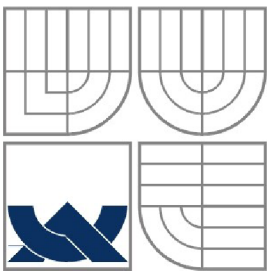
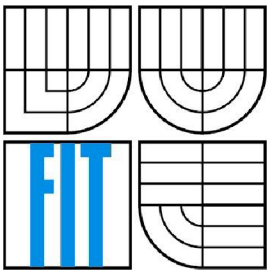


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

AJAX – AKTIVNÍ JAVASCRIPT KOMUNIKUJÍCÍ V XML

AJAX – ACTIVE JAVASCRIPT COMMUNICATING VIA XML

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Jana Firlová

VEDOUCÍ PRÁCE
SUPERVISOR

Prof. Ing. Tomáš Hruška, Csc.

BRNO 2007

Abstrakt

Práce pojednává o nástrojích používaných v současnosti při tvorbě dynamických webových aplikací a soustřeďuje se především na Ajax. Představuje komponenty této techniky, její princip, užití, výhody a nevýhody. Je zde popsán systém, který využívá funkcionalitu Ajaxu při realizování OLAP operací pomocí dynamické tabulky. Nechybí charakteristika těchto operací a úvod do problematiky zobrazení výsledků těchto operací ve dvourozměrném prostoru.

Klíčová slova

Ajax, DOM, XML, JavaScript, OLAP, dynamická tabulka

Abstract

I discuss in this work tools used to build dynamic web-based aplications, focusing on Ajax technology. I introduce the components of this technique, its principles, usage, advantages and disadvantages.I describe a system, which uses Ajax for helping with realization of OLAP operations through dynamic tables. There is also description of these operations included, as well as introduction to possibilities of visual representation of their results.

Keywords

Ajax, DOM, XML, JavaScript, OLAP, dynamic table.

Citace

Firlová Jana: *Ajax – aktivní JavaScript komunikující v XML*. Brno, 2008, diplomová práce, FIT VUT v Brně.

Ajax – aktivní JavaScript komunikující v XML

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně pod vedením prof. Ing. Tomáše Hrušky, Csc.
Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

.....
Jméno Příjmení
Datum

Poděkování

Děkuji panu Hruškovi za podporu a vedení pči vypracovávání DP a Lubomírovi Hurtečákovi za významnou pomoc při ladění programové části DP.

Obsah

1 Úvod

Webové aplikace získávají již dlouhou dobu na oblibě a pronikají do oblastí dříve vyhrazených pouze „klasickým“ desktopovým aplikacím. Pronikly do komerční sféry, kde přímo pomáhají propagovat a prodávat zboží, i do informační sféry, kde se můžeme setkat s opravdu rozsáhlými a sofistikovanými informačními systémy, realizovanými jako webové aplikace. Své místo mají i v zábavním průmyslu a juplatní se jako prezentace prakticky čehokoliv nebo kohokoliv.

Tyto aplikace mají oproti desktopovým výhodu v tom, že pro jejich zavedení u zájemce odpadá potřeba instalace nového softwaru. Jejich distribuce a aktualizace je snadná a levná, aplikace jsou přístupnější. Pokud je webová aplikace správně napsaná a funguje ve všech hlavních moderních webových prohlížečích, je velice snadno rozšiřitelná. Je vstřícnější ke svému uživateli, který spíše vyzkouší novou aplikaci bez nutnosti zdlouhavé instalace a případně stahování řady balíčků.

Oproti klasickým alikacím však mají webové i jednu zásadní nevýhodu. Je to citelná ztráta interaktivity, kterou poskytuje uživatelské rozhraní, a nutnost neustálého obnovování celých stránek. Ajax se nabízí jako technika, která potlačuje tuto nevýhodu.

Tento negativní rys webových aplikací je zakořeněn v samotném zrodu Internetu, v jeho počáteční koncepci. Pro pochopení toho, kam moderní webové technologie směřují a proč, je nutno znát, odkud vycházejí. Úvodní podkapitola stavu problematiky se tedy věnuje historii Internetu a hlavně webových technologií. Zmiňuje a charakterizuje hlavní technologie, které se ve webových aplikacích používaly a často stále používají.

Následující kapitoly se věnují technologiím, které přímo souvisí s Ajaxem.

Druhá podkapitola stavu problematiky se věnuje modelu DOM, který významně ovlivňuje chápání struktur dokumentů používaných na webu a práci s nimi, následuje kapitola o skriptovacím jazyce JavaScript, dnes nejrozšířenější technologii na straně klienta a srdci Ajaxu.

Čtvrtá podkapitola popisuje formát XML, který se stal v pravém slova smyslu fenoménem zasahujícím do oblastí nejrůznějších aplikací, obzvláště pak webových, pro které byl primárně určen.

Pátá podkapitola zužitkovává předchozí informace a představuje zblízka techniku Ajaxu.

Součástí řešení mé diplomové práce je využití Ajaxu tak, že pomáhá realizovat OLAP operace a zobrazuje na stránce výsledná data. Poslední podkapitola, která uvádí do problematiky této práce, je proto věnována stručné charakteristice OLAP systémů, základním operacím, které v nich probíhají, a problému vizualizace výsledných dat.

2 Stav problematiky

2.1 Historie a vývoj webových technologií

Na konci 70. let 20. století vyústily dlouholeté výzkumy a experimenty na poli propojení počítačů do sítí v první zárodek Internetu. Jednalo se o projekt ARPANET, který probíhal v rámci počítačového výzkumu úřadu DARPA. V 70. letech se ARPANET rozvíjel oproti tomu, co mělo přijít, poměrně málo. Byl ale vyvinut TCP, zavedený do ARPANETu v roce 1980, kdy se událo v této síti více podstatných změn. Zvětšující se počet uživatelů dal vzniknout systému DNS. ARPANET přešel z protokolu NCP na rodinu protokolů TCP/IP, dosud užívaných armádou. V průběhu 80. let se objevily další sítě. Za stvoření Internetu bývá považován vznik sítě NSFNet, která byla roku 1985 otevřena pro komerční sféru. Samotný pojem Internet se objevil přitom až o dva roky později.

Sítě vznikaly pod křídly univerzit a státních institucí a přirozeně byly využívány převážně pro výměnu statických dokumentů mezi vědci a institucemi. V roce 1989 oživil Tom Berners-Lee myšlenku hypertextu, tedy textových dokumentů vzájemně provázaných odkazy. Dále vytvořil podmnožinu značkovacího jazyka SGML zvanou HTML, jednoduchý protokol HTTP a první webový prohlížeč zvaný WorldWideWeb.

Tyto prostředky (HTML spolu s HTTP) plně postačovaly k realizaci funkce Internetu - k výměně dokumentů a zpřístupnění základních informací (termíny, kontakty,..). Webové stránky byly vlastně jen kopií běžného textu. Se stoupajícím počtem uživatelů, se šířícími se osobními počítači a s uvolněním Internetu v roce 1993 pro komerční sféru se zvedly nároky na webové stránky a objevily se požadavky na zvýšení jejich dynamičnosti.

Brzy se také objevily prostředky realizující tyto požadavky. Objevily se jazyky a nástroje pracující jak na straně klienta, tak na straně serveru. Spolu s nimi se objevily příslušné normy, modely a rozhraní (někdy norma předcházela vznik takového prostředku, jindy následovala až po něm). Silné kombinace pro vytváření dynamických webových aplikací pak vznikají právě spojováním různých těchto prostředků.

Technologie a jazyky na straně klienta umožňují webovému prohlížeči nějak ozvláštnit, zpříjemnit či rozšířit práci s dokumenty. Obvykle jsou rozšířením HTML. Jsou to například Java ve formě appletů, JavaScript, Flash, DHTML jako kombinace HTML, CSS, JavaScriptu a DOM.

Technologie na straně serveru jsou pak ty, které umožňují mít na serveru uloženou logiku pro vytváření dokumentů za běhu. Jsou to například CGI skripty, Java servlety a JSP, PHP, ASP či dnes spíše ASP.NET a další.

Mezitím se objevují DOM, model pro zpřístupnění struktury stránky a nový, univerzální značkovací jazyk XML.

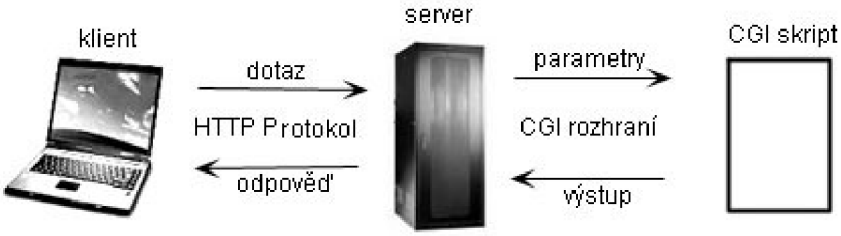
Modelu DOM, jazyku XML a JavaScript, které přímo souvisí s technikou Ajax, budou věnovány samostatné kapitoly. Ostatní prostředky jsou přehledově uvedeny dále [1, 2].

2.1.1 CGI skripty

Jako první řešení zvýšení dynamičnosti webových stránek se objevila specifikace CGI (Common Gateway Interface). Jedná se o rozhraní mezi webovým serverem a programem, o způsob jejich komunikace při předávání parametrů a výsledků.

Tato norma umožňuje vznik programů v nejrůznějších programovacích nebo skriptovacích jazycích, které mohou dynamicky generovat dokumenty přenositelné přes protokol HTTP (nejčastěji pochopitelně HTML dokumenty). Tyto samostatné, externí programy se nazývají CGI skripty.

Když klient vyšle serveru CGI požadavek a server jej rozpozná (a povolí), založí server CGI proces, který má na starost komunikaci mezi serverem a CGI skriptem. Skriptu jsou předány parametry a proměnné prostředí, ten na jejich základě vygeneruje dokument a předá jej serveru. Server pak dokument pošle klientovi (viz obr. 2.1).



obrázek 2.1 - obsloužení požadavku CGI skriptem

CGI skripty jsou vhodným a dodnes používaným nástrojem pro interakci s uživateli. Je nutné však dodržovat určitá bezpečnostní opatření, protože mohou představovat jisté riziko. Jedná se konec konců o spouštění programů na serveru nahraných sem a používaných různými uživateli s různými úmysly. Je nutné například kontrolovat vstup uživatele (speciální znaky) , jména a cesty. Tyto programy bývají uloženy ve zvláštním adresáři, kam nemívají běžní uživatelé právo umisťovat své soubory [1, 6, 8].

2.1.2 Applety

Vznikly v roce 1995 spolu s první verzí jazyka Java. Tehdy dominantní prohlížeč Netscape Navigator nabízel podporu pro Javu a s tím, že Java byla ke stažení na Internetu, se tento jazyk dočkal rychlého rozšíření.

Díky rozšířenosti Netscape Navigatoru se staly rozšířené také applety psané v Javě - menší aplikace, používané k rozšíření funkčnosti prohlížeče, uložené či načtené u klienta a pracující na jeho straně. Typicky nějak zvyšují komfort webové aplikace nebo ji vizuálně zatraktivňují.

Applety mají výrazně omezené možnosti, zejména kvůli bezpečnosti. Typicky nesmějí pracovat se souborovým systémem, spouštět programy, načítat knihovny. Dále nesmí navazovat síťové spojení jinam než na domovský server. Bezpečnost je zajišťována také spouštěním appletů s použitím bezpečnostního modelu *sandbox*.

Java applet je spouštěn na virtuálním stroji JVM, obsaženém v prohlížeči (pokud prohlížeč Javu podporuje), který zajišťuje spustitelnost na různých platformách. Osudnou slabinou těchto appletů jsou však rozdílné verze Javy, které mohou mít uživatelé nainstalovány. Vývojáři se musí pro správný běh svého appletu ubezpečit, že klient má tu správnou verzi Javy. Navíc některé virtuální stroje v prohlížečích nebyly právě ideální - jmenovitě virtuální stroj od Microsoftu nikdy nepodporoval vyšší verzi Javy než 1.1. To se stalo překážkou masivnějšímu rozšíření této technologie. Některé uživatele odradil od používání appletů také výskyt špatně naprogramovaných appletů, které způsobovaly na klientských počítačích problémy. Applety jsou navíc poměrně náročné na zavedení a spuštění a při použití pro jednoduché aplikace mohou být příliš těžkopádné [1].

2.1.3 Servlety, PHP, ASP a další

Java servlety byly uvedeny ve stejném roce jako Java applety. Jak již napovídá název, servlet se vykonává na straně serveru, takže o správnou verzi Javy je třeba se starat právě jen tam. Navíc není potřeba přenášet klientovi kvanta kódu, jak se u některých složitějších appletů může stát. Servlety s kompletní knihovnou pro obsluhu HTTP se staly silnou konkurencí CGI skriptů.

Jedinou nevýhodou těchto servletů byla nešikovná struktura kódu, pokud měl servlet generovat delší textový výstup (například stránku HTML). Servlet pak tvořily z větší části nepřehledné a špatně udržovatelné příkazy pro výpis textových řetězců. Příklad takového servletu ukazuje výpis 2.1:

```
response.setContentType("text/html;charset=UTF-8");
PrintWriter vystup = response.getWriter();
vystup.println("<html>");
vystup.println("<head>");
vystup.println("<title>titulek stranky</title>");
vystup.println("</head>");
vystup.println("<body>");
vystup.println("<h1>nadpis</h1>");
vystup.println("<p>text</p>");
vystup.println("</body>");
```

výpis 2.1 - generování HTML struktury servletem

Tento nedostatek odstraňuje technologie JSP (Java Server Pages), která umožňuje lépe oddělit zpracování dat od vzhledu výsledného dokumentu. Dokument JSP pak může vypadat v případě generování HTML stránky stejně jako ta HTML stránka, jen obsahuje příkazy volající logiku, zodpovědnou za dynamiku stránek, uloženou v jiném souboru. JSP kompilátor je pak schopen takovýto soubor převést na klasický servlet v jazyce Java (který je dále kompilován Java kompilátorem) nebo přímo na bajtkód servletu. .

Solidním konkurentem JSP je technologie ASP (Active Server Pages) od Microsoftu. Jde v ní také o provádění skriptů na serveru, ASP je programový stroj na straně serveru, který skripty přijímá a zpracovává. Takovýto skript může být například VBScript nebo JScript. Stejně jako u JSP se uplatňuje princip oddělení zpracování dat a výsledného vzhledu vygenerovaného dokumentu.

Softwarových řešení na straně serveru je dále celá řada. Rozšířeným je například jazyk PHP, který je na webovém serveru zpracováván buď pomocí nainstalovaného PHP modulu nebo CGI skriptu. PHP dokument je vlastně také velmi podobný JSP dokumentu - běžný text s vyznačenými místy, kde se nachází funkce daného jazyka. Konkurují mu dále například Perl, Ruby nebo Python [1, 10].

2.1.4 Flash

Tato technologie se zrodila v roce 1996, když společnost Macromedia koupila společnost FutureWave, spolu s jejím produktem FutureSplash Animator, který pak byl vydán jako Flash 1.0. Dnes technologii Flash vlastní společnost Adobe, která koupila v roce 2005 Macromedia.

Jedná se o klientskou technologii, pracující s vektorovou grafikou a s jazykem ActionScript.

Flash aplikace vyžadují software (flash plug-in v prohlížeči) na straně klienta, který je v současnosti dodáván spolu s většinou prohlížečů a je hojně rozšířen. Flash umožňuje vytvářet velice atraktivní a vysoce dynamické aplikace, je vhodný pro kombinaci obrázků, animací, videí a zvuků. Díky vektorové povaze jsou Flash aplikace při rozumném použití poměrně malé a tedy vhodné pro přenos. Mohou obsahovat klasické HTML prvky zajišťující interaktivitu s uživatelem - tlačítka, vstupní pole, výběry a podobně, a svým vzhledem a chováním se dokáží dostat na úroveň desktopových aplikací.

Šíření Flashe zpomaluje jeho komerční povaha - sada nástrojů pro vývoj Flash aplikace je poměrně drahá. Dále, i když je plug-in pro přehrávání Flash souborů poměrně dobře rozšířen, nemusí mít všichni uživatelé jeho poslední verzi (která bývá často vyžadována) a mohou používat nástroje, které Flash částečně nebo úplně blokují. Díky malé velikosti se totiž Flash začal hojně užívat pro klasické reklamní bannery a různé, někdy vizuálně velmi agresivní propagační animace na stránkách, které se uživatelé často snaží filtrovat. Flash aplikace také představují velkou zátěž na procesor.

Flash zůstává velmi populárním hlavně na poli dynamických prezentací (ne jen webových - Flash je dostupný i ve formátu spustitelném bez prohlížeče), menších jednodušších her a různých animací (reklamních a zábavných) [1, 11].

2.2 DOM

DOM (Document Object Model) je sada specifikací konsorcia W3C. Nejedná se o žádný jazyk, za který bývá někdy mylně považován, ale o definované rozhraní. Jeho vznik je spjat se skriptovacím jazykem JavaScript, resp. se snahou sjednotit přístup tohoto jazyka k prvkům dokumentu, což často implementovaly různé prohlížeče různě - podle svého vlastního modelu dokumentu. Brzy se však JavaScript a DOM oddělily ve dva samostatné koncepty. Vztah mezi JavaScriptem a DOM je takový, že DOM definuje rozhraní pro přístup k dokumentu a JavaScript jej implementuje a používá.

Toto je definice DOM uveřejněná na stránkách W3C:

„Document Object Model je na platformě a jazyce nezávislé rozhraní, které umožňuje programům a skriptům dynamicky přistupovat a aktualizovat obsah, strukturu a styl dokumentů. Dokument může být dále zpracován a výsledek tohoto zpracování může být zahrnut zpět do prezentované stránky.“

JavaScript je tedy jen jedním z *mnoha* jazyků, které využívají *jedno* rozhraní DOM [4].

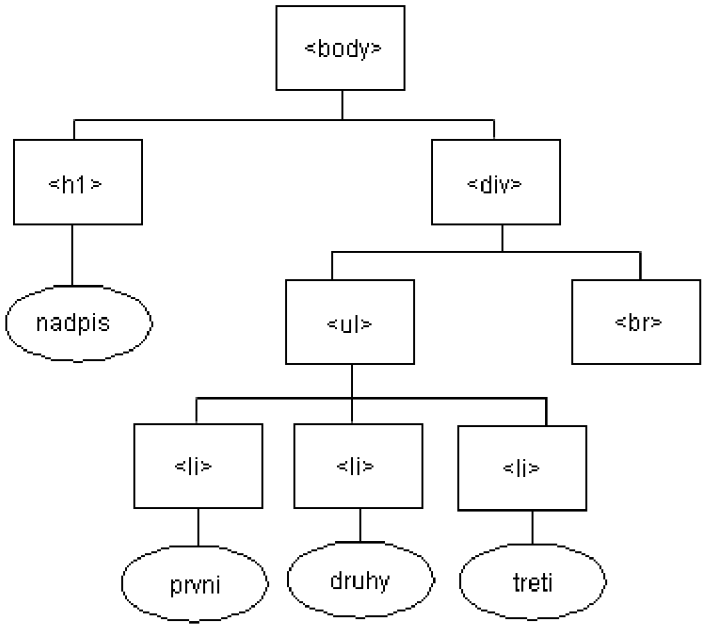
2.2.1 Charakteristika DOM

DOM poskytuje standardní množinu objektů, které charakterizují HTML a XML dokumenty. Určuje chování těchto objektů, jak jsou objekty kombinovány a jaké mají mezi sebou vztahy, a rozhraní, přes které jsou zpřístupňovány a používány. Téměř cokoliv v HTML a XML může být přes toto aplikační rozhraní zpřístupněno, změněno, smazáno nebo přidáno.

Logická struktura DOM dokumentu je podobná stromu. DOM však nikde nespecifikuje jak má být tato struktura implementována, ve skutečnosti tedy tato struktura nemusí být „pravým“, implementovaným stromem. Obrázek 2.2 ukazuje, jak lze znázornit strukturu DOM vytvořenou podle výpisu kódu 2.2:

```
<body>
  <h1>nadpis</h1>
    <div>
    <ul>
      <li>prvni</li>
      <li>druhy</li>
      <li>treti</li>
    </ul>
    <br />
  </div>
</body>
```

výpis 2.2 - jednoduchý HTML kód



obrázek 2.2 - jednoduchý DOM

Uzly tohoto strukturálního modelu nereprezentují datovou strukturu, ale objekty s vlastním chováním a identitou. Pokud dvě aplikace používají různé implementace DOM a každá vytváří reprezentaci stejného dokumentu, musí být výsledný model těchto reprezentací stejný. Tato vlastnost strukturálního modelu se nazývá *strukturální izomorfismus*.

Stromová podoba DOM vyžaduje pro práci s dokumentem načtení celého dokumentu. Proto je tento model vhodný zejména tam, kde se k datům přistupuje v nepředvídatelném pořadí a opakovaně. V případě, že aplikace požaduje pouze sekvenční průchod dokumentem nebo jen jedno čtení/zápis do dokumentu, může být použití DOM zbytečně náročné. Na tomto poli má DOM konkurenta v podobě modelu SAX, používaného pro sekvenční průchody XML [3, 4].

2.2.2 Úrovně DOM

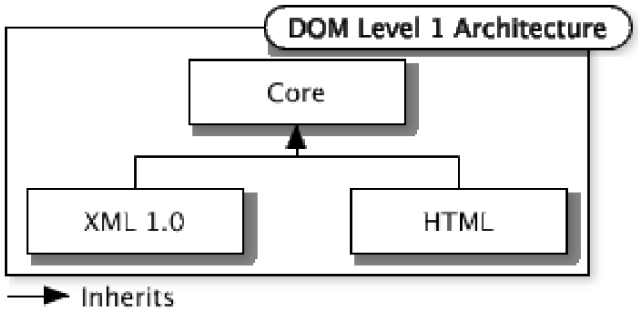
Specifikace DOM jsou rozčleněny do více modulů, které jsou obsaženy ve třech úrovních (DOM Levels), které vznikly postupně po sobě. Základní částí každé úrovně je modul *Core*, na němž pak závisí další moduly, které nejsou povinné. Už od první úrovně je definováno v *Core* rozhraní , pomocí něhož lze jednoduchým dotazem zjistit, zda je daný modul implementován (*DOMImplementation*).

2.2.2.1 DOM Level 1

V jádře této úrovně jsou popsány objekty, které vytváří samotný základ stromové struktury DOM (*Node, Document, Attr,...*). Nad jádrem jsou definovány vyšší úrovně programového rozhraní pro dokumenty HTML a XML (pro HTML například rozhraní pro odkazy, tělo, hlavičku, styly, metatagy,

formulářové prvky, rámce atd.). I když obsahuje ze všech tří úrovní modulů nejméně, jeho význam je zásadní. Jádro se dočkalo u pozdějších úrovní dalších rozšíření.

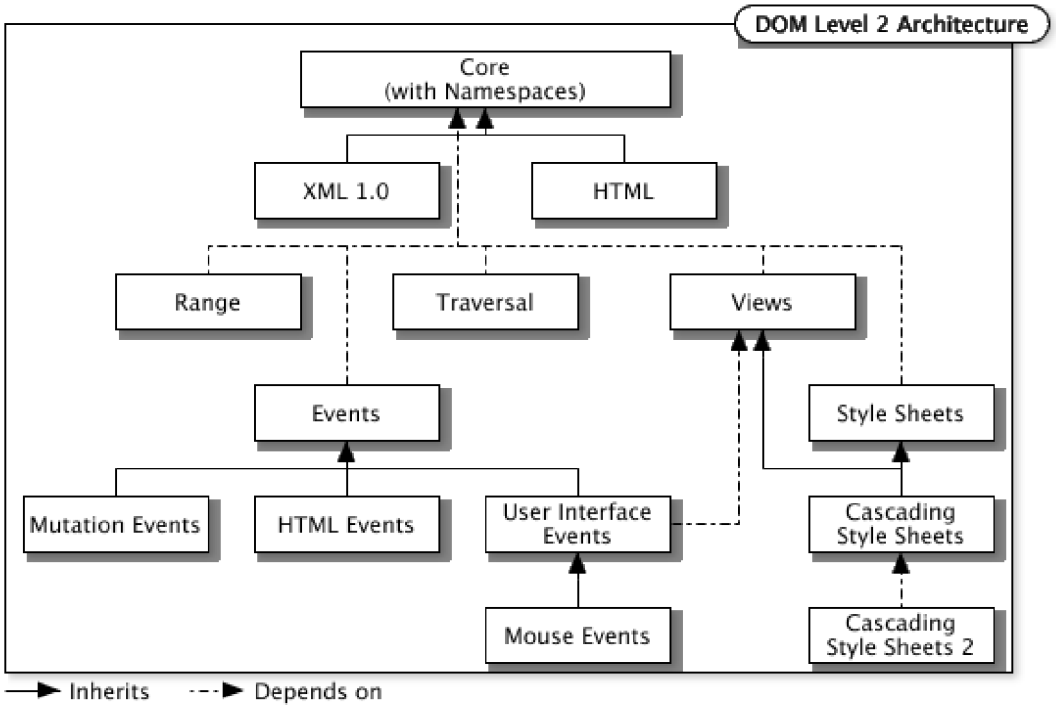
Přílohy DOM Level 1 obsahují kompletní rozhraní implementované v jazycích Java a ECMAScript (JavaScript podle specifikace asociace ECMA) [4].



obrázek 2.3 - moduly DOM Level 1 [7]

2.2.2.2 DOM Level 2

Obrázek 2.4 mapuje strukturu modulů a jejich částí druhé úrovně DOM:



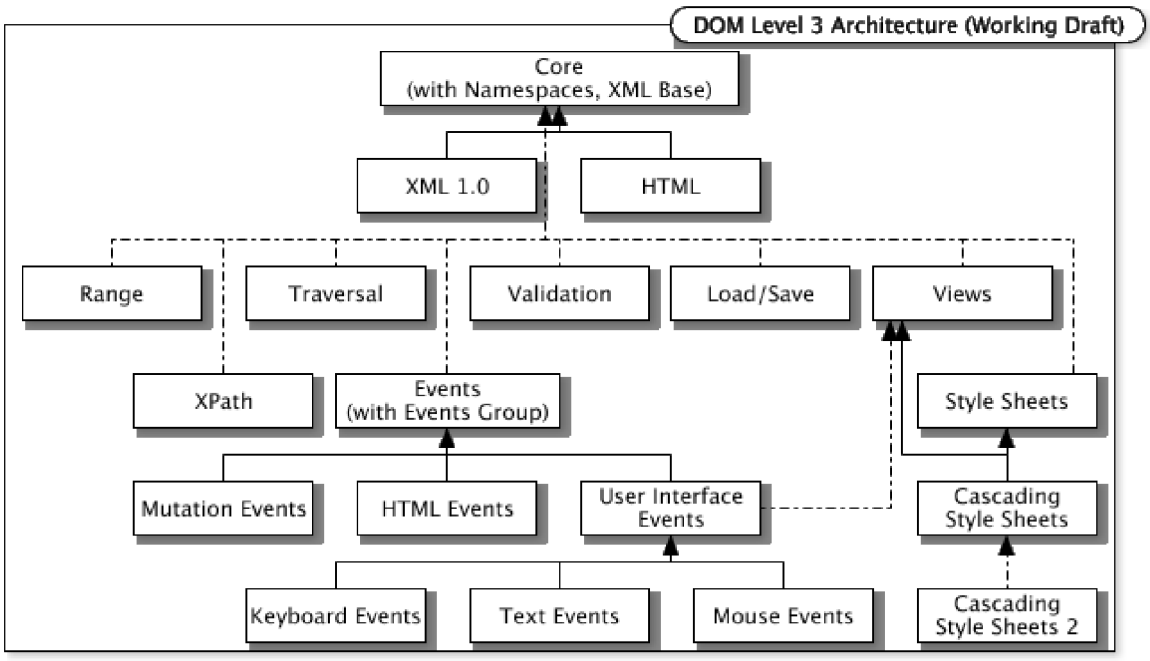
obrázek 2.4 - moduly DOM Level 2 [7]

DOM Level 2 navazuje na předchozí úroveň. Jádro je obohacenou verzí z první úrovně, nově definuje například objekt DOM dokument a přináší podporu jmenných prostorů XML. Dalšími volitelnými moduly jsou [4]:

- *Events* s rozhraními pro události - definuje události, jak vznikají, jejich zpracování, atributy - umožňuje práci s nimi. Neobsahuje zatím ještě události pro klávesnici.
- *Views* umožňující dynamicky zpřístupnit a měnit obsah reprezentace dokumentu.
- *Style* zpřístupňuje konstrukce jazyka CSS (vytváření, čtení, změna, odstranění) a umožňuje kontrolu nad vkládáním jednotlivých CSS stylů.
- *Traversal and Range: Traversal* poskytuje rozhraní pro selektivní procházení skrz strukturu DOM, *Range* umožňuje definovat vzdálenosti v dokumentu (mezi elementy) a manipulovat pak s takto vymezenými úseky.
- *HTML* které obsahuje rozhraní pro HTML 4.01 a XML 1.0 a není zpětně kompatibilní s HTML modulem předchozí úrovně.

2.2.2.3 DOM Level 3

Tato úroveň je zatím zčásti pořád ve vývoji. Některé části mají již status hotového standardu (*Recommendation*), jiné mají formu jen zprávy (*Working Group Notes*). Jádro staví opět na Core předchozí úrovně, doplňuje jej zejména o funkce pro práci s XML. Obrázek 2.5 zobrazuje strukturu modulů a jejich částí třetí úrovně DOM:



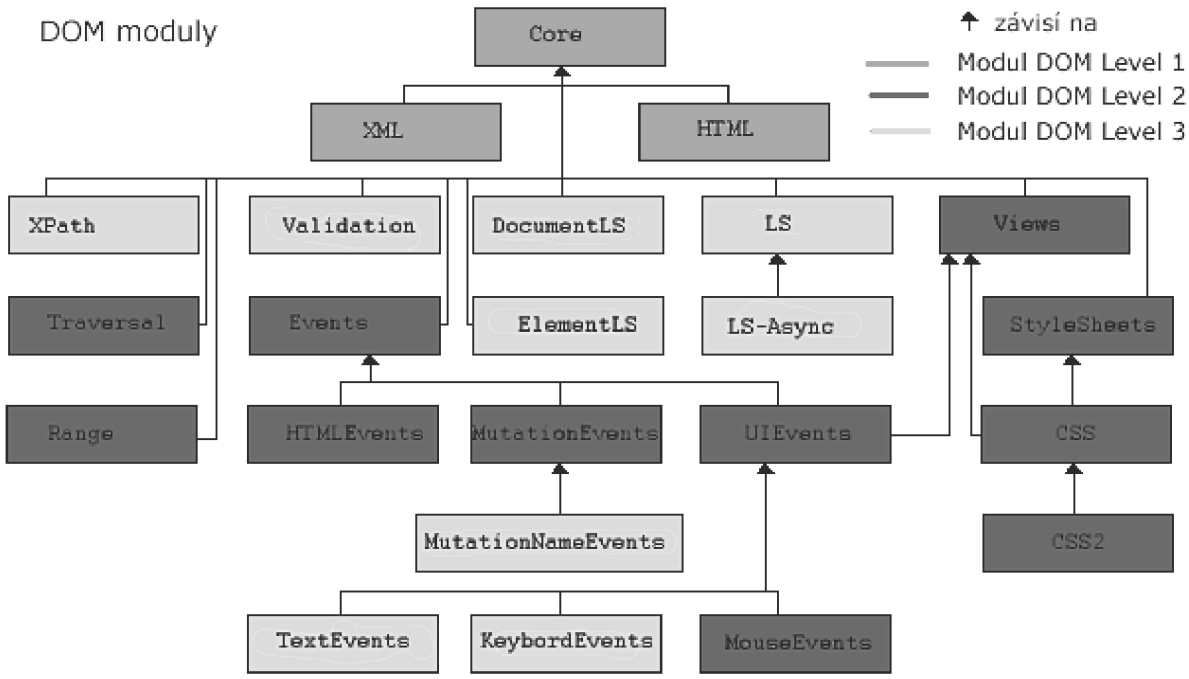
obrázek 2.5 - moduly DOM Level 3 [7]

Tato úroveň je také bohatá na zcela nové moduly, které přináší nové možnosti funkčnosti [4]:

- *Load and Save* (*Recommendation*) definuje rozhraní pro dynamické načítání obsahu XML do DOM dokumentu a naopak pro serializaci DOM dokumentu do XML. Také zahrnuje možnosti filtrování obsahu během převodu oběma směry mezi XML a DOM dokumentem.

- *Validation* (*Recommendation*) obsahuje rozhraní pro získání informací o gramatice dokumentu. Tento modul by měl umožnit programům a skriptům dynamicky měnit obsah a strukturu dokumentu zároveň s kontrolou, zda výsledek bude validní dokument. Měl by tedy umožnit odpovědi na dotazy typu „pokud sem něco vložím, bude dokument stále validní?“ nebo „Co všechno mohu zde smazat, aby dokument zůstal validní?“
- *XPath* (*Working Group Notes*) definuje, jak zpřístupnit stromovou strukturu DOM pomocí XPath 1.0.
- *Events* (*Working Group Notes*) staví na *Events* Level 2, doplňuje je o nové události, především o události klávesnice.
- *Views and Formatting* (*Working Group Notes*) vychází s modulu Views Level 2 a je zatím v hodně ranném stadiu vývoje.

Obrázek 2.6 dává přehled o tom, ve které úrovni DOM se poprvé objevily které moduly, a jak na sobě tyto moduly závisí:



Obrázek 2.6 - DOM Moduly [9]

2.3 JavaScript

2.3.1 Historie vzniku

Tento skriptovací jazyk byl vytvořen v roce 1995 společností Netscape. Měl sloužit jako jazyk pro webové designéry a programátory, kteří neovládali Javu, k vytváření appletů. Jeho autorem je Brendan Eich, který dostal za úkol tento nový jazyk navrhnout a implementovat. JavaScript měl sloužit jako doplněk h HTML a Java appletům. Brzy po svém vzniku se však dal na svou vlastní cestu užití a místo kontrolování appletů začal být využíván k manipulaci s obrázky a s prvky dokumentů.

I když je JavaScript mocným nástrojem pro vytváření dynamických webových aplikací, trpěl od svého vzniku mnoha problémy. Prvním z nich byla absence vývojových a ladících nástrojů a užitečných chybových hlášení. Druhým byly rozdílné modely dokumentů, které používaly různé prohlížeče. JavaScript je právě silně závislý na tomto svém modelovém prostřed a s nekompatibilními modely mezi prohlížeči byly nekompatibilní i skripty psané pro různé prohlížeče. Psaní přenositelných skriptů bylo obtížné a vyžadovalo spoustu kódu navíc pro ošetření rozdílů mezi prohlížeči. Dodnes ještě tato věc webové vývojáře občas trápí. Nutno podotknout, že to ovšem není způsobeno nedokonalostí JavaScriptu, ale často nedokonalostí prohlížeče v implementaci dnešního standardního modelu DOM. O co horší musela být situace v době, kdy žádný DOM ještě neexistoval.

JavaScript, jazyk, který je snadný k užívání a nepotřebuje žádné zvláštní znalosti, který neměl tehdy pořádné podpůrné vývojové nástroje a který šel ladit vlastně jen v kontextu daného prohlížeče, byl mnohými odsunut stranou jako „hračičkovský“ jazyk bez většího potenciálu pro začátečníky.

Postupného zlepšení situace se JavaScript dočkal s nástupem prohlížečů implementujících DOM, kdy umožnil zkombinovat sílu tohoto modelu s rysy, které se již dříve osvědčily u jiných jazyků a které jsou mu vlastní (volně typované proměnné, regulární výrazy, možnosti objektové orientace, dobře vyvinuté knihovny pro práci s řetězci, časy a matematickými výrazy,...). JavaScript se i dočkal v roce 1997 ratifikace své specifikace s názvem ECMAScript společností ECMA a o něco i později organizací ISO. Objevily se dodatečně i prostředí pro vývoj a ladění JavaScriptu.

V současnosti JavaScript dominuje mezi technologiemi užívanými webovými aplikacemi na straně klienta [1, 5].

2.3.2 Charakteristika a užití

Jedná se o samostatný skriptovací jazyk, který je nicméně užíván hlavně při vývoji webových aplikací. Jeho obrovskou výhodou je, že je podporován dnes na dobré úrovni všemi nejpoužívanějšími prohlížeči a uživatel nemusí pro funkčnost JavaScriptu stahovat žádný další podpůrný software, žádné nové plug-iny, žádné aktualizace.

Jedná se dnes o opravdu multiplatformní, objektově orientovaný jazyk. Syntaxí je podobný například jazyku Java nebo C. Není to kompilovaný jazyk, bývá interpretován prohlížeči spolu s HTML kódem stránky, do kterého bývá přímo vkládán - spouští se tedy až po stažení na straně klienta. Z toho vyplývají bezpečnostní omezení pro JavaScript, v první řadě to, že nesmí na straně klienta pracovat se soubory. Kompenzuje to tím, že umí a může pracovat s cookies.

Nejběžnější použití JavaScriptu bývá tvorba dynamických dokumentů - JavaScript využívá obrovského množství možností, které mu nabízí DOM pro dynamický přístup k dokumentu a k jeho částem (objektům), které může různě upravovat, dále má díky modelu DOM možnost reagovat na velkou škálu událostí. Vznikají tak různé animace a prvky vizuálně oživující web, mechanismy pro ověřování formulářových dat bez nutnosti jejich odeslání na server, prvky umožňující počítání a práci s časem, dynamická menu a nápovědy a různé jiné doplňky, které jen dokáže člověk vymyslet.

K těmto „klasickým“ formám využití se v posledních letech přidává jeho novější, velice populární a „vážnější“ úloha - role ústředního prvku techniky Ajax (více viz kapitola Ajax).

Webový programátor však musí mít na paměti, že uživatel, i když má prohlížeč dobře podporující JavaScript, může mít JavaScript ve svém prohlížeči zakázaný, nebo ho dokonce může u sebe lokálně měnit k obrazu svému. Proto by měl zvážit použití jakési „záchranné sítě“, pojistného kódu pro případ, že JavaScript nebude vykonáván (nebo vykonáván jinak), který zajistí alespoň základní funkcionalitu a případně i bezpečnost [1, 2]. Typickým příkladem je dvojí ověřování platnosti formulářových dat – nejdříve na straně klienta JavaScriptem, poté znovu při zpracování vstupů na serveru. Pokud uživatel JavaScript povolený má, dojde k ověření ihned po vyplnění príslušných polí, aniž by se cokoli kamkoli odesílalo – to šetří čas uživatele i síťové prostředky. Pokud uživatel JavaScript povolený nemá, zabrání logika až na straně serveru dalšímu zpracování chybných vstupů (a nejspíš o tom i informuje uživatele).

2.4 XML

2.4.1 Původ XML

Jazyk XML (v překladu „rozšiřitelný značkovací jazyk“) je následníkem jazyka SGML. Standard SGML vznikl již v roce 1986. SGML byl vytvořen jako sémantický a strukturovaný značkovací jazyk určený pro textové dokumenty. Největším úspěchem SGML je jeho aplikace jazyk HTML, který je specializovaný čistě pro popis webových stránek a přirozeně nenabízí celou množinu schopností SGML. Pro jiné využití SGML, například jako prostředku pro správu velkého množství dokumentů nebo jako formát pro výměnu dat, se nabízejí jiné aplikace. Přenositelnosti a kompatibilitě jednotlivých aplikací SGML však stála v cestě překážka v podobě velice složité a obsáhlé specifikace SGML, která pokrývá mimo jiné i velké množství zvláštních a nepravděpodobných situací. Dá se říci, že neexistuje aplikace, která plně splňuje specifikaci SGML a rozdíly mezi aplikacemi bývají často významné - vlastnost, kterou jedna aplikace považuje za podstatnou, druhá vůbec takto vidět nemusí a neimplementuje ji.

V roce 1996 započaly práce na vývoji odlehčené verze SGML - takové verze, která by zachovala většinu užitečných vlastností SGML a vynechala rysy, které byly mnoha uživatelům nejasné, byly příliš komplikované na implementaci nebo se neprokázala jejich užitečnost. Na tomto projektu pracoval tým 11 lidí s podporou mnoha desítek dalších osob. Výsledkem se stala v roce 1998 první verze XML 1.0, která slavila okamžitý úspěch. Standard XML definuje obecnou syntaxi, která se používá pro označování dat jednoduchými, člověku srozumitelnými značkami. Ustanovuje flexibilní formát pro počítačové dokumenty.

Následovaly další standardy nějakým způsobem rozšiřující či doplňující XML, jako například XML Namespaces, jazyk pro stylové šablony XSL nebo odkazovací jazyk XLL. Tyto standardy pak prošly dalším vlastním vývojem. K samotné základní specifikaci XML přibyla časem řada dalších rozšíření jako například XML Schemas nebo XHTML [3].

2.4.2 Základní rysy XML

Dokument XML se skládá z textového obsahu označkovaného pomocí textových značek. Značek je neomezené množství, uživatel si může vytvářet značky jaké sám chce. Neexistuje žádná množina daných značek. Jediné, čím je uživatel limitován, jsou striktní pravidla pro strukturu dokumentu a určitá omezení v tom, jak může být značka tvořena a co může obsahovat. O dokumentech XML, které tato syntaktická pravidla splňují, se říká, že jsou *správně formulovány* (*well-formed*). Pokud dokument není well-formed, není ani považován za XML dokument a analyzátory XML jej odmítnou dále zpracovávat.

Názvy značek mají popisovat pouze strukturu dokumentu a sémantiku elementu (značka spolu s textovým obsahem) - například vyznačují, že nějaký element obsahuje jméno, město, datum, souřadnice atd. (viz výpis 2.3). Různé organizace či jednotlivci se mohou dohodnout, jaké přesně značky budou používat pro určitou sémantiku. Vznikají tak speciální aplikace XML, které mohou popisovat například chemické sloučeniny, grafiku, knihy v nějakém katalogu apod.

```
<?xml version="1.0"?>
<studenti fakulta="FIT">
  <student>
    <jmeno>Jana</jmeno>
    <prijmeni>Firlová</prijmeni>
    <login>xfirlo00</login>
  </student>
  <student>
    <jmeno>Lubomír</jmeno>
    <prijmeni>Hurtečák</jmeno>
    <login>xhurte00</login>
  </student>
</studenti>
```

výpis 2.3 - XML: sémantické názvy značek

Takovéto aplikace XML se popisují a standardizují pomocí definice typu dokumentu - DTD. Tato definice pak určuje přesná pravidla pro to, jaké značky a jak mohou být v dokumentu použity (například pravidla pro vnoření značek). U dokumentu, který obsahuje DTD, je pak při analýze vyhodnocováno nejen jestli správně formulován (splňuje základní XML standard), ale jestli je i platný (validní), tedy v souladu s příslušnými pravidly DTD. Platnost dokumentu závisí čistě na tom, proti jaké definici DTD je porovnáván. Ověřování validity funguje na principu, že co není povoleno, je zakázáno. Vše v dokumentu tedy musí odpovídat pravidlům deklarovaným v DTD. Neplatnost dokumentu není zdaleka tak fatální, jako když dokument není správně formulován, a analyzátory XML nemusí mít problém jej i přes upozornění zpracovat.

Elementy v XML mohou mít atributy, jak je vidět na výpisu 4.1 u elementu „studenti“. Velká část informací by se dala vlastně zakódovat jen do atributů elementů, a naopak, některé atributy by se daly převést na samostatné elementy. O tom, kdy je vhodné použít pro uložení informací synovské elementy a kdy atributy, se vedou diskuse. Obecně by se měly používat atributy pro metadata elementu a elementy pro data, někdy ale nemusí být jasné rozlišení dat a metadat. Atributy se obecně dají použít pro doplňkové informace v dokumentech orientovaných na sdělení nebo třeba pro odkazy (viz výpis 4.2).

Obsah dokumentu XML je vždy textový, nikdy neobsahuje binární data. Dokáže jej tedy číst jakýkoliv program, který dokáže číst textové dokumenty. To z něj dělá maximálně přenositelný formát mezi platformami a různými programy. Navíc je díky tomu snadno čitelný člověkem, který je schopen jej i přímo kontrolovat a editovat.

Dokumenty XML mají strukturu shodnou s DOM stromovou strukturou. Obsah je tvořen kořenovým elementem, který dále obsahuje synovské elementy nebo znaková data nebo obojí. Pro synovské elementy dále platí rekurzivně stejná pravidla pro jejich obsah jako pro kořenový element. Výpis 4.1 ukazuje jednoduchý XML dokument, ve kterém obsahem elementů jsou vždy buď čistě další synovské elementy nebo znaková data. XML lze však aplikovat i na běžný text a vznikají tak přirozeně elementy s kombinovaným obsahem, jak ukazuje výpis 2.4:

```
<?xml version="1.0"?>
<student><jmeno><krestni_jmeno svatek="24.5.">Jana</krestni_jmeno>
<prijmeni>Firlová</prijmeni></jmeno> studuje již pátým rokem na
fakultě <fakulta xlink:type="simple" xlink:href=
"http://www.fit.vutbr.cz">FIT<fakulta>. Po třech letech studia
bakalářského programu získala titul <titul
zkratka="Bc">bakalář</titul>. Nyní je v <rocnik>2.</rocnik> ročníku
studijního programu <stud_program>MIS</stud_program>. </student>
```

výpis 2.4 - XML dokument uspořádaný jako sdělení

Dokumenty XML mohou dále obsahovat tzv. zpracovací instrukce, což jsou instrukce určené pro konkrétní aplikace. Nejedná se o elementy a můžou se nacházet kdekoliv v XML dokumentu, mimo vnitřku značek - dokonce i před nebo za kořenovým elementem. Zpracovací instrukce začíná znaky „<?“ následovanými označením aplikace, pro kterou jsou instrukce určeny. Výpis 2.5 ukazuje příklady takovýchto zpracovacích instrukcí, které by mohly být webovým vývojářům z praxe blízké.

```
<?php ....jakkoli dlouhý php kód ... ?>
<?xml-stylesheet href="styl.css" type="text/css"?>
<?robots index="yes" follow="no"?>
```

výpis 2.5 - příklady zpracovacích instrukcí

Poznámka: úvodní řádek ve výpisu 4.1 a 4.2 má sice tvar zpracovací instrukce, ale jeho funkce je čistě jen deklarativní.

Nyní, když jsou osvětleny základní rysy XML, mohu shrnout základní pravidla (pouze základní!), která musí XML dokument splňovat - tedy podmínky pro správně formulovaný XML dokument [3]:

- každá počáteční značka musí mít odpovídající ukončovací značku
- elementy se nesmí překrývat
- musí existovat právě jeden kořenový element
- hodnoty atributů jsou uzavřeny do uvozovek
- jeden element nesmí mít více atributů stejného názvu

- komentáře a zpracovací instrukce musí být mimo značky
- ve znakových datech elementu nebo atributu se nesmí nacházet nenahrazené znaky ‘<‘ nebo ‘& ’ (nahrazování pomocí entitních referencí)

2.4.3 Dvojí role XML dokumentů

XML dokument nemusí být chápán pouze jako dokument někde pevně uložený. Velice často má podobu dynamicky sestavovaného dokumentu, který může existovat jen v rámci trvání nějakého procesu.

I když škála využití XML dokumentů je široká a velice pestrá, lze rozlišit dva hlavní způsoby využití tohoto formátu [3].

2.4.3.1 Formát dokumentů

XML byl vytvořen především jako formát pro ukládání dokumentů, zejména pro webové stránky, dále pak knihy, povídky, vědecké články, příručky, učebnice, smlouvy a podobné dokumenty, které čtou lidé. XML kombinuje volnější a srozumitelný styl požadovaný pisateli článků, a pevnou datovou strukturu, která vyhovuje programátorům.

Této roli XML - formát dokumentů, které mají číst lidé, napomáhá řada technologií [3]:

- XSL, stylový jazyk, který se skládá ze dvou částí: XSLT a XSL-FO. XSLT je aplikací XML, která dokáže transformovat dokumenty XML na dokumenty jiného formátu (např. jiný XML, HTML, holý text). XSL-FO představuje úplný popis uspořádání a prezentace dokumentu. Téměř nikdy se nepíše přímo a vzniká klasicky jako výsledek aplikace transformace XSLT na výchozí XML dokument.
- XPath, jazyk mimo XML používaný pro identifikaci a odkazování se na různé části XML dokumentu. Výrazy XPath se užívají například v XSLT pro vyhledání a výběr elementů, které mají být zpracovány.
- XLink, syntaxe užívaná pro vytváření odkazů v XML dokumentu.
- XPointer, syntaxe užívaná pro určení bodů a rozsahů uvnitř XML dokumentu.

2.4.3.2 Datový formát

XML se neprosazuje pouze jako formát dokumentů „pro lidi“, k čemuž byl určen, ale stále více také jako datový formát pro přenos a ukládání informací. Má potenciál nabídnout formát přenositelný mezi platformami i programovacími jazyky, do kterého by se daly uložit i složité datové struktury. Toto užití XML je určeno primárně pro dokumenty, které bude číst program.

Bez XML programátor musel sám rozhodovat, jak budou vypadat uložená nebo přenášená data programu. Data často nebyla určena k použití mimo konkrétní program, proto programátor ukládal data způsobem, který byl zrovna pro něj nejvýhodnější a vznikalo tak nepřeberné množství velice speciálních formátů. XML nabídlo programátorům další možnost, jak ukládat svá aplikační data.

Výhody této možnosti převyšují mírně zvýšenou režii, která vzniká rozšířením aplikace o analyzátor XML:

- jednoduchá syntaxe umožňující snadno generovat a zpracovávat jazyk
- podpora vnořování, umožňující snadnou reprezentaci různých struktur
- snadné ladění, tento běžně čitelný formát lze snadno simulovat a vytvářet vzorky pro ladění
- nezávislost na jazyku a platformě - vhodné i pro smíšená prostředí
- rozšířený XML analyzátor, rozsáhlé knihovny pro práci s XML

Také XML jako datový formát má k dispozici několik podpůrných technologií [3]:

- XML Schema, jazyk, který obohacuje mechanismus DTD a umožňuje důkladnější kontrolu obsahu XML dokumentu. Přináší možnost kontrolovat typ znakových dat. Validní dokument pak musí nejen splňovat příslušný DTD předpis, ale i omezení definovaná pravidly XML Schema.
- DOM
- SAX, alternativní rozhraní pro XML k DOM. XML dokument je zpracováván sekvenčně a je při tom využíván mechanismus zpětných volání, které oznamují klientské aplikaci, že se analyzuje nějaká konkrétní konstrukce. Není zde nutnost načítat do paměti celou strukturu zpracovávaného dokumentu, což podstatně zlepšuje výkon při zpracovávání velmi rozsáhlých dokumentů. Tento přístup je použitelný tam, kde postačuje jeden sekvenční průchod dokumentem. Dokument se programu předává po částech, přičemž program může začít pracovat s částmi dokumentu ihned, jakmile je obdrží. V současnosti je využíván řadou XML analyzátorů.

2.5 Ajax

Nyní si můžeme představit Ajax, mladou techniku používanou pro vytváření interaktivních webových aplikací, která využívá spoustu z výše představených technologií či na ně navazuje.

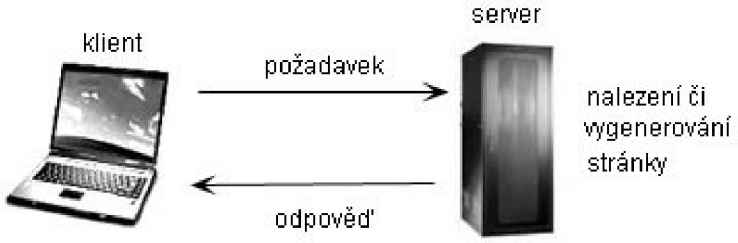
Ajax je zkratkou pro asynchronní JavaScript a XML. Hlavní přínos Ajaxu pro dynamické webové aplikace spočívá v tom, že je to malá, jednoduchá a elegantní technika, umožňující webové aplikaci na straně klienta získávat data ze serveru asynchronně bez nutnosti znovunačítání stránky. Využívá jen starých známých prostředků a pro používání Ajaxu se vývojář nemusí učit žádný nový jazyk, ani zavádět nové nástroje. Taktéž prohlížeče obsahují všechny potřebné prostředky pro fungování Ajaxu.

2.5.1 Princip Ajaxu

Ústřední komponentou Ajaxu je JavaScript spolu se svým objektem XMLHttpRequest (dále jen XHR). Tento objekt se objevil poprvé zcela nenápadně v IE 5, a i když byl jen málo známým, převzala jej brzy i Mozilla a později po ní ostatní prohlížeče.

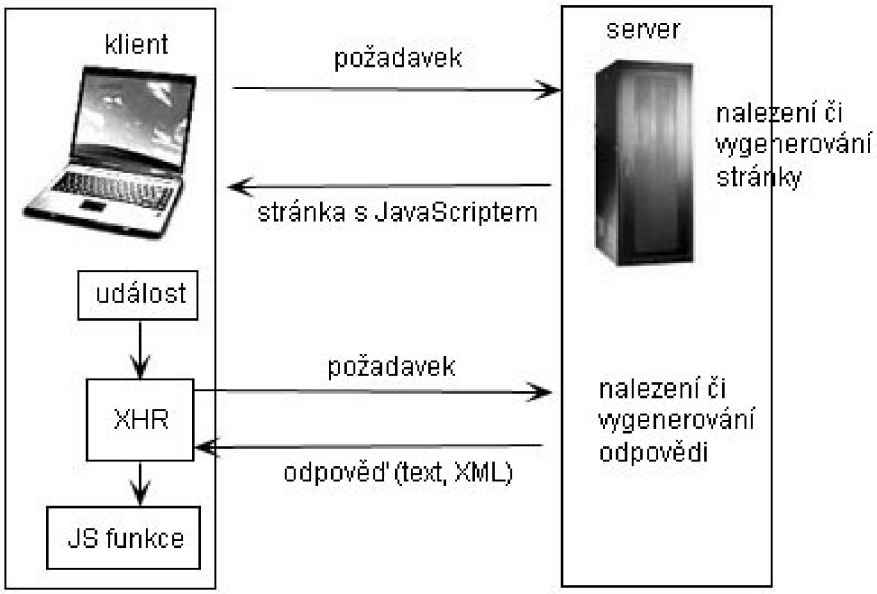
Objekt XHR umožňuje JavaScriptu zcela novou věc - požadovat pro sebe od serveru další data. Dosud mohla být data dodána jedině spolu s celou webovou stránkou. XHR dokáže připravit a poslat serveru požadavek, vyčkat na odpověď a tu si pak sám zpracovat. Požadavky dokáže vytvářet synchronní, kdy do přijetí odpovědi ustrne veškerá činnost JavaScriptu, a asynchronní, které představují právě onen přínos Ajaxu. Zatímco je vyřizován požadavek, uživatel může normálně se stránkou pracovat. V okamžiku, kdy dorazí od serveru odpověď, si ji JavaScript zpracuje a na jejím základě může aktualizovat právě jen ty části stránky, které jsou potřeba. Při zpřístupňování částí dokumentu a stylu dokumentu využívá JavaScript funkce implementující DOM rozhraní, při komunikaci se serverem obvykle formát dat XML a pro vyřízení požadavku prakticky jakoukoliv technologii na straně serveru, ať už JSP nebo PHP nebo cokoliv jiného. Ajax víceméně nezajímá a není pro něj nijak podstatné, jak je mu na serveru vytvářena odpověď, důležitá je jen odpověď a znalost názvu prostředku, po kterém ji má požadovat.

Obrázek 2.7 ukazuje klasický model komunikaci prohlížeče s webovým serverem:



obrázek 2.7 - běžná komunikace bez použití Ajaxu

Obrázek 2.8 znázorňuje, co se děje, když pracuje stránka obsahující Ajax. Nejprve je načtena ze serveru stránka obsahující JavaScript, který umožňuje použití techniky Ajax. JavaScript pak v reakci na nejrůznější události (klasické *DOM Events* nebo například uplynutí nějakého časového intervalu) může vytvořit objekt XHR, určit mu podle parametrů, co má požadovat, a kterou funkci má volat, až přijde odpověď. Jakmile je objekt XHR volný (nečeká na odpověď), může generovat další požadavky [1, 2].



obrázek 2.8 - komunikace stránky obsahující Ajax

2.5.2 Objekt XMLHttpRequest zblízka

Objekt XHR není žádným W3C standardem, i když mnoho z jeho funkcionality je popsáno v DOM Level 3: Load and Save. Z tohoto důvodu se může jeho chování v různých prohlížečích lišit, většina metod a atributů je však široce podporovaná a implementována podobně. XHR je v současnosti implementován ve všech nejrozšířenějších aktuálních prohlížečích a jeho použití je v nich velice podobné.

Jediný rozdíl v použití XHR, který jsem byla nucena překlenout javascriptovým kódem, je hned při vytváření instance tohoto objektu. IE totiž implementuje XHR jako objekt ActiveX (kterým byl v době svého vzniku), ostatní prohlížeče jako nativní objekt JavaScriptu. I tak však zůstává vytvoření XHR snadnou záležitostí, jak ukazuje výpis 2.6:

```
function vytvorXHR() {
  var XmlHttp = null;
  try { XmlHttp = new XMLHttpRequest(); }
  catch (e){
    try { XmlHttp = new ActiveXObject("Microsoft.XMLHTTP"); }
    catch(e) {}
  }
  return XmlHttp;
}
```

výpis 2.6 - možný způsob vytvoření XHR

2.5.2.1 Metody a atributy

Sada metod a atributů objektu XHR potřebných k plnému využití v Ajaxu není nijak velká, což usnadňuje vývojářům seznámení se s Ajaxem. Jejich kompletní přehled uvádí následující tabulka. Zvýrazněny jsou ty atributy, bez kterých se neobejde ani nejjednodušší implementace Ajaxu - mají převahu [1, 2].

tabulka 2.1 - Metody a atributy XHR

Metoda	Popis
`abort()`	přeruší aktuální požadavek
`getAllResponseHeaders()`	vrací všechny hlavičky HTTP požadavku jako řetězec
`getResponseHeader("hlavička")`	vrací hodnotu zadané hlavičky jako řetězec
`open``("metoda", "url", async, "login", "heslo")`	nastaví parametry volání serveru: • metoda: GET, POST, PUT, HEAD • url: požadované URL • async: true/false, nepovinné, implicitně true • login, heslo: pokud je vyžadováno pro přístup k url
`send``(obsah)`	odešle požadavek serveru
`setRequestHeader("hlavička","hodnota")`	nastaví určenou hlavičku na požadovanou hodnotu
Atribut	**Popis**
`onreadystatechange`	ukazatel na obslužný kód (funkci), který je spuštěn po změně stavu požadavku
`readyState`	stav požadavku, hodnoty 0 až 4, důležité především hodnoty 0 (neinicializováno) a 4 (dokončeno), které signalizují, že XHR je volný a stav 4 navíc, že přišla odpověď od serveru
`responseText`	odpověď serveru ve formě řetězce
`responseXML`	odpověď serveru ve formě XML
`status`	stavový kód získaný od serveru (200 - vše v pořádku)
`statusText`	textová verze stavového kódu, vhodné pro čitelná chybová hlášení

Klasické pořadí použití těchto metod a atributů je následující:

1. Je vytvořen XHR objekt.
2. Zavolání metody **open** s nastavením prvních tří parametrů (kam posíláme asynchronní požadavek a jakou metodou - v případě GET jsou součástí URL i parametry)

3. Nastaví se obslužná funkce v atributu **onreadystatechange**.
4. Pošle se požadavek pomocí metody **send**, obsahem je v případě použití GET null, v případě POST zasílaný řetězec parametrů.
5. Obslužná funkce při detekci změny stavu XHR ověří hodnotu jeho stavu pomocí atributu **readystate**.
6. Pokud readyState indikuje dokončení požadavku (4), ověří se, zda odpověď byla v pořádku vyřízena pomocí atributu **status**.
7. Pokud status říká, že vše bylo vyřízeno v pořádku (kód 200), je možné načíst odpověď buď pomocí **responseText** nebo **responseXML**. Odpověď získaná přes responseText je považována za prostý řetězec a hodí se proto jen pro velice jednoduché a nestrukturované odpovědi. Odpověď získaná přes responseXML může být dále zpracována a zpřístupňována jako XML dokument, pokud je správně formulovaným XML dokumentem. Pokud je něco v nepořádku s odpovědí, je dobré využít řetězec obsažený ve **statusText** pro zobrazení čitelného chybového hlášení.

2.5.2.2 Ajax a vlákna

V některých aplikacích používajících Ajax se může stát, že několik rychle po sobě jdoucích událostí vyvolá více požadavků, než jeden XHR objekt může stihnout zvládnout. To se může snadno stát při pomalém připojení, kdy XHR objekt je dlouho zaměstnán čekáním na odpověď od serveru nebo nějakou rychlou posloupností akcí od uživatele, které asynchronní požadavky vyvolávají (například rychlé projetí políček formuláře, která jsou kontrolována pomocí Ajaxu na serveru). Výsledkem může být chyba, která může vést až k ukončení aplikace.

Je několik možností, jak tuto situaci vyřešit [2]:

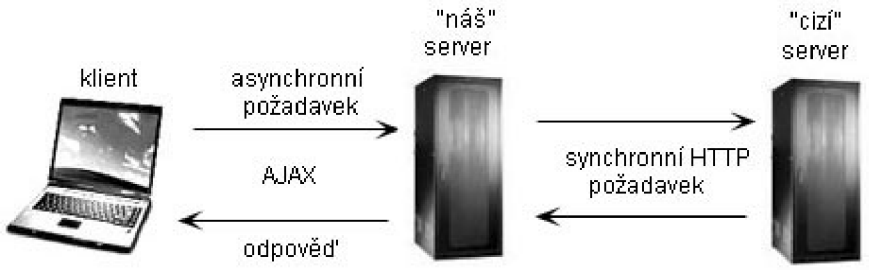
- Pro každý nový požadavek vytvořit nový XHR objekt, což je jednoduché na implementaci, ale náročnější pro server a nezaručuje to pořadí, ve kterém se nám odpovědi vracejí.
- Po uplynutí nějaké doby naplánovat automatický pokus o vytvoření nového požadavku. V jednu chvíli existuje jen jeden XHR objekt, ale není zaručeno pořadí posílání požadavků a už vůbec ne pořadí přijímání odpovědí.
- Uložit požadavek do fronty, nejlépe FIFO, ze které jsou požadavky odebírány a odesílány průběžně, kdykoliv je to možné. Zaručuje zachování pořadí odesílání i přijímání odpovědí.
- Pokud je objekt XHR právě zaměstnán, požadavek ignorovat.
-

Upřednostňuji poslední dvě možnosti (podle situace), protože představují nejmenší zátěž pro klienta i server a umožňují přesnou kontrolu nad požadavky a odpověďmi. Varianta s ukládáním požadavků do fronty navíc není díky javascriptovému objektu *Array* ani náročná na implementaci.

2.5.3 Bezpečnost

Objekt XHR spadá do kategorie zabezpečení prohlížeče. Hlavním bezpečnostním omezením JavaScriptu a tedy i Ajaxu je, že jakýkoliv požadovaný prostředek se musí nacházet ve stejné doméně, ze které pochází volající skript. Parametr *url* metody *open* bude tedy ve většině případů obsahovat jen krátký relativní odkaz, i když umožňuje použití také absolutního odkazování.

Pokud chceme pomocí XHR přistupovat k prostředkům či službám uloženým jinde, můžeme k tomu využít zástupný skript na serveru. Nejlépe to znázorní schéma na obrázku 2.9:



obrázek 2.9 - použití zástupného skriptu v přístupné doméně

Klient posílá pomocí Ajaxu normální asynchronní požadavek do přístupné domény, kde je uložen skript, který může už bez omezení volat služby či vyžadovat prostředky v jiných doménách. Celá procedura zůstává přitom pro klienta perfektně asynchronní, protože místo, kde se čeká na vyřízení synchronního HTTP požadavku, je zástupný skript na serveru [1, 2].

2.5.4 Využití

Praktických využití Ajaxu je tolik, kolik jen fantazie dovolí. Stručně tedy alespoň ta, co se přímo nabízejí, a která už existují v praxi [1, 2]:

- ověřování formulářových dat: umožňuje porovnávat vstup před odesláním například s údaji uloženými v databázi. Když třeba vyplňujete název svého nového e-mailu, dokáže ihned zjistit, zda daný název již není registrován. Dále může na základě vyplnění nějakého vstupu automaticky předvyplnit další vstupy. Umožňuje mít celou ověřovací logiku uloženou na serveru bez ztráty komfortu pro uživatele.
- automaticky se obnovující části stránky: udržuje aktuálnost prvků stránky, například počitadel nebo oznámení novinek
- vytváření rozsáhlého (objemného) systému oken s nápovědou, který není potřeba stahovat celý ke klientovi
- přístup k webovým službám (vyhledávání, získávání náhodných čísel, ...)
- automatické dokončování
- aktualizace výpisů a tabulkových dat

- realizace webového chatu
- efektní změna pořadí seznamů nebo prvku seznamů (přidání/odebrání) pomocí přetahování myší (iluze přidávání a vyhazování zboží z nákupního košíku)

2.5.5 Výhody a nevýhody Ajaxu

Výhody jsou celkem zřejmé - Ajax šetří síťové prostředky tím, že jsou zasílána pouze potřebná data namísto celých stránek. To je cenné hlavně u stránek s rozsáhlými tabulkovými daty. Přináší komfort a úsporu času uživateli, který může se stránkou pracovat během vyřizování požadavků (klasickým příkladem je vyplňování vstupů formuláře, zatímco již vyplněné části jsou průběžně kontrolovány). Umožňuje novou funkcionalitu webových aplikací.

Nic však není dokonalé a i Ajax má své slabiny. Problémy jsou následující:

- Dynamicky generovaná stránka se nezaznamenává do historie prohlížeče, a tak šipka „zpět“ nebude fungovat, pokud bude uživatel chtít zobrazit stránku, kterou viděl před poslední změnou provedenou Ajaxem. Také takováto dynamicky vygenerovaná stránka nemusí jít uložit jako odkaz.
- Zpoždění sítě je v případě použití Ajaxu mnohem viditelnější (ne delší), než při načítání celé stránky. Uživatel například použije tlačítko a může se stát, že se pro něj z neznámého důvodu několik sekund nic neděje. Doporučuje se používat nějaké vizuální signalizace probíhající aktivity.
- Nenápadnost změn provedených Ajaxem - uživatel navyklý na změnu obsahu stránky pouze při načtení celé nové stránky si nemusí často ani všimnout změn, které provede Ajax. Pokud je nebezpečí, že uživatel nová data může opravdu snadno přehlédnout, doporučuje se tato data při zobrazení na malou chvíli nějak zvýraznit (například kontrastní pozadí, které vybledne za pár sekund do normálního).
- Optimalizace pro vyhledávače - webové aplikace, které načítají pomocí Ajaxu data, u kterých je žádoucí, aby byla zaindexována, musí poskytnout tato data také jako normální přístupný odkaz a ve formátu, aby si s nimi poradily vyhledávací programy. Tento problém nemají jen stránky používající Ajax, ale i jiné stránky s dynamicky generovaným obsahem [1, 2, 12].
-

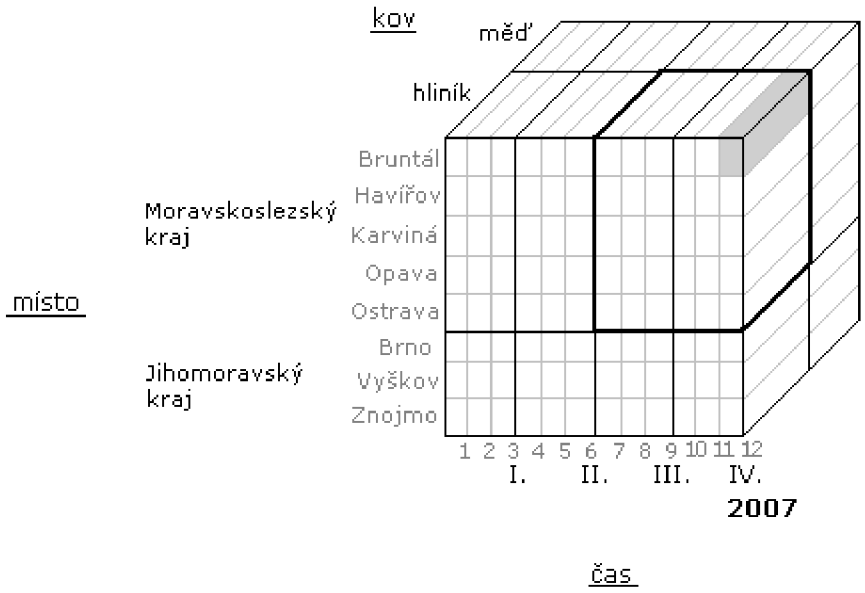
2.6 OLAP

Jedná se o analytický přístup k datům, od nějž se odvozuje struktura databáze – v tomto případě tzv. databázového skladu, charakteristika uložených dat a typické dotazy, které nad databázovým skladem vznikají. Zkratka samotná pochází z anglického výrazu *on-line analytical processing* (on-line analytické zpracovávání). OLAP systémy jsou takové databázové systémy, které jsou navrženy pro analýzu dat a podporu rozhodování. Charakteristika OLAP systémů bývá vymezována obvykle oproti charakteristice OLTP systémů (transakční zpracovávání dat), které představují starší a tradiční způsob práce s daty v databázích. Cíle OLAP a OLTP systémů jsou odlišné, každý slouží jinému účelu a bývají nasazeny v jiných situacích. Tyto dva přístupy si nijak nekonkurují. Zde jsou hlavní odlišnosti OLAP systému od OLTP systému, se kterým je předpokládám čtenář seznámen:

- pracuje se typicky s mnohem větším objemem dat než v OLTP systému, často různě sebraná a integrovaná historická data,
- data nemusí být (a nebývají) v normálních formách,
- pracuje se výrazně více s agregovanými hodnotami,
- uplatňují se jiné logické pohledy na data – schémata hvězdy, souhvězdí, model datové kostky, rozlišení dimenzí a faktů,
- data jsou relativně neměnná, drtivá většina dotazů jsou vyhledávací, často velmi složité dotazy, s tím souvisí hojně užívaná indexace záznamů,
- výsledky dotazů bývají používány pro podporu rozhodování a plánování v dlouhodobějším horizontu, v datech se nepátrá po jednotlivých přesných položkách, ale po různých vzorech, tendencích, zajímavých obvyklých a neobvyklých jevech – tímto se zabývá celý obor *data mining* (dolování dat),
- je požadována rychlá reakce na složité dotazy (průchodnost dotazů).

2.6.1 Datová kostka

Je to logický pohled na data, která mohou, ale nemusí, být uložena v této formě. Obecně je kostka multidimenzionální, kdy stěny kostky představují tzv. *dimenze* a obsah kostky *fakta*. Fakta se v kostce seskupují podle dimenzí, které jsou nad nimi definovány. Každá dimenze má definovanou svou konceptuální hierarchii úrovní. Zadání úrovně dimenze pro zobrazení faktů určuje, jak hodně nebo málo budou data agregována. Názorněji než slovní nebo matematický popis dle mého přibližuje datovou kostku následující obrázek 2.10:



obrázek 2.10 – datová kostka, výkupy kovů

Jedná se o kostku s daty o výkupech barevných kovů. Dimenzemi jsou čas, místo a kov. Data můžeme s ohledem na čas seskupovat po úrovních roky, čtvrtletí, měsíce (případně týdny, dny ...to na obrázku pro přehlednost již není). U místa výkupu kovu můžeme volit úroveň krajů nebo měst. Druh kovu má pouze jednu úroveň.

Obsahem kostky – faktem – mohou být v tomto případě váhy různých kovů, které byly vykoupeny za daný čas na daném místě. Část kostky ohraničená tlustou čárou bude například obsahovat údaj, kolik hliníku vykoupila firma v 3. a 4. čtvrtletí roku 2007 v celém moravskoslezském kraji (úroveň pro čas bude čtvrtletí a pro místo kraj). Šedá buňka zase kolik hliníku bylo vykoupeno v prosinci 2007 v Bruntále (úroveň pro čas měsíc, pro místo město).

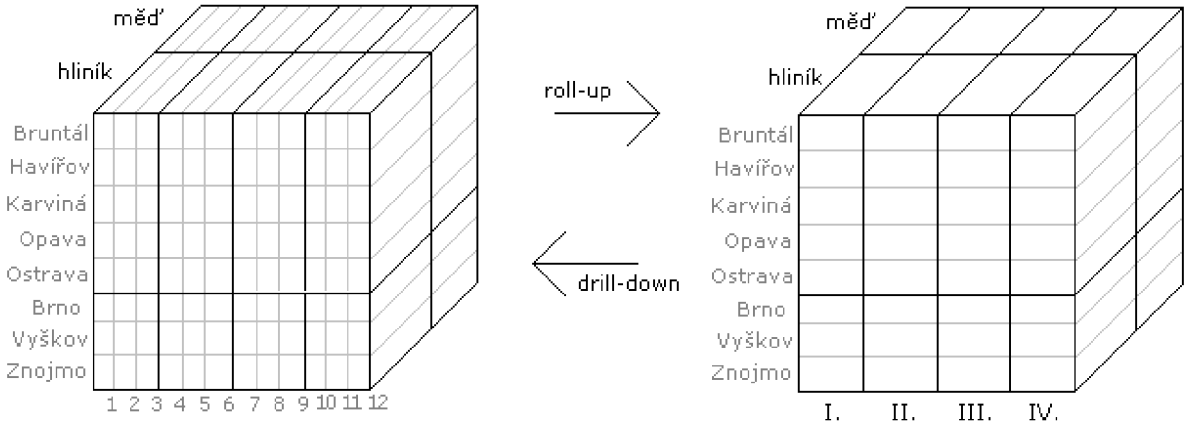
Faktů může být samozřejmě více, spolu s váhou kovu může buňka obsahovat například informaci o tom, za kolik peněz byl materiál vykoupen, nebo kolik kovu donesl průměrně jeden zákazník.

2.6.2 Základní operace nad datovou kostkou

Rozeznáváme pět typických operací nad datovou kostkou: roll-up a drill-down - změny stupně agregace, slice a dice – výběr podkostek, pivoting – změna pohledu na data.

2.6.2.1 Roll-up a drill-down

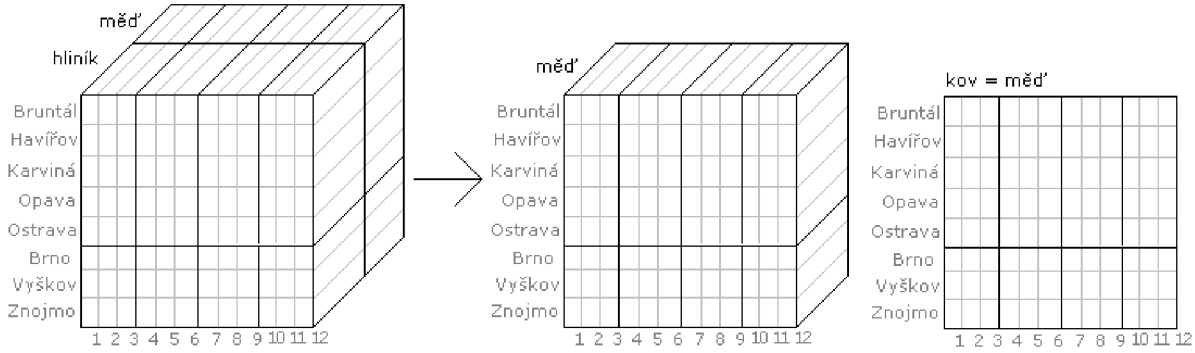
Roll-up zvýší stupeň abstrakce dat posunem v konceptuální hierarchii jedné nebo více dimenzí směrem nahoru. Drill-down je operací inverzní, v konceptuální hierarchii se pohybujeme směrem dolů. Díváme se přitom stále na stejná data (celková suma je pořád stejná).



Obrázek 2.11 – roll-up, drill-down

2.6.2.2 Slice

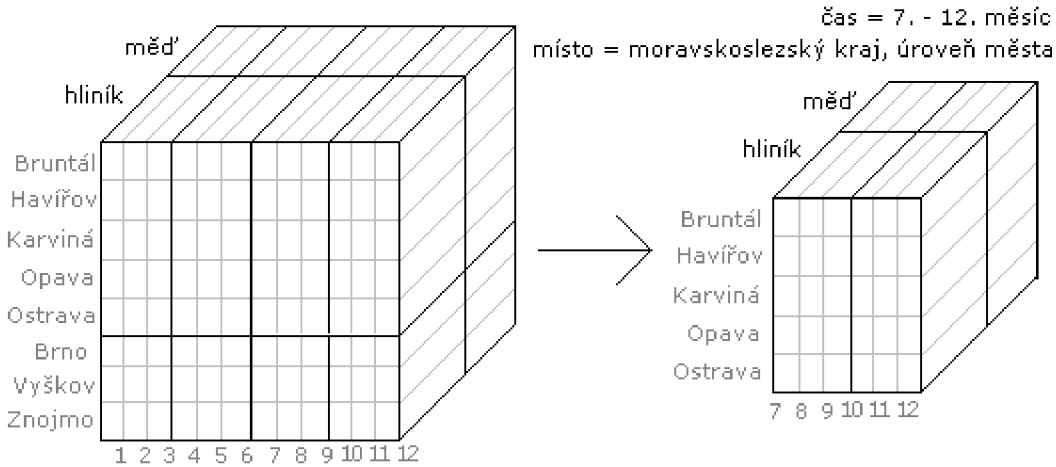
Dochází ke zploštění některé dimenze. Té je přiřazena určitá hodnota, pro kterou se mají data hledat. Při znázornění výsledku operace lze tuto dimenzi vypustit.



Obrázek 2.12 – slice

2.6.2.3 Dice

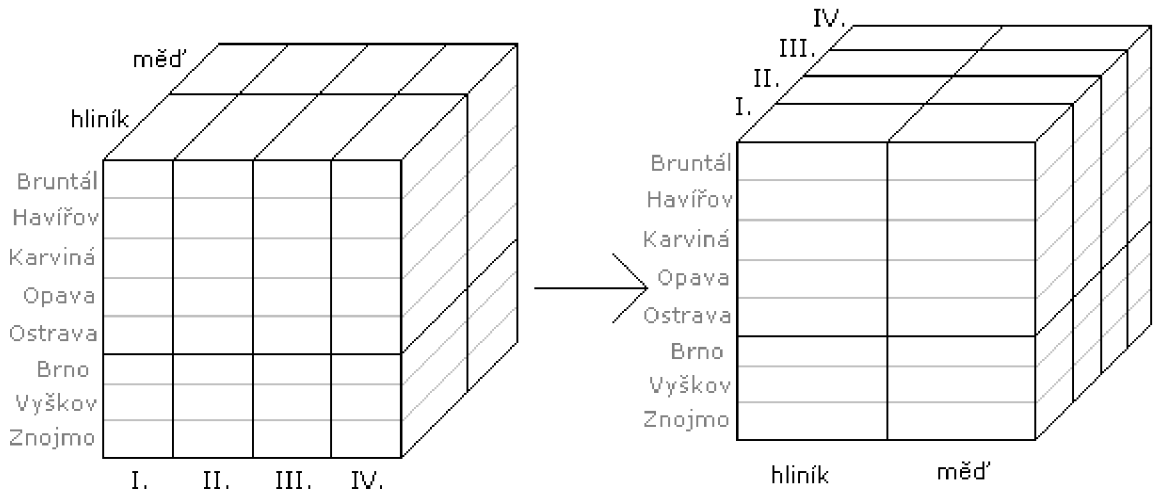
Pro výběr dat jsou nastaveny určité podmínky, je vybrána podmnožina dat původní kostky.



Obrázek 2.13 – dice

2.6.2.4 Pivot

Změna pohledu na data, mění se uspořádání dimenzí.



Obrázek 2.14 – pivot

2.6.3 Reprezentace výsledků OLAP operací

Datové kostky mohou mít mnoho dimenzí a nabízí se zde otázka, jak je znázornit v dvourozměrném prostoru tak, aby byly čitelné a srozumitelné.

Pro maximálně dvourozměrné kostky se dají použít běžné **křížové tabulky**, které bývají doplněny v posledním sloupci a řádku o mezisoučty a celkové součty. Ty však často nestačí, proto jsou zde další možnosti, z nichž chci zmínit kontingenční tabulky a dynamické tabulky.

Tabulky reprezentující data je vhodné dále kombinovat pro větší přehlednost s histogramy. Existuje řada dalších speciálních grafů, které se používají především při vizualizaci výsledků OLAP analýzy (koláčové grafy, box plots, scatter plots aj.) [13].

2.6.3.1 Kontingenční tabulky (pivot table)

Pro vícerozměrné kostky se nejčastěji využívají kontingenční tabulky, běžně známé z různých tabulkových procesorů. Ty oproti klasickým tabulkám umožňují snadno měnit strukturu sloupců a řádků - zaměňovat je, od toho se odvozuje jejich anglický název. Umožňují také pohyb po zadaných hierarchiích sloupců či řádků, automatické třídění, seskupování a sumarizování dat. Obrázek 2.14 ukazuje obyčejnou tabulku, která by se dala považovat za kus seřazeného výpisu dat z databáze, kde sloupce A-D by mohly být dimenzemi a E-G fakty:

	A	B	C	D	E	F	G
1	**Region**	**Gender**	**Style**	**Ship Date**	**Units**	**Price**	**Cost**
2	East	Boy	Tee	1/31/2005	12	11.04	10.42
3	East	Boy	Golf	1/31/2005	12	13	12.6
4	East	Boy	Fancy	1/31/2005	12	11.96	11.74
5	East	Girl	Tee	1/31/2005	10	11.27	10.56
6	East	Girl	Golf	1/31/2005	10	12.12	11.95
7	East	Girl	Fancy	1/31/2005	10	13.74	13.33
8	West	Boy	Tee	1/31/2005	11	11.44	10.94
9	West	Boy	Golf	1/31/2005	11	12.63	11.73
10	West	Boy	Fancy	1/31/2005	11	12.06	11.51
11	West	Girl	Tee	1/31/2005	15	13.42	13.29
12	West	Girl	Golf	1/31/2005	15	11.48	10.67

Obrázek 2.14 – obyčejná tabulka

Pokud nás nezajímají všechna data pro nejnižší úrovně abstrakce (což většinou nebývá), je tato tabulka málo přehledná a lze zní jen těžko vyčíst nějaké zajímavé informace. Zde je příklad, jak by mohla vypadat kontingenční tabulka vytvořená z těchto dat, pokud by nás zajímaly z faktů pouze počet prodaných kusů (Units) v závislosti na oblasti (Region) a datum zásilky (Ship date):

Sum of Units	Ship Date					
Region	1/31/2005	2/28/2005	3/31/2005	4/30/2005	5/31/2005	6/30/2005
East	66	80	102	116	127	125
North	96	117	138	151	154	156
South	123	141	157	178	191	202
West	78	97	117	136	150	157
(blank)						
Grand Total	363	435	514	581	622	640

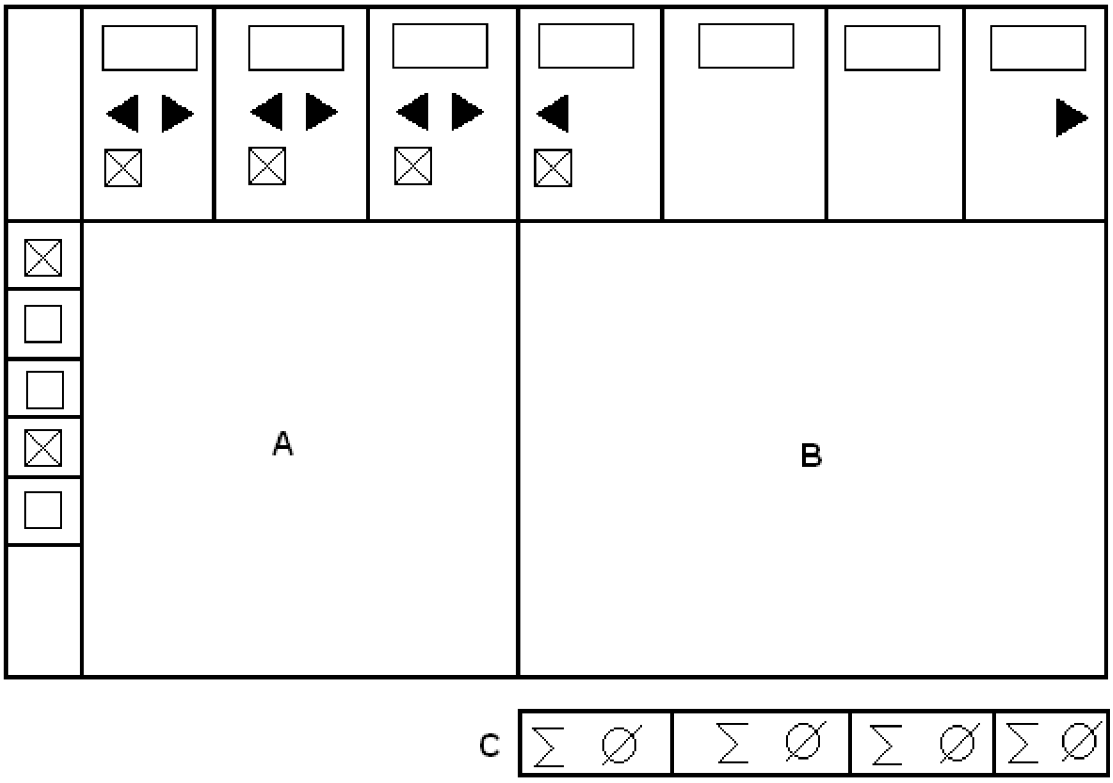
Obrázek 2.15 – kontingenční tabulka

Pro více dimenzí začínají být ale i kontingenční tabulky méně přehledné [14].

Poznámka: v českém jazyce je nebezpečí záměny významu spojení „kontingenční tabulka“ užívaného v IT za tabulky užívané ve statistice. V angličtině mají oba tyto typy tabulek odlišné názvy – contingency table (statistika) a pivot table (IT).

2.6.3.2 Dynamické tabulky (dynamic tables)

Jedná se o novější metodu, jak zobrazit multidimenzionální data pomocí tabulky. Dynamické tabulky zároveň umožňují realizaci OLAP operací. Úplnou strukturu této tabulky schématicky znázorňuje obrázek 2.16:



Obrázek 2.16 – dynamická tabulka

Tabulka se skládá ze dvou základních částí - **oblasti dimenzí** (A) a **oblasti faktů** (B). Oblast dimenzí obsahuje sloupce s položkami dané dimenze, seskupenými s ohledem na zvolenou úroveň dimenze. Každý řádek této oblasti představuje jednu podkostku.

Oblast faktů obsahuje data určená oblastí dimenzí, různě agregovaná dle konkrétních požadavků (například suma, průměr).

Pod fakty se může objevit další část tabulky, **oblast statistik** (C). Zde se objevují různé statistické metriky pro příslušné sloupce faktů.

Záhlaví tabulky je **oblastí filtrů** pro dané sloupce. Šipky vpravo a vlevo nad oblastí dimenzí umožňují prohazování sloupců (operace pivot), zaškrtávací box určuje, zda bude dimenze použita při konstruování podkostky (operace slice). Vyplňovací textové pole v záhlaví umožňuje zadávat různé vyhledávací filtry a spolu s uloženou informací o úrovních dané dimenze realizuje dice operace.

Šipka vlevo nad oblastí faktů umožňuje roll-up, šipka vlevo inverzní drill-down. Textový vstup nad sloupci s fakty umožňuje opět filtrování.

Zaškrtávací políčka na začátku každého řádku umožňují filtrování řádků.

Z nejlevějšího sloupce dimenzí a vybraného sloupce faktů lze generovat histogram [13].

3 Cíl práce

Jak jsem již popsala výše, technika Ajax není nijak zvlášť obtížná pro webového vývojáře na zvládnutí a základní použití. Na konci však záleží vždy na konkrétní aplikaci a její složitosti.

Aby bylo dodrženo zadání diplomové práce a zároveň aby řešený problém nebyl pro DP příliš triviální, stanovili jsme s vedoucím DP jako cíl vytvoření stránky či stránek, které zobrazují výsledky OLAP operací v nějaké „inteligentní“ tabulce a zároveň umožňují vytváření nových OLAP dotazů. Data mají být do tabulky načítána pomocí Ajaxu. Přenášená by měla být pomocí formátu XML, který se bude zpracovávat s využitím funkcí implementujících DOM rozhraní. Mělo by být možné vytvářet dynamicky nějaké grafy z dat v tabulce (histogramy). Tabulka by měla být přehledná a interaktivní. Na konci bych měla zhodnotit, jak se v této aplikaci uplatňuje Ajax, zda je přínosný či nikoli. Aplikace by měla fungovat v prohlížečích Mozilla, Opera, IE 6+.

Vedoucí DP mi k tomu poskytl zdrojové kódy internetového obchodu s vínem a k němu příslušnou databázi naplněnou reálnými daty. Zdrojové texty zahrnovaly kompletní logiku obchodu napsanou v jazyce PHP, ovšem „holou“, téměř bez stylu, nějakého vizuálního rozvržení stránek a čehokoliv jiného. Nad dodanou relační databází jsem měla za úkol vytvořit si nějaký logický pohled na data odpovídající strukturám OLAP systému, tedy určit si nějaké zajímavé dimenze a fakta, se kterými budu dále pracovat.

Stránku s OLAP daty a ovládáním jsem se rozhodla zakomponovat do dodaného systému, jehož funkčnost jsem proto musela nastudovat. Dále jsem se rozhodla dát systému nějakou „tvář“ a pomocí Ajaxu vylepšit a rozšířit stávající vyhledávání v katalogu zboží.

4 Řešení

Mé řešení DP je vestavěno do hotového systému internetového obchodu s vínem, který mi byl dodán. Části systému, které jsem vytvořila úplně nové nebo dotvořila ze starých, zapadají do dvojjazyčné povahy systému a snaží se zachovat styl a techniky, jakými je systém napsán. Pracuje se s vnitřními proměnnými původního systému, jsou využívány původní funkce a konstrukce.

Systém je realizován pomcí těchto prostředků:

- logika na straně serveru – PHP
- výsledné dokumenty ve formátu XHTML + CSS
- JavaScript na straně klienta
- MySQL databáze

Systém původně generoval dokumenty typu HTML 4.01, což jsem změnila na čistší a modernější XHTML 1.0 (Strict).

Při tvorbě nebyly použity žádné cizí knihovny či funkce, všechny funkce jsou buď mé dílo nebo bylo součástí původního systému. Pokud budu dále popisovat nějakou funkcionalitu a nebude se jednat o můj výtvor, bude to vždy explicitně uvedeno.

4.1 Databáze

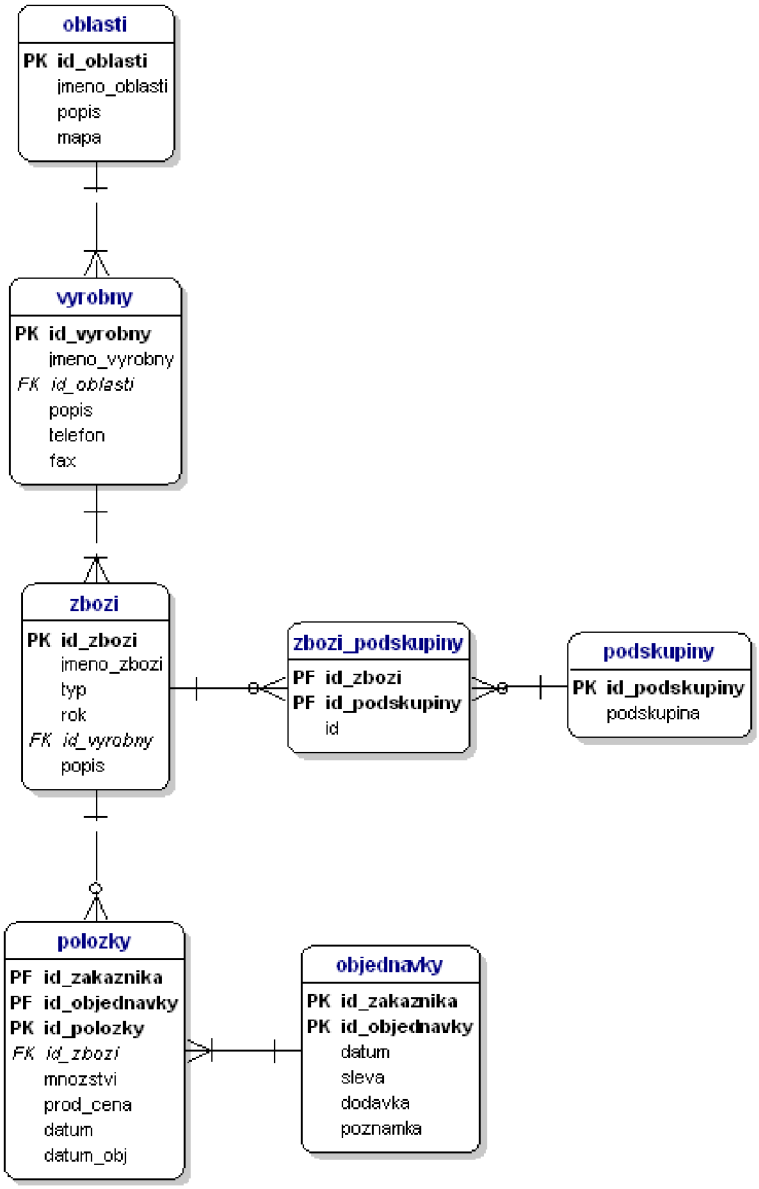
Uvedu pouze ty části databáze, které mají význam pro tuto práci. Databáze není nijak složitá, jedná se relační databázi s ústředními prvky **zboží**, **oblast** jeho původu a **typ**, zákazníci, **objednávky** a sklad. Se skladem (určuje kolik je zboží na skladě a jaká je jeho aktuální cena) ani zákazníky dále nepracuji, proto je v dalším popisu vynechám.

4.1.1 Struktura

Části databáze podstatné pro tuto práci ukazuje obrázek 4.1. Středem, ke kterému se vše vztahuje, je tabulka **zboží**. V ní jsou uloženy nějaké základní informace o víně, jako jeho název, ročník, výrobna a základní typ. Pod typem se skrývá základní rozdělení vín do několika málo skupin jako například červené, bílé, šumivé aj. Dále je se zboží může řadit do více **podskupin**, které dávají o víně podrobnější informace, jako například odrůda hroznů. Informace o podskupinách jsou důležité pro výpisy pro uživatele, protože mnoho vín se liší právě jen jimi (a pochopitelně svým id, které ovšem běžně člověku nic neřekne). Ve výpisu by tedy mohlo být zavádějící uvést jen název a ročník, proto se k těmto dvěma údajům v systému připojuje ještě výčet podskupin, do kterých je zboží zařazeno.

Zboží má danou svou **výrobnu**, k té se váže informace o **oblasti**, ve které se výrobna nachází.

Databáze obsahuje záznamy o prodejích zboží v tabulkách **objednávky** a **položky**. Na jednotlivé druhy zboží odkazují položky objednávky, v nichž je také uveden počet odebraných kusů a aktuální cena za tuto celou položku. Jedna či více položek se pak váží k objednávce, ve které pro nás bude zajímavé především datum jejího vytvoření..



Obrázek 4.1 – struktura databáze

4.1.2 Nalezení dimenzí a faktů

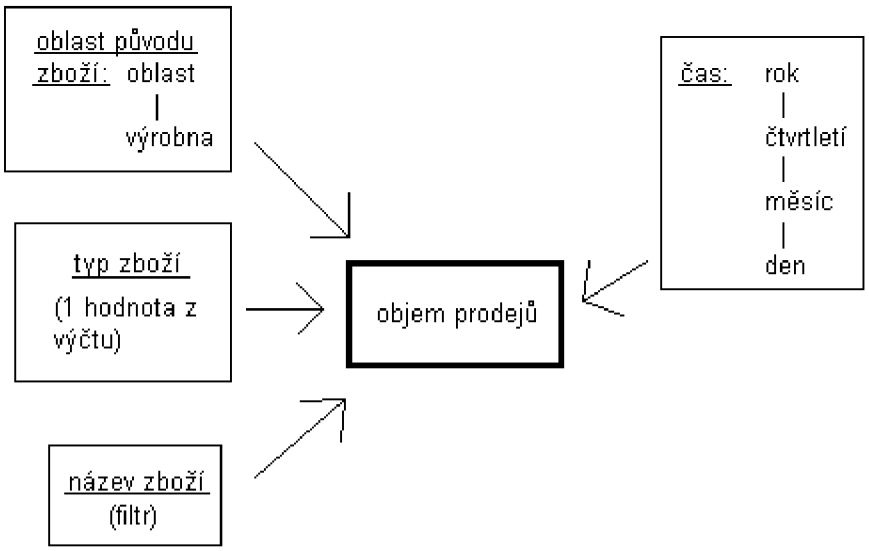
Na výsledky OLAP operací se většinou chce dívat provozovatel nebo manažer, řídící osoba uvnitř firmy či organizace. Takové osobě půjde vždy o to vidět nějaké obchodní výsledky, pohyb peněz, v závislosti na různých faktorech. S výběrem jsem proto začala u faktů, kde se nabízí jako asi nejzajímavější finanční objem **prodejů**.

U výběru dimenzí je nabídka možností obsáhlejší. Určitě nesmí chybět časový rozměr, bez kterého se neobejde snad žádná obchodní analýza. Ten je možné získat u objednávek. Dále se nabízí určovat objemy prodejů v závislosti na druhu zboží, typu zboží, ročníku zboží, oblasti jeho původu, zařazení do podskupin nebo na informacích o zákazníkovi, tedy například když vezmeme jeho adresu, tak jaké bylo cílové místo dodávky zboží (země, město, ..).

Nakonec jsem vybrala čtyři kritéria, kterí by mohla být pro obchodníka zajímavá a která mají určovat objem prodejů:

- doba prodeje,
- odkud zboží pochází - výrobna a oblast
- jakého je typu - pouze základní typ zboží,
- konkrétní druh zboží - druh určuje trojice název, ročník, výčet podskupin.

Hierarchii nad údaji o čase prodeje jsem stanovila na rok – čtvrtletí – měsíc – den (od nejvyšší úrovně abstrakce po nejnižší). Hierarchie nad původem zboží je oblast – výrobna. Typ zboží hierarchii nemá (resp. jen jednoprvkovou) a druh zboží také ne – tam by ale mělo být umožněno filtrovat požadované zboží nějakými výrazy.



Obrázek 4.2 – vybrané dimenze a fakt

4.1.3 Nutné úpravy databáze

Po stanovení toho, jaká data na základě jakých podmínek budu hledat, jsem zkoušela vytvářet a spouštět příslušné SQL dotazy nad databází obchodu. Zde se ovšem projevila transakční povaha databáze a její uspořádání podle normálních forem, oboje ne příliš vhodné pro rozsáhlejší čtecí dotazy, typické pro OLAP systémy.

Při pokusu spojit informace pomocí spojení INNER JOIN z 5 tabulek byly doby trvání dotazu v řádu několika sekund a i když databáze obsahuje data o prodejích v řádu pouze tisíců, výsledek obsahoval téměř 3,7 miliónů položek - z nich samozřejmě mnoho se opakujících. Kolik přesně z nich bylo redundantních se asi již přesně nedovím , protože tentýž dotaz, ale s odstraněním duplicitních hodnot (klíčové slovo DISTINCT), má MySQL databáze ani jednou nebyla schopna dokončit.

Problém způsobovalo spojení nejobsáhlejších tabulek položky a objednávky. Vzhledem k tomu, že z objednávek jsem potřebovala zjistit jen jediné datum, a tato tabulka byla na konci pomyslného řetězu propojování tabulek, rozhodla jsem se přesunout datum z objednávek do položek objednávky a s tabulkou objednávky přestat pracovat úplně. Nový atribut *datum_obj* je zaznamenán i na schématu databáze na obrázku 4.1. Pro mé účely tato úprava postačila, pokud by ale byl objem dat v databázi větší a požadavky na průchodnost dotazů přísnější, určitě by bylo potřeba strukturu přepracovat více (nebo úplně), aby se blížila více charakteristikám datového skladu.

4.2 Realizace stránky s prodeji

Pro zobrazení dat jsem si vybrala dynamickou tabulku, která slouží zároveň jako rozhraní pro zadávání nových dotazů. Stránka pracuje čistě pomocí javascriptu a během celé práce s tabulkou na ní pořád zůstáváme. Jediné dvě akce, které vyvolají znovunačtení celé stránky, je úplný reset tabulky a změna počtu jejích řádků. Jinak jsou vždy nové, aktuálnější části vsazovány do stránky javascriptem, který je získává pomocí asynchronní komunikace ze serveru.

Tabulka má tyto dynamické pvky:

- výměna sloupců dimenzí,
- možnost úplně zavřít sloupec a vyloučit ho z generování podkostky,
- možnost změnit řazení v rámci sloupce (vzestupné a sestupné),
- odeslání požadavku a dat z vyplněných filtrů v záhlaví tabulky pomocí stisknu klávesy enter,
- načítání nabídky výroben dle vybrané oblasti (sloupec pro oblast původu zboží).

Na stránce s prodeji jsou kromě samotné tabulky dále tyto dynamické prvky:

- formulář, kterým lze nastavit počet řádků tabulky a zresetovat tabulku,
- navigace tabulky,
- tlačítko pro zobrazení uživatelské nápovědy k tabulce,
- pokud v tabulce není zobrazen některý ze sloupců, pak se zobrazí formulář pro vložení chybějících sloupců,
- histogram, pokud jej uživatel nechal vygenerovat.

Jakákoliv změna struktury tabulky, tedy veškeré změny sloupců a řazení ve sloupcích, vyvolá automaticky i nové načtení dat. Načtení dat je vyvoláno také odesláním hodnot vyplněných ve

filtrech v záhlaví tabulky, pokud se hodnoty ve filtrech nějak změnily od posledního dotazu. V těchto případech se generují nejen data, ale i html kód různých částí stránky - záhlaví tabulky, navigace tabulky, nadpis tabulky, případně formulář pro vložení chybějících sloupců.

Při pohybu v tabulce pomocí navigace se generují pouze kód navigace a data pro datové řádky tabulky, které se dále zpracovávají javascriptem, který z nich generují jednotlivé řádky.

Při požadavku o vytvoření histogramu se vytváří pouze html kód histogramu podle posledního realizovaného dotazu a aktuální pozice v tabulce.

Poslední ajaxovou akci vyvolá změna výběru oblasti ve sloupci určujícím oblast vína – zjistí se výrobny v dané oblasti a ty se vyplní javascriptem do výběru pro výrobny.

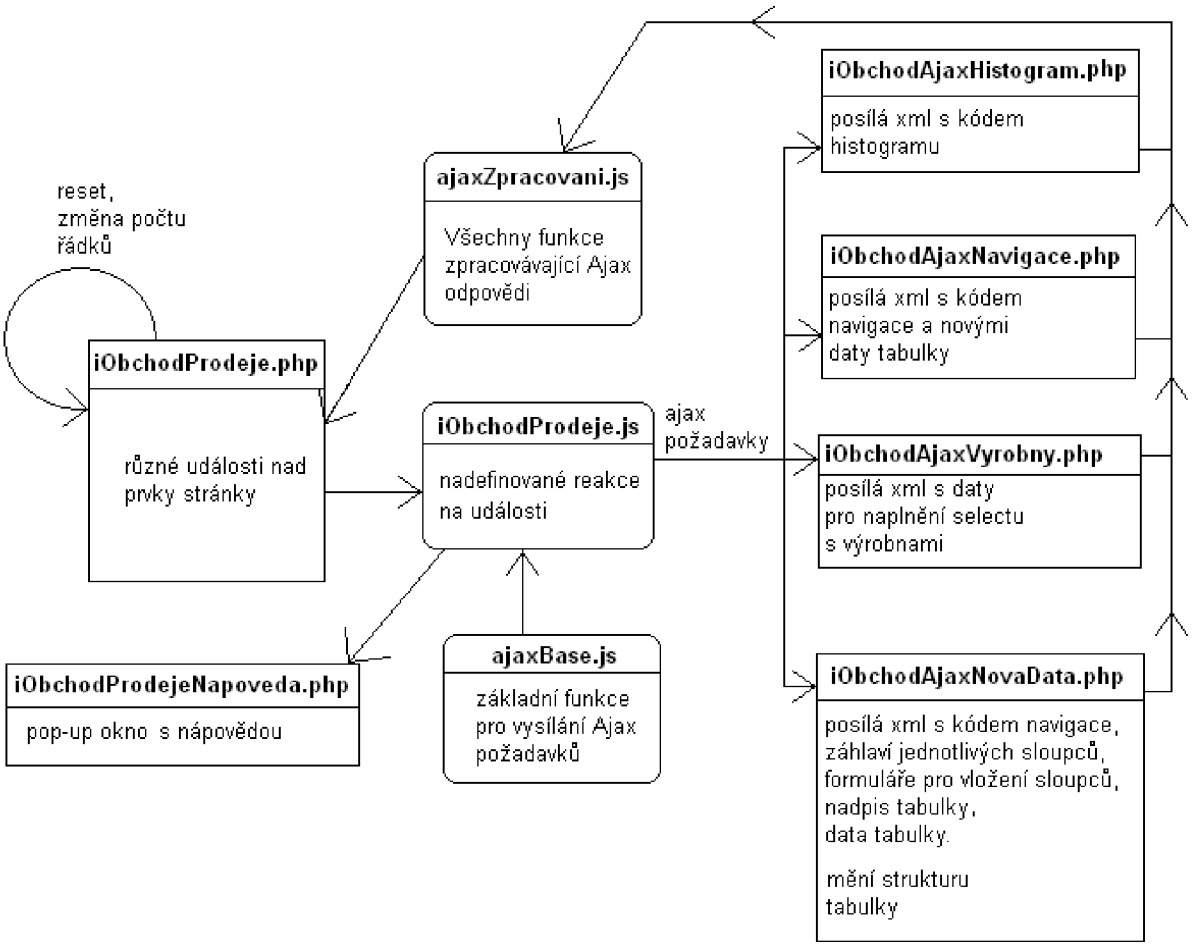
4.2.1 Soubory se skripty a jejich spolupráce

Celý systém je založen na těsné spolupráci javascriptu a PHP. Nicméně, snažila jsem se více funkcionality ponechat v PHP a nezatěžovat klientskou část aplikace nutností generovat rozsáhlé a členité úseky html kódu nebo složitě parsovat a zpracovávat komplikované XML dokumenty.

Javascriptové části jsou co nejvíce odděleny do samostatných souborů (externí javascript), což zaručuje přehlednější a výrazně udržovatelnější výsledný kód. Jediným styčným bodem mezi javascriptem a stránkou je zavolání inicializační funkce při události *onload* v elementu *body* dokumentu.

Všechny PHP skripty sdílejí a uchovávají stav stránky v proměnných prostředí. Javascriptové části toho „ví“ poměrně málo o nějakém stavu stránky a data, která jsou jim posílána, bývají co nejlépe předpřipravena (seřazena, zformátována, ..) PHP skriptem. Javascript se tak nemusí o mnoho starat. PHP skripty, které generují stránku, a ty, které generují XML pro javascript, mají přístup ke stejným funkcím a kontextu. Je vnich mimo jiné zahrnut důležitý knihovní soubor *iObchodProdejeFce.php*, který obsahuje především funkce pro generování různých úseků html kódu stránky, pro tvorbu dotazu a formátování dat.

Obrázek 4.3 ilustruje spolupráci javascriptových a PHP souborů a jejich úlohu v posloupnosti činností (neurčuje vzájemnou vnořenost souborů!). Soubor *iObchodProdeje.php* je výchozím, generuje celou stránku prodejů.



4.2.2 Jak to funguje

Středobodem celého fungování jsou tři konfigurační proměnné, jejichž hodnoty jsou trvale uloženy v PHP superglobální proměnné $_SESSION. Pokud uživatel navštíví stránku poprvé, tyto proměnné se naplní defaultními hodnotami. Všechno jsou to asociativní pole. Kombinací těchto proměnných lze zjistit většinu potřebných informací pro vygenerování celé stránky. Jsou využívány většinou hlavních funkcí jako vytváření SQL dotazu, html kódu tabulky a generování formulářů.

4.2.2.1 Proměnná $sloupce

Jednoduché pole, které obsahuje v klíčích identifikaci sloupců a v hodnotách jejich aktuální pořadí v tabulce. Obsahuje sloupce dimenzí i faktů. Záporná hodnota znamená, že sloupec není v tabulce zobrazen. Z této proměnné si lze jednoduše vytvářet seřazený seznam viditelných a neviditelných sloupců. Například seznam viditelných by podle proměnné $sloupce uvedené ve výpisu 4.1 vypadal takto:

cislo, kdy, co_typ, co_oblast, castka.

```
$sloupce = array( 'cislo'     => 0,
                  'kdy'       => 1,
                  'co_oblast' => 3,
                  'co_typ'    => 2,
                  'co_nazev'  => -1,
                  'castka'    => 4 );
```

výpis 4.1 – proměnná $sloupce

4.2.2.2 Proměnná $sloupceAtributy

Složené pole, které obsahuje v klíčích nejvyšší úrovně identifikaci sloupců a v hodnotách strukturu s různými údaji o sloupcích. Lze se tak dopátrat nadpisu sloupce, zda je dimenzí nebo ne, jak je aktuálně nastaveno jeho řazení a který sloupec které tabulky v databázi mu odpovídá – to se využívá při vytváření SQL dotazu. V kombinaci s proměnnou $sloupce se dá například vytvořit seznam viditelných nebo neviditelných dimenzí.

4.2.2.3 Proměnná $params

Pole, které obsahuje hodnoty vyplněné do záhlaví sloupců při posledním odeslání dotazu. Hodnoty jsou vždy již vyfiltrovány pomocí různých funkcí a zformátovány do požadovaného tvaru (správný tvar platného data, filtru pro název zboží, ..). Při prvním načtení stránky se vyplní defaultními hodnotami.

4.2.2.4 Další důležité proměnné

V kontextu se dále uchovává **poslední vytvořený dotaz**, který se používá ve chvíli, kdy je vytvořen nějaký požadavek na data, ale ani parametry, ani struktura sloupců se nijak nezměnila. To se stává při posunu v tabulce pomocí její navigace nebo při generování histogramu. Vytváření nového dotazu je poměrně složitá funkce a pokud to jde, je dobře se jí vyhnout.

Nutný je také **offset**, který určuje, jak daleko jsme v prohlížení dat v tabulce od počátku. Když se sečte s číslem celkově prvního záznamu, určuje pořadí prvního záznamu na stránce. Při změně dat se vždy nastaví na hodnotu 0.

4.2.2.5 První načtení stránky

Při prvním vygenerování stránky se nejdříve naplňují kofigurační proměnné (přednostně ze session, poté defaultními hodnotami). Poté se generují jednotlivé prvky stránky a první dotaz na základě již naplněných parametrů. Ajax se zde nijak nedostává ke slovu, vše probíhá synchronně. Po vytvoření celé stránky se pomocí javascriptu (událost onload v těle stránky) navěší události na všechny ovládací prvky na stránce a práce se stránkou může začít.

Při vyvolání nějaké události, kterou javascript registruje, se zavolá obslužná funkce. Pokud je potřeba nějaká data ze serveru, resp. z databáze, funkce vytváří většinou jen ajaxový požadavek a

připravuje pro něj parametry, odesílá jej a o víc se nestará. Pokud taková data nepotřebuje, promítne potřebné změny sama ihned do stránky.

Až je požadavek na serveru obsloužen a dostane se zpět ke klientovi, začne se provádět obslužná javascriptová funkce, zaregistrovaná k tomuto požadavku při jeho vytváření. Zpracuje se přijatý XML dokument a upraví podle něj stránka. V této aplikaci se často stává, že jsou přegenerovány elementy stránky, u kterých byly předtím zaregistrovány nějaké události. Nesmí se tedy zapomenout je zaregistrovat u nových elementů znova.

4.2.2.6 Vytváření SQL dotazu

Vytvoření dotazu spočívá ve vygenerování několika jeho částí, které se pak připojí k pevné „kostře" dotazu. Těmito částmi jsou seznam sloupců po klauzuli SELECT, GROUP BY a ORDER BY a řetězec s podmínkami pro část WHERE.

Pomohou nám konfigurační proměnné. Nemá cenu se vůbec zabývat v dotazu sloupci, které nejdou vidět – nebere se na ně žádný ohled. Dále, sloupce s fakty mají již své pevné místo v dotazu v části SELECT, v GROUP BY a ORDER BY se vůbec neobjevují, takže o ty se také starat není třeba. Proto si vytvoříme seřazený seznam viditelných dimenzí, pořadí zde přirozeně hraje důležitou roli.

Procházíme seznam od počátku a vždy zjistíme, o jakou dimenzi se jedná (pomocí identifikace sloupce) a podle toho ji zpracujeme a připojíme k řetězci, který nakonec bude vložen za příslušnou klauzuli. Pod zpracováním se skrývá transformace identifikace sloupce (PHP jméno) na název sloupce v databázi (DB jméno), v některých případech se ještě ke sloupci připojí způsob, jakým bude databází naformátován.

Výsledkem jsou tři řetězce, z nichž jeden se vloží za SELECT, další za GROUP BY a poslední za ORDER BY. Pokud je některý z řetězců pro GROUP BY nebo ORDER BY prázdný, tyto klauzule se k dotazu vůbec nepřipojí.

Část WHERE je naplněna, pokud byly zadány nějaké omezující podmínky na sloupce v jejich záhlaví. Jako první věc při vytváření řetězce pro podmínky se provede ošetření vstupů pro databázový dotaz. Poté jsou zkoumány parametry jednotlivých sloupců. Pokud nemají prázdnou hodnotu (což by znamenalo, že sloupec nejde vidět) nebo hodnotu označující vše (žádná omezující podmínka), je parametr dále zpracován, zformátován pro SQL dotaz a připojen k řetězci pro WHERE. Při zpracování se zde například ověřuje a formátuje datum pro porovnání v databázi.

4.2.2.7 Vytváření kódu tabulky

Jednotlivé buňky tabulky, data i záhlaví, si lze představit jako díly stavebnice. V knihovním souboru *iObchodProdejeFce.php* se nachází funkce pro vytvoření všech těchto jednotlivých dílků, tedy kompletního kódu dané buňky. Jednotný přístup k těmto funkcím z hlavního programu zajišťují dvě další funkce (pro datové buňky a buňky záhlaví), kterým stačí předat identifikaci sloupce, ve kterém

se buňka nachází, a další dodatečné parametry sloužící k vygenerování obsahu (pole s daty pro celý řádek u datové buňky, proměnná $sloupceAtributy pro buňky záhlaví).

Pořadí, v jakém budou díly sestavovány do řádku, máme v proměnné $sloupce. Zjistíme si z ní všechny viditelné sloupce, seřadíme je a poté voláme pro každý z nich stejnou funkci vygenerování buňky, které se vždy předá jiný parametr s identifikací sloupce.

4.2.2.8 Řešení kolizí asynchronních požadavků

Ajaxové požadavky jsou generovány tak, aby co nejméně zatěžovaly obě strany. Objekt XmlHttpRequest (XHR) má vždy jen jednu instanci. Při vytváření požadavku se kontroluje, zda je objekt XHR připraven jej přijmout a následně začít zpracovávat nebo zda je právě zaneprázdněn vyřizováním jiného. Pokud je objekt zaneprázdněn, požadavek se prostě ignoruje. V tomto případě je to nejbezpečnější cesta, jak zaručit správné pořadí odesílání a přijímaní odpovědí a zároveň nezahltit zdroje na serveru. Pokud bychom řadili požadavky do fronty FIFO a svědomitě vyřizovali vše, mohly by vznikat časově dlouhé a nepříjemné fronty, vzhledem k náročnějším a déle trvajícím operacím, které se v této aplikaci mohou na serveru odehrávat. Také to řeší situaci, kdy uživatel z nějakého důvodu (omyl či pokus otestovat odolnost stránky?) vyvolá rychle po sobě několik událostí, obsluhovaných ajaxem.

4.2.3 Jednotlivé funkce

4.2.3.1 Posun sloupců dimenzí

Pohybovat doleva a doprava lze pouze se sloupci obsahujícími dimenze. Děje se tak prostřednictvím ikonek šipek směřujících do stran v záhlaví sloupce. Změna pozice sloupce realizuje OLAP operaci pivot, kdy se změní způsob pohledu na data.

Je zde dvojí kontrola, zda lze sloupec posunout požadovaným směrem. První zajišťuje javascriptová funkce vyvolaná událostí kliknutí na ikonu. Na úvodu kapitoly řešení bylo řečeno, že javascript na rozdíl od PHP skriptů nemá příliš povědomí o struktuře tabulky a stavu sloupců. Na tomto místě se ovšem využívá jakási odlehčená verze konfigurační proměnné *$sloupce* – totiž atribut objektu *cm* (*columnManager*), *cm.sloupce*. Objekt *cm* nám zde slouží jako obal pro proměnné a funkce (atributy a metody) kolem manipulace se sloupci tabulky. Funkce pro posun sloupce jsou rovněž metodami objektu *cm* (*cm.toLeft(), cm.toRight()*).

Informace o sloupcích si *cm* načte sám ze struktury dokumentu. Dále se tyto informace aktualizují pokaždé, když úspěšně dorazí od serveru odpověď na požadavek na nová data.

Pokud je vyhodnoceno, že sloupec lze posunout požadovaným směrem, je vytvořen asynchronní požadavek na nová data. V odesílaných parametrech jsou data sebraná z formulářových prvků v záhlaví tabulky, dále informace o tom, že je požadována akce posun doprava nebo doleva

(*akce=toleft|toright*) a identifikace sloupce, který se má posunovat – získaná pohybem po DOM struktuře dokumentu z atributu id daného sloupce.

PHP skript reagující na tento požadavek je *iObchodAjaxNovaData.php*. Na jeho počátku se připraví všechny potřebné proměnné pro takřka kompletní přegenerování obsahu stránky. Na základě hodnoty parametru *akce* je vybráno zpracování změny tabulky. Ověřuje se znovu, zda lze sloupec posunout požadovaným směrem (tentokrát s pomocí starých známých proměnných *$sloupce* a *$sloupceAtributy*). Pokud to možné je, upraví se adekvátně tomu hodnoty v proměnné *$sloupce*. Pokud ne, je celá akce „degradována" na akci *novadata*, kdy se sloupce nijak nemění a jen se generují data pro nově zadané hodnoty filtrů. Změny v klíčových proměnných se uloží do session.

Po aktualizaci konfiguračních proměnných je možné na jejich základě vygenerovat SQL dotaz a začít tvořit XML odpověď. V XML odpovědi požadavku na nová data je obsažen:

- nadpis tabulky – v něm je obsažena informace, s jakými hodnotami filtrů byl poslední dotaz sestaven, které sloupce byly brány v úvahu a v jakém pořadí,
- html kód formuláře pro přidání sloupce (může být prázdný),
- html kód navigace tabulky, v případě nových dat začínající první stránkou,
- informace o sloupcích – seřazené elementy <*sloupec*>, obsahující hodnotu id atributu sloupce v html kódu, a kód pro obsah záhlaví sloupce,
- data – v elementech <*radek*>, odpovídajících jednomu řádku tabulky, a rozčleněnému na elementy <*bunka*> (seřazenými tak, jak budou ve výsledné tabulce).

Pokud pro zadaný dotaz nebyla nalezena žádná data, neposílá se kód navigace, žádná data, přibude ale element <*nic*> s textovou zprávou o tom, že nebylo nic nalezeno.

Javascriptová funkce po obdržení výsledného XML smaže staré obsahy navigace, formuláře pro přidávání sloupců a celý obsah tabulky. Zatímco do navigace a formuláře prostě vloží html kód získaný z XML, tabulka je kousek po kousku vygenerována znovu, s hojným využitím funkcí DOM rozhraní.

Na konci jsou přegenerovaným prvkům na stránce přiřazeny reakce na události, které zmizely se smazáním starých elementů. Změny ve sloupcích jsou promítnuty do objektu *cm*.

4.2.3.2 Smazání sloupce

Sloupec dimenze je možné z tabulky odstranit kliknutím na ikonku křížku. Toto odstranění dimenze realizuje OLAP operaci slice.

Událost opět obsluhuje metoda objektu *cm* – *cm.colClose()*. Zde není třeba nic ověřovat, funkce posbírá parametry ze záhlaví sloupců a spolu s parametrem *akce* nastaveným na hodnotu *colclose* odešle ajaxový požadavek na nová data. Vše probíhá shodně jako u posunu sloupce, pouze na serveru je vybráno jiné zpracování akce. Ověřuje se, zda je sloupec mezi viditelnými dimenzemi.

Pokud ano, upraví se proměnná *$sloupce* - sloupci je nastavena hodnota -1 a všem ostatním, které byly v tabulce za ním, sníženo pořadí o 1.

4.2.3.3 Přidání sloupce

Sloupec dimenze je možno přidat pomocí formuláře pro přidání sloupců, který se nachází mimo tabulku a objevuje se, pouze pokud některý ze sloupců v tabulce chybí. Je možné zvolit, za který sloupec přesně bude nový přidán.

Operace je inverzní k předchozí operaci smazání sloupce. Požadavek na tuto operaci vysílá při kliknutí na potvrzovací tlačítko formuláře metoda *cm.addColumn()*, která přečte hodnoty formuláře a odešle je spolu s obyvklými parametry ze záhlaví tabulky a udáním akce (*akce=addcolumn*).

PHP skript pro generování nových dat vyhodnotí správnost požadavku – sloupec, který se má vkládat, musí být mezi neviditelnými dimenzemi. Pokud není nalezen sloupec, za která se má vkládat, mezi viditelnými, je nový sloupec vložen automaticky na konec. Je upravena proměnná *$sloupce* a *$params* – do parametrů se přidá defaultní hodnota nového sloupce. Dále jako u posunu sloupců.

4.2.3.4 Změna řazení sloupce

Událost je vyvolána klinkutím na ikonu svislé šipky v záhlaví sloupce a zpracována metodou *cm.dirAsc()* nebo *cm.dirDesc()*.

Skriptu pro nová data se posílají obvyklé proměnné a identifikace akce *dirasc* nebo *dirdesc*. Podle toho se upraví proměnná *$sloupceAtribuy*, ve které je uloženo řazení sloupce. Dále jako u posunu sloupců.

4.2.3.5 Odeslání hodnot filtrů

Děje se prostřednictvím zmáčknutí klávesy enter nad dokumentem. Podle toho, jak jsou vyplněna vstupní pole, jsou realizovány operace dice (výběry a zadání omezujících hodnot) a roll-up a drill-down (měnění úrovně dimenze pomocí select prvku u času, měnění hodnot v select prvku výrobny z obecných na konkrétní a naopak).

Žádné změny se u sloupců nedějí, takže zpracování události v javascriptu se obejde bez *columnManageru* a požadavek na nová data spolu s parametry se serveru posílá rovnou. Hodnota akce je tentokrat *novadata*. Po přípravě potřebných proměnných se na serveru ověří, zda se data od posledního dotazu nějak změnila. Pokud ne, nenastala v tabulce vůbec žádná změna a vygeneruje se ihned pouze krátké a stručné XML s informací, že je vše při starém. V tomto případě pak zpracovávající javascriptová funkce neuskuteční na stránce žádné změny. Pokud jsou vstupní parametry jiné než posledně, přejde se rovnou k vytvoření dotazu a následnému generování XML.

4.2.3.6 Histogram

Událost vzniká kliknutím na ikonu histogramu v záhlaví sloupce. Ikona je viditelná, pouze pokud tabulka obsahuje jeden sloupec s dimenzí. Je obsluhována jednoduchou funkcí *sendHistogramRequest()*, která zasílá požadavek na vygenerování histogramu skriptu *iObchodProdejeAjaxHistogram.php*. Parametry není potřeba zasílat žádné.

Skript generuje jednoduché XML obsahující html kód histogramu, pokud úspěšně ověří, že je zobrazena skutečně jen jedna dimenze. Data pro histogram jsou získána vyvoláním posledního provedeného dotazu a offsetu dat ze session, počet položek histogramu je roven počtu datových řádků tabulky.

Funkce generující histogram přijímá jako jeden ze svých parametrů krok na ose X, zadaný ve stejných jednotkách, jako jsou hodnoty položek histogramu. Je potřeba nějak dynamicky tento krok stanovit. Vycházím přitom z maximální hodnoty položky v grafu. Z ní zjistím, v jakých řádech se čísla pohybují (tisíce, sta tisíce, …). Dále zjistím hodnotu největšího řádu (první číslice zleva). Pro různé intervaly této hodnoty stanovím hodnotu kroku jako násobek o jedna nižšího řádu, než jsem zjistila u maximální hodnoty a stanovené konstanty pro tento interval. Příklad: největší hodnotou je číslo 17850, řád jsou desetitisíce, tedy 10^4. Hodnota největšího řádu je 1. Konstantu pro interval <1,2) mám stanovenu číslo 2. Výsledný krok bude tedy 2*10^3=2000. Z toho vychází dělení osy X na 9 částí, což není ani příliš husté (hrozí ztráta čitelnosti) ani příliš řídké (hrozí ztráta informační hodnoty). Konstanty jsou voleny tak, že počet dělení osy X se zde bude pohybovat vždy mezi 5-10.

Dalšími parametry histogramu jsou jeho rozměry a popisek, zde stanoveny napevno.

S ohledem na to, že histogram je tvořen ostylovaným html kódem, jeho položky nejsou zobrazovány jako svislé sloupce, ale jako řádky.

Javascriptová funkce zpracovávající odpověď s histogramem vloží celý jeho kód do stránky a vygeneruje k němu funkční zavírací tlačítko.

4.2.3.7 Navigace

Taktéž pohyb po stránkách tabulky je implementován čistě pomocí javascriptu. Při kliknutí kdekoliv v navigaci se vyvolá javascriptová reakce. Je zkontrolováno, zda zdrojem události byl prvek typu anchor, pokud ne, zpracovávání končí.

Z atributu 'href' zdrojového prvku události je získána hodnota offsetu, poté se vygeneruje požadavek o novou stránku dat skriptu *iObchodProdejeAjaxNavigace.php.* V požadavku je samozřejmě zaslána i získaná hodnota offsetu.

V tomto případě není potřeba generovat nový dotaz, proto použijeme starý, uložený v session, a na výsledek dotazu aplikujeme jen operaci posunutí kurzoru podle zadaného offsetu. Výsledné XML obsahuje nový kód navigace a data tabulky. Na rozdíl od skriptu pro získání nových dat (*iObchodProdejeAjaxNovaData.php*), zde zasílám data jako atributy elementu *<radek>*, ne jako seřazené potomky tohoto elementu, a zkouším pak odlišnou (mírně složitější) techniku zpracovávání

dat na straně klienta. K tomu ovšem potřebuji informace o seřazení a viditelnosti sloupců, které jsou proto ke XML dokumentu také připojeny.

4.2.3.8 Nabídka výroben

Poslední ajaxovou funkcí je vyplňování nabídky výroben v záhlaví sloupce Zboží – oblast podle zvolené oblasti. Jedná se o typické využití Ajaxu.

Celý proces se spouští událostí změna výběru v nabídce oblastí. Jednoduchá funkce odesílá skriptu *iObchodProdejeAjaxVyrobny.php* údaj o vybrané oblasti. Ten vybere z databáze všechny rozdílné hodnoty výroben pro danou oblast a vrátí je v elementech <*vyrobna*>. Z těch pak zpracující javascriptová funkce vytvoří prvky option a vloží je do select prvku pro výběr výroben.

4.3 Úpravy vyhledávácí stránky

V rámci svého zadání jsem se rozhodla kromě vytvoření přehledu prodejů ještě vylepšit a obohatit vyhledávání zboží. Ve skutečnosti jsem pracovala na těchto úpravách dříve, než na přehledech prodejů, a při této práci jsem se seznamovala detailně se systémem obchodu a se strukturou jeho databáze. Zde jsem si taky začala psát první funkce, které jsem později využila u prodejů (především javascriptové).

Jedná se o stránku *iObchodHledani1.php*. Původní vyhledávací filtr se skládal pouze ze dvou výběrových prvků: pro typ zboží a oblast jeho původu. Rozšířila jsem jej o možnost zadávat ročníky a intervaly ročníků, a vybírat požadované výrobny.

Na této stránce stojí asi nejvíce za zmínku právě formulářový prvek pro **výběr výroben**. Skládá se ze dvou selectů, které umožňují výběr více hodnot (atribut *multiple*). V levém je kompletní seznam výroben (generovaný s ohledem na vybranou oblast, stejně jako u přehledu prodejů), pravý je na počátku prázdný a je určen pro seznam vybraných, požadovaných výroben. Označené výrobny se přesouvají mezi selecty pomocí dvou tlačítek umístěnými mezi nimi. Při přesunu výroben výrobny zmizí z původní nabídky a abecedně se zařadí do nové. Dále je zde možnost přesouvat jednotlivé výrobny dvojitým poklikáním na výrobnu. Bohužel s tímto má problém prohlížeč Opera, ale zdá se, že ve verzi 9.50 (momentálně beta verze, ještě nevydána oficiálně) byl tento problém odstraněn. Uživatelé jiných verzí Opery mají každopádně stále možnost přesouvat výrobny pomocí tlačítek.

Funkce, které realizují načítání seznamu výroben v závislosti na vybrané oblasti, jsou odlišné od těch, které dělají to samé v přehledu prodejů. Je to dáno zcela odlišnou strukturou a funkcionalitou tothoto výběru výroben. Proto má vyhledávání vlastní zpracovávací skript na serveru i vlastní obslužnou funkci, která výsledná data integruje do stránky.

Další výraznou úpravou byla implementace **získávání dat** tabulky pomocí **Ajaxu**. V tomto případě je vše jednodušší než u prodejů, protože sloupce jsou fixní a není potřeba vytvářet žádné struktury, které by se o ně staraly. Výrazně jednodušší na vytváření jsou i SQL dotazy.

Na rozdíl od prodejů je tato stránka téměř plně funkční i bez povoleného javascriptu.- jediné, co zůstane nepřístupné, je právě výběr konkrétních prodejen.

4.4 Další úpravy a zásahy

Mé další zásahy do systému souvisí převážně se vzhledem stránek a jejich transformací z HTML 4.01 do validního formátu XHTML 1.0. Upraveny byly všechny funkce a části generující html kód, výrazně změněna funkce generující html hlavičku. Soubor s obecnými funkcemi potřebnými na různých místech systému jsem rozšířila o své funkce (soubor *iObchodObecne.php*; ve zdrojovém kódu jsou označeny viditelně mým loginem). Přibylo také hodně nových hodnot do dvojjazyčných textů systému (*iObchodTexty.php*). Zde je výčet dalších souborů, které jsem vytvořila, a které nebyly zmíněny nikde výše:

- *histogram.css* – ostylování histogramu
- *napoveda.css* – ostylování nápovědy
- *styl.css* - styl celého internetového obchodu

- *iObchodEven.js* – umožňuje přístup k parametrům událostí nezávislý na prohlížeči, obsahuje reakci na událost přejetí myši nad tlačítky
- *iObchodHledani1.js* – reakce na události na stránce *iObchodHledani1.php*
- *iObchodUtil.js* – různé pomocné javascriptové funkce

- *iObchodHledani1Fce.php* – funkce užívané při vyhledávání zboží
- *iObchodHledaniAjaxHledani.php* – generování XML dokumentu s výsledky hledání
- *iObchodHledaniAjaxVyrobny.php* – generováni XML dokumentu se seznamem výroben

5 Závěr

Hlavním výsledkem mé práce je funkční stránka, která zobrazuje výsledky OLAP operací a umožňuje vytvářet nové, ve které je téměř veškerá funkcionalita implementována pomocí Ajaxu. Stránka je součástí systému internetového obchodu, ve kterém jsem dále vylepšila pomocí Ajaxu stávající vyhledávání zboží.

Při implementaci se mi podařilo odstínit rozdíly mezi prohlížeči, které dosud ještě přetrvávají v interpretování javascriptového kódu.

Užití Ajaxu v případě obyčejného vyhledávání zboží hodnotím jako vhodné. Stránka nabyla mírně na komfortu a atraktivnosti pro uživatele, výsledky požadavků jsou do stránky zasazovány rychle a plynule, bez známého bliknutí obrazovky, které často provází i rychlé překreslování stránky. Ukázkové přínosné využití Ajaxu je vyplňování pole formuláře s výrobnami podle zadané oblasti, které by při realizování synchronní komunikací bylo příliš těžkopádné a nepraktické.

Na stránce s přehledem prodejů bych však byla s používáním Ajaxu v praxi opatrnější. Výsledky většiny operací v tabulce (posuny sloupců, přidávání a mizení sloupců, změna řazení, odeslání filtrů) vracejí poměrně rozsáhlá data. Velikost struktury XHTML stránky je díky jejímu formátu a přísnému oddělení vzhledu do externích stylových dokumentů velice malá a zanedbatelná v poměru k těmto datům. Jednou z výhod Ajaxu je možnost aktualizovat některé části stárnky, aniž by bylo nutné načítat a přenášet jiné, tedy šetření síťových prostředků, zvyšování rychlosti zobrazení nových informací na stránce a ušetření uživatele od neustálého pozorování znovunačítající se stránky po sebemenší akci.

V tomto případě jsou ale některé klady Ajaxu úplně setřeny. Objem asynchronně přenášených dat do stránky není o mnoho menší, než velikost celé stránky. Přegenerovávají se všechny složitější prvky stránky, úplně znovu se vykresluje tabulka.

I když jsem se snažila mít javascriptovou logiku na straně klienta malou, abych zachovala zadání práce, musela jsem přeci jen data přenášet v nějakém strukturovaném XML dokumentu, zpracovávaném dále pomocí DOM funkcí , ne jen jako nestrukturovaný kus textu, který se někam do stránky prostě vloží. Zpracování takového dokumentu a vygenerování nových elementů na jeho základě představuje přeci jen jakousi režii na straně klienta. U pomalejšího prohlížeče IE 6 se to projevuje občas tím, že na místech formulářových vstupů problikává při jejich překreslení pozadí stránky. V případě IE 6 jsme tedy vyměnili jedno probliknutí obrazovky při znovunačtení stránky za jiné.

Dále, OLAP operace jsou typické tím, že pracují nad velkými objemy dat. I když jsou OLAP databáze optimalizovány na rychlou odezvu dotazů a budou většinou pravděpodobně běžet na velice výkonných strojích, pokud odezva bude delší než dejme tomu 1.5-2 sekundy, Ajax bude mít problém – díky jeho charakteristice, že se nenačítá znovu celá stránka, se zdají všechny prodlevy mezi

začátkem a koncem akce delší, než při vyvolání synchronního požadavku s překreslením. I když to tedy nebude pravda, Ajax by se mohl zdát uživateli „línější“. Navíc je potřeba zakomponovat do systému to, aby při delším vyřizování požadavku, který uživatel vyvolal (zmáčknutí tlačítka ..), bylo na stránce nějaké viditelné znamení, že se něco děje.

Co chci říci je, že v případě tabulky prodejů nám při mnohých operacích Ajax nic neušetří nebo nezpohodlní. Sice ani naopak neuškodí, ale proč zavádět a implementovat něco, co nemá užitek a přináší ještě některé negativní rysy typu „nejde použít tlačítko zpět k předposlední zobrazené stránce“.

Dle mého se Ajax nejlépe hodí pro aktualizace opravdu *malých* bloků informací na stránce (malých vzhledem k objemu celé stránky).

V případě stránky s přehledem prodejů bych ho jednoznačně ponechala u již zmíněného generování výběru výroben v závislosti na vybrané oblasti, u vkládání histogramu do stránky a možná i u pohybu ve stránkách tabulky pomocí navigace – hodně by záleželo na průměrné odezvě konkrétní databáze a tím i webového serveru.

Ajax ponechávám zatím formulářům, různým políčkům na stránkách, které toho vědí překvapivě hodně (našeptávače, nápovědy aspol.), vkládání a obnově menších prvků stránky. To je jeho zatím nejvhodnější běžné využití.

Literatura

[1] Asleson, R., Schutta, N. T.: *AJAX - vytváříme vysoce interaktivní aplikace*. Brno, Computer Press, a. s., 2006.

[2] Darie, C. aj.: *AJAX a PHP - tvoříme interaktivní webové aplikace PROFESIONÁLNĚ*. Brno, ZONER Software, s.r.o., 2006.

[3] Harold, E. R., Means, W. S.: *XML v kostce*. Praha, Computer Press, 2002.

[4] W3C: *Document Object Model (DOM)*. Dokument dostupný na URL http://www.w3.org/DOM/ (leden 2007).

[5] Champeon, S.: How did we get here?. 2001. Dokument dostupný na URL http://www.oreillynet.com/pub/a/javascript/2001/04/06/js_history.html (leden 2007).

[6] Satrapa, P.: HTML v příkladech - 8 Common Gateway Interface. *Softwarové noviny*. 1997, 3. . Dokument dostupný na URL http://www.kai.vslib.cz/~satrapa/docs/wwwprikl/html8.html (leden 2007).

[7] Altmann, U.: *Einführung in das Document Object Model (DOM)*. Dokument dostupný na URL http://www.med.uni-giessen.de/akkk/xml/DOM/domeinfuehrung.html (leden 2007).

[8] Kosek, J.: CGI-Skripty. Dokument dostupný na URL http://www.kosek.cz/clanky/iweb/06.html (leden 2007).

[9] Pichlík, R.: *Poznáváme DOM III. - moduly a úrovně*. Dokument dostupný na URL http://www.sweb.cz/pichlik/archive/2004_01_11_archive.html (leden 2007).

[10] Java Servlet. In *Wikipedia, The Free Encyclopedia*. Dokument dostupný na URL http://en.wikipedia.org/w/index.php?title=Java_Servlet&oldid=96076447 (29. 12. 2006).

[11] Adobe Flash. In *Wikipedia, The Free Encyclopedia*. Dokument dostupný na URL http://en.wikipedia.org/w/index.php?title=Adobe_Flash&oldid=97684737 (30. 12. 2006).

[12] Ajax (programming). In *Wikipedia, The Free Encyclopedi*. Dokument dostupný na URL http://en.wikipedia.org/w/index.php?title=Ajax_%28programming%29&oldid=97900309 (1. 1. 2007).

[13] Stryka, L., Hruška, T., Máčel, M.: Algebra multidimenzionální datové kostky a její grafické projevy v prostředí OLAP. In *DATAKON 2007*. Popelínský, L., Výborný, O. (eds.) Brno, 2007. s. 187-197. Dokument dostupný na URL: http://www.datakon.cz/2007/sbornik_datakon07.pdf (leden 2008).

[14] Pivot table. In Wikipedia, The Free Encyclopedia. Dokument dostupný na URL http://en.wikipedia.org/wiki/Pivot_table (leden 2008).

Použité zkratky:

AJAX - Asynchronní JavaScript a XML

ARPANET - Advanced Research Projects Agency Network

ASP - Active Server Pages

CGI - Common Gateway Interface

CSS - Cascading Style Sheets

DARPA - Defense Advanced Research Projects Agency

DHTML - Dynamic HyperText Markup Language

DNS - Domain Name Server

DOM - Document Object Model

DTD - Document Type Definition

ECMA - European Computer Manufacturers Association

HTML - HyperText Markup Language

HTTP - HyperText Transfer Protocol

IP - Internet Protocol

JSP - Java Server Pages

JVM - Java Virtual Machine

NSFNet - National Science Foundation Network

OLAP – on-line analytical processing

PHP - PHP: Hypertext Preprocessor

SAX - Simple API for XML

SGML - Standard Generalized Markup Language

SQL – Structured Query Language

TCP - Transmission Control Protocol

XHTML - Extensible HyperText Markup Language

XLL - XML Linking Language

XML - Extensible Markup Language

XSL - Extensible Stylesheet Language

XSL-FO - Extensible Stylesheet Language - Formatting Objects

XSLT - Extensible Stylesheet Language Transformations

Seznam příloh

Příloha 1 – Přehled prodejů

Jste přihlášen jako admin. Změnit detaily Odhlášení

Na domovskou stránku

Řádků tabulky: 20 nastavit resetovat tabulku nápověda k tabulce

Přidat sloupec: Zboží - typ po: Číslo OK

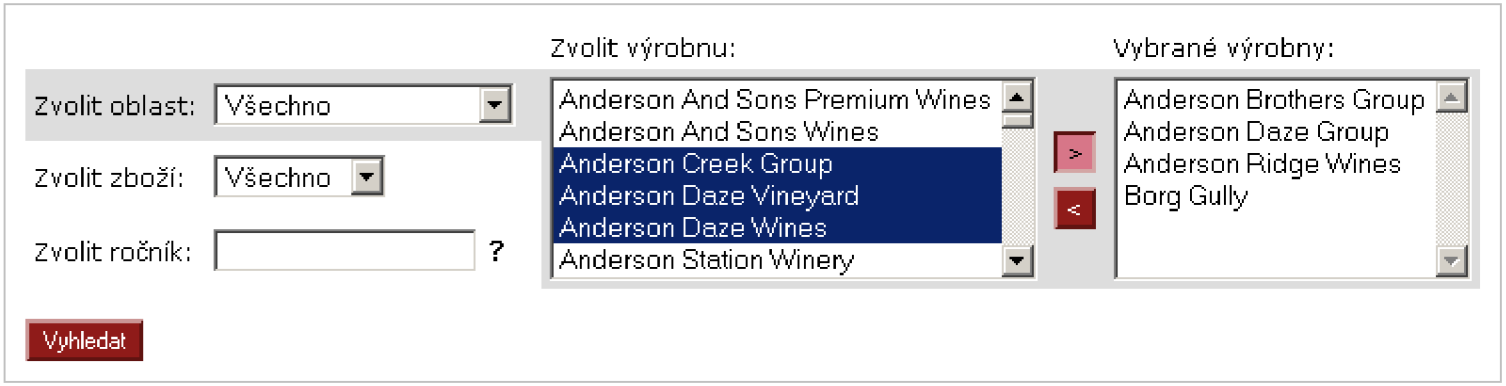
Číslo
Období
Zboží - oblast
Zboží - název

[1, 2]

všechno (měsíc); zboží - oblast: Barossa Valley, všechny výrobny; zboží - název: k*

Číslo	Období	Zboží - oblast	Zboží - název	Částka (Kč)
	období: úroveň: měsíc	oblast: Barossa Valley výrobna: Všechno	název: k*	
1	06.2000	Barossa Valley, Borg's	Kinsala, 1971 Semillon Sauvignon Blanc	21.84
2	06.2000	Barossa Valley, De Morton Hill Group	Krennan, 1986 Sparkling White	39.04
3	06.2000	Barossa Valley, Macdonald Brook Vineyard	Krennan, 1995 Port	84.81
4	06.2000	Barossa Valley, Rogerson Gully	Krennan, 1993 Grenache	49.32
5	05.2000	Barossa Valley, Bell Daze Premium Wines	Kinsala, 1989 Grenache	331.24
6	05.2000	Barossa Valley, Borg's	Kinsala, 1971 Semillon Sauvignon Blanc	152.88
7	05.2000	Barossa Valley, De Morton Hill Group	Krennan, 1986 Sparkling White	58.56
8	05.2000	Barossa Valley, Macdonald Brook Vineyard	Krennan, 1995 Port	30.84
9	05.2000	Barossa Valley, Macdonald's Group	Kinsala, 1980 Chardonnay	142.23
10	04.2000	Barossa Valley, Bell Daze Premium Wines	Kinsala, 1989 Grenache	152.88
11	04.2000	Barossa Valley, Gosk Winery	Keisling, 1982 Sparkling White	258.94
12	04.2000	Barossa Valley, Lord Daze	Krennan, 1970 Sauvignon Blanc	155.32
13	04.2000	Barossa Valley, Macdonald's Group	Kinsala, 1980 Chardonnay	38.79
14	04.2000	Barossa Valley, Rogerson Gully	Krennan, 1993 Grenache	24.66
15	03.2000	Barossa Valley, Bell Daze Premium Wines	Kinsala, 1989 Grenache	356.72
16	03.2000	Barossa Valley, Borg's	Kinsala, 1971 Semillon Sauvignon Blanc	120.12

Příloha 2 - Hledání



[1, 2]

Všechno ze všech oblastí, všechny ročníky				
#	**rok**	**výrobna**	**název, typ**	**cena**
1	1986	Anderson Brothers Group	Marzalla, Cabernet Sauvignon	13.58 Kč (162.96 Kč za tucet)
2	1997	Anderson Brothers Group	Mellili, Port	6.30 Kč (75.60 Kč za tucet)
3	1983	Anderson Brothers Group	Mettaxus, Merlot	5.78 Kč (69.36 Kč za tucet)
4	1979	Anderson Brothers Group	Pattendon, Sparkling Red	8.79 Kč (105.48 Kč za tucet)
5	1975	Anderson Daze Group	Barneshaw, Champagne	26.52 Kč (318.24 Kč za tucet)
6	1970	Anderson Daze Group	Kinsala, Grenache	29.89 Kč (358.68 Kč za tucet)
7	1996	Anderson Daze Group	Mettaxus, Semillon	25.68 Kč (308.16 Kč za tucet)
8	1979	Anderson Daze Group	Morfooney, Pinot Noir	27.30 Kč (327.60 Kč za tucet)
9	1999	Anderson Ridge Wines	Galti, Muscat	14.63 Kč (175.56 Kč za tucet)
10	1975	Anderson Ridge Wines	Chester, Sauvignon Blanc	19.50 Kč (234.00 Kč za tucet)
11	1979	Anderson Ridge Wines	Rosenthal, Shiraz Cabernet	28.53 Kč (342.36 Kč za tucet)
12	1996	Anderson Ridge Wines	Sorrenti, Sparkling Red	25.38 Kč (304.56 Kč za tucet)

1-12 z nalezených 15 druhů zboží splňujících Vaše kritéria