která má být zavolána, pokud se kliklo do plochy grafu.

- OnClickCollapse - název JavaScriptové funkce, která má být zavolána, pokud se kliklo na tlačítka „+“ nebo „-“.
- OnClickMenu - název JavaScriptové funkce, která má být zavolána, pokud se vybrala položka z menu.
- Type – typ grafu (point – bodový, bar – sloupcový, area - plošný)
- ChangeType – funkce, která změní za běhu typ grafu, může být volána z JavaScriptu
- SetXml – funkce která provede nastavení nového XML, může být volána z JavaScriptu
- AddXmlToCategory – funkce, která vloží za běhu do dané kategorie nové XML zadané uživatelem, může být volána z JavaScriptu
- ReloadChart – funkce pro znovu načtení grafu, může být volána z JavaScriptu
- Show – funkce pro zobrazení grafu
- DrawSubCategories – metoda, která provede vykreslení podkategorií
- DrawContentArea – metoda pro vykreslení plošného grafu
- DrawContentBar – metoda pro vykreslení sloupcového grafu
- DrawContentPoint – metoda pro vykreslení bodového grafu

Třída prostředí JavaScript obsahuje podobné metody a atributy až na některé odlišnosti.

- Id – je unikátní identifikace daného grafu, která se vytváří automaticky při vytvoření grafu
- ParentNode – určuje element, do kterého se má graf vložit, zadaný uživatelem
- ThisElement – obsahuje samotný objekt daného grafu ve stránce
- Scale – nastavení měřítka grafu

4.3 Schéma vstupního datového XML

Při návrhu vstupního XML jsem vycházel z XML, které používá firma Fusion Chart [6] u svých grafů. XML jsem si trochu upravil, přidal a odebral některé elementy. Postupně popíšu všechny elementy ve vstupním XML řetězci. U každého atributu a elementu také napíšu, zda se jedná o povinnou nebo nepovinou položku.

- **Chart** – hlavní element – povinný
  - Vlastnosti
    - Caption – název grafu – nepovinný
  - Podelementy
    - Datasets
    - Categories

- **Datasets** – element sdružující elementy dataset – povinný
  - Podelementy
    - Dataset

- **Dataset –** obsahuje nastavení pro jednotlivé řady dat – nepovinný
  - Vlastnosti
    - seriesName – popis řady – povinný
    - color – barva řady – nepovinný

- **Categories –** element sdružující kategorie – povinný
  - Vlastnosti
    - Type – udává, jaký typ grafu bude mít daná úroveň - nepovinný
  - Podelementy
    - nextCategoryType
    - category

- **Category** - element kategorie v daném bodu – nepovinný
  - Vlastnosti
    - Label – název kategorie – povinný
    - Id – id dané kategorie, musí být unikátní – povinný
  - Podelementy
    - Set
    - subcategories

- **nextCategoryTypes** – obsahuje výčet všech možných kategorií, kam se dá v dané úrovní zanořit - nepovinný
  - Podelementy
    - categoryType

- **categoryType** – typ kategorie do které se dá v dané úrovni zanořit – nepovinný
  - Vlastnosti
    - Label – název daného typu - povinný

- **Set** – obsahuje hodnotu pro danou řadu v dané kategorii – nepovinný
  - Vlastnosti
    - Value – daná hodnota - povinný

- **Subcategories** – Obsahuje kategorie, které jsou podkategoriemi kategorie, ve které je tento element umístěn – nepovinný
  - Vlastnosti
    - Type – udává typ grafu pro daný podgraf – nepovinný
  - Podelementy
    - Category
    - nextCategoryTypes

A zde je příklad takového XML.

```
<chart caption="Sales cell phone">
  <datasets>
    <dataset seriesName="HTC touch cruise" color="#1548dc" />
    <dataset seriesName="Iphone" color="#aa4a9a" />
  </datasets>
  <categories>
    <nextCategoryTypes>
      <categoryType label="Half months" />
      <categoryType label="Cities" />
    </nextCategoryTypes>
    <category label="Jan" id="1" >
      <set value="1500" />
      <set value="2000" />
      <subcategories>
        <category label='H1' id="13" >
          <set value='700' />
          <set value='500' />
          <subcategories  type='point'>
            <category label='NY' id="15" >
              <set value='200' />
              <set value='250' />
            </category>
            <category label='London' id="16" >
              <set value='500' />
              <set value='250' />
            </category>
          </subcategories>
        </category>
        <category label='H2' id="14" >
          <subcategories>
            <category label='NY' id="17" >
              <set value='200' />
              <set value='850' />
            </category>
            <category label='London' id="18" >
              <set value='600' />
              <set value='650' />
            </category>
          </subcategories>
          <set value='800' />
          <set value='1500' />
        </category>
      </subcategories>
```

```
    </category>
    <category label="Feb" id="2" >
      <set value="3000" />
      <set value="2000" />
    </category>
    <category label="Mar" id="3" >
      <set value="1000" />
      <set value="1200" />
    </category>
  </categories>
</chart>
```

Na tomto příkladu můžeme vidět, že jde o víceúrovňový graf, který se jmenuje „Sales cell phone“ a má dvě datové řady „HTC touch cruis“ a „Iphone“. Obě tyto řady mají určenou barvu. Na první úrovni je zobrazena kategorie času a to v úrovni měsíců. Tato úroveň umožňuje vybrat v nabídce ze dvou typů možných přechodů do další kategorie a to buďto „Half months“ nebo „Cities“. Kategorie „Jan“ má navíc již vnořené dvě podkategorie a ty mají další dvě podkategorie v sobě.

Pomocí tohoto XML se dá graf jednoduše vygenerovat v jakémkoliv stavu a stylu.

5 Popis implementace knihovny

Samotnou implementaci knihovny jsem rozdělil do dvou částí. V části první se zabývám vytvořením samotného vykreslovacího jádra v ActionScriptu a v části druhé potom propojení komunikace s uživatelem pomocí JavaScriptu. Samotný objekt Flashe je schopný fungovat samostatně, ale vývojář, který by ho chtěl použít, musí vědět, jaké funkce může u objektu používat a jaké parametry může nastavit.

5.1 Implementace vykreslovacího jádra v ActionScriptu

Při vytváření vykreslovacího jádra jsem postupoval od základních částí k těm složitějším. Popíšu postupně knihovny, které jsem použil pro vytvoření aplikace a poté popíšu některé zajímavé funkce, které tvoří podstatu celého systému.

5.1.1 AnimationPackage

Nejdříve jsem musel přijít na způsob, jak správně vykreslovat jednotlivé objekty pomocí ActionScriptu. Při pátrání po nejjednodušším způsobu jsem narazil na knihovnu napsanou pro ActionScript pod jménem AnimationPackage[10]. Tato knihovna funguje tak, že na začátku jejího použití se zavolá inicializace jádra a potom už stačí jen volat funkce pro vykreslení. Knihovna během inicializace vytvořila prázdný MovieClip a vše potom zakresluje do tohoto MovieClipu. Stará se například o hloubku daného obrazce, nebo umístění.

Knihovna umí potom s objekty vykreslenými pomocí ní pracovat a vytvářet animace. Každý obrazec se vytváří jako objekt, kterému se dají nastavit jeho vlastnosti a poté už jen říci, aby se vykreslil nebo svoje vykreslení animoval.

AnimationPackage existuje ve funkčních verzích pro ActionScript 1.0 a 2.0. Pro verzi 3.0 je zatím jen ve verzi beta.

5.1.2 XML2Object

Jelikož jsem potřeboval jednoduchým způsobem přistupovat k XML, které bylo zasíláno jako vstupní parametr, musel jsem najít nejlepší řešení. ActionsCript má v sobě již zabudovaný objekt, který umí zpracovávat XML, ale bohužel je procházení tímto objektem velmi obtížné a nemá implementován XPath. Proto jsem hledal alternativu, která by tento objekt nahrazovala, a našel jsem knihovnu XML2Object.

Na začátku se této knihovně předá již vytvořený objekt XML. Knihovna poté toto XML zpracuje a vytvoří strukturu objektů a polí objektů, ke kterým lze velmi rychle a bez obtíží přistupovat. Objekt potom vrátí na výstup.

Knihovnu jsem si upravil tak, abych si některé potřebné informace z XML zjistil už ve chvíli vytváření objektu a nemusel zbytečně tento objekt znovu procházet. Mezi takovéto hodnoty patří například maximální hodnota řad. Také jsem do této knihovny naimplementoval částečnou kontrolu na správnost vloženého XML.

5.1.3 Funkce pro definování správného měřítka.

Po získání maximální hodnoty z grafu je zapotřebí i tuto maximální hodnotu zpracovat a vytvořit adekvátní měřítko, pomocí něhož se pak bude celý graf vykreslovat. Proto jsem vytvořil funkci, která tuto problematiku řeší.

```
// funkce ktera vypocita podle nejvysi hodnoty vhodne meritko grafu
function CalculateMaxH(maxV) {
	var pomV:Array = new Array( 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0,
9.0);
	var maxH = 0, i = 0;
	// pokud je hodnota vetsi jak 1 tak hledame postupnym zvetsovanim
hledanou hodnotu
	if (maxV > 1.0) {
		i = 0;
		while (maxH < maxV)
		{
			if (i == 9)
			{
				i = 0;
				for (var j = 0; j < pomV.length; j++)
					pomV[j] = pomV[j] * 10;
			}
			maxH = pomV[i];
			i++;
		}
	}
	else // jinak provadime to same postupnym zmensovanim
	{
		i = 9; maxH = 1.0; var pomS = 1;
		while (pomS > maxV)
		{
			if (i == 0)
			{
				i = 9;
				for (var j = 0; j < pomV.length; j++)
					pomV[j] = pomV[j] / 10;
			}
			i--;
			maxH = pomS;
			pomS = pomV[i];
		}
	}
	return maxH;
}
```

Funkce má na vstupu maximální hodnotu získanou ze vstupních dat a na výstupu potom maximální hodnotu výšky grafu. Celý výpočet se provádí tak, že se na začátku inicializuje pole, které obsahuje čísla od 1 do 9. Je-li maximální hodnota větší než 1, tak se postupně prochází vytvořené pole a hledá se číslo, které je jako první větší než maximální hodnota. Pokud se takové číslo po projití celého pole nenajde, provede se vynásobení všech prvků pole číslem 10 a cyklus se spouští nanovo. Toto se opakuje, dokud není číslo nalezeno. Pokud se číslo najde, cyklus končí a toto číslo se posílá na výstup.

Pokud je ale maximální hodnota menší než 1, provede se podobný postup jako u předchozího popisu, ale místo aby se hledala hodnota větší než je maximální hodnota, tak se hledá hodnota první menší. Celé pole se pokaždé vydělí 10. Jakmile je hodnota nalezena, vrací se hodnota, která předcházela hodnotě nalezené.

Díky této funkci se definuje nejvyšší zobrazovaná hodnota v grafu, a všechny hodnoty se potom pomocí níže uvedeného vzorce (viz: Rovnice 5-1) přepočítávají.

Rovnice 5-1. Rovnice pro výpočet výšky bodu v grafu pro danou hodnotu.

$$h = value/(max\_value/h\_graf)$$

h – hledaná výška dané hodnoty v pixelech od základny grafu
value – hodnota míry
max_value – maximální hodnota celého grafu.
h_graf – velikost vykreslovací plochy grafu v pixelech

5.1.4 Vykreslovací funkce

Samotné vykreslení grafu potom spočívá na čtyřech funkcích. Tři z nich jsou určeny k vykreslení jednotlivých typů grafu a čtvrtá má za úkol obsluhovat případy, je-li nalezena podkategorie.

Na začátku se zjistí typ grafu, který se má vykreslit - buďto tak, že byl zadán jako vstupní parametr, nebo že je napsán jako hodnota v categories v XML. Hodnoty v XML mají vždy větší přednost než hodnoty obecné. Podle zjištěného typu se zavolá příslušná funkce pro vykreslování. Aby celý systém vykreslování věděl, kam má vykreslit další hodnoty na ose x, byla vytvořena globální proměnná xPos. Do této hodnoty se po každém vykreslení zapsala hodnota místa, kde se bude provádět další vykreslování. Díky tomu ví funkce podgrafu, kde bude vykreslovat svůj obsah.
Vykreslovací funkce procházejí XML a vykreslují dané hodnoty řad. Pokud je nalezen v kategorii element subcategories, je zavolána funkce čtvrtá, která rozhodne podle nového typu, jaká funkce se zavolá pro další vykreslování. Toto se provádí rekurzivně, dokud není dosaženo nejnižší podkategorie v daném stromu XML.

5.1.5 Interakce s okolním prostředím.

Každý interaktivní objekt ve vykresleném grafu, má přiřazený název. Tento název je složený z informací o daném objektu. Ve chvíli kdy je na tento objekt kliknuto, se z názvu zjistí, o jaký bod se jedná a pomoci ExternalInterface se provede volání JavaScriptové funkce, která byla na danou událost navázána.

Díky ExternalInterface se přichystají i funkce, které budou reagovat na volání z JavaScriptu.

5.2 JavaScriptový objekt

Jelikož je potřeba zajistit, aby se dal objekt jednoduše používat, byla k němu vytvořena i JavaScriptová knihovna. Tato knihovna obsahuje definici třídy OLAPChart. Třídě se nadefinuje, jak velký graf má být a kde má být ve stránce umístěn a předá se vstupní XML. Třída sama zabezpečí vložení Flashe do stránky ve správném formátu. Pomocí funkcí volaných nad tímto objektem se můžeme propojit i s ActionScriptem ve Flashi.

Pro vkládání Flash objektu do stránky využívám JavaScriptové knihovny SWFObject. Tato knihovna zajistí správné vložení do stránky a vrátí odkaz na objekt, který se dále používá pro komunikaci s Flashem.

6 Správný postup použití knihovny

Knihovna byla koncipována tak, aby se odstranilo zbytečné dotazování na server. Graf se umí podle akce uživatele interaktivně překreslovat bez potřeby znovu načítat stránku. Proto je nejlepší všechna data pro graf získávat přes AJAX a ty pak už jen posílat grafu samotnému.

Díky tomuto přístupu je možné rychle zobrazovat nové úrovně pohledu. Velkou výhodou této knihovny je, že dokáže zobrazovat více detailů vedle sebe. Například mám-li zobrazený graf prodejnosti aut za celý rok, tak v případě, kdy chceme porovnat detailně dva měsíce po dnech, můžeme si otevřít oba detaily vedle sebe v jednom grafu. Navíc díky možnosti libovolně měnit typ grafu podle úrovně, kterou zobrazujeme, je možné na úrovni nejvyšší mít data zobrazena v grafu sloupcovém a na úrovni nižší pak v grafu bodovém. Toto se dá s výhodou použít třeba i pro vizuální oddělení úrovní pohledu.

Správné použití knihovny tedy spočívá na tom, aby programátor vytvořil pozadí, které bude dodávat data potřebná pro graf a aby správně volil barvy a typy grafů v jednotlivých úrovních. Samozřejmě se dá s grafem pracovat, jako s grafy jednoduchými, ale to nebude využívat plnou sílu možností mnou vytvořené knihovny.

7 Závěr

Jelikož v dnešní době je již mnoho společností, které se zabývají vývojem grafů pro zobrazování OLAP dat, bylo velmi složité přijít s něčím novým. I přesto bylo zadání diplomové práce splněno.

Vytvořil jsem systém, do kterého jsem zaimplementoval všechny hlavní body, které jsem v rámci své práce navrhl. Avšak výsledný systém lze nadále do budoucna rozšiřovat. Mnou navržená zlepšení, které jsem implementoval v knihovně, jsou určena pro zjednodušení práce s grafem ze strany uživatele.

Knihovna neřeší problémy samotného zobrazení velkého množství dat. Proto by se jako pokračovaní mé práce mohlo brát vylepšení samotného zobrazení dat, třeba podle modelu *Table Lens* nebo *Fisheye table*. V takovém případě by byly jednotlivé úrovně pohledu od sebe vizuálně odděleny a v každé úrovni by se data zobrazovala podle těchto modelů ve vertikální nebo horizontální poloze podle směru grafu. Toto rozšíření může být zadáním nové diplomové práce.

Do knihovny se dají zapojit i další typy grafů jako je koláčový, burzovní nebo radarový. Dále by pak mohla umožňovat zobrazovat data ve vertikální i horizontální poloze.

Literatura

[1] Lacko, L.: Datové sklady, OLAP a dolování dat, Computer Press 2003, 488 stran, ISBN: 80-7226-969-0.

[2] Data Warehousing. URL: http://datamining.xf.cz/index.php, [10.12.2007].

[3] Dundas Chart for .NET - OLAP Services. URL: http://www.dundas.com/Products/Chart/NET/OLAp/index.aspx, [10.12.2007].

[4] Chart FX OLAP. URL: http://eu.softwarefx.com/EXTENSIONS/FEATURESOLAP.ASPX?USSfxRef=www.google.cz%2fsearch%3fhl%3dcs%26q%3dchart%2bolap%26btnG%3dHledat%26lr%3d, [10.12.2007].

[5] OpenI. URL: http://openi.sourceforge.net/, [10.12.2007].

[6] Fusion Chart. URL: http://www.fusioncharts.com, [2.1.2008].

[7] Ing. Ladislav Ruttkay: Vizualizácia dát, FIT VUTBR, 17.12.2007, 36 stran.

[8] http://msdn2.microsoft.com/en-us/library/ms717005.aspx, [3.5.2008].

[9] Maniatis, A., Vassiliadis, P., Spiros Skiadopoulos, Vassiliou, Y.: CPM: A Cube Presentation Model for OLAP, Springer Berlin / Heidelberg 2003, 10 stran, ISBN: 978-3-540-40807-9.

[10] Animation Package, http://www.alex-uhlmann.de/flash/animationpackage/index.htm, [3.5.2008].

[11] FishEye Table, http://iv.slis.indiana.edu/sw/fisheye.html#Details, [3.5.2008].

[12] Table lens demo, http://www.avizsoft.com/contents/table-lens-demo.htm, [3.5.2008].

[13] Flash Help, Martin Hozík, http://flash.jakpsatweb.cz/index.php?page=download, [3.5.2008].

[14] Lacko, L.: Business Intelligence v SQL Serveru 2005, Computer Press 2006, 391 stran, ISBN: 80-251-1110-5.

[15] Fotr, J.: Naprogramujte si vlastní hru v Macromedia Flash MX 2004, Computer Press 2005, 224 stran, ISBN: 80-251-0576-8.