TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: N2612 – Elektrotechnika a informatika

Studijní obor: 1802T007 - Informační technologie

TVORBA RICH INTERNET APPLICATION PRO SPRÁVU ZÁKAZNICKÝCH REFERENCÍ

CREATION OF RICH INTERNET APPLICATION FOR MANAGING CUSTOMER REFERENCES

Diplomová práce

Autor: Bc. Jitka Dařbujanová

Vedoucí práce: RNDr. Klára Císařová Ph.D.

V Liberci 1. 1. 2008

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Ústav mechatroniky a technické informatiky Akademický rok: 2007/08

ZADÁNÍ DIPLOMOVÉ PRÁCE

Jméno a příjmení: **Jitka Dařbujanová**

studijní program: N 2612 - Elektrotechnika a informatika

obor: 1802T007 - Informační technologie

Vedoucí ústavu Vám ve smyslu zákona o vysokých školách č.111/1998 Sb. určuje tuto diplomovou práci:

Název tématu: **Tvorba Rich Internet Application pro správu zákaznických referencí**

Zásady pro vypracování:

1. Analyzuje současné možnosti uživatelského komfortu u webových aplikací
2. Charakterizujte RIA aplikace a prostudujte současné možnosti vývoje takto označených aplikací
3. Vyhodnoťte jednotlivé postupy, se kterými jste se seznámila a pokuste se o klasifikaci přístupů vztaženou k zaměření zamýšlené webové aplikace – např. podniková databázová aplikace, či graficky orientovaná reklamní prezentace. Diskutujte míru oddělení aplikační a prezentační logiky v jednotlivých přístupech.
4. Své závěry a zkušenosti prakticky prověřte realizací databázové webové aplikace s třívrstvou architekturou.

PROHLÁŠENÍ

Byl(a) jsem seznámen(a) s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé DP a prohlašuji, že s o u h l a s í m s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom(a) toho, že užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Diplomovou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum: 16.5. 2008

Podpis : Dalibor Janoš

Poděkování

Na tomto místě mám příležitost poděkovat těm, kteří jakýmkoliv způsobem pozitivně přispěli do mého života a mnohdy, třeba nevědomky do této práce. Dušanovi, za jeho bezmeznou trpělivost, podporu a uživatelské podněty. Dále svému otci Bohumilu Daňbujanovi, jenž mě přivedl k zaměstnání programátora a nejvíce celé své rodině za celoživotní podporu a lásku.

Za množství podnětů, materiálů, rad a důvěru bych na tomto místě chtěla poděkovat také své vedoucí práce RNDr. Kláře Císařové Ph.D..

ABSTRAKT

Cílem teoretické části práce je podat ucelený přehled nejaktuálnějších serverových a klientských technologií určených pro tvorbu RIA aplikací, srovnat je v oblastech uživatelského komfortu, rychlosti zpracování dat, uživatelské podpory a dostupných IDE, jejich potenciálu na významnější prosazení na trhu a čtenáři představit cesty pro výběr RIA technologie pro jeho specifické řešení. Dále čtenářovi doporučuje používání několika významných návrhových vzorů.

Získané informace pak práce aplikuje do praxe, kdy na čistě demonstračním tématu "správa referencí" ukazuje rozšířenou CRUD aplikaci ve dvou RIA technologiích – Asynchronní javascript (AJAX) a Flex, vše s ohledem na uživatelský komfort. Pro aplikaci tak existuje jeden front-end a dva back-endy.

Serverová část je založena na PHP5 s použitím nejnovějšího Zend framework 1.5 s efektivním použitím novinky Zend_Form. Pro komunikaci s Flex klientskou aplikací je dále do serverové části implementován aplikační open-source Flex server AMFPHP, který je inovativním způsobem začleněn přímo do struktury framework Zend.

Klientská část porovnává Ajaxový komponentový framework DOJO a technologii Flex. Dalším přínosem práce v této oblasti je automatizované vytváření Dojo forms s klientskou validací za pomocí serverově generovaných Zend_Form se serverovou validací a implementace těchto formulářů do modálních oken známých z desktopových aplikací. Technologie Flex využívá pro komunikaci s Flex serverem binární protokol AMF3.

Klíčová slova: software, internet, vývoj, RIA, AJAX, DOJO, Zend framework, Remoting, Flex, WebML, MVC

Abstrakt

The main goal of theoretical part of this work is to give global summary of most actual server and client technologies developed for creation of RIA, compare them in user comfort manner, speed of data processing, user support and available IDE, their potential on getting to a global market and to introduce ways of choosing RIA technologies to reader to apply them on his specific solution. Also it is recommending the use of some important design patterns.

Work applies this theory to praxis then and shows wider CRUD application "managing of references" in two RIA technologies – asynchronous javascript (AJAX) and Flex, all preserved to make user comfort. For application there is one front-end and two back ends at all.

Server side is based on PHP5 with use of newest Zend framework 1.5 and very effectively uses one news: Zend_Form library. For communication with Flex application there is implemented application open-source Flex server AMFPHP on the server side, which is by innovating way included nearly to structure of framework Zend.

Client side compares Ajax component framework DOJO and Flex technology. Of this work in this area is automated creation of Dojo forms with the client validation with the help of server-generated Zend_Form with server validation and implementation of these forms to modal windows well-known from the desktop applications. Flex technology uses binary protocol AMF3 for communication with Flex.

Keywords: software, internet, development, RIA, AJAX, DOJO, Zend framework, Remoting, Flex, WebML, MVC

OBSAH

SEZNAM ZKRATEK

AIR	Adobe Integrated Runtime - integrované běhové prostředí Adobe
AJAX	Asynchronnous Javascript and XML, označení principu tvorby RIA
AMF	Action Message Format
AOP	Aspektově orientované programování
API	Application programming interface, rozhraní pro programování aplikací
ASP.NET	Active Server Pages.NET
AVM	ActionScript Virtual Machine
AWA	Architektury webových aplikací
BAML	Binary XAML
BLL	Business Logic Layer
CASE	Computer-aided software engineering; programové nástroje pro vývoj software
CLR	Common Language Runtime
CORBA	Common Object Request Broker Architecture
CRUD	Create, Read, Update, Delete – zkratka pro čtyři základní operace nad daty
CSRF	Cross Site Request Forgery, typ útoku na síti
CSS	Cascade styesheets, tabulky kaskádových stylů
DAL	Data Access Layer, vrstva pro přístup k datům
DAO	Data access Object, objekt pro přístup k datům
DCOM	Distributed komponent object model
DLR	Dynamic Language Runtime
DOM	Document object model, objektový model dokumentu
DOM	Distributted Object Middleware
DTD	Document Type definition
ECMA	European Computer Manufacturers Association
EJB	Enterprise Java Beans
FPS	Frames per seconds, rámce za sekudu
GUI	Graphical user interface, grafické uživatelské prostředí
HTML	HyperText Markup Language, hypertextový značkovací jazyk
HTTP	HyperText Transfer protokol
IDE	Integrated Development Envirometn, integrované vývojářské prostředí
IE	Internet Explorer
J2EE	Java Enterprise Edition

JDBC	Java Database connectivity
JDO	Java diabase obejcts
JIT	Just in time compilation
JMS	Java messaging systém
JNPL	Java Network Launch Protocol
JRE	Java Runtime Environment
JSON	JavaScript object Notation
LINQ	Language Integrated Query
MOM	Message oriented middleware
MVC	Model view controller
MXML	Magic eXtensible Markup License
OOP	Objěktově orientované programování
OOHDM	Object Oriented Hypermedia Design Method
ORM	Object relational mapping
P2P	Peer to peer
PHP	Personal Home Page, později Hypertext Preprocesor
REST	Representational state transfer
RFC	Request for Comments
RIA	Rich Internet Application
RoR	Ruby on Rails
RPC	Remote procedure call
RSS	Really Simple Syndication, dříve RDF Site Summary
SGML	Standard Generalized Markup Language
SMTP	Simple Mail Transfer Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SOM	Service oriented middleware
SWF	Shockwave flash
UDDI	Universal Description, Discovery and Integration
UI	User interface, uživatelské rozhraní
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

UTF-8	Unicode Transformation Format 8 bit
VO	Value object
VOM	Virtual Sharp memory
WebML	Web Modeling Language
WOA	Web oriented architecture
WPF	Windows presentation foundation
WSDL	Web service description langure
XAML	eXtensible Avalon Markup Language
XHR	XMLHttpRequest
XHTML	eXtensible HyperText Markup Language
XML	eXtensible Markup Language
XSL	eXtensible Stylesheet Language

ÚVOD

Cílem této práce je obeznámit čtenáře s dostupnými prostředky pro vývoj uživatelsky bohatých webových aplikací, takzvaných Rich Internet Applications. V úvodní analýze – teoretické části práce se snažím zhodnotit všechny hlavní technologie, jejich výhody a nevýhody. Aby byl čtenář uveden do jednotlivých použitých technik, jsou v teoretické části rozebrány všechny použité termíny a technologie. Jelikož je v práci použito například techniky RPC (remote procedure calling), vysvětluje teoretická část jednotlivé architektury webových aplikací, kde RPC je podkapitolou architektury DOM (distributed object middleware), což je nejčastější případ jeho použití. Stejně tak ale může být použit v rámci jiné architektury např. SOA, kterou používá tato práce. Veškeré důležité termíny jsou vysvětleny v teoretické části, a ta též odpovídá na otázku, jakou technologii zvolit pro konkrétní řešení určité RIA aplikace, nebo obecně webové aplikace. K nalezení odpovědi je použito jak slovního popisu a srovnání, tak též výkonnostních testů, které najdete v závěru teoretické části.

Praktická část práce ukazuje vlastně čtyři vytvořené aplikace, kde tři z nich běží nad databází MySQL a čtvrtá nad MS SQL 2005. Aplikace běžící nad MySQL jsou: „Frontend" – webové stránky, které ukazují výsledky práce v backendech, „Backend1" – ajaxové administrační rozhraní, vytvořené jako „Single-page application" s použitím frameworku Dojo, „Backend2 " – administrační rozhraní nad technologií Flex s použitím binárního protokolu AMF3 a voláním distribuovaných objektů pomocí RPC. Práce přinesla též vlastní inovace v této oblasti, které jsou zmíněny v závěru práce.

TEORETICKÁ ČÁST

1. HTTP A HTTPS PROTOKOL OBECNĚ

HyperText Transfer protocol je internetový protokol, který byl zpočátku navržen pro výměnu HTML dokumentů. HTTP server nejobvykleji očekává požadavky na portu TCP 80, port pro zaslání odpovědi pak sám určí. Komunikace je založená na principu nešifrovaný dotaz-odpověď, přičemž veškerá aktivita musí být zahájena klientem. Klient navazuje spojení se serverem po TCP protokolu metodami GET, POST, PUT, DELETE, HEAD, TRACE, OPTIONS, CONNECT. Nejčastější jsou GET a POST, u složitějších aplikací, zvláště webových služeb, se setkáme i s PUT, DELETE, HEAD. HTTP je bezestavový protokol, server nezná pojem „session s klientem X". Díky MIME rozšíření je možno přenášet definované typy souborů.

Pro detekci adresy zdroje využívá jednotný lokátor prostředků URL (Uniform resource locator).

Aktuální verzí protokolu HTTP je 1.1 (1) a přidává možnost perzistentního připojení, takže se kanál okamžitě neukončuje a klient tak může žádat o další data (HTML, obrázky,...). Spojení je ukončeno po určitém časovém intervalu nečinnosti nebo klientem.

Klient zasílá požadavek s určením metody (POST, GET ...), URL a portu, dále zasílá hlavičky dotazu, z nichž nejvýznamnější jsou:

- Accept – definuje, jaká data je schopen klient zpracovat, server mu pak data přizpůsobí. Najdeme zde MIME typy, znakovou sadu, kódování přenášených dat a jazyk dokumentu.
- Connection – v http 1.1 umožňuje definovat vynucení ukončení připojení po zpracování dotazu
- Referer – hlavička, jež využívají statistické společnosti – sděluje URI stránky, ze které byl odkaz vygenerován, bývá zneužívána pro referer spam a referer spoofing
- User-Agent – definuje klienta, jenž dotaz vyvolal

Server odpovídá odesláním verze protokolu, kódu stavu, textové zprávy, hlaviček a těla zprávy:

```
HTTP/1.1 200 OK
Date: Mon, 01 Feb 2008 14:18:22 GMT
Server: Apache/2.0.59 (Unix)  (Debian/Linux)
Content-Type: text/html

<HTML>
...
</HTML>
```

Kód stavu je rozdělen do pěti skupin:

- 1xx – informační zprávy

- 2xx – úspěch – zpráva obdržena a server jí porozuměl
- 3xx – přesměrování, např. kód 301 – přesměrováno trvale, 303 - see other – shlédni na jiném URL metodou GET (http 1.1)
- 4xx- chyba klienta – např. 403 – Forbidden - pokus o přístup k existujícímu zdroji bez dostatečných práv, 404 not found – pokus o přístup k neexistujícímu zdroji
- 5xx –chyba serveru, např. 500 – interní chyba serveru

Nejdůležitějšími hlavičkami jsou:

- Cache-control – server říká klientovi, zda se má obsah kešovat (no-cache = zákaz)
- Content-encoding – kódování dat – možnost komprese např. pomocí gzip
- Content-Language – jazyk dat
- Content-Length – délka v bytech
- Content-Type – MIME typ obsahu (`Content-Type: text/html; charset=utf-8`)
- Location – určení URL pro přesměrování

HTTPS je URI schéma pro šifrovanou komunikaci přes aplikační protokol HTTP přes vrstu SSL (Secure Socket Layer) či TLS. Komunikace implicitně probíhá na chráněném portu 443. Obě vrstvy komunikace mají své vlastní ověřování, které tak lze kombinovat:

	HTTP		
HTTPS		**Anonymní HTTP**	**Jméno/heslo (neanonymní HTTP)**
	Jen certifikát serveru (Anonymní HTTPS)	Anonymní uživatel si je jist, s jakým serverem komunikuje. Hodí se např. pro distribuci software, ceníků, kurzovních lístků apod.	Jméno a heslo je přenášeno bezpečným kanálem. Hodí se pro většinu komerčních aplikací, které používají neanonymní klienty a nechtějí klienty obtěžovat vydáváním certifikátů
	I certifikát klienta (Neanonymní HTTPS)	Hodí se pro nejnáročnější aplikace (např. v oblasti domácího bankovnictví). Snad jediným nedostatkem je, že SSL/TLS neprovádí elektronický podpis.	Nesmyslná kombinace, protože bezpečnost na úrovni certifikátů se již použitím hesel nezvýší.

SSL je vrstva mezi vrstvou HTTP a TCP/IP, která zabezpečuje autentizaci komunikujících stran a asymetrické šifrování této komunikace. TCL (transport layer security) 1.0 má být nástupcem SSL 3.0, zatím se však příliš významně neliší.

Pro funkci HTTPS tedy server potřebuje vlastnit certifikát podepsaný certifikační autoritou (ta zaručí autentičnost správce serveru ve vztahu k jeho doméně). Podpisové vzory těchto autorit jsou zabudovány ve webových prohlížečích (nebo je možné je zabudovat).

Podpis certifikátu autoritou je placený, přičemž důvěryhodnost samotných certifikačních autorit je samostatnou otázkou. I proto mají možnost správci serveru podepsat si certifikát sami.

Pak je však uživateli zobrazena hláška o nedůvěryhodném certifikátu, kterou musí pozitivně potvrdit, aby jej prohlížeč na webové stránky pustil. Také podepsaný certifikát zobrazuje podobnou hlášku. Je to proto, aby si uživatel sám ověřil, zda je v certifikátu skutečně uvedena ta firma, které chce poskytnout svá data. Mnoho uživatelů však tuto kontrolu neprovádí, hlášku akceptují s mylným pocitem, že jsou na zabezpečeném serveru a nehrozí jim žádné riziko.

Výhodou použití HTTPS pak zůstává spíše jeho druhá stránka, a to šifrování komunikace vrstvou SSL či TLS.

2. Stavovost webových aplikací, metody Get a post a PRG pattern

2.1. Stav webové aplikace

Protokol HTTP je bezestavový protokol, což způsobuje značné problémy při návrhu webových aplikací, jelikož je programátor nucen udržovat stav aplikace svépomocí. Mnohé programovací jazyky se pokouší programátorovi usnadnit práci, např. ASP.NET s technikou ViewState apod. Všechny ovšem mohou využívat pouze základních existujících principů, na kterých jsou postaveny. Popisuji je v následujících kapitolách.

Významnou překážkou při sledování stavu webové aplikace představuje pro programátora samotný tenký klient- webový prohlížeč. Ten totiž zpravidla obsahuje ovládací prvky "Obnovit" a "Zpět", které znesnadňují kontrolovaný chod webové aplikace a tím pádem i jejího stavu. Vývojáři webových prohlížečů si tohoto jsou vědomi, a proto vznikají nové progresivní projekty typu Adobe AIR, které nechávají navigaci v aplikaci (přechod mezi stavy aplikace) čistě na programátorovi.

2.2. Typy web aplikací z hlediska udržení stavu

Webové aplikace se dají **z hlediska udržení stavu** rozčlenit na 3 skupiny:

1) **Webové aplikace bezestavové**

   Představitelem budiž statické (x)HTML stránky nebo malé funkční bloky typu „aktuální přesný čas", které nemají nutnou návaznost na předchozí kroky (stavy).

2) **Stav je dán jen klientovou pozicí na určité stránce/pohledu/stavu v aplikaci**

   Jedná se například o webové aplikace bez nutnosti autentizace (např. *program kina > přechod na program konkrétního dne > přechod na detail filmu*), kdy je zobrazená stránka závislá na předchozím kroku, ovšem informace o předchozím stavu je pro aplikaci ztracena. K dispozici ji má pouze samotný tenký klient (tlačítko "Zpět"). Stav může být dán také aktuálním pohledem v Flash animaci, či State v Flex aplikaci.

3) **Stav je dán mnoha proměnnými**

   Jedná se o stav z bodu 2 doplněný o další přenášené informace. Typicky po autorizaci uživatele (cookies, sessions, certifikáty, ...), vícestránkové formuláře s hidden elementy atd.

2.3. Úloha klienta

Udržení stavu aplikace záleží pouze a jenom na klientovi, server pouze reaguje na události. Klient má následující možnosti:

1) Stav je dán vygenerovanými HTML odkazy na straně klienta, na které může uživatel z aktuálního stavu přejít.
2) Stav může být uložen do hidden prvku formuláře.
3) Stav může být uložen do HTTP cookie, má-li je uživatel povoleny. Server jej poprvé posílá hlavičkou set-cookie a klient je poté s každým dotazem na server zasílá zpět (hlavičkou Cookie).

2.4. Úloha serveru

Zatímco veškeré informace o stavu aplikace udržuje klient, server má v některých případech umožnit jednoznačnou identifikaci klienta. Při prvním požadavku od klienta HTTP server vygeneruje klíč, kterým se poté v každém kroku klient identifikuje. Udržení tohoto klíče je pak opět záležitostí klienta. Klíč lze získat vlastními silami nebo připravenými technikami.

1) Http autentizace

Je standardní součástí protokolu HTTP. Nelze změnit podobu přihlašovacího okna, obtížně se řeší odhlášení a automatické odhlášení po určité době. HTTP autentizace bývá implementována na úrovni webového serveru a hesla jsou přenášena v nekódované podobě.

2) **Vlastní autentizace**

Tento typ autentizace využívá HTML formuláře a session proměnné a je mnohem flexibilnější oproti HTTP (vlastní přihlašovací stránka, hesla uložená na libovolném místě). V session proměnné se uchovávají informace o přihlášeném uživateli až do doby jeho posledního přístupu. Pro odhlášení tak stačí zrušit session proměnnou. V dnešní době je vhodné ji kombinovat s dalšími bezpečnostními prvky (autorizační token, Zend_Form_Element_Hash), které zabrání útoku Cross Site Request Forgery. Stoprocentní spolehnutí může být pouze na komunikaci přes SSL (Secure Socket Layer).

2.5. METODA GET („GUI SKRIPT")

Při posílání dat metodou GET sdělujeme tenkému klientu, že při požadavku má data (z formuláře apod.) kódovat na konec URL. URL užívá k předání dat serveru a zisk odpovědi od serveru. To, že server posílá odpověď, je jediný rozdíl proti metodě PUT (kdy klient žádnou odpověď nedostává). Užívat by se měla v případech, kdy je celá transakce idempotentní, což v tomto významu znamená, že akce může být bez poškození dat vyvolána vícekrát za sebou. Jinými slovy, když nemění stav na serveru. Výjimkou potvrzující pravidlo jsou formuláře, jejichž URL by po doplnění přesahovalo maximální rámec délky URL (*max délka v IE : 2083 znaků*). Data posílaná metodou GET jsou běžně viditelná, což může způsobovat bezpečnostní rizika. Vkládají se také do logů serveru. Příkladem je situace, kdy má uživatel vypnutá cookies a SESSIONID se předává v URL. SessionID pak může útočník ukrást a pod tímto SessionID se vydávat za uživatele (útok se jmenuje cross site request forgery). Posílaná data musí být ASCII a jsou vždy kódována do Content-type `application/x-www-form-urlencoded`.

2.6. METODA POST („AKČNÍ SKRIPT")

Při odesílání požadavku metodou POST dochází k navázání dalšího spojení. Oproti metodě GET, je tedy náročnější na komunikaci. Tenkému klientu říká, aby data zabalil do těla zprávy. Používat by se měl při procesu s jiným postranním efektem (ukládání, editace dat, poslání emailu). Posílaná data se nevkládají do logu, metoda je bezpečnější. Posílaná data jsou kódována dle atributu ENCTYPE, a to buď jako `application/x-www-form-urlencoded` či `multipart/form-data (pro odesílání souborů)`. Zaslání vstupních dat může změnit stav serveru. Pokud pošleme více POST příkazů za sebou, může dojít k poškození dat, poškození stavu serveru a k dalším kritickým chybám webové aplikace. Znám je termín: „double submit problem".

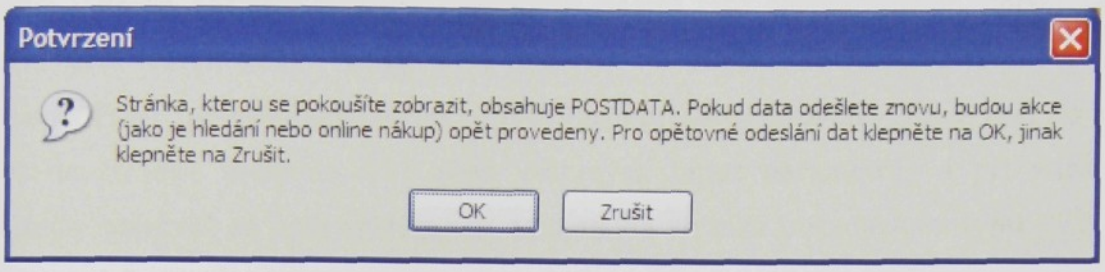
Dojde k němu při vrácení dat požadavkem POST těmito třemi způsoby:

1) Obnovení stránky tlačítkem OBNOVIT prohlížeče
2) Stisk tlačítka ZPĚT a následně VPŘED

3) Opětovné stisknutí tlačítka Submit

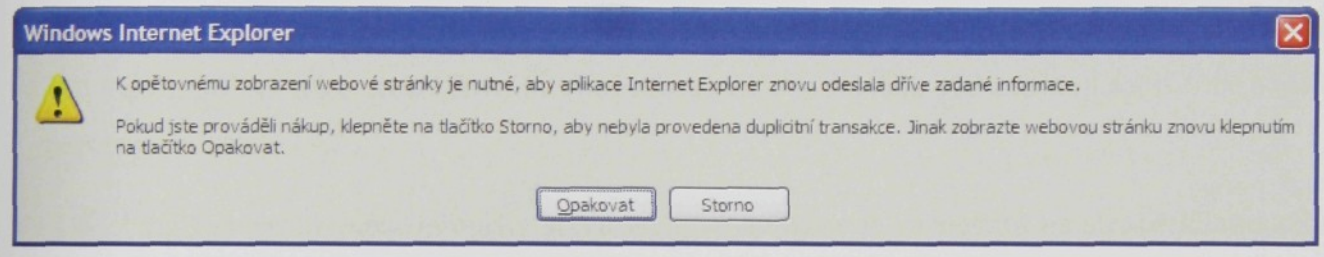
Vývojáři browserů si tohoto rizika všimli, a proto do prohlížečů implementovali nutnost potvrzení opětovného odeslání POST dat. Chtěli tím tak přenést zodpovědnost za porušení stavu na uživatele.

FireFox:



Obrázek 1 - Firefox - hláška pro potvrzení "double submit"

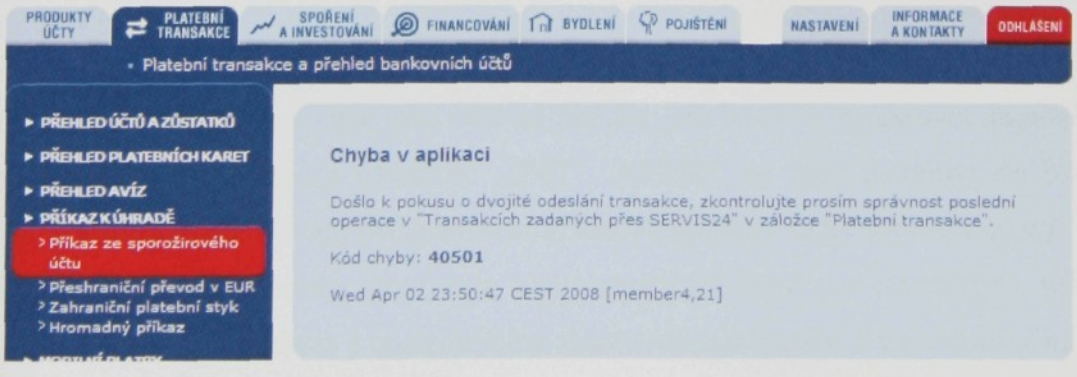
IE:



Obrázek 2 - IE 7 - hláška pro potvrzení "double submit"

Opera uživatele neupozorňuje.

Aplikace může mít vlastní logiku na to, jak zjistit „double submit problem":



Obrázek 3 - Double submit problem při zadání transakce přes Servis24

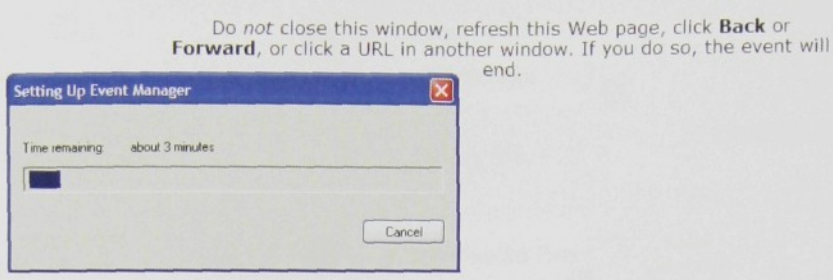
2.7. PRG PATTERN

Uživatel ovšem v devadesáti procentech případů neví, co může potvrzením této hlášky způsobit. Internet je především sociální médium, určené pro lidi (ačkoliv módní web 2.0 jej optimalizuje spíše pro stroje). Ze sociálního hlediska uživatele hlášky naopak stresují, znemožňují mu příjemné prohlížení, nákup a upozorňují na to, že může udělat chybu. Proto platí pravidlo: v dobře napsané webové aplikaci se **nesmí** tato hláška objevit. Docílit se toho dá implementací návrhového vzoru: PRG pattern: POST-REDIRECT-GET, tzn. po provedení požadavku POST (po vykonání celého skriptu) provést přesměrování. Tím je zpět do prohlížeče přenesena stránka metodou GET (bez veškerých input parametrů) a při stisku tlačítka prohlížeče „obnovit" se opět vrátí pouze tato stejná stránka požadavkem typu GET. Jediným způsobem, jak serveru znovu předat input data, je uživatelovo stisknutí tlačítka „Submit" formuláře s metodou POST.

Doporučovanou hlavičkou takovéto HTTP odpovědi je hlavička s kódem "303: See other". Mnohé MVC frameworky tento problém vývojářům velmi usnadňují a vedou ho, aby požadavek POST po skončení akce přesměroval na požadavek GET a hlavičku 303 zajišťují sami. Více o MVC pattern v kapitole 14.2..

Dalším problémem integrity stavu webové aplikace je cachování na straně klientského prohlížeče. Ačkoliv v dnešních dobách rychlého připojení v podstatě postrádá smysl, většina webových prohlížečů tuto techniku implementuje a implicitně nastavuje. Můžou tak znovuobnovit námi pracně vyřešený „Double submit problem" či zobrazovat neaktuální stav aplikace na serveru. Tento problém se řeší zakázáním cachování v sekci head : `<meta HTTP-EQUIV="Pragma" content="no-cache">` a `<meta HTTP-EQUIV="Expires" content="-1">`.

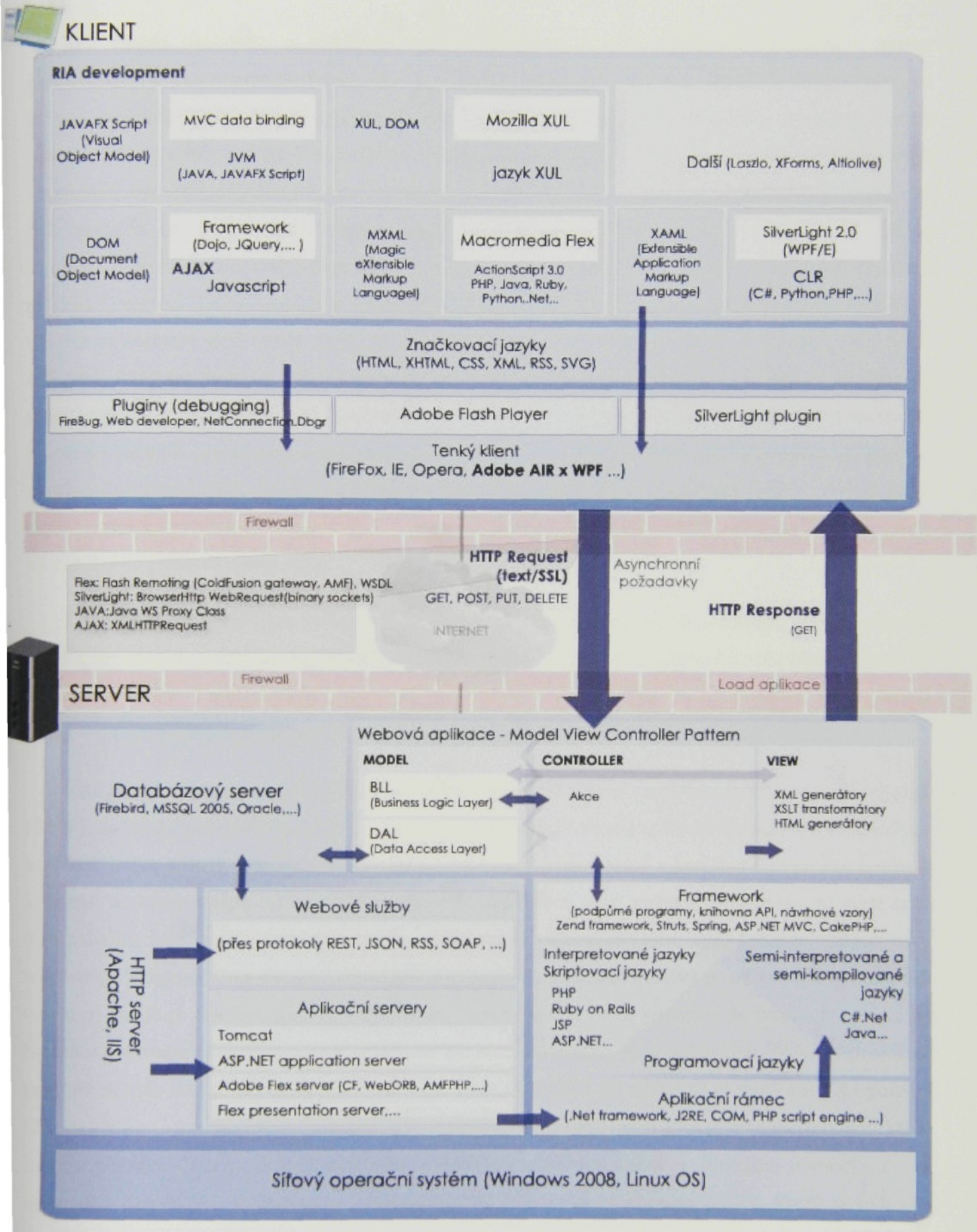
Pokud bude mít programátor tyto věci na paměti, pak bude programovat stavově bezpečné webové aplikace. Problémům s tlačítky „ZPĚT", „VPŘED" se však nevyhne do té doby, než prohlížeče umožní toto chování programovat. (Flexové aplikace to již např. umožňují, otázkou je, kdy se příslušná funkcionalita zanalyzuje a implementuje např. pro javascript). Ještě nějakou dobu se tak budeme setkávat s následujícími doprovodnými texty a varováními na webových stránkách:



Obrázek 4 - start webového semináře (webináře)

3. Obecný přehled jazyků a protokolů pro vývoj webových aplikací

Vývoj webových aplikací může být časově náročnější než vývoj desktopových aplikací. Komunikovat spolu totiž musí mnoho technologií na straně klienta, serveru, přes množství komunikačních protokolů. Následující obrázek se může zdát na první pohled složitým, nicméně, dle mého názoru, uceleně a přehledně vystihuje jednotlivé technologie používané při vývoji webových aplikací. V dalším textu budu popisovat jednotlivé jeho části od serverových technologií až po RIA technologie na klientu.



Obrázek 5 - přehled technologií pro tvorbu webových aplikací

Aby mohla vzniknout a existovat webová aplikace je nutné vlastnit hardware – server s nainstalovaným serverovým operačním systémem. Množství OS dává administrátorovi serveru mnoho možností volby, mezi nejvýznamnější se řadí RedHat Linux, FreeBSD, Windows server (2003/2008), Debian a další. V operačním systému by měl být nainstalován vhodný webový server s možností debuggingu (někdy nutno doinstalovat rozšíření). Dle náročnosti a použité technologie se dále zavádějí různé aplikační servery, které automatizují rozličné úkoly. Při použití databáze v aplikaci je další nutnou technologií databázový server, který může být umístěn i na jiném stroji než je webový server. Každý serverový jazyk má svůj aplikační rámec, virtuální stroj (Virtual Machine) (2). Na aplikačním virtuálním stroji pracují samotné serverové programovací jazyky (PHP, J2EE, ASP.NET, Ruby on Rails,...). Pro ně existují nejrůznější frameworky, které si kladou za cíl oprostit programátora od programování všeobecně se opakujících funkčních celků. V dnešní době zpravidla umožňují rozdělit architekturu aplikace na tři vrstvy: Prezentační, aplikační a datovou za pomocí návrhového vzoru MVC (model-view-controller), který je popsán v kapitole 14.2. . Tento výčet technologií na serveru není samozřejmě vyčerpávající a neuvažuje různé zabezpečovací mechanizmy, nástroje pro správu serveru a jiné.

Mezi klientem a serverem se nachází prostředí internetu. Nebezpečné prostředí, které je vhodné oddělit firewallem a dalšími zabezpečovacími prvky. Komunikace zde probíhá výhradně pomocí již popsaného HTTP či e-mailového protokolu SMTP. Přes HTTP protokol se podle MIME typu přenáší obsah v mnoha různých formátech – HTML, prostý text, JSON, RDF, RSS, SOAP, AMF a další.

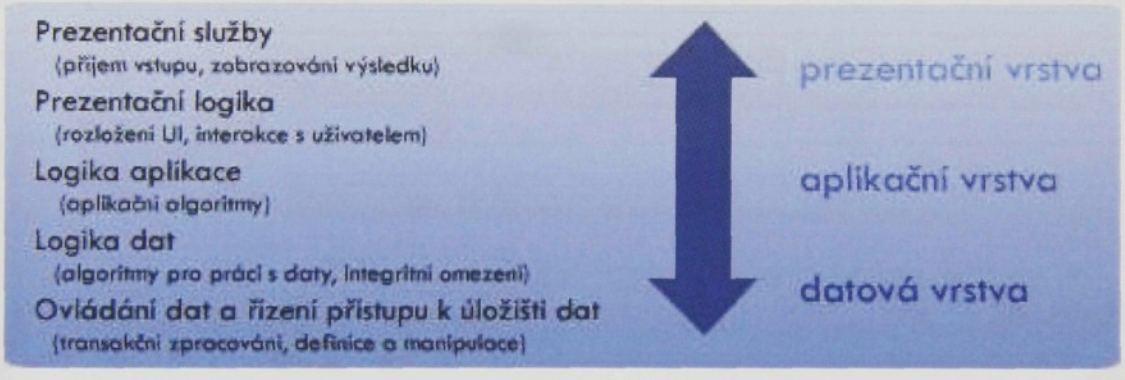
Vstupním bodem aplikace na klientu je webový prohlížeč. Mezi nejvýznamnější se řadí FireFox 2, IE 6 a 7, Safari a Opera, vznikají však nové přístupy přenosu webové aplikace na desktop s následnou možností práce offline – Adobe AIR a MS WPF, o kterých se rozepisuji v kapitole 8. . Pro RIA aplikace je nutná podpora technologií integrovaných do prohlížeče – to znamená nutnost uživatele tuto funkcionalitu sám dodat. Jednat se může pouze o zapnutí javascriptu, vyžadovat se můžou také nejrůznější pluginy (Flash Player, SilverLight plugin) nebo dokonce běhová prostředí (JRE), které tak vytvářejí Sandbox pro RIA aplikace. Z pohledu marketingu je volba klientské technologie závislá na hledisku procentuálního zastoupení těchto technologií na webových prohlížečích. Webový prohlížeč pak sám dokáže interpretovat značkovací jazyk HTML a xHTML do výsledné podoby často za pomocí CSS. Stručně o těchto jazycích v kapitole 6.2. . Jazyk HTML je konečným bodem, RIA technologie, které jsou na obrázku Obrázek 5 běží nejčastěji v nějakém sandboxu, který je spouštěn pomocí pluginu webového prohlížeče. Výjimkou je přístup tvorby RIA aplikací pomocí AJAXu, který je jediným nativním jazykem webového prohlížeče a nevyžaduje tak žádný dodatečný runtime. Jednotlivé RIA technologie popisuji v kapitole 11. a srovnávám v kapitole 12. .

Pro debugging na klientu bohužel neexistuje mnoho sofistikovaných nástrojů. Světlou výjimkou je však plugin FireBug, který je vyvíjen společností Yahoo pro prohlížeč FireFox a díky němuž lze efektivně měřit a ladit funkcionalitu webové aplikace (html, css, skripty), měřit a optimalizovat výkonnost (počet dotazů, velikost přenášených dat, vše přehledně rozděleno dle typu requestu – například samostatný profiling pro XHR komunikaci.) či pomocí automatizovaných testů zjistit doporučení pro vyladění aplikace.

4. ARCHITEKTURY APLIKACÍ

Architektura aplikace je základem kooperativního zpracování. Definuje strukturu z rozložení služeb, jež poskytuje daná aplikace. Volba architektury je klíčovou částí vývoje aplikace, při níž se má brát ohled na budoucí rozšiřitelnost aplikace nebo hardware v případě distribuovaných systémů.

Základní služby aplikací ve vztahu k vrstvám aplikace:



Obrázek 6 - základní služby distribuovaných systémů

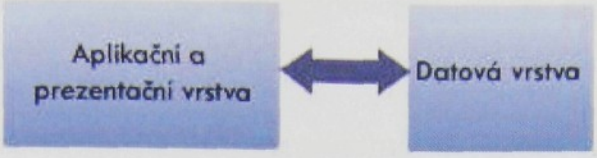
4.1. DVOJVRSTVÁ ARCHITEKTURA KLIENT/SERVER

Klient-server je architektura pro komunikaci mezi dvěma subjekty. Server (sluha) pouze obsluhuje požadavky, které vyvolá klient. Příkladem budiž webový prohlížeč (klient) – webový server (server) nebo také webový server (klient) – databázový server (server). Data byla uložena v proprietárním formátu aplikace a nebyla přenositelná.

Architektura klient-server může být trojího typu:

4.1.1. Klient-server se vzdálenými daty (souborový server)

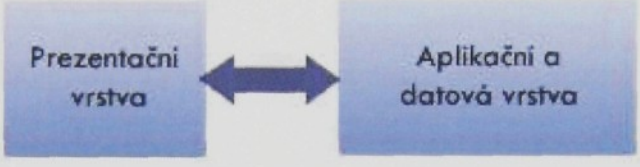
a. **Klient** – poskytuje prezentační služby, prezentační logiku, logiku aplikace a logiku dat
b. **Server** – poskytuje datové služby, V/V operace, souborové operace
c. **Rozdělení zátěže** – Velká zátěž pro klienta a přenosový kanál, malá zátěž na serveru
d. **Příklad**: Subversion



Obrázek 7 - Klient-server se vzdálenými daty (souborový server)

4.1.2. Klient-server se vzdálenou prezentací

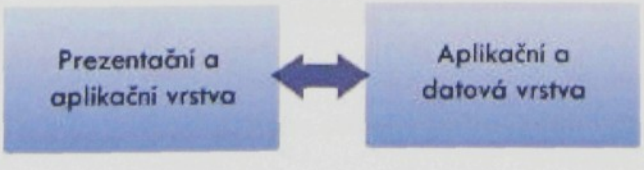
a. **Klient** – poskytuje prezentační služby, prezentační logiku
b. **Server** – poskytuje logiku aplikace, logiku dat, datové služby, V/V operace a souborové operace
c. **Rozdělení zátěže a vlastnosti** – Velká zátěž pro server, malá zátěž pro klienta, pro klienta nerozeznatelné od třívrstvé architektury
d. **Příklad**:Obecně prohlížeče



Obrázek 8 - Klient-server se vzdálenou prezentací

4.1.3. Klient-server s rozdělenou logikou (hybridní architektura)

a. **Klient** – poskytuje prezentační služby, prezentační logiku a spolu se serverem logiku aplikace a logiku dat
b. **Server** – poskytuje spolu s klientem logiku aplikace, logiku dat, sám pak datové služby, V/V operace a ovládání souborů
c. **Rozdělení zátěže a vlastnosti** – Vyvážená zátěž ovšem znamenající horší rozšiřitelnost a přenositelnost, pro klienta nerozeznatelné od třívrstvé architektury
d. **Příklad**: Webový prohlížeč, firemní systémy



Obrázek 9 - Klient-server s rozdělenou logikou (hybridní architektura)

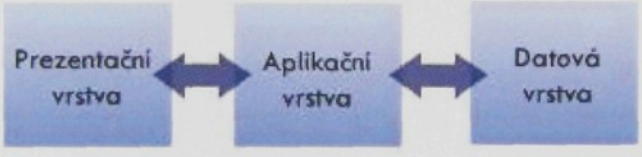
4.2. TŘÍVRSTVÁ ARCHITEKTURA

Při nástupu univerzálních technologií již bylo možné aplikaci rozdělit do tří vrstev a umožnit tak lepší škálovatelnost (horizontální – více serverů i vertikální – výkonnější server), přenositelnost,

rozdělení zátěže a kooperaci mezi systémy. Univerzálnost si samozřejmě vyžádala větší požadavky na správu a údržbu aplikace jako celku.

a. **Klient** – poskytuje prezentační služby a prezentační logiku
b. **Aplikační server** – spravuje logiku aplikace a logiku dat
c. **Datový server** – poskytuje datové služby, souborové operace
d. **Rozdělení zátěže a vlastnosti** – rovnoměrně zatížený, plně distribuovaný systém s mnoha klienty, aplikačními i databázovými servery. Umožňuje sdílení aplikačních objektů (business objects). Komunikační kanál je též zatížen rovnoměrně. Systém je snadno rozšiřitelný a flexibilní. Prezentační úroveň je zcela nezávislá na datové, lze změnit prezentační část (prohlížeč) bez toho, aby si toho všiml datový server. Lze vytvořit různé reprezentace dat bez nutnosti změn struktur dat. Aplikace mohou běžet na levnějších zařízeních.
e. **Příklad**: Webové aplikace

Využití třívrstvé architektury je v současné době tedy již v podstatě standardem a nahradila klasickou architekturu klient/server.



Obrázek 10 - třívrstvá architektura

4.3. Vícevrstvé architektury

Pojem vícevrstvé architektury není dosud (ani v akademickém prostředí) přesně definován , a to také z důvodu, že se velmi často jedná o klasickou třívrstvou architekturu, která je ve svých základních částech interně rozdělena do podvrstev. Z marketingového hlediska tak firmy tvoří své vlastní „nejlepší" architektury s heslem: čím víc vrstev, tím víc enterprise aplikace. Svým způsobem lze například za čtyřvrstvou architekturu klasifikovat systémy běžící na nějakém klientském runtime (.NET framework, JRE) nebo systémy používající aplikační server, ovšem vždy se jedná o rozdělení logiky v rámci daného stroje a nelze je distribuovat na různé stroje.

5. Architektury webových aplikací (AWA), distribuce v SW systémech

Architektury webových aplikací ve svém principu vycházejí z obecných architektur aplikací, ovšem zaměřují se primárně na distribuované systémy. Architektura webových aplikací v sobě zahrnuje popis struktury aplikace, přechod od analýzy k návrhu, použité návrhové vzory, použité frameworky a aplikační servery.

Základní typy distribuce v softwarových systémech uvádím zde, pro více informací doporučuji zdroj (3). U vybraných popisuji významnou implementaci dané architektury, ovšem neznamená to, že se tato implementace při vhodném návrhu nedá využít v rámci jiné architektury, např. při dobrém návrhu je typicky RPC systém CORBA možné implementovat například architekturou SOA.

5.1. DOM (Distributed Object Middleware)

Ke vzdálenému objektu se přistupuje transparentně. Je založen na principu RPC – Remote procedure calling. Mezi významné představitele se řadí CORBA, DCOM (Distributed component object model) od Microsoftu, či EJB (Enterprise Java Beans) od Sunu.

5.1.1. RPC (Remote procedure call)

Aneb komunikace typu METODA-METODA

RPC je webová architektura, která definuje způsob vzdáleného volání procedur, zpravidla uložených na počítači v síti bez nutnosti implementace detailů potřebných k takové interakci. Komunikace je **synchronní**. Programátor píše kód stejně, jako kdyby proceduru spouštěl lokálně.

Definována byla v RFC 707 v roce 1976. Od té doby vzniklo množství jeho implementací, vzájemně nekompatibilních. Např. Microsoft RPC, na kterých Microsoft postavil technologie DCOM a CORBA, MS .NET Remoting jež je jeho nástupcem pro platformu .NET, JAVA RMI (remote method invocation) od Sunu, XML-RPC, které kóduje názvy a parametry procedury do XML a zasílá je přes HTTP, RPyC pro Python či AMF pro Flash/Flex. Přenos tedy může být buď v textové nebo binární formě.

RPC je založen je na aplikační architektuře klient-server. Klient vyvolává procedury s příslušnými parametry, server je zpracuje a zašle klientovi výsledek. V době zpracování požadavku serverem je klient zablokován a v práci pokračuje až po obdržení výsledku procedury (v případě OOP metody –remote procedure calling se označuje také jako remote method invocation).

Proces volání vzdálené procedury vypadá následovně:

1. KLIENT - Zabalení parametru a identifikace procedury do formy, která je vhodná pro přenos po síti (marshalling)
2. KLIENT - Posláni vytvořeného balíčku k jinému procesu
3. SERVER - Vybalení a identifikace procedury a jejích parametrů (unmarshalling)
4. SERVER - Provedení procedury
5. SERVER - Zabalení návratové hodnoty do formy, která je vhodná pro přenos po síti
6. SERVER - Poslání vytvořeného balíčku k volajícímu procesu
7. KLIENT - Vybalení návratové hodnoty

8. KLIENT - Vrácení této hodnoty volající proceduře

Omezení vyplývající z architektury:

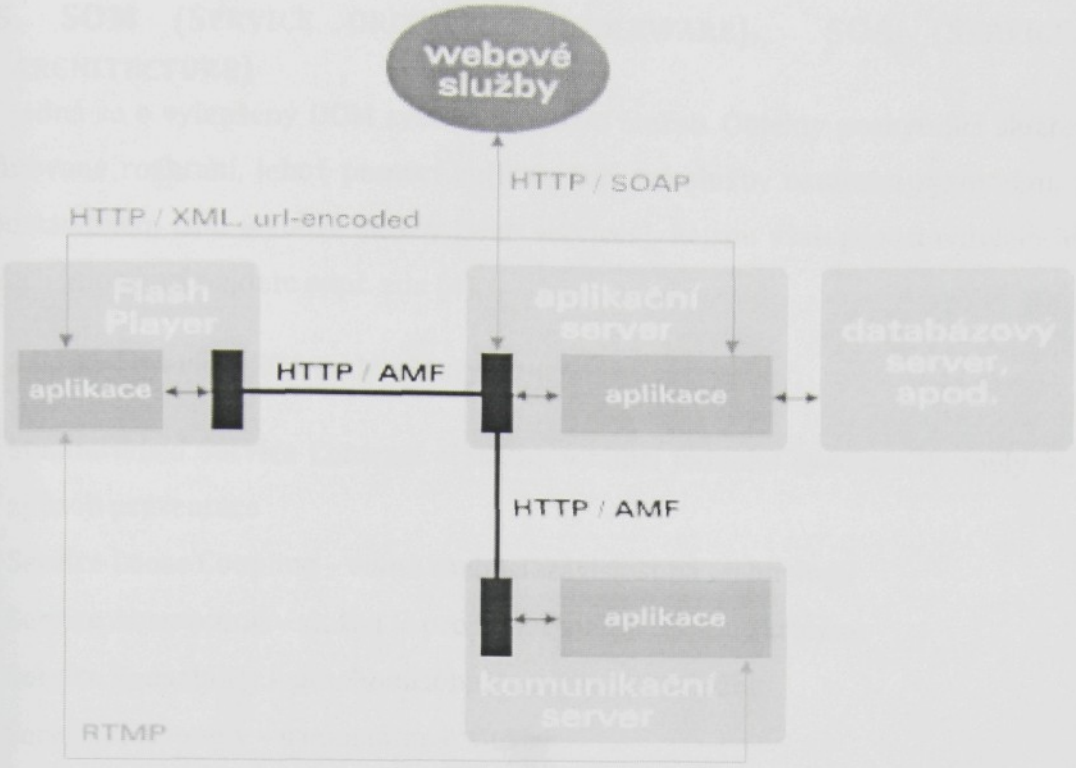
1. Nelze předávat ukazatele
2. Nelze předávat odkaz na soubory na disku lokálním disku
3. Přenášení rozsáhlých parametrů hodnotou je pomalé

Rozdíl oproti lokálnímu volání spočívá zejména v tom, že na síti se mohou vyskytnout problémy a klient odpověď nezíská. Nemůže pak počítat s tím, že procedura byla nebo nebyla spuštěna. Architektura je tedy vhodná převážně pro idempotentní procedury, tedy takové, které mohou být (bez poškození integrity aplikace či dat) vyvolány vícekrát za sebou (například zisk dat).

5.1.1.1. AMF

Pro komunikaci mezi serverovým programovacím jazykem a klientem napsaném ve Flashi se dá použít také SOAP, ovšem ten je velmi popisný a velikost přenášených dat je značná. Proto Adobe přišel se svým vlastním řešením – binárním protokolem pro Flash Remoting postaveným na SOAP:„Action Message Format" (AMF). Tento protokol umožňuje serializovat objekty ActionScriptu (používaném ve Flashi/Flexu). Až do verze ActionScriptu se používala jeho podoba, která se nyní nazývá AMF0. Od nástupu Flash Player 9 a ActionScript 3.0 (postaveném na ActionScript Virtual Machine - AVM) se již využívá protokol AMF3 a umožňuje tak využití nových datových typů a nových vlastností jazyka. Textové řetězce kóduje pomocí UTF-8, čeština je tedy podporována a nejsou s ní žádné problémy. AMF neumožňuje serializaci objektů, proto se v moderním způsobu používání hodnoty předávají jako ByteArray s využitím patternu Value object.

Je zjevné, že na straně serveru je potřebný engine, který serializované objekty a typy ActionScriptu převede do objektů daného serverového jazyka. Tento engine je součástí Flash Remotingu, ačkoliv se nemusí jednat nutně o produkt od firmy Adobe, a běží nejčastěji jako součást aplikačního serveru. PHP má opensource engine AMFPHP a komerční WebORB. Ten je k dispozici také pro .Net a Javu. .Net má nadále možnost užití engine Fluorine, JRun (též Java). Pro Javu existuje opensource projekt Red5, jehož součástí je též Remoting přes AMF, navíc umožňuje další funkcionalitu Flash serveru (streaming videí atd.). Adobe nabízí svůj firemní komerční aplikační server ColdFusion.



Obrázek 11 - princip fungování AMF protokolu

5.1.1.2. .NET Remoting

.NET Remoting je proprietární řešení firmy Microsoft pro komunikaci mezi procesy, proto jej uvádím pouze jako doplněk, pro doplnění. Jedná se o nástupce technologie DCOM. Vhodné je využít pouze v případě, že na straně klienta i serveru je .Net framework, v opačném případě je vhodné použít klasické web services. Jeho výhodou je rychlost oproti webservices, nevýhodou je menší přehlednost kódu. Nelíbí se mi, že pro odesílání dat vždy využívá metodu HTTP POST.

5.2. VOM (Virtual shared memory)

Paralelní procesy mohou přistupovat ke společné sdílené paměti.

5.3. MOM (Message oriented middleware)

Systémy pro asynchronní výměnu zpráv. Zaručují doručení zprávy, i když je klient nějakou dobu offline. Příkladem je JMS (Java messaging service) od Sunu či MSMQ od Microsoftu.

5.4. P2P (peer to peer)

Přímá komunikace mezi klienty bez použití serveru. Stanice jsou si rovnocenné, každý klient je zároveň serverem pro ostatní klienty. Se vzrůstajícím počtem klientů roste přenosová kapacita, na rozdíl od komunikace přes centrální server, kdy se o jeden stroj dělí všichni klienti.

5.5. SOM (Service oriented middleware), SOA (Service oriented architecture)

Jedná se o vylepšený DOM systém o použití služeb. Objekty poskytující službu mají přesně definované rozhraní, jehož pomocí zpřístupňují své služby ostatním systémům. Jeho hlavním představitelem jsou webové služby (web services), nejsou však představitelem jediným. Zdroj dalších informací najdete např. zde (4).

Základní pravidla SOA architektury jsou:

- Standardized Service Contract – služby v rámci jednoho systému by měly používat stejný způsob prezentace
- Service Loose Coupling – volná vazba,nezávislost na technologii
- Service Abstraction – služba je pro konzumenta černou skříňkou
- Service Reusability – mnohonásobná použitelnost služby
- Service Autonomy – samostatnost služby
- Service Statelessness – minimalizace potřeby zdrojů odložením managementu stavu
- Service Discoverability – nalezitelnost – architektura definuje prvky pro nalezení služby
- Service Composability – služba by měla být schopna efektivním způsobem spolupracovat s jinými službami

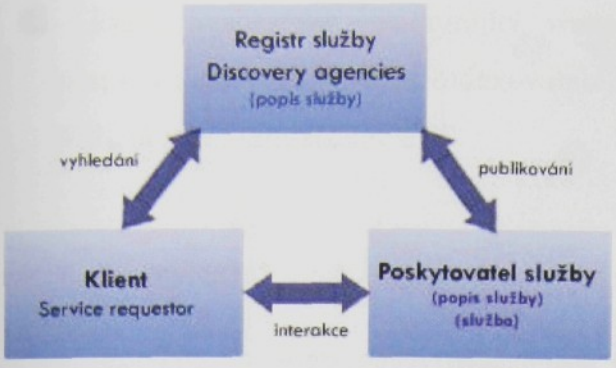
Jelikož složité systémy psané architekturou SOA bývají příliš robustní, těžko upravitelné, při jejich použití v rámci lokálních zdrojů též neefektivní, je patrný záměr přechodu na podmnožinu SOA se zkratkou WOA – web oriented architecture.

5.5.1. Webové služby (Web services)

Webové služby jsou modulární obchodní aplikací se specifikovanými rozhraními, přístupnými a provozovanými v internetovém prostředí. Webové služby jsou často postaveny na architektuře SOA (service oriented architecture) proto je zařazuji sem. Trendem je však stavět webové služby na architektuře REST, viz 5.6.1.

Webová služba se skládá ze tří částí:

- Poskytovatel služby (service provider) – vlastní softwarové řešení uložené na webserveru
- Registr služby (discovery agencies) – místo, kde jsou uloženy informace o webových službách a jejich poskytovatelích
- Klient (service requestor) – Aplikace požadující data z webové služby



Obrázek 13 - implementace web services pomocí SOA Obrázek 12 - web services na bázi protokolu SOAP

5.5.1.1. Protokoly SOAP, WSDL, registr UDDI, WSIL

Jedná se o jedny z prvních implementací protokolů pro předávání zpráv v rámci webservices. Jejich použití není nutné, ale velmi rozšířené. Oba tyto protokoly jsou založeny na XML, s čímž souvisí „popisnost" přenosu, ať už v pozitivním tak i v negativním smyslu (velikost přenášených dat, náročnost na operační paměť). Při požadavku na velký objem přenášených dat je proto vhodné využívat binární protokoly.

- SOAP (5) = **Simple Object Access** Protocol – je nástupce XML-RPC (vzdálené volání procedur) – komunikační protokol k přenosu zpráv mezi aplikacemi (nejčastěji přes RCP) architekturou peer-to-peer či SOA přes protokol HTTP či SMTP (nejčastěji HTTP). Aktuální verze je 1.2 .
  - Ukázka kódu:

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
     <getProductDetails xmlns="http://warehouse.example.com/ws">
       <productID>827635</productID>
     </getProductDetails>
   </soap:Body>
 </soap:Envelope>
```

- **WSDL** (Web Services Description Language) popis rozhraní služby. Zpravidla popisuje SOAP komunikaci přes HTTP. Operace a zprávy jsou popisovány na abstraktní úrovni a teprve poté jsou svázány s konkrétním síťovým protokolem a datovým formátem. WSDL vzniklo až po SOAP společnou iniciativou firem Microsoft a IBM.
- **UDDI** (Universal Description, Discovery and Integration, česky ***univerzální popis, objevování a spojování***) – veřejná databáze webových služeb (žluté stránky pro webové služby). Dvě takové spravují firmy IBM a Microsoft. Data jsou ovšem až z 2/3 neaktuální a nezaručují autentičnost poskytovatele. Proto se UDDI příliš neujalo a pravděpodobně bude nahrazeno jiným konceptem: **WSIL** (také IBM a Microsoft). Její princip je více distribuovaný

- každý webserver, poskytující webové služby, má v kořenovém adresáři soubor `inspection.wsil`, jenž je indexovatelný vyhledávacími roboty (Google apod.). UDDI ani WSIL nejsou standardem W3C.

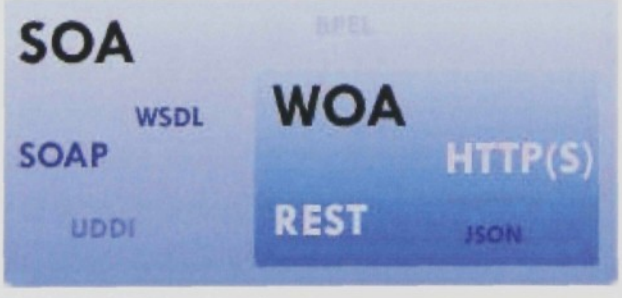
Formát SOAP využívá pro své webové služby (kromě jedné s XML-RPC) firma Google.

Výměna dat mezi poskytovatelem a klienty funguje na protokolu HTTP či HTTPS, na standardních portech 80 a 443, které mívá povolené každý firewall, což je výhodou oproti např. distribuovanému DCOM, jenž bývá na firewallech zakázán.

Díky moderním IDE bývá často programátor od ručního psaní XML dokumentů ušetřen. Obvykle zahrnují generátor WSDL, generátor kódu pro klienta služby, generátor kódu pro serverovou implementaci služby...

5.6. WOA (WEB ORIENTED ARCHITECTURE)

Architektura WOA je podmnožinou SOA architektury a s trochou nadsázky by se jí dal dát přívlastek „zpátky na stromy", jelikož se vrací k základům fungování HTTP protokolu, používání standardních metod GET, POST, DELTE, HEAD, PUT a využívá je moderním způsobem. Současné webové stránky jsou identifikovány pomocí jejich URL a mají většinou jedinou reprezentaci – tu pro webový prohlížeč. Další častou reprezentací je RSS feed. WOA definuje webové zdroje jako webové služby, jejichž reprezentace se mění dle přistupujícího klienta (User-Agent).



Obrázek 14 - WOA ve vztahu k SOA

5.6.1. REST (REPRESENTATIONAL STATE TRANSFER)

aneb komunikace typu ZDROJ-METODA

REST je softwarová architektura pro distribuované systémy. I world-wide-web je distribuovaným systémem postaveným na principu REST. Architekturu REST navrhl jediný člověk Roy Fieldeing, jehož jméno je spojeno s počátkem HTTP protokolu, jako výsledek disertační práce v roce 2000.

Princip této architektury je takový, že aplikační stav i funkcionalita jsou uloženy ve zdrojích (resources) a každý zdroj je unikátně adresovatelný (např. pomocí URL). Všechny zdroje využívají jednotný interface pro transport stavu mezi klientem a zdrojem.

V současnosti existuje na WWW trend pro převod aplikací SOA architektury na architekturu REST, hlavně pro menší systémy, jelikož REST je přímo postaven na principu fungování HTTP protokolu. To má za následek rapidní zvýšení komunikační rychlosti oproti SOA architektuře a lepší škálovatelnost aplikace.

Uvedu příklad webové služby psané architekturou REST. Na něm je vidět, že jelikož je HTTP bezestavový, též REST je bezestavový a stavové informace má možnost ukládat pouze v URL. Jelikož architektura definuje strukturu, je doporučována určitá struktura URL tak, že je tento systém efektivní, navíc většina frameworků již nabízí knihovny pro psaní webových služeb architekturou REST, např. JAVA API for REST, Zend_Client/Server_Rest (PHP). Jejich použití ovšem není nutné.

Zdroje:

```
http://example.com/users/
http://example.com/users/{user} (pro každého jeden)
  nebo
http://domena.com/{kosik}/
```

Metody:

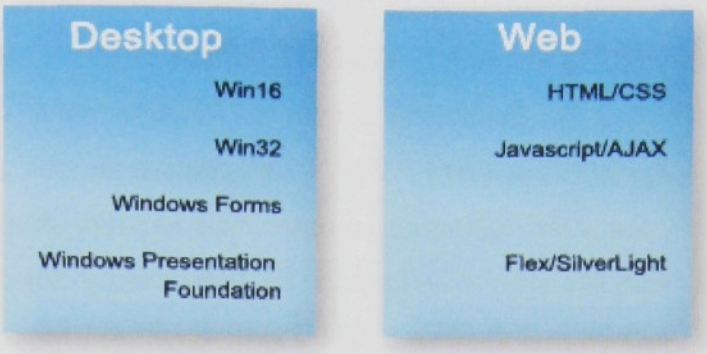
```
userResource = new Resource('http://example.com/users/001')
userResource.delete();
 nebo
userResource = new Resource('http://domena.com/kosikASD354DFQW7/')
userResource.pridejPolozku();
```

REST již využívají společnosti jako Yahoo a Flickr pro veškeré své webové služby, eBay a Amazon jej používají pro některé, v kombinaci se SOAP.

6. Příklady klientských jazyků a technologií pro tvorbu webových aplikací

6.1. Souvislost historie desktopových a webových aplikací

Programování pro desktopové aplikace prošlo během mnoha let dlouhým vývojem. Z tohoto ohledu jsou webové aplikace stále na startovní čáře, ale v této chvíli máme to štěstí být u toho, když se webové aplikace stávají modernějšími a použitelnějšími. Situace se dá připodobnit k přechodu z 16ti bitových OS na 32bitové. Pro mnohé firmy bude novou příležitostí, pro jiné velkou zátěží a mnohými výdaji.



Obrázek 15 - souvislost historie desktopových a webových aplikací

6.2. Základní značkovací jazyky –HTML, xHTML, CSS, HTML5.0 vs. xHTML 2.0

Jelikož jsou tyto jazyky všeobecně známy a zdrojů je dostatek, popíšu je jen stručně a zaměřím se spíše na jejich, jak se zdá, velmi blízkou budoucnost.

HTML – Hypertext Markup Language (5) – je hypertextový značkovací jazyk pro psaní webových stránek vycházející z jazyka SGML. Definuje strukturu pomocí tagů – značek, obsah uvnitř značek, formuláře, možnost vložení skriptů a kaskádových stylů. Původní předpoklad, že jeho vývoj skončil současnou poslední verzí 4.01 a bude zcela nahrazen xHTML, zdá se nebude naplněn, díky aktivitám mimo W3C, jak popíši v následujícím textu.

xHTML –eXtensible HTML - značkovací jazyk, jež měl umožnit plynulý přechod z HTML na XML formu prezentace dat na webu. Poslední verzí je 1.1. XHTML, které umožňuje lepší parsování obsahu.

CSS – Cascade StyleSheet – umožňuje oddělení vzhledu aplikace od jeho struktury a obsahu, přidává nové možnosti formátování dokumentu oproti (x)HTML. Byly vydány zatím dvě verze specifikace CSS1 a CSS2 (relase candidate CSS 2.1), pracuje se na verzi CSS3.

HTML 4.01 je 8 let starý standard. xHTML 1.0 je jazyk starý 7 let. Co bude následovat? Během psaní této práce došlo skutečně k přelomu v boji o trůn mezi dvěma hráči: **HTML5 a xHTML 2.0**. Konsorcium W3C vyvíjí (vyvíjel) xHTML2.0 a zapomněl na velmi podstatnou věc: zpětnou kompatibilitu. A to je důvod, proč se k podpoře tohoto standardu nepřihlásil žádný výrobce klientského prohlížeče. Ba naopak, založili samostatnou skupinu vyvíjející svůj standard HTML 5.0. Její součástí jsou například Mozilla, Opera, Google, Apple. Ti podali nabídku konsorciu, aby se HTML 5.0 stalo společným základem pro další vývoj a na konci ledna 2008 bylo W3C nuceno jejich nabídku přijmout (6).

Oficiální informace tvrdí, že vydání HTML 5.0 standardu je plánováno za 18-22 let, což se ovšem nekryje s aktivitami výrobců webových prohlížečů. Microsoft a jeho nová verze IE 8 přichází s nečekanou novinkou-podporou HTML 5, který je v současnosti pouze v DRAFT verzi. Díky tomu má IE 8 bude výrazně lépe reagovat na prvky vytvořené v AJAXu, například tím, že tlačítko zpět nebude jen přeskakovat po jednotlivých URL, ale bude korektně zohledňovat i dynamické prvky stránek. Stejně tak díky HTML 5 bude IE 8 kontrolovat připojení k Internetu a umožňovat uložit obsah formulářů v případě, že přechodně přijde o připojení k Internetu. Tuto část HTML 5.0 bude umožňovat též nový FireFox 3.0. Safari 3.1 již s podporou HTML 5 vyšlo.

Z pohledu této práce je zajímavostí, že HTML 5.0 si klade za cíl sjednotit existující UI jazyky a zamezit tak závislosti na konkrétním dodavateli technologie (Flash, SilverLight atd.), ovšem tato otázka je pro DRAFT verzi ještě zcela otevřena.

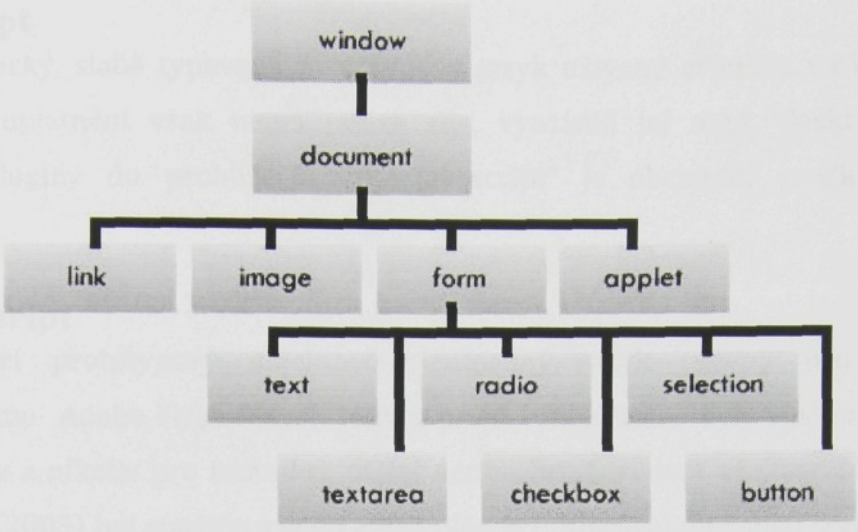
6.3. DOM (Document Object Model)

Document Object model je standard w3c, jehož úspěšně splněným cílem bylo sjednotit přístup k prvkům dokumentu a jejich editaci. Měnit můžeme jak obsah, tak strukturu, atributy či styly. V současné době existují 3 úrovně modelu DOM: Level 1, Level 2 a Level 3, čím vyšší úroveň, tím vyšší požadavky jsou kladeny na implementaci v prohlížeči.

Před rokem 2000 měl každý prohlížeč svoji implementaci DOM, a tak byly prohlížeče vzájemně nekompatibilní. IE měl své IE-DHTML, Netscape svůj model nepojmenoval. Programátor byl tak nucen programovat pro každý prohlížeč zvláštní kód či omezit funkcionalitu na určitý typ prohlížeče a přijít o velkou část zákazníků. Moderní webové prohlížeče již interně parsují obsah stránky do standardního DOM modelu (+ přidávají své specifické prvky – nejvíce se ujalo užitečné innerHTML od Microsoftu) a umožňují tak psát jeden kód pro všechny prohlížeče.

Je tedy možné bez obav psát dle DOM Level 1, při požadavku na vyšší stupeň je nutné srovnat vybavení prohlížečů (7). Prohlížeč dle DOM modelu interpretuje zobrazovaný obsah stránky v prohlížeči. Když chce tedy programátor umožnit uživateli interaktivní komunikaci s jeho aplikací, měl by DOM model ovládat.

DOM model reprezentuje strukturu stejně jako XML dokumenty – jako strom. Přístup po jeho prvcích je pak možný přes množství funkcí jako appendChild, childNodes apod. Základní typy jsou: Document (kořenový element dokumentu), Element (uzel je zapouzdřující objekt), Attr (uzel je typu atribut) a Text (uzel je typu text).



Obrázek 16 - DOM struktura

6.4. Skriptovací jazyky založené na ECMA skriptu: JavaScript, ActionScript 3.0, JavaFX script

ECMAScript je skriptovací prototypový jazyk standardizovaný organizací ECMA (European Computer Manufacturers Association) ve standardu ECMA-262. ECMAScript vznikl v roce 1997 jako snaha o sjednocení dvou, do té doby nejednotných jazyků JavaScript (od firem Sun Microsystems, Inc. a Netscape Communications Corporation) a JScript (od firmy Microsoft).

K plnému sjednocení těchto dvou jazyků nedošlo dodnes, ECMAScript by měl však být zárukou, že pokud budou programy napsány v souladu s tímto standardem, budou fungovat v obou jazycích. Od roku 1997 vzniklo několik dalších jazyků, které implementují standard ECMAScript, například ActionScript.

Skriptovací jazyk (též interpretovaný) je takový programovací jazyk, který je interpretován přímo, bez nutnosti kompilace. Prototypové jazyky jsou podmnožinou objektově orientovaných jazyků, v nichž neexistují třídy, ale jen klonování existujících objektů, které fungují jako prototypy. ECMAskript se tak spíše blíží funkcionálnímu programování než např. jazyku C. ECMAScript se většinou používá jako skriptovací jazyk pro webové stránky XHTML nebo HTML nebo obecně pro XML dokumenty. Skript se vkládá přímo do kódu stránky/dokumentu (nebo je na něj odtud odkazováno) a spouští se na straně klienta (tzv. klientský skript). Protikladem ke klientskému skriptu je serverový skript, který se spouští na straně serveru (např. programovací jazyk PHP, Hypertext Preprocessor). ECMAScript sám o sobě nemá prostředky k tomu, aby mohl pracovat se stromovou strukturou XML. Za tímto účelem byl vytvořen DOM (Document Object Model), který byl popsán v předchozí kapitole.

6.4.1. Javascript

Je dynamický, slabě typovaný prototypový jazyk užívaný zejména v rámci webového prohlížeče. Své uplatnění však nemá pouze zde, využívají jej např. desktopové „widget miniaplikace", pluginy do prohlížečů atd. „Javascript" je obchodní značkou firmy Sun microsystems.

6.4.2. ActionScript

Skriptovací prototypový objektově-orientovaný jazyk určený pro běh v rámci klientského pluginu Adobe Flash Player. Jelikož první verze Flashe byly vyvíjeny zejména pro prezentační účely a nikoliv pro interakci, přišel ActionScript na svět až s verzí Flash Player 5. Actionscript 2.0 (2003) byl spuštěn s verzí Flash Player 7, přidával možnosti stylizace a přístup založený čistě na třídách. Dal se zkompilovat i pro verzi 6.0. Verze FP 8.0 přidala funkční API a současná verze FP 9.0 přišla s novým moderním jazykem ActionScript 3.0 (2006), Ten již běží v rámci virtuálního stroje AVM2 (ActionScript Virtual Machine), které zrychluje běh Flash aplikací a přidává možnost JIT kompilace pomocí virtuálního stroje Tamarin, binárních socketů, reguláních výrazů, ful-screen módu a dalších... V mnohém přesahuje rámec ECMA standardu, navíc se z něj ECMA učí a v příští verzi ECMA 4 již bude též existovat možnost JIT kompilace Javascriptu verze 2.

6.5. JSON – JAVASCRIPT OBJECT NOTATION

Javascript Object Notation je nízkoobjemový textový formát pro přenos dat vycházející z podmnožiny programovacího jazyka Javascript. Dá se přečíst lidským okem a zároveň dobře parsovat. Zvláště jeho parsování do struktury DOM je záležitostí jednoho řádku kódu. Proto se při komunikaci s javascriptovým klientem jedná o velmi dobrou volbu formátu přenosu dat. Většina frameworků pro serverové programovací jazyky již nabízí knihovny pro převod mezi tímto formátem a např. polem či objektem v daném serverovém jazyce (Zend_Json::Encode/Decode). Pro webové služby s ajaxovým frameworkem tento formát upřednostněte před formátem XML.

V hlavičce content-type se přenáší mime typ `application/json`. Z důvodu velké náchylnosti JSONu na útok CSRF (cross site request forgery) je nutné data vždy zasílat jako objekt s obalovými znaky {}.

Příklad kódu:

```
{"menu":                                                    {
  "id": "file",
  "value": "File:",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
```

```
    ]
  }
}}
```

A parsování:

```
var mujObj = eval("(" + JSON_data + ")");
```

6.6. Pluginy pro běh RIA a Moderní značkovací jazyky pro tvorbu RIA - MXML, XAML

Pro dosažení interakce na klientském prohlížeči existuje jediná obecná technologie - javascript. Více o něm píši v předchozí kapitole. Javascript jez však nechvalně znám pro řadu nekompatibilit v prohlížečích a omezené možnosti prezentace, což se nelíbilo společnosti Macromedia (nyní pod Adobe), která přišla s vlastním pluginem pro prohlížeče a technologií Flash.

6.6.1. (Adobe) Flash Player

Flash player je multiplatformní klientský runtime, jenž umožňuje přehrávání vektorových a rastrových animací, pro což byl původně navržen, ale s postupem času přidává i další pokročilé funkcionality jako např. streamování videa, full-screen, JIT kompilaci programů napsaných v ActionScriptu 3 atd. Jelikož Flash Player přišel na scénu v roce 1996, měl dostatek času najít si své místo v prohlížečích a v současnosti dosahuje 95% penetrace v prohlížečích. Pro Flash player se vyvíjí v IDE Flash grafickým způsobem, v IDE Flex (založeném na Eclipse) deklarativním způsobem v jazyce MXML a v již zmíněném skriptovacím jazyce ActionScript. Zdrojové soubory se kompilují do binární podoby SWF souboru, který umí tento plugin zpracovávat. Více o samotné technologii Flash v kapitole 11.2..

6.6.2. (MS) Silverlight Plugin

Pro Silverlight se vyvíjí v značkovacím jazyce XAML, který je založen na XML. Plugin nejčastěji interpretuje kód přímo v prohlížeči, což může být pomalejší než jeho kompilovaný protivník : formát SWF. XML dokument je nejdříve po jeho stažení nutné vždy parsovat, odpadá tak možnost streamování ještě před dokončením stáhnutí celé aplikace. MS na toto již reagoval uvedením binárního formátu pro XAML: BAML.

SilverLight plugin nyní pokulhává v penetraci v prohlížečích. Existují pluginy pro všechny významné prohlížeče (kromě Safari pro Windows, na tom se pracuje), avšak verze IE8 má mít plugin integrovaný a pravděpodobně si instalaci pluginu Microsoft vynutí i v nějakém update OS Windows.

V současnosti je stabilní verze 1.0, ve vývoji je beta 2, která má přinést podporu jazyků C# a Visual Basic, protokolů JSON, REST, webových služeb RSS, LINQ, knihovnu komponent, a další. Poslední z těchto prvků má již Flash Player 9 zahrnutu od roku 2006 a prověřenu časem.

Co bude pro SilverLight plugin velkým plusem je zahrnutí CLR do klientského Runtime. Umožní tak nejen použití výše zmíněných jazyků na straně klienta, ale i oddělení kódu na straně klienta a JIT kompilaci těchto skriptů. Pravděpodobně tak vznikne další MVC framework pro SilverLight (klientský). O běh skriptovacích jazyků JavaScript, Ruby a Python se postará DLR - Dynamic Runtime Library. Díky CLR a DLR nebude nutností, aby měl klient nainstalován .Net framework.

Mínusem je nutnost zapnutého javascriptu, který se používá pro zavedení pluginu v HTML stránce a též nejistota, zda až se MS SilverLight prosadí, neukončí podporu svého pluginu pro prohlížeče třetích stran. V každém případě se jedná o velmi zajímavou technologii.

6.6.3. MXML

Magic eXtensible Markup Language je značkovací jazyk založený na XML, vyvinutý firmou Macromedia v roce 2004. Nejčastěji se používá s Flex server, který jej dynamicky kompiluje do podoby SWF souboru, lze je však také zkompilovat jednorázově (z příkazové řádky - zcela zdarma). Deklarativním způsobem se v něm definuje UI, události prvků, ale také částečně logika aplikace. Do MXML lze vkládat obsluhu událostí jazykem ActionScript, a to buď do skriptu a sekce CDATA, či do zvláštního souboru (code behind) s příponou .AS.

```
Příklad:
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Array id="sampleArray">
        <mx:String>Sample Label 1</mx:String>
        <mx:String>Sample Label 2</mx:String>
    </mx:Array>
    <mx:Panel title="Example Panel">
        <mx:ComboBox dataProvider="{sampleArray}"></mx:ComboBox>
    </mx:Panel>
</mx:Application>
```

6.6.4. XAML

XAML (eXtensible Avalon Markup Language) je značkovací jazyk založený na XML, který se používá pro vývoj pro SilverLight Runtime, WPF a WP. Deklarativním způsobem se v něm definuje uživatelské rozhraní a události prvků. Užívá se v kombinaci s CLR jazykem nebo skriptovacími jazyky JavaScript, Python a Ruby, které se starají o vykonávací kód události.

```
Příklad:
<Page
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Class="MyNamespace.MyPageCode">
  <Button Click="ClickHandler" >Click Me!</Button>
</Page>
```

7. Příklady serverových jazyků, frameworků a technologií pro tvorbu webových aplikací

7.1. PHP, Zend framework

Personal Home Page, později Hypertext Preprocesor je skriptovací dynamicky typový interpretovaný programovací jazyk, určený pro tvorbu webových stránek, nezávislý na platformě. V oblibu vešel verzí PHP3 pro svou jednoduchost. Až od verze 5 se dá mluvit o objektově orientovaném jazyce (podobný Javě), v současnosti existuje verze 5.2 a chystá se PHP6. V kritikách PHP je nutno sledovat verzi diskutovaného PHP. PHP je z většiny napsáno v C - sdílené knihovně. PHP neběží nad žádným aplikačním serverem, tedy platnost všech proměnných je omezena na jeden požadavek. Výhodou oproti všem nevýhodám může být to, že je proces ve svém průběhu těžké porušit (žije jen pár milisekund). PHP tedy nezatěžuje server paměťovými dírami (memory leaks) a je možné jej snadno přerušit. Procesy jsou izolovány, jedna PHP aplikace neohrožuje jinou PHP aplikaci. PHP je konfigurovatelné přímo z kódu, není nutno žádat administrátora. Bezestavovost protokolu HTTP řeší pomocí cookies/sessions.

Za jeho vývojem stojí firma Zend, která v roce 2006 přišla na trh s novým webovým aplikačním frameworkem Zend framework (Framework = podpůrná knihovna, program, API, návrhové vzory atd.). Všimla si totiž velkého nedostatku PHP - neexistence jakéhokoliv oficiálního frameworku, který by usnadňoval programátorům práci. Ačkoliv existuje množství frameworků, svou komplexností i větších, Zend po svém oficiálním frameworku požadoval vlastnosti, které mu žádný existující framework nenabídl, a proto se rozhodl vytvořit vlastní. Zend framework tak má splňovat požadavky na jednoduchost používání, neexistenci nutných konfiguračních souborů, komponentovém přístupu, MVC architektuře, objektovém přístupu založeném na PHP5, stabilitě a silné podpoře komunity.

Hlavní vybrané komponenty frameworku:

Pro implementaci MVC vzoru:

- `Zend_Controller`, `Zend_Controller_Action`, `Zend_Controller_Dispatcher`, `Zend_Controller_Plugin`, `Zend_Controller_RewriteRouter`, `Zend_View`
- `Zend_Http_Request`, `Zend_Http_Response`

Pro implementaci DAL a ORM/ Table/Row data gateway vzoru:

- `Zend_Db`, `Zend_Db_Table`, po přidání late static binding v PHP 5.3 se bude snažit o implementaci ActiveRecord vzoru.

Pro autentifikaci, autorizaci a správu session:

- `Zend_Acl, Zend_Authentication, Zend_Session`

Pro vnitřní infrastrukturu:

- `Zend_Cache, Zend_Config, Zend_Console_Getopt, Zend_Log, Zend_Memory`
- `Zend_Debug, Zend_Environment, Zend_Loader, Zend_Registry,Zend_Version`
- `Zend_Filter, Zend_Validate`

A další:

```
Zend_Json, Zend_Pdf, Zend_Mail, Zend_Mime, Zend_Search_Lucene,Zend_Date,
Zend_Locale, Zend_Measure, Zend_Translate, Zend_Feed, Zend_Rest_Client,
Zend_Service, Zend_XmlRpc_Client, Zend_Gdata, Zend_Http_Client, ...
```

Zend framework nabývá na významu, zejména kvůli tomu, že je vyvíjen pod hlavičkou firmy Zend a existují domněnky na zařazení frameworku přímo do hlavní instalace PHP v nějaké příští verzi. Svým charakterem je velmi podobný Javovskému frameworku Struts.

Na Zend frameworku je postavena i tato práce, původně navržena pro verzi 1.0. V průběhu vývoje ale sledovala všechny beta a release candidate verze, takže se spuštěním Zend Framework 1.5 na začátku března již běží pod tímto frameworkem.

7.2. CLR, ASP.NET FRAMEWORK, ASP.NET MVC

Common language Runtime umožňuje volbu programovacího jazyka ze všech jazyků platformy .Net, které jej podporují (nejčastěji C#). ASP.NET je webovým frameworkem pro CLR jazyky, jedná se o řadu nástrojů pro tvorbu webových aplikací a služeb.

Aplikace založené na ASP.NET se předkompilovávají do DLL knihoven, a není zde tak nutnost opětovného parsování skriptu (PHP lze nakonfigurovat též pro jednorázovou kompilaci do opade). Při kompilaci se nalezne více chyb a mohou se opravit ještě před deploymentem. ASP.NET pro tvorbu uživatelského rozhraní využívá serverové ovládací prvky namísto klasického HTML. Jedná se o dobře testovanou bohatou sadu komponent s předpřipraveným chováním a s možností odchytávání událostí (eventHandlers) - čímž je ASP.NET unikátní (událostní programování na nestavovém webu). Použití těchto komponent velmi zrychluje proces tvorby a zjednodušuje přechod od vývoje desktopových aplikací.

Bezstavovost HTTP protokolu řeší ASP.NET pomocí prvků VIEWSTATE a SESSIONSTATE. View state binárně kóduje (base64-encoding) do hidden elementu informace ze stránky a přenáší se mezi postbacky. V případě špatného používání (objemné gridy, mnoho web komponent s EnableViewState) se pak mezi klientem a serverem odesílá příliš velké množství dat (nevyhovující jsou již kilobajty dat), které zpomaluje chod celé aplikace a zhoršuje uživatelský komfort (Microsoft ani Atlas - dva velcí uživatelé ASP.NET ViewState vůbec nepoužívají). ViewState má výhodu v tom, že nevyžaduje žádné nadstavbové prvky na straně

klienta. Důležité je, že ViewState neuchovává obsah formulářových prvků, ty jsou standardně uchovány klasicky, v proměnných post a get. Je tedy velice dobře možné programovat většinu aplikací bez ViewState.

SessionState uchovává informace na serveru, k čemuž potřebuje ukládat session id, a to buď pomocí cookies(bezpečnější) nebo v URL (nebezpečné). SessionState si klade velké nároky na server, v případě nesprávného užití je pak server náchylný k DoS útoku. Řešením může být oddělení SessionState do samostatného procesu. Další technikou, která méně zatěžuje klienta a více server je ukládání Viewstate přímo do session. Užívá se například pro mobilní zařízení.

Verze ASP.NET jsou číslovány stejně jako celý .Net framework, jehož je ASP.NET součástí. Celý .Net framework působí dojmem velmi ucelené technologie, která však programátora nenutí užívat předpřipravené chování a umožňuje mu optimalizovat requesty na míru aplikace (za cenu delší práce než s připravenými koncepty). Na druhou stranu lehce zastírá co je processing na serveru a co již zpracovává tenký klient, proto si programátor rozdělení logiky mezi klient a server musí sám dobře hlídat.

Renderování výstupu provádí asp.net na serveru pomocí již zmíněných web-controls, programátor nezapisuje přímo HTML kód. Server pak optimalizuje výstup pro konkrétní klientský prohlížeč, který požadavek zadal, tedy na každém klientu se zobrazuje specifický Html kód.př.

IE6 – podpora CSS

```
<span id="Label1"
   style="color:Blue;">
   Ahoj
</span>
```

Netscape 4 – nepodporuje CSS

```
<span id="Label1">
   <font
   color="Blue">Ahoj</font>
</span>
```

Nutno podotknout, že generovaný kód HTML není zpravidla syntakticky čistý (avšak validní) a přehledný, za což je ASP.NET kritizováno. Komponenty též nejsou přístupné (accessible), existují však add-ons, které toto napraví.

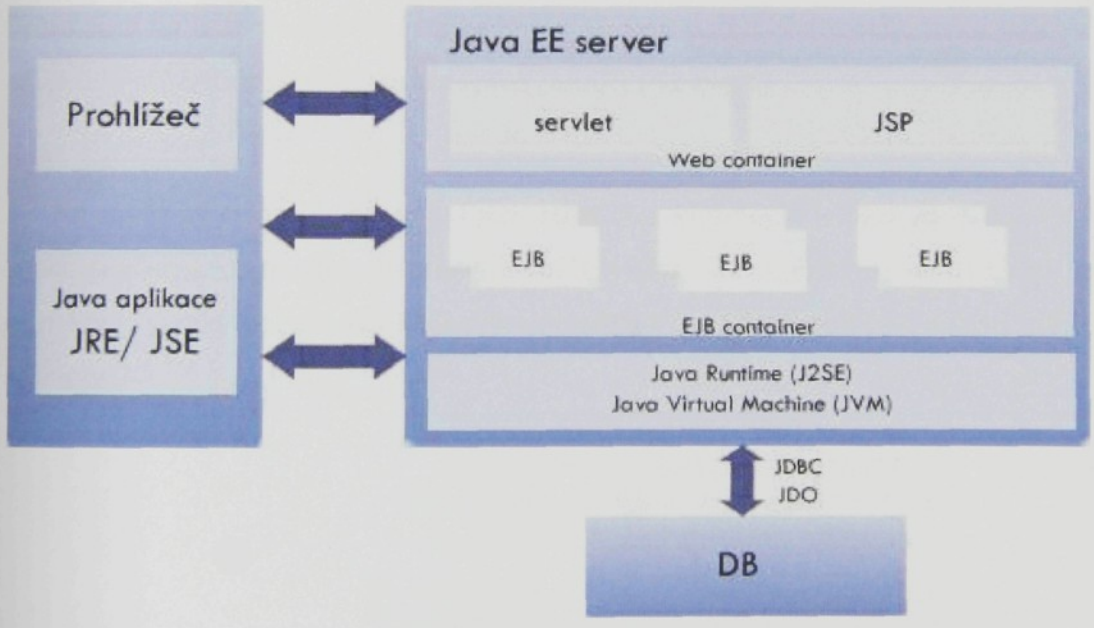
Aktuální verzí je ASP.NET 3.5 (vyšlo v únoru) a přináší dlouho očekávanou a uživateli vyžádanou novinku - implementaci MVC vzoru ASP.NET MVC (prvním a jediným MVC frameworkem pro ASP.NET byl MonoRails). Oproti WebForms, Code behind a Code beside tak nově nabízí lepší oddělení prezentace, dat a logiky, ale také lepší testovatelnost a routování. Microsoft jím však neplánuje nahradit WebForms. ASP.NET MVC má být pouze alternativou, navíc takovou, jenž si mnoho své funkcionality bude propůjčovat z WebForms. Do ASP.NET 3.5 se musí pro jeho použití doinstalovat MVC Toolkit a je nutné využít nového Visual Studia 2008 v minimální verzi Standard.

7.3. JAVA EE, SPRING FRAMEWORK

Java Platform Enterprise Edition je velmi rozšířenou platformou pro serverové programování. Oproti Java SE přidává možnost vícevrstvé architektury a distribuovaných systémů a běží nad aplikačním serverem. Aktuální verze se jmenuje Java EE 5 v letošním roce má vyjít verze 6. Java EE 5 je standardem, jehož oficiální prvky určuje JCP (Java Comunity Process). Chce-li společnost třetí strany vyvinout Java EE technologii, musí být jejím členem a postupovat dle JSR (Java Scpecification Request). Java EE poskytuje mnoho různých API a mnoho frameworků (viz dále). Nabízí také vlastní specifické principy jako Enterprise JavaBeans, servlety, portlety, JSP stránky a podobně.

Základní oblasti, jež Java EE zahrnuje:

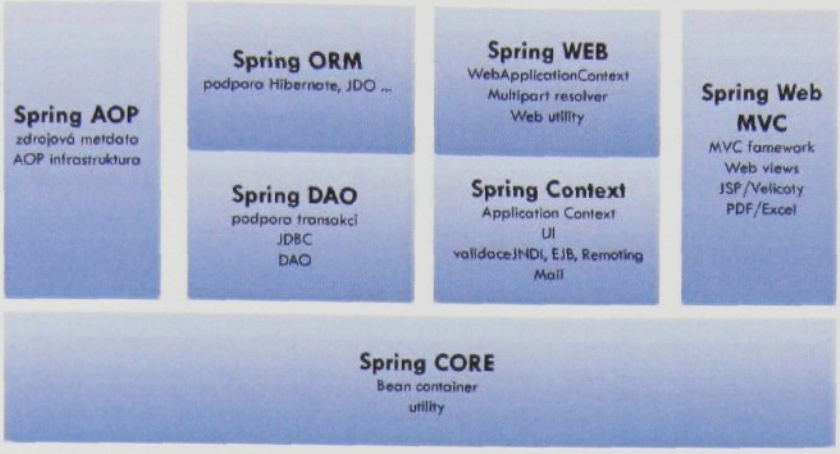
- vývoj webových aplikací - Java Servlets, Java Server Pages (JSP)
- vývoj sdílené business logiky - Enterprise Java Beans (EJB), Spring
- přístup k legacy systémům - Java Connector Architecture (JCA), Hibernate
- přístup ke zprávovému middleware - Java Messaging Services (JMS)
- podpora technologií Web Services



Obrázek 17 - Java EE

Spring framework se objevil jako alternativa k přístupu EJB, jelikož EJB se jeví jako příliš komplexní, postavené na dřívějších požadavcích. Implementuje vzor MVC s podporou integrace různorodých technologií (JSP+JSTL+Tiles, Velocity, FreeMarker, XSLT, JasperReports), nabízí možnost test-driven-developmentu- podporu AOP (aspektově orientované programování) a nenutí k použití aplikačního serveru tak, jako EJB. Spring framework se ale v určitých případech dá použít s EJB. EJB nezůstává pozadu, a proto nová příchozí verze 3.0 by měla podstatně zjednodušovat vývoj a poučit se z úspěchu Spring frameworku.

Vlastnosti a uspořádání Springu by se daly začlenit do sedmi modulů:



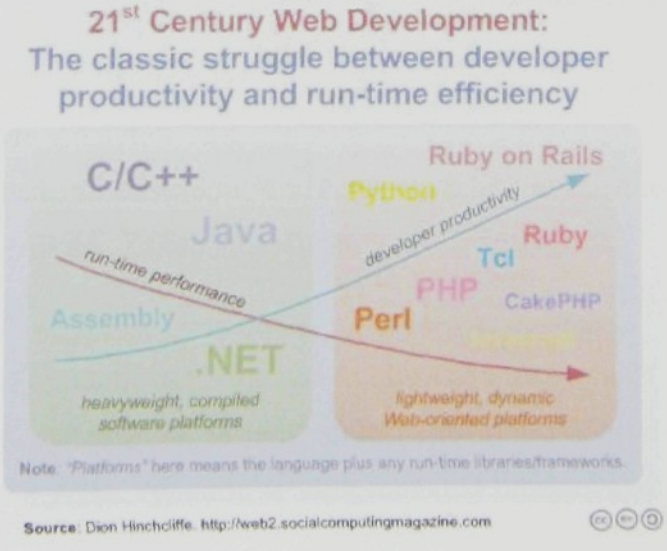
Obrázek 18 - Spring framework

7.4. Ruby, Ruby on Rails

Programovací skriptovací jazyk Ruby je vznikl v roce 1995, avšak momentální věhlas si získal díky rámci Rails, který nabízí rychlý vývoj, rychlé zvládnutí technologie, je velmi dobře systémově navržen. Pokulhává ovšem ve výkonu a je tak těžké pro něj sehnat webhosting. Optimálně na jednom serveru může běžet 10 RoR webů. Kritizován je za svůj OOP přístup, kdy objektem může být i číslo: 5.times {puts „ahoj"} (5x vypiš).

7.5. Srovnání

Na závěr by se hodilo srovnání jednotlivých technologií. Jelikož se však každý hodí na určitý specifický projekt, nelze jednoznačně zvolit vítěze. Na internetu jsem ovšem nalezla zajímavý graf, který jazyky porovnává z hlediska poměru výkonnosti aplikace a času nutného k vytvoření aplikace.



Obrázek 19 - porovnání serverových jazyků

8. PŘENESENÍ WEBOVÝCH (RIA) APLIKACÍ NA DESKTOP

8.1. AIR, WPF

Z důvodů popsaných v kapitole 0a také ze snahy sloučit vývoj pro web a pro desktop, vznikly nové platformy AIR (Adobe) a WPF (Microsoft). Jejich snahou je vyplnit mezeru na trhu mezi desktopovými aplikacemi a webovými službami.

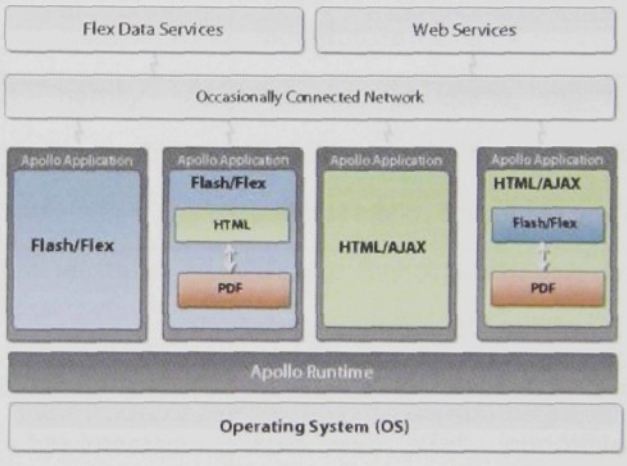
8.2. AIR(APOLLO)

„Adobe Integrated Runtime", tedy integrované běhové prostředí Adobe. Jedná se o multiplatformní řešení vývoje RIA aplikací na desktopu. Běhové prostředí AIR je založeno na třech pilířích:

- WebKit framework(prezentace HTML a JavaScriptu)
- Flash technologie (v9 pod AVM - JIT kompilátor Tamarin)
- SQLite databáze (úložiště persistentních dat)

Oproti desktopovým aplikacím nabízí větší volnost ve tvorbě GUI, využití stávajícího webového frameworku také pro desktop a obsahuje také bezpečnostní prvky – při instalaci nové aplikace se dotazuje, zda bude moci daná aplikace přistupovat k souborům na disku a ověřuje digitální podpis.

Oproti webovým aplikacím nabízí automatický drag and drop prvků a práci offline.



Obrázek 20 - platforma Adobe AIR

Adobe Air se prosadilo u firem jako Google se svou verzí Google Analytics AIR, E-bay pro jejich aukce na desktopu, YouTube – přehrávání a download videa přímo z desktopu, Offline blog WordPress, Yahoo Messenger.

8.3. WPF (AVALON)

Windows Presentation Foundation je reakcí na Adobe AIR a též se snaží o přenesení webové aplikace na desktop. Jeho nevýhodou je, že není multiplatformní. Popravdě - existuje oficiální podpora pouze pro Windows Vista. Výhodou je však možnost integrovat WPF komponentu do Windows Forms.

WPF běží na DirectX, veškerá grafika je tak renderována na grafické kartě – to platí v případě, že grafická karta hardwarově podporuje vše od DirectX 7 po DirectX 9. Vertex a Pixel

shadery musí být ve verzi 2.0, jinak může nastat situace, kdy je aplikace částečně renderována na GPU a částečně na CPU.

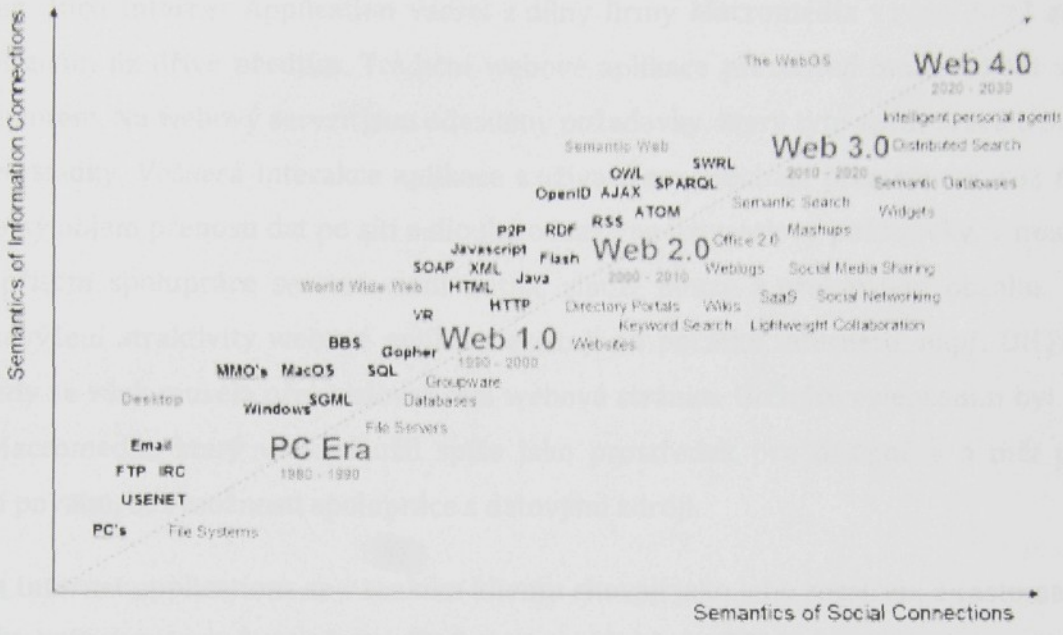
9. WEB 2.0, SÉMANTICKÝ WEB A WEB 3.0

Myšlenka sémantického webu existuje již velmi dlouho a již ona prorokovala příchod toho, co je nyní velmi volně překládáno jako web 2.0. Sémantický web je nadstavbou současného webu a jeho cílem je přiřadit datům přesný význam, na základě kterého mezi sebou budou umět komunikovat jak lidé, tak software. Sémantický web doplňuje do současného webu metadata, která by měla popisovat sémantické informace o webovém zdroji a byla zapsána v strojově srozumitelném jazyce.

Web 1.0 versus Web 2.0		
	Web 1.0	**Web 2.0**
OBSAH	Obsah je vytvářen převážně jeho vlastníkem	Návštěvníci se aktivně podílejí na tvorbě obsahu – vlastník je v roli moderátora
INTERAKCE	Interakce vytváří nároky na vlastníka, proto jen v nezbytné míře	Interakce je vítána, má formu diskuzí, chatu, propojení s messengery, sociálních profilů
AKTUALIZACE	Odpovídá možnostem vlastníka	Web je živý organizmus – tvůrců obsahu mohou být miliony
KOMUNITA	Neexistuje, návštěvník je pasivní příjemce informace bez interakcí	Návštěvník je současně ten, „o kom web píše", jednotlivec je součástí rozsáhlé komunity
PERSONALIZACE	Weby neumožňují implicitní personalizaci	Umožňují vytvářet a využívat sociální profily čtenářů

Tim O´Reilly v roce 2005 sepsal, co by měl splňovat Web 2.0. Opírá se o nutnost decentralizace webových služeb. V současnosti se využívají zejména tyto komponenty Web2.0: tagování RSS Feeds, AJAX, blogování, zpřístupnění API pro třetí strany, webové služby, wikipedia,... Web 2.0 tato decentralizovaná data umí oproti sémantickému webu atraktivně prezentovat. Oproti sémantickému webu však neumí integraci mezi aplikacemi. Jejich spojením bychom v blízké budoucnosti mohli získat velmi atraktivní kooperující webové aplikace. Toto spojení se již dnes označuje jako web3.0.

Následující statistika, převzatá v roce 2007 z Radar Networks &Nova Spivack ilustruje přechod k webu 4.0 a byla použita též na významné konferenci, zaměřující se právě na budoucnost webu (9):



Obrázek 21 - historie webu a přechod k webu 4.0

10. DEFINICE RIA APLIKACE

RIA (Rich Internet Application) jsou webové aplikace, které se snaží překlenout rozdíly mezi klasickou webovou aplikací a desktopovou aplikací. RIA aplikace se snaží v rámci webového prohlížeče napodobovat desktopové aplikace svým vzhledem i chováním a poskytnout vyšší uživatelský komfort.

RIA aplikace se nedrží tradičního request/response principu, nemusí se instalovat, nehrozí útoky viry a červy, uživatel má vždy nejaktuálnější verzi programu a programátor spravuje pouze jednu verzi programu. V podstatě se jedná o opak k programům pro mobilní zařízení, kdy je na klienta potřeba přenášet nejmenší možné množství dat. Pozor si vývojář musí dát na zachování co největší míry přístupnosti, aby přes všechnu interakci nepřišel o významné procento jakkoliv indisponovaných zákazníků (navigace pomocí klávesnice, kontrast barev atd.)

Instalací RIA aplikace je pouze stáhnutí určitého objemu dat a instrukcí na klientský stroj. Pak již může uživatel s aplikací pracovat téměř tak, jak je zvyklý ze standardních desktopových aplikací. V průběhu uživatelovy práce s aplikací může docházet k asynchronním přenosům dat mezi klientem a serverem. V žádném případě by se neměla stahovat celá aplikace při každém požadavku. Tato oblast vývoje pro web je nyní v nejprogresivnějším vývoji, jsme

svědky velkého konkurenčního boje velkých hráčů jako je Microsoft se svým SilverLightem a Windows Presentation Foundation a Adobe Macromedia se svým Flexem a Adobe AIR.

10.1. Historie:

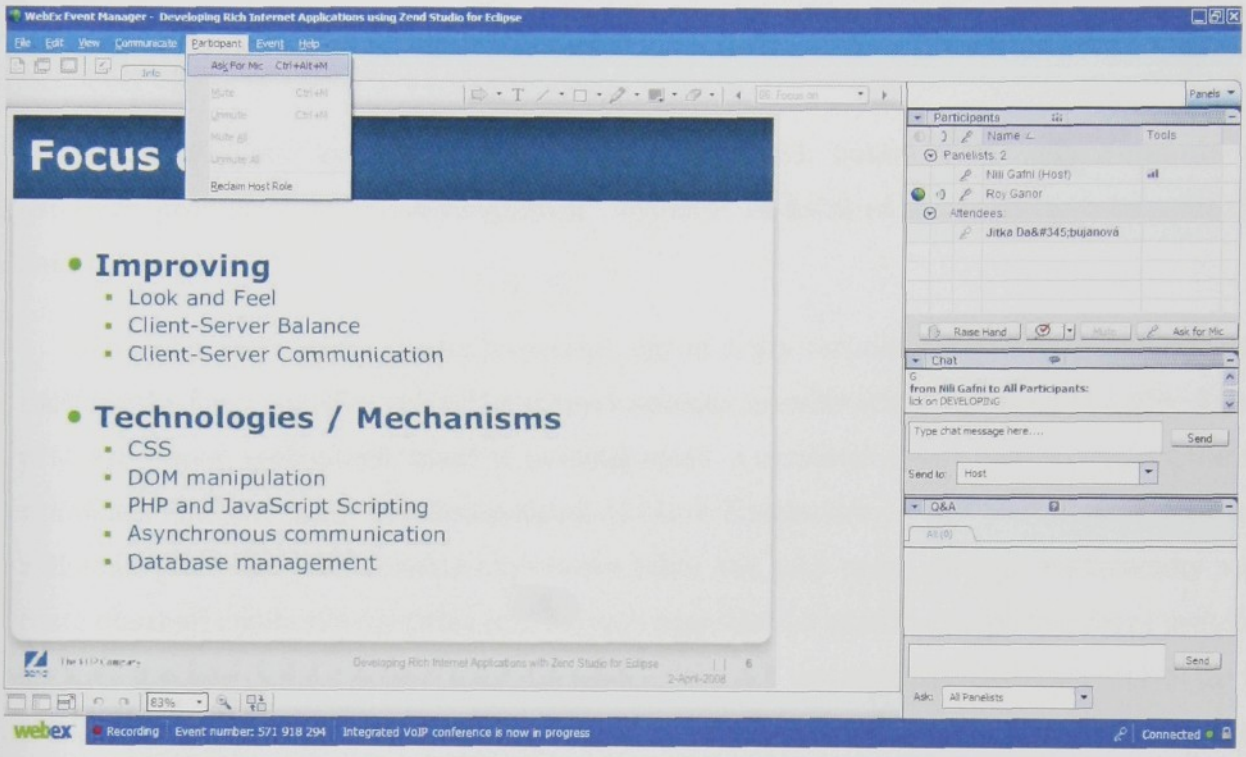
Pojem Rich Internet Application vzešel z dílny firmy Macromedia v roce 2002 ačkoliv koncept byl znám již dříve předtím. Tradiční webové aplikace prezentují model klient-server s tenkým klientem. Na webový server jsou odesílány požadavky, který tyto zpracovává a navrací na klienta výsledky. Veškerá interakce aplikace s uživatelem prochází přes server, což má za důsledek velký objem přenosu dat po síti a dlouhé odezvy na uživatelovy požadavky. V mnohých případech přitom spolupráce serveru není nutná, slouží pouze k překreslení obsahu. Jisté pokusy o navýšení atraktivity webové aplikace existují od počátků internetu, např. DHTML či iFrames. Vždy se však musela překreslovat celá webová stránka. Určitým vylepšením byl Flash od firmy Macromedia, který však sloužil spíše jako prostředek pro designéry a měl pouze prezentační povahu, bez možnosti spolupráce s datovými zdroji.

Rich Internet applications se v tenkém klientu chovají jako jeho rozšíření a často na sebe berou i úlohu volání serveru (asynchronně). Oproti standardním HTML komponentám přidávají své vlastní, implementující např. drag and drop, kalkulace na klientovi, validace na klientovi. Rozdělují tak úlohy mezi klient a server tak, že klient není přetěžován.

10.2. Omezení RIA:

- RIA aplikace běží (většinou) v Sandboxu, nemají tedy přístup ke všem systémovým zdrojům.
- Problémy při vypnutém javascriptu / nenainstalovaném pluginu – aplikace bez nich nemůžou běžet.
- Rychlost na klientovi – zvláště skriptovacích nekompilovaných jazyků
- Prvotní stáhnutí aplikace = download velkého objemu dat.
- SEO optimalizace – u některých technik nemá vyhledávací stroj možnost indexovat stránky.
- Mnohdy nutno, aby byl uživatel aplikace stále připojen k internetu
- Přístupnost – čtečky obsahu mohou mít problém s dynamickými změnami na klientu.
- Nutnost znalosti mnoha technologií, vyšší nároky na programátory, vyšší nároky na testování
- Nutnost existence krizových plánů, např. při výpadku serveru - musí existovat plán okamžité reakce na danou situaci – zapojení záložního serveru, clustering, rychlá obnova zálohy,...
- Pro běh aplikací jsou vyžadovány nejmodernější prohlížeče, popř. pluginy.
- Velké možnosti při tvorbě GUI nesmí mít za následek znepřehlednění aplikace a práce s ní

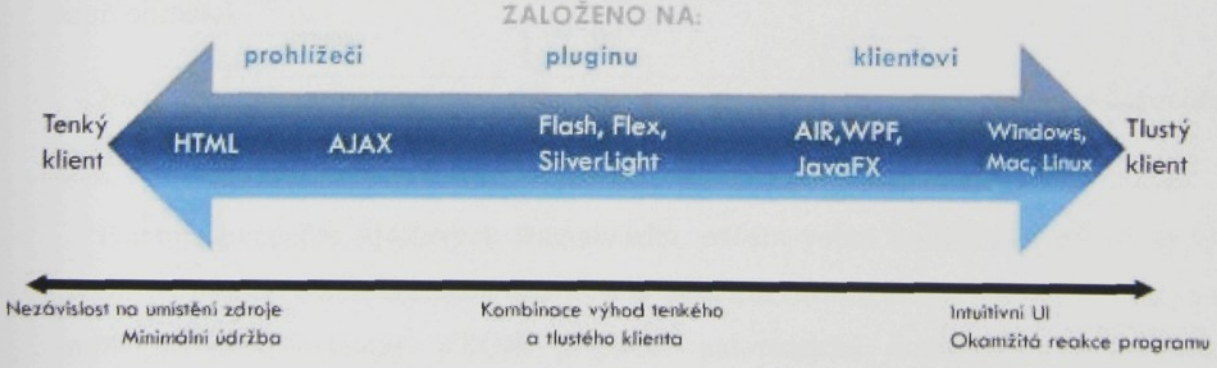
I takto může vypadat RIA Aplikace. Screenshot byl pořízen 2.4.2008 při web konferenci na téma vývoj RIA aplikací pořádané firmou Zend přes RIA aplikaci od firmy WEBEX (postaveno na javascriptu).



Obrázek 22 - příklad RIA aplikace - webový seminář (webinář) na téma vývoj RIA aplikací přes RIA aplikaci Webex

11. TECHNOLOGIE PRO VÝVOJ RIA APLIKACÍ

Z pohledu běhu aplikace se nejběžnější dostupné technologie dají rozdělit takto:



Obrázek 23 - Technologie pro RIA aplikace

11.1. AJAX

Název Asynchronnous javascript and XML uvedl James Garrett ve své práci (8) a je jakýmsi návrhovým vzorem pro používání následujících technologií:

- Document Object Model (DOM) – vytváření a manipulace s elementy stránky

- XMLHttpRequest - asynchronní dotazy na server
- HTML, CSS a JavaScript (JScript) – prezentace a aplikační logika

Věhlas si získal díky modernějším prohlížečům(IE od verze 5.5), které podporují objekt XMLHttpRequest, jež umožňuje asynchronní volání serveru.

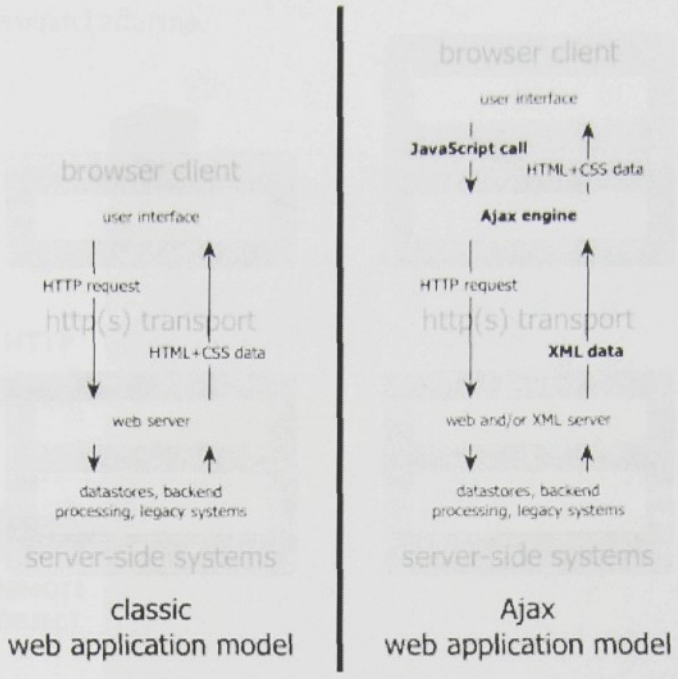
XMLHttpRequest byl jako první interpretován v IE5, ovšem kvůli neexistující podpoře ostatních prohlížečů jej nikdo nevyužíval. Využívat se začal až s jeho zavedením v Mozille a Safari.

Uživatel je nucen mít zapnutý javascript, oproti jiným technologiím však nevyžaduje žádný další plugin. Programování velkých ajaxových aplikací se může stát velmi složitým, jelikož do hry přichází mnoho technologií, které je potřeba spojit a otestovat jejich běh pro nejvýznamnější prohlížeče. Do jisté míry toto zjednodušují AJAXové frameworky, jejich použití je vhodné nejen z důvodu přehlednějšího a méně chybového kódu, tak také proto, že tyto frameworky v sobě často obsahují logiku pro specifika jednotlivých prohlížečů a na základě zjištěné verze prohlížeče aplikují správné funkce a data (např. samotné vytvoření instance XMLHttpRequest). Stále však platí, že vyvíjet RIA aplikace na AJAXu je velmi bolestivé a jen díky tomu se rozmohly alternativní technologie SilverLight, Flex a JavaFX, ve který je vývoj daleko příjemnější a do budoucna budou moci nabídnout funkcionalitu, při níž může AJAXu docházet dech. AJAX je také jedinou technologií z těch, které uvádím v této kapitole, která přímo nepodporuje tvorbu GUI pomocí vektorové grafiky, což zvyšuje jeho náročnost právě o nutnost stáhnutí množství jiných zdrojů, např. obrázků.

Naproti ostatním klientským technologiím, není pro AJAX do jisté míry problém otázka SEO (search engine optimalization), ačkoliv XHR požadavky též nejsou indexovány.

Existuje bezpočet AJAXových frameworků, ovšem velmi se liší. Hlavně co se týče jejich určení. Jedny jsou nízkoobjemové a umožňují základní rutinní úkoly, jako např. práci s objektem XmlHttpRequest, orientaci v DOM, případně automatické doplňování ve formuláři apod. (MooTools, JQuery, ASP.NET AJAX framework).

Jiné jsou určeny pro vytváření RIA aplikací, obsahují složité widgety (komponenty) a pokročilou funkcionalitu (drag and drop). Vykoupeno je to větším množstvím stahovaných dat na klienta (DOJO, ExtJs, Bindows, YUI, ...).



Obrázek 24 - porovnání přístupu klasické webové aplikace a AJAXu

11.2. Flex

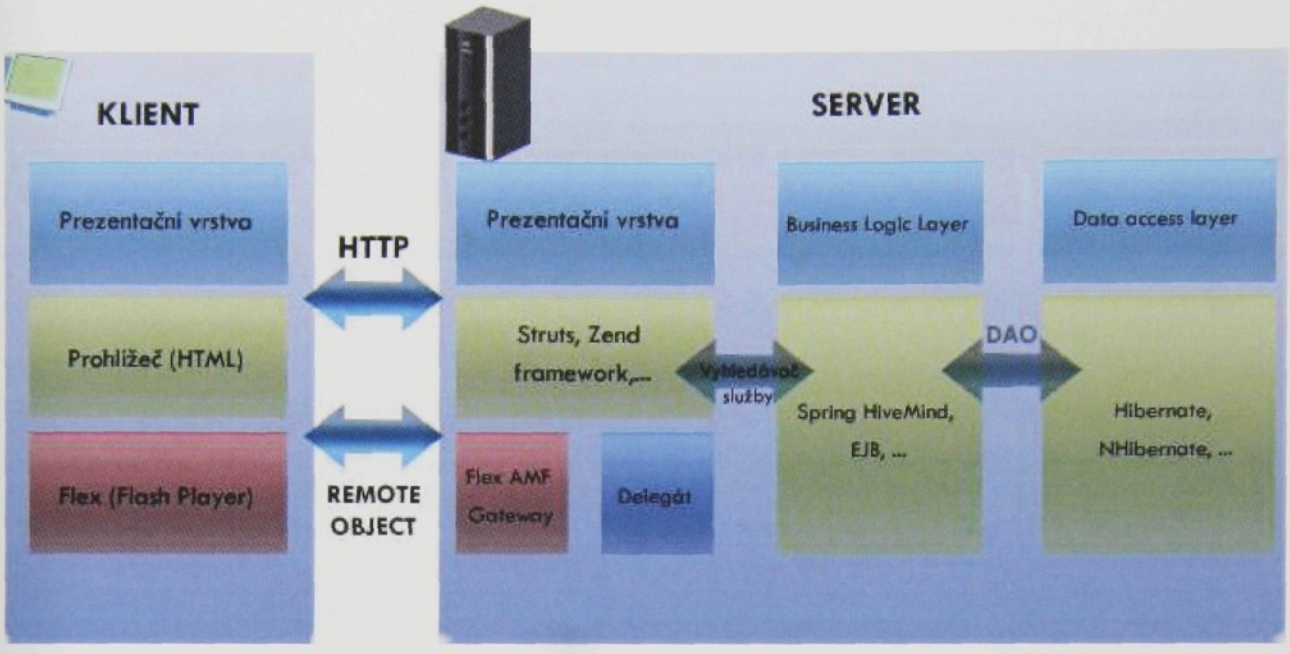
Jedná se o technologii z dílny Adobe, která pohltila dřívější Macromedia. Zpřístupňuje ActionScript 3.0 a možnosti Flash playeru programátorům, které proti použití Flashe odrazovala známá „timeline" a zcela odlišný přístup k programování, než na který byli a jsou programátoři zvyklí. Jeho výhodou je obrovský náskok v procentuálním zastoupení Flash pluginu v prohlížečích. Dle mnoha nezávislých statistik má nejnovější Flash plugin (v9) nainstalováno přes 85% prohlížečů, což je velmi významným faktem při výběru technologie.

Oproti AJAXu či SilverLightu řeší také serverovou část pomocí Flex Server(pro výměnu dat je nutná serverová část – Flex AMF Gateway). Tato serverová část se liší dle použité serverové technologie, v dnešní době existuje Flex server pro všechny významné serverové jazyky. Úkolem flex serveru je převážně serializace/deserializace objektů a typecasting z typů serverového jazyka na typ ActionScriptu. Při asynchronním volání dále kompiluje aktuální skript do binárního formátu SWF.

Komunikace mezi klientem a serverem může probíhat jak pomocí http, SOAP, ZLIB, socket, tak AMF, který je pro ActionScript vhodnou volbou a jeho bližší popis je možné najít v kapitole 5.1.1.1.

Flex umožňuje programovat ve značkovacím jazyku MXML (obdoba XAML) viz. Kapitola 6.6.3 a skriptovacím objektově orientovaném jazyce ActionScript(v současnosti ve verzi 3.0), který je založen na standardu ECMAScript. Jako vhodné vývojové prostředí je doporučen Flex

Builder, ačkoliv MXML kód lze psát v textovém editoru a kompilovat s použitím command-line kompilátoru, který je k dispozici zdarma.



Obrázek 25 - Architektura Flex RIA aplikace

Flex, na rozdíl od SilverLightu, umožňuje vlastní změnu vzhledu komponent pomocí standardního CSS. Rozsáhlá komunita vývojářů pracuje na vývoji mnoha kvalitních komponent použitelných ve Flexu (9). Ten je od verze 3.0 publikován jako open source pod licencí Mozilla Public License a pro Flex existuje již bezpočet aplikací, příkladů a návodů. Aplikace psaná ve flexu se dá zkompilovat do jediného AIR souboru použitelného na desktopu pomocí aplikace Adobe AIR.

Uživateli technologie Flex jsou například E-bay (spolu s sandboxem air), google a další.

11.3. SilverLight

Microsoft vloni přišel s novinkou, která má pokořit největšího vládce trhu RIA – Adobe Flash. Verze SilverLight 1.0 umožňuje pouze práci s jednoduchými médii a je postaven na jazyce javascript. SilverLight 2, který je v beta verzi, již naproti tomu na klientovi využívá .NET CLR včetně managed-javascriptu a pythonu, které jsou kompilovány do binární podoby a na klientu běží velmi rychle. Díky tomu, že podporuje Python, podporuje také Dynamic Language Runtime (DLR), díky čemuž SilverLight nabízí nejbohatší výběr programovacího jazyka na klientu. Stahování tohoto engine na klienta však samozřejmě trvá nějakou dobu, stahují se jednotky MB. AJAXové frameworky se těžko dostávají přes 600KB(i to znamená 5s stahování).

Většina dostupných Silverlight aplikací v současné době vyžaduje javascript, který je též doporučovaným způsobem pro detekci a případnou instalaci pluginu.

SilverLight aplikace se skládá ze tří částí:

- XHTML stránka
- XAML soubor
- soubory s aplikační logikou

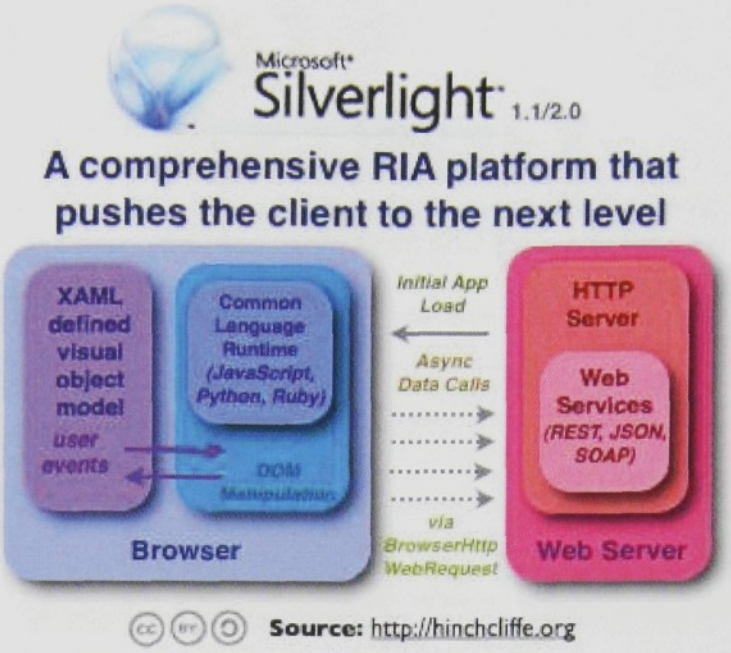
XHTML stránka zavádí plugin pomocí dodávané javascriptové knihovny Silverlight.js.

```
<script type="text/javascript" src="js/Silverlight.js"></script>
```

Samotné zavedení může vypadat takto:

```
Silverlight.createObject(
        "plugin.xaml",          // Source property value.
        parentElement,          // DOM reference to hosting DIV tag.
        "myPlugin",             // Unique plug-in ID value.
        {                       // Plug-in properties.
            width:'1024',       //Width of rect region of plug-in in pixels.
            height:'530',       //Height of rect region of plug-in in
pixels.
            inplaceInstallPrompt:false,     // Determines whether to
                                //display in-place install prompt if invalid
                                //version is detected.
            background:'white',   // Background color of plug-in.
            isWindowless:'false', // Determines whether to display in
                                //windowless mode.
            framerate:'24',                 // MaxFrameRate property
value.
            version:'1.0'                   // Silverlight version.
        },
        {
            onError:null, // event-handler function name.
            onLoad:null   //event-handler function name.
        },
        Null,// initParams -- user-settable string for information
passing.
        null);
```

Pro Silverlight se vyvíjí v značkovacím jazyce XAML, o kterém píši v kapitole 6.6.4.



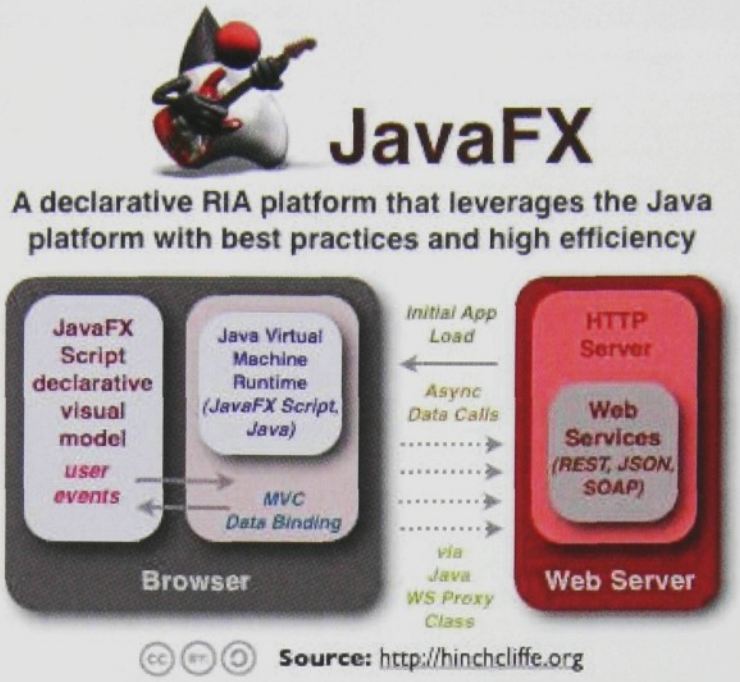
Obrázek 26 - architektura SilverLight RIA aplikace

Pro SilverLight existují vynikající IDE(placená), které též oddělují práci designera a programátora (MS Visual Studio 2008 vs. Expression Blend 2).

V SilverLightu se nachází obrovský potenciál, jelikož komunita vývojářů pro .NET framework je větší než množství vývojářů v Flexu a ActionScriptu. Už dnes je SilverLight na výši v oblasti videa. V USA existují televize, které svůj program vysílají po internetu a pomocí SilverLightu jej lze přehrávat v HD kvalitě.

11.4. JavaFX

Sun nezůstal pozadu a pro vývoj RIA aplikací přinesl technologii JavaFX zcela odlišnou od konceptu Flashe a SilverLightu.



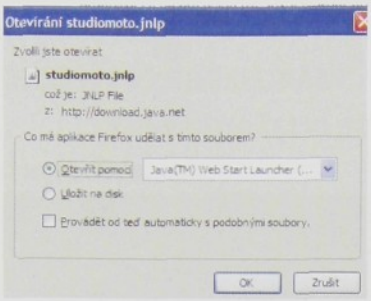
Obrázek 27 - architektura JavaFX RIA aplikace

Vývoj Sun směřuje převážně k zvýšení produktivity programátora při vývoji RIA, zejména tím, že jako jediný implementovali MVC architekturu přímo v skriptovacím jazyce pro klienta, jímž je jazyk nazvaný JavaFX Script. Ten je staticky typovaný, deklarativní a kompilovaný a umožňuje automatický data-binding, nabízí podporu 2D grafiky a ověřených komponent Swing a navíc potlačuje nutnost užití javascriptu, jehož interpretace je v každém prohlížeči odlišná. Je postaven na vrcholu platformy Java, tudíž umožňuje snadné využití javovských tříd přímo v JavaFX Script.

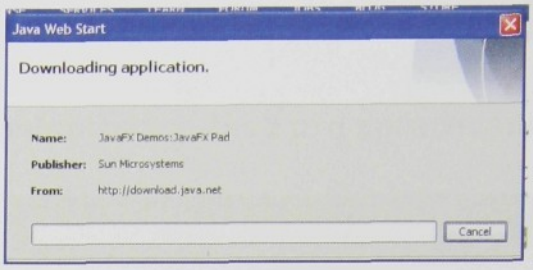
Ke svému běhu potřebuje aplikace nainstalovaný JRE na klientském počítači, kde interpreter JavaFX generuje Java bytecode.

Kód JavaFX Script velmi připomíná přístup ActionScriptu. Výhodou JavaFX je uživatelova možnost práce offline.

V procesu instalace aplikace je nejméně user-friendly, jelikož se spouští pomocí Java web start(součást JRE od verze 1.4), není součástí klientského prohlížeče a uživatel musí projít následujícími kroky před spuštěním aplikace:



Obrázek 28 - JavaFX start 1



Obrázek 29 - JavaFX start 2

Obrázek 30 - JavaFX start 3

Obrázek 31 - JavaFX start 4

JNLP = Java Network launching protokol je, jak již název napovídá, protokolem pro zavádění aplikace na síti. Jedná se o XML formát, který definuje, kde se nachází spouštěný JAR balíček, jméno třídy s metodou main a parametry.

11.5. Reakce W3C na roztříštěnost technologií RIA

Konsorcium W3C si všimlo roztříštěnosti technologií a tak, jak to udělala např. s DOM, se nyní snaží přístup vývoje web aplikací na straně klienta standardizovat pomocí nově vzniklé Web API Working Group (únor 2006). Ten má zajistit standardizované postupy pro upload souboru přes XMLHttpRequest, Drag and Drop, mapování jazyků klienta na DOM, atd. Druhá vzniklá skupina sjednotí jazyk pro definici UI.

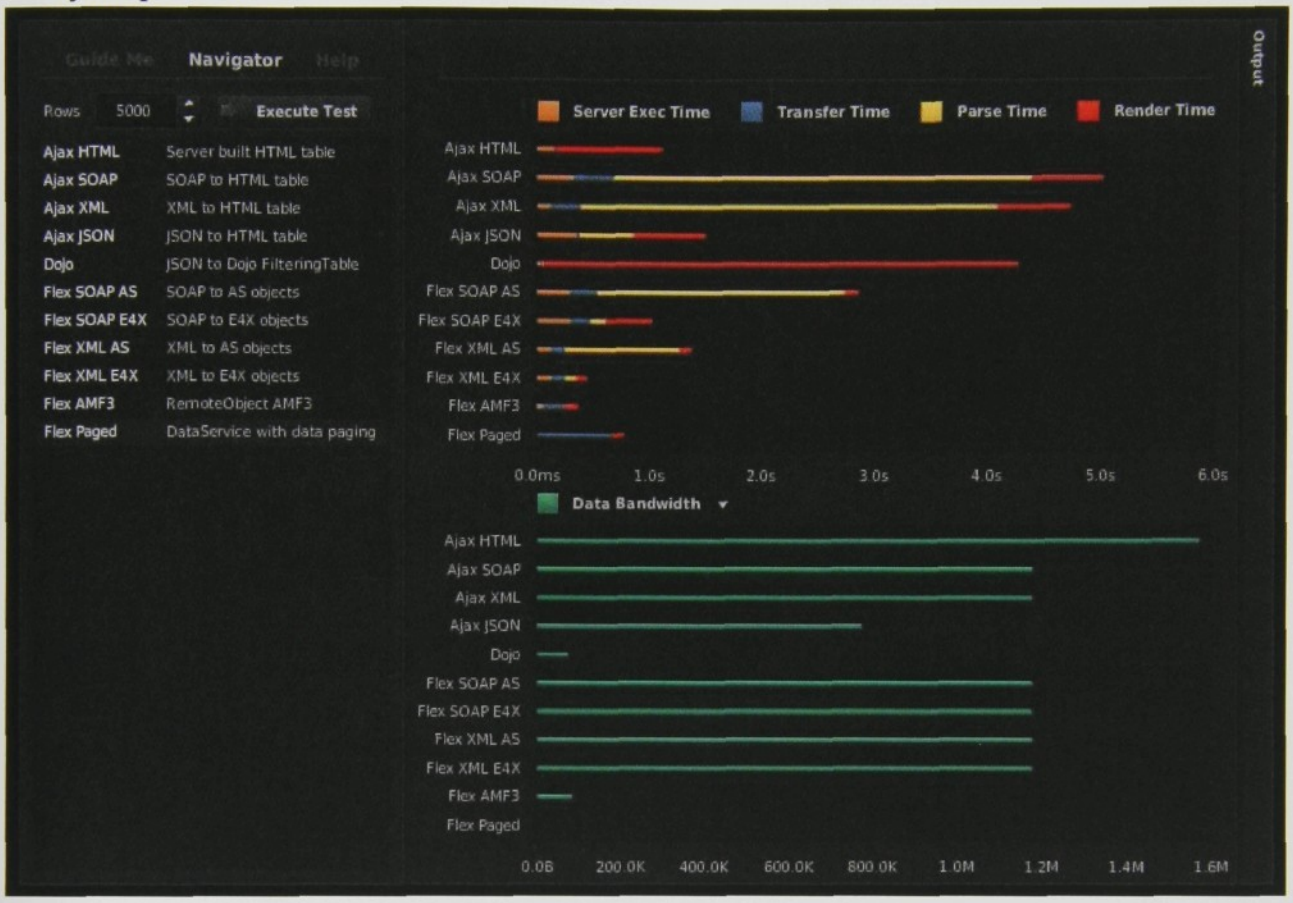
12. Výkonnostní srovnání technologií pro vývoj RIA aplikací

Benchmarking RIA technologií poskytuje zcela nový pohled, jakým by se měl též řídit výběr RIA platformy. Existuje několik studií a několik programů, zabývajících se tvorbou statistik pro různé RIA platformy.

Data, která zde uvedu, budou vycházet z testu Bubblemark (10), který je určen hlavně pro test vykreslování a z testu Census (11), jehož určení se zaměřuje spíše na stahování dat. V testu se zaměřím na technologie AJAX, Flex, SilverLight (CLR/JavaScript) a JavaFX.

12.1. Z HLEDISKA MNOŽSTVÍ STAHOVANÝCH DAT, PARSOVÁNÍ OBSAHU V PROHLÍŽEČI A RYCHLOSTI VYKRESLOVÁNÍ

12.1.1. Test množství stahovaných dat a jednotlivých časů pro zpracování výstupu Census:



Obrázek 32 - výkonnostní test RIA platforem Census

Tento test bohužel dokáže srovnat pouze 2 technologie – Ajax a Flex s použitím různých přenosových protokolů. Veškeré testy proběhly při stahování 5000 řádků dat, DOJO test nad 500 řádky v prohlížeči IE 6.

Obecně z něj vyplývá tvrzení, že zatímco nutný serverový čas je menší pro AJAX, renderování výstupu na klientu je neúměrně zdlouhavé oproti technologii Flex, který běží v rámci JIT kompilátoru Tamarin.

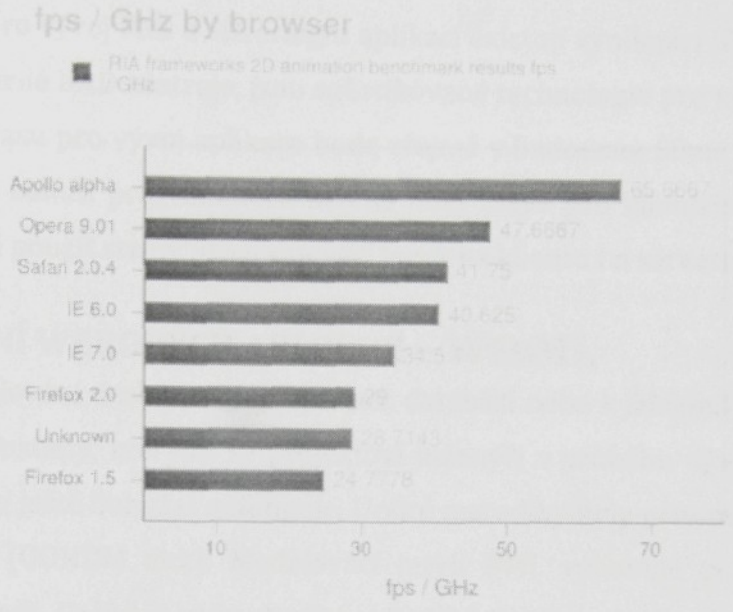
12.1.2. Testu rychlosti vykreslování Bubblemark

Z hlediska technologií:

```
JavaFX – 14 fps
Firefox + Silverlight (JavaScript) – 56 fps
Firefox + Flex – 62 fps
Adobe AIR – 62 fps
Firefox + Silverlight2 (CLR) – 202 fps
```

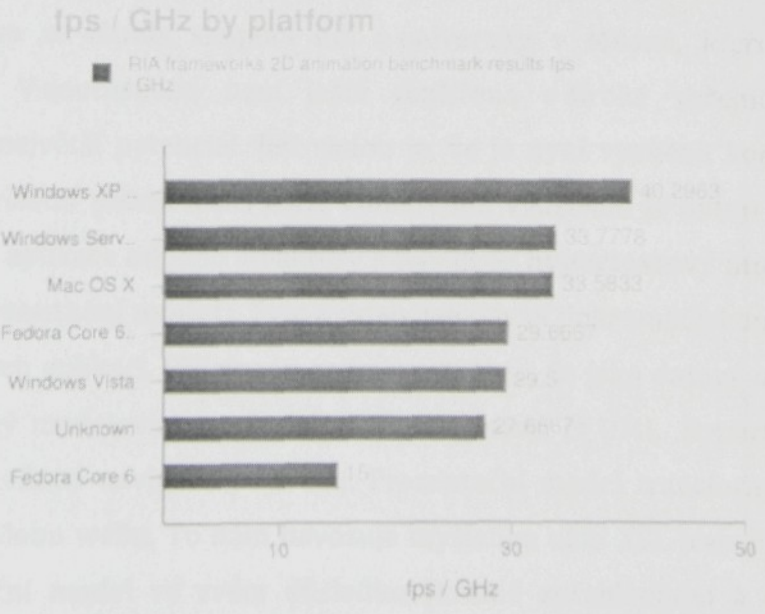
Vykreslování obsahu se měří s pomocí snímkové frekvence, což je frekvence, s jakou zobrazovací zařízení zobrazuje jednotlivé unikátní snímky. Jednotkou je Hz nebo FPS (frames per second).

Z hlediska rychlosti zpracování prohlížečů:



Obrázek 33 - Vykreslovací Bubblemark test dle prohlížečů

Dle platformy:



Obrázek 34 - Vykreslovací Bubblemark test dle OS

Z výše uvedených výsledků testu vyplývá, že SilverLight 2 bude se svým kompilovaným klientským kódem několikanásobně rychlejší než ostatní technologie a nejhůře na tom co do rychlosti bude JavaFX. Dále je patrné, že nejhorším prohlížečem pro RIA aplikace je

programátory oblíbený FireFox, jelikož velmi dlouho renderuje. Dále je vidět, že co do rychlosti vykreslování v OS předbíhá starší WinXP nové Windows Vista.

12.2. Z hlediska doby vývoje aplikace a stability

V tomto ohledu je významně pozadu technologie AJAX. Vývoj v javascriptu je poměrně časově náročný, náchylný k chybám a nutí testera k testům na všech významných webových prohlížečích. Zatímco pro vývoj Flex a SilverLight aplikací existují vynikající IDE, komponentové knihovny a další podpůrné RAD nástroje, jsou sofistikované technologie pro vývoj AJAX aplikací nedostupné. V oblasti času pro vývoj aplikace bude zřejmě v budoucnu SilverLight lepší volbou zejména díky tomu, že nenutí programátora učit se nový jazyk pro klientské programování a programátor bude moci použít stejný programovací jazyk na klientu i n serveru (CLR).

13. Modelování webových aplikací – WebML

Zatímco k modelování desktopových aplikací, databází nebo k jakémukoliv projektování (např.staveb) existují postupy, kterými se projektoví inženýři v průběhu vývoje řídí, na webu tento inženýrský postup ještě bohužel nefunguje. Určité metodiky byly přitom vymyšleny dříve než existoval Internet (OOHDM atd.). Modelovací jazyk UML vyhovuje podstatě webu jen v některých jeho modelech. Určitě je vždy vhodné začít USE CASE modelem, který nastíní, kdo a jak chce pracovat s aplikací. Zde ale vhodnost užití UML jako modelovacího nástroje pro webové stránky končí. V základu totiž nestaví na hypertextové povaze webu. Přitom v dnešní době, kdy se na web kompletně přenášejí velké informační systémy, je analýza chování webové aplikace vyloženě nutností. Toho si všimla skupina lidí z univerzity v Miláně, kteří přišli na trh s metodologií WebML. V současnosti není ještě rozšířena v široké veřejnosti, má však s komerčního hlediska největší potenciál. Její vadou je, že je nyní vyvíjena komerčně firmou WebRatio, která na ní vlastní patent a při jejím komerčním využívání je nutno platit. WebML odděluje **datový model** aplikace od jeho struktury a navigace(**hypertextový model**) a od jeho konečné prezentace(**prezentační model**). Tyto modely tak dávají dohromady komplexní nástroj pro modelování webových aplikací. Příjemnou zajímavostí je to, že jako datový model můžeme využít jakýkoliv oblíbený modelovací jazyk, např. ER diagramy, či UML. Hypertextový model zjednodušeně slouží k definici navigace po webu. Prezentační model transformuje předchozí modely na konečnou podobu webu. To nám navozuje myšlenku užití XSL transformací. A to je přesně to, co prezentační model ve svém důsledku (kromě zpřehlednění a dokumentace) umožňuje. Model WebML je totiž možné prezentovat do podoby XML, pro který existují DTD. Z něj je možné XSL transformací vygenerovat šablony jednotlivých stránek, dokonce spolu s značkovacím serverovým jazykem(Java, ASP.NET).

S WebML je tedy možné z modelu vygenerovat celý web i jeho dokumentaci. Určitou nevýhodou je samozřejmě menší svoboda kontroly nad generovaným kódem. Výhodou je čistota a rychlost tvorby. Ve vývoji je verze, která bude umět automaticky generovat Rich Internet Applications s ajaxovým a flashovým interfacem a bude zároveň kompatibilní se standardy sémantického webu. První implementace AJAXového WebML webu již je ke stažení a její dema je možné shlédnout na internetu[1]. Zajímavostí je jejich implementace „oken", tak typických pro desktopy na jedné z podstránek.

Jelikož backend1 mé aplikace využívá přístup „oken", otestovala jsem si toto demo a k mé radosti zjistila, že backend1 má oproti této implementaci několik výhod:

- Když velikost modálního okna u WebRatio přeteče klientskou výšku okna prohlížeče, formulář je natolik „vysoký", že se stává velmi nepřehledným a navíc díky tomu, že se vycentruje na obrazovku dle své výšky, skryje tak svou okenní lištu a uživatel nemá možnost jej uzavřít, popř. uložit tlačítkem vespod atd.
- Přístupnost (accessibility) – tj. navigace po formuláři pomocí klávesnice, nikoliv myši. Zatímco v mé aplikaci je možné okno uzavřít standardním ESCAPE tlačítkem, WebRatio tuto možnost neposkytuje.
- Možnost libovolného layoutu ve formuláři – zatímco v této práci je implementována možnost Flow Layoutu, tedy layoutu, kdy jsou prvky řazeny horizontálně za sebe dle své vlastní šířky, webratio vždy striktně řadí prvky horizontálně pod sebe.
- Použití nápověd „hint" k jednotlivým prvkům formuláře zcela chybí
- Chybí indikce práce na pozadí při asynchronním volání

WebML je technologií, která se zejména pokud přejde na open-source, rychle rozšíří. Nyní se však bohužel využívá převážně v akademickém prostředí.

14. Doporučované návrhové vzory pro vývoj webových aplikací, oddělení vrstev, vzory použité v praktické části

Vzor je obecné řešení problému, které se týká tvorby software. Nejedná se o knihovnu nebo kusy zdrojového kódu, nýbrž o doporučení jak tento typ problému realizovat. Vzory jsou dvojího druhu:

1. **Architekturální** – jsou vzory, které určují architekturu aplikace v jejím základu. Ze známých jmenujme klient-server, pipeline, peer to peer, service-oriented architecture, třívrstvá, MVC atd. O poslední jmenované se zmíním v dalším textu, jelikož se na webu stává v současné době doslova standardem tvorby webových aplikací.

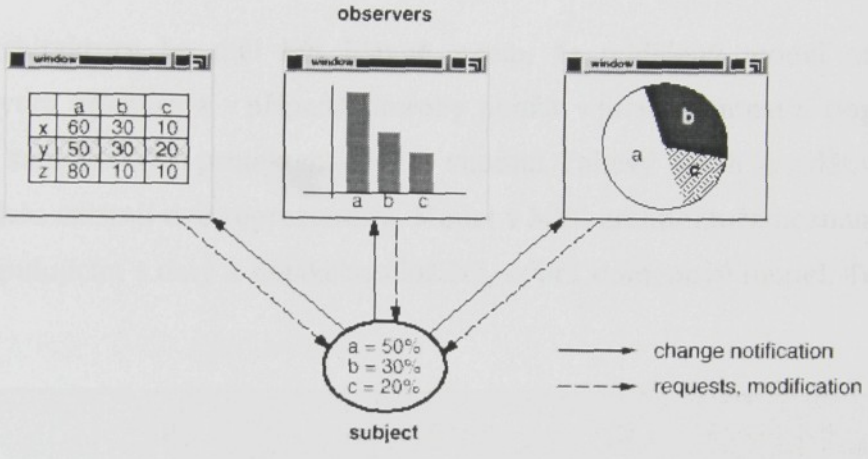
[1] http://131.175.57.119/ajaxdemo/sv1.do

2. **Design patterns** – návrhové vzory: typické pro tvorbu metodou OOP. Jedná se o doporučení realizace dílčí části kódu.

V této kapitole popíši vybrané vzory, které používá praktická část této práce. Přehled dalších návrhových vzorů používaných v této aplikaci je uveden např. zde: (8).

14.1. ObServer pattern

Zavádí možnost sledování změn u objektu tak, že když objekt změní stav, ostatní objekty na tuto změnu zareagují, přičemž nedojde k přímé vazbě od sledovaného objektu k těmto objektům. Vice například zde: (8).



Obrázek 35 - ObServer pattern

14.2. MVC (Model-view-controller) pattern, další rozdělení modelu na DAO, DAL a BLL

Viz příslušná část obrázku v kapitole 0. Jedná se o architekturální vzor založený na návrhovém vzoru observer. Jeho cílem je rozdělit aplikaci na tři samostatné dílčí jednotky, a to: Model – datová vrstva, Controller – Logická vrstva, View – Prezentační vrstva.

- "Modely" jsou MVC architektuře komponentami zodpovědnými za udržení stavu. Často je tento stav uchováván v databázi.
- "Pohledy" jsou v MVC architektuře komponentami zodpovědnými a zobrazování uživatelského rozhraní. Typicky zobrazují data z modelu.
- "Kontrolery" jsou v MVC architektuře komponentami zodpovědnými za zpracování uživatelských požadavků, manipulaci s modelem a vybírajícím vhodný pohled (view) pro vykreslení uživatelského rozhraní.

Správně navržená aplikace by měla umožňovat znovupoužitelnost vrstvy model, popřípadě controller. To znamená, že model vrstva nepřestane fungovat, změníme – li databázi nebo vzhled

webu. Vrstvy model a controller by měly tvořit základ pro více view (jedno pro web, druhé pro mobilní zařízení, jiné pro pda,...).

Význam MVC při vývoji webových sídel stoupá natolik, že si toho všiml i Microsoft, který upustil od svého pojetí oddělení aplikační logiky (code behind, code beside) a na začátku roku uvedl nový MVC framework pro ASP.NET. Java EE s tímto vzorem pracuje již velmi dlouhou dobu a jednotlivé vrstvy tvoří jednotlivé technologie: Model = JavaBeans, Hibernate, Spring, Controller = Servlety, JSF; View = JSP, JSF stránky. PHP framework Zend pro všechny vrstvy používá jazyk PHP, ovšem View vrstvu je možné prezentovat pomocí nějakého dostupného šablonovacího systému, např. Smarty.

Přínos MVC architektury by měl být hlavně v tom, že oddělený model má jít zcela osamostatnit od zbytku aplikace a v případě potřeby použít v jiném kontextu. Doporučuje se tedy, aby byl model samostatným projektem, je-li to vhodné. Takový návrh je vidět v této práci, kdy k jednomu modelu existují dvě reprezentace. Model v MVC architektuře neznamená přímo datový model manipulujícím s daty z nějakého úložiště, nýbrž doménový model. Ten se může dělit dále na:

- Controller
- View
- Model
  - Data access object (DAO)
  - Domain
    - Repositories - Data Access Layer (DAL)
    - Entities - Business Logic Layer (BLL)
    - Services - Business Logic Layer (BLL)

Následující termíny jsou pro profesionální programování databázových webových aplikací zásadní:

DAO je vrstva starající se o napojení dat. Může se jednat o různé persistentní frameworky, např. Hibernate a JDBC pro Javu, NHibernate pro .NET, či komponenta Zend_Db pro Zend framework PHP.

DAL je vrstva pro přístup k datům. ORM frameworky tuto vrstvu implementují a nabízejí programátorovi k použití. V případě, že není žádný ORM nástroj použit, je programátor nucen psát přístup k datům ručně (10): getKategorie(), getProdukty(), atd. Je tedy velmi vhodné nějaký ORM framework využít, už z důvodu ušetření času. Pro srovnání ruční časotvorby DAL a použití ORM frameworku, bych doporučila srovnání (11), které ukazuje, že při požadavku na přidání jednoho atributu do entity se dostaneme na poměr nutného času 1:10 ve prospěch ORM.

BLL – Bussiness Logic Layer. Business Logic – „obchodní logika" znamená v počítačové terminologii funkční logiku, jež řídí a spravuje přenos informací mezi databází a UI. Překlad je to volný, v praxi často používaný, ovšem ani v akademickém světě nelze nalézt jeho přesnou definici. **Business Logic Layer** již naproti tomu definovaná je a jedná se o vrstvu oddělující UI od Data access vrstvy. Vyměníme – li DAL, např. z důvodu použití jiného RDBMS, BLL musí zůstat funkční. BLL se nejčastěji používá pro validaci dat před jejich uložením do databáze.

14.3. Factory

Návrhový vzor, který určuje jakým způsobem rozhodnout až v průběhu programu o vytvoření instance konkrétní třídy. Je definován objekt, který se stará o způsob vytváření instancí podřízených tříd (implementuje factory metodu vytvářející produkty). Třída vytvářejícího objektu je definována jako abstraktní nebo jako konkrétní (přímo implementující factory method).

14.4. PRG (Post-redirect-Get) pattern

PRG pattern podrobně rozepisuji v kapitole 0.

14.5. ORM – objektově relační mapování

Je technikou pro konvertování dat mezi dvěma navzájem nekompatibilními technikami – OOP objektově orientovaném přístupu a relační databází. Vzniká tak jakási virtuální objektová databáze, k níž lze přistupovat pomocí OOP.

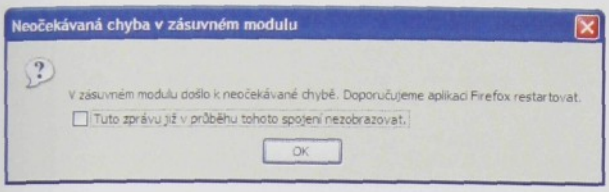
14.6. VO (Value object) pattern

Value object vzor je znám též pod zkratkou DTO – Data transfer object. Jedná se o návrhový vzor, který definuje obal pro přenos dat mezi dvěma softwarovými subsystémy a hraje velkou roli při návrhu RIA aplikací.

PRAKTICKÁ ČÁST

1. ÚVOD

Z počáteční analýzy vyvstala potřeba testování především dvou technologií – AJAXu a Flexu. Jakkoliv lákavé by bylo v této chvíli zahrnout též Microsoft SilverLight, mluvil proti němu fakt, že verze 2.0, která má právě přinést zajímavé možnosti v oblasti vývoje RIA aplikací je nyní ve fázi beta. Tento fakt, ale též stav demo aplikací a chybové hlášky, které se při testování beta verze projevily mě nepřesvědčily použít SilverLight pro součást této práce. SilverLight 2 by měl teoreticky vyjít v průběhu roku 2008, přesnější informace jsou nedostupné.

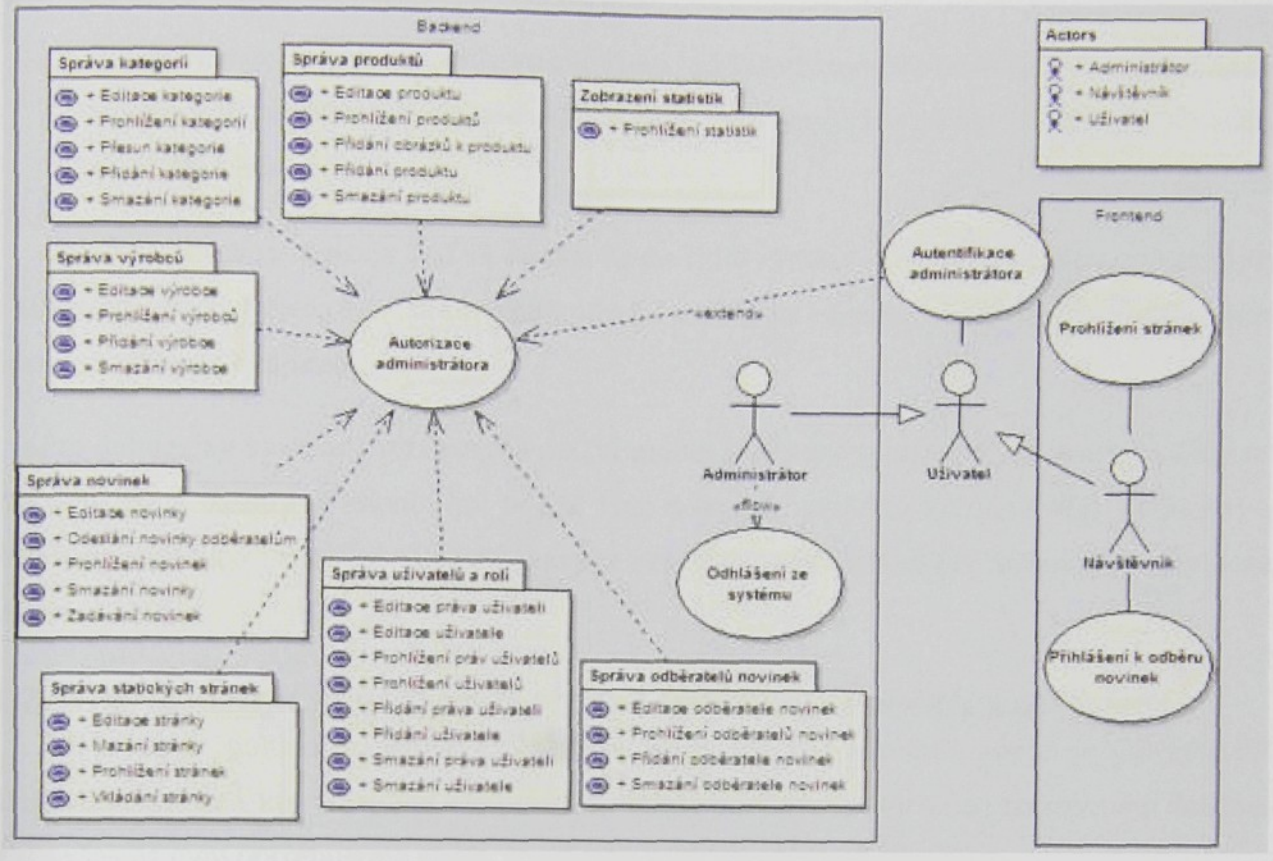


Obrázek 36 - Chybová hláška při startování SilverLight demo aplikace

AJAX naproti tomu je již velmi ostřílenou technologií, někteří kritici webových RIA aplikací dokonce tvrdí, že je to technologie zastaralá. S tím tak zcela nesouhlasím, i tato oblast se velmi progresivně vyvíjí a dá se s ní vytvořit aplikace podobné desktopu. Hlavním problémem je ve však doba vykreslování DOM struktury prohlížečem, proto mají AJAX aplikace pro uživatele delší dobu odezvy než pluginová řešení.

Flex jako nenápadnější představitel pluginové technologie je od února ve verzi 3.0 a nabízí největší možnosti v tvorbě RIA aplikací.

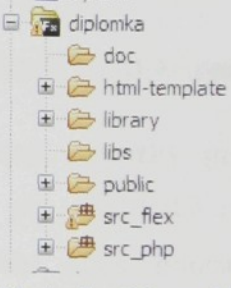
Modelový případ – správa referencí jsem pojmula jako správu několika základních modulů (funkcionalit) aplikace a přidala správu dalších funkcionalit. K zobrazení základního modelu aplikace se nám hodí UML diagram – Use Case model:



Obrázek 37 - use case package diagram

1.1. STRUKTURA PROGRAMU A POUŽITÁ IDE A DEBUGGERY

Program je tvořen jako „Flex project" v pluginu pro IDE Eclipse jménem Flex Builder 3. Eclipse je vhodným IDE pro vývoj RIA, jelikož díky možnosti užití mnoha různých kompilátorů máme možnost debuggingu klienta i serveru v jedné debug session.



Obrázek 38 - hlavní adresářová struktura aplikace

Hlavními adresáři jsou:

- **Public** – obsahuje soubory viditelné z internetu, hlavně však zaváděcí vstupní soubor (bootstrap file) a všechny zdroje aplikace (obrázky, javascripty, ...). Čítá 189 řádků vlastního kódu v PHP (2 soubory), 324 řádků v javascriptu (2 soubory), celkem obsahuje 1745 souborů (spolu s knihovnou Dojo)
- **Src_flex** – obsahuje klientskou část backendu 2 – především MXML a AS soubor, dále konfigurační XML soubor pro definici připojení ke službám. Čítá 4193 řádků kódu v 68 souborech.
- **Src_php** – obsahuje backend 1, frontend, webové služby, value objects a gateway k amfphp serveru. Čítá 7399 řádků kódu v 66ti souborech.
- **Library** – obsahuje knihovnu společný funkcí Dar_XXX. Dále obsahuje použité frameworky a amfphp server. Knihovna DAR_XXX čítá 4049 řádků kódu v 44 souborech.

Pro zajímavost uvedu, že knihovna amfphp (kód třetí strany) obsahuje 8804 řádků kódu v celkem73 souborech a knihovna Zend (kód třetí strany) obsahuje 377076 řádků kódu v 1514ti souborech.

Celkem jsem tedy napsala **16154 řádků kódu** (PHP+MXML+ActionScript+XML+Javascript) **v 182 souborech**. Nejsou započítány konfigurační soubory a backend v ASP.NET, o němž se více dozvíte v příslušné kapitole.

Pro debugging na straně serveru byl použit modul XDebug integrovaný do Apache a Eclipse. Pro debugging na straně klienta byl použit Flex debugger pro Flex (součást Flex Builderu) a FireBug plugin pro prohlížeč FireFox, který je skvělým pomocníkem při ladění Javascriptu na straně klienta.

1.2. Implementace CRUD aplikace pro správu zákaznických referencí

Konkrétnější pohled do jednotlivých částí, které aplikace uživateli nabízí je patrný z ER diagramu databáze, který najdete v kapitole 2.. Pro úvodní představu a pro porozumění dalšímu textu však již nyní potřebujeme vědět, co vlastně aplikace uživateli umožňuje.

Jednou z funkcionalit je **správa uživatelů**. Uživatelům se dají též **přidělovat práva** ADMIN/MEMBER/GUEST, přičemž do Backendu má přístup pouze ADMIN.

Spravovat se dají uživatelé, kteří se prostřednictvím frontendu přihlásili k **odběru novinek**.

Samotná správa referencí je rozdělena do čtyř částí: **Produkty, obrázky produktů, kategorie, výrobci**. Kategorie představují hierarchickou strukturu pro tématické rozdělení vytvořených produktů (referencí). Počet vnořených kategorií není nijak omezen. Část „Produkty" představuje vlastní předmět pro prezentaci ve frontendu. Každý produkt může být v libovolném počtu kategorií a nastavuje se u něj velké množství parametrů a ke každému produktu se dá uploadovat teoreticky libovolné množství obrázků, které se ukládají na server až po jejich úpravě programem. Část „výrobci" představuje výrobce produktů. Nepředpokládala jsem pouze prezentaci produktů jedné firmy/osoby, proto lze ke každému produktu přiřadit jiného výrobce.

Dalšími možnostmi je **správa** (vkládání, editace, mazání) **novinek**, které se zobrazují na frontendu a zároveň rozesílají odběratelům e-mailem **a správa statických stránek** – malé CMS.

Jelikož tato práce porovnává, kromě jiného, míru interaktivity s uživatelem, je velmi důležité JAK se jí docílí, méně již samotné množství nebo složitost jednotlivých spravovaných částí. Aplikace je navržena tak, že přidání dalších možností pro uživatele je velmi snadná a nevyžaduje jakoukoliv změnu architektury aplikace, jelikož ta je napsána velmi obecně. Většinou stačí

rozšířit nějakou hotovou abstraktní třídu či implementovat rozhraní a nový formulář je připraven k použití. Samozřejmě s narůstající složitostí spravovaného formuláře vzrůstá množství programového kódu. Pro ilustraci je tedy možné část „Produkty", která je velmi obsáhlá a umožňuje měnit značné množství parametrů, porovnat se složitostí malého objektu „Výrobci".

1.3. Routování URL

Routování je proces, kdy se vezme poslední část url (vše po tzv. base url) a dekomponuje parametry tak, abychom dokázali zobrazit požadovanou stránku. V ZF se nejčastěji používá pro oddělení modulu, kontroleru a akce, která má daný požadavek obsloužit.

Aplikace používá jak standardní routování balíčku Zend_Controller_Router_Rewrite, kdy se předpokládá URL ve tvaru modul/kontroler/akce/parametryAkce, tak vlastní routování na frontendu, kdy jsem potřebovala docílit, aby se v url zohledňovaly též vnořené kategorie (vhodné pro SEO). O toto routování se stará vlastní router Dar_Controller_Router_Route_Katalog a výsledná url může mít například tvar: `katalog/osobni-stranky/o-hudbe/vlastni-tvorba/pocatky/`, a akceptuje na konci i detail jedné jediné položky (reference): `katalog/osobni-stranky/o-hudbe/vlastni-tvorba/pocatky/prvni-kapela.html`.

Dále využívá vlastní routování pro součást „statické stránky"(malé cms), které je definováno předpisem: `:module/stranka/:urlrewrite/*` a též pro routování požadavků přes gateway(backend 2- Flex):

```
$router->addRoute('gate', new Zend_Controller_Router_Route(':name/gateway/*',
        array('controller' => 'gateway', 'action' => 'index')));
```

O další zpracování se pak stará objekt Zend_Controller_Request_Http.

Nastat může situace, kdy požadavek neodpovídá ani jedné definované routě (není „dispatchable"), proto jsem vytvořila plugin Dar_Controller_Plugin_NoRoutePlugin, který požadavek přesměruje na chybu 404- nenalezeno.

1.4. Společná knihovna Dar_

Aplikace používá některé obecné principy, které nejsou přímo její součástí a jsou použitelné v jakémkoliv jiném webovém projektu. Tyto funkce jsou proto odděleny do knihovny DAR a jejich význam je dán slovem za podtržítkem DAR_ (např. Dar_Image). Některé jsou zcela samostatné, jiné dědí z frameworku Zend a upravují jeho chování na míru aplikace (aplikací).

1.4.1. DAR_Base, e-maily

Třída Dar_Base obsahuje základní pomocné funkce. Definuje finanční operace (výpis měny v korunách, práce s DPH), operace s regulárními výrazy (vrací reg. výrazy pro validaci čísel, cen atd.), práce s URL(tvorba SEO friendly URL), export do CSV, odesílání e-mailů atd.

1.4.2. DAR_Image

Třída Dar_Image dle svého názvu poskytuje funkce pro práci s obrázkem. V aplikaci slouží k konverzi a kompresi, tvorbě zmenšených náhledů, ukládání obrázků na server a mazání obrázků ze serveru.

1.4.3. Dar_Controller, Dar_Db, Dar_Form, DAR_Validate, Dar_View_Helper, ...

Tyto balíčky modifikují a doplňují defaultní chování Zend Frameworku pomocí dědičnosti. Například třída Dar_Controller_Action dědí z Zend_Controller_Action a podobně. Pluginy jsou v bootstrap file zavedeny do aplikace a provádějí se při obsluze každého požadavku.

Dar_Controller

- Dar_Controller_Action_Helper_ViewRenderer - třída, starající se o jednotný layout aplikace. Jednotlivé akce pak definují pouze obsah content prvku.
- Dar_Controller_Plugin_AclPlugin - plugin, který se stará o autorizaci přístupu uživatele do právě požadované akce. Jedná se o prvek aspektově orientovaného programování a *více o něm se dozvíte v kapitole XXX(1.5).*
- Dar_Controller_Plugin_AuthPlugin - plugin, který se stará o autentifikaci uživatele - jeho identitu.
- Dar_Controller_Plugin_DbConnectPlugin - Nastavuje připojení k databázi a vysílá prvotní query „SET NAMES"
- Dar_Controller_Plugin_NoRoutePlugin - přesměruje na chybu 404 v případě nerozpoznaného URL
- Dar_Controller_Router_Route_Katalog - rozebírá URL dle regulárních výrazů v případě, že URL začíná „katalog"
- Dar_Controller_Action - velmi důležitá třída, ze které pak dědí jednotlivé kontrolery aplikace a sdílejí tak její funkcionalitu. Předává společné „view proměnné" do view, zasílá hlavičku UTF 8, poskytuje chráněné metody: _posliForm (pošle form na klienta, buď ajaxem či do view proměnné $frm), _posliStranku($title, $stranka = null) (pošle view na klienta, buď ajaxem či do view proměnné $frm), provádí stránkování záznamů a navrací offset, umožňuje export do PDF

Dar_Db

- Dar_Db_Table - Bázová třída pro modely, defaultní automatické ORM mapování nebylo v mnohém dostačující, proto přidány metody: fetchSum, fetchCount, fetchAllWithJoins, a několik funkcí pro práci se session daty objektu modelu
- Dar_Db_Table_Row - umožňuje naplnit řádek daty z pole

Dar_Form

Třídy v tomto balíčku rozšiřují funkcionalitu Zend_Form. Většinou nastavují decoratory, které potřebuje ke svému výstupu aplikace, případně atributy jednotlivých elementů. Tak se automaticky generují též atributy pro dojo forms.

Dar_Validate

Obsahuje vlastní validátory, které nenabízí ZF. Jedním z nich je Dar_Validate_PasswordConfirmation, který porovnává 2 hesla na to, zda se shodují, druhým Dar_Validate_Unique, který validuje, zda je daná hodnota unikátní proti databázi.

Dar_View_Helper

Sada pomocníků pro použití ve views: Stránkování, odesílací tlačítko, parsování zpráv, samotný obsah pro začlenění do jednotného layoutu

1.4.4. Rozhraní

Každý formulář šči strana, která chce používat ajaxové volání musí implementovat rozhraní:

Dar_IAjaxForm – metody pro zobrazení: getForm, insertForm, editForm, a pro obsluhu požadavku: vlozAction, smazAction, indexAction

Dar_IAjaxStrana – pošle stránku a její titulek přes JSON na klienta pomocí metody _posliStranku

Knihovna DAR_xxx obsahuje ještě další třídy, které jsou zdokumentovány přímo v kódu a není třeba je zde popisovat.

1.5. Moduly aplikace, Modul default

Aplikace je typicky rozdělena do modulů:

- Default – obsahuje funkce společné pro všechny moduly v rámci architektury MVC
- Admin – obsahuje backend pro administraci referencí
- Reference – obsahuje frontend

Modul default je navržen tak, aby funkce, které obsahuje, byly použitelné v jakékoliv (podobné) aplikaci. Proto obsahuje v podstatě hlavně:

- správu uživatelů, jejich přihlašování
  - přihlásit se totiž může i člověk ve frontendu. Ten nemá sice definováno právo pro vstup do administrace, ale frontend pro něj může nabízet prvky jako např. personalizace stránky, možnost rezervace volného designu apod. Je tedy zapotřebí sjednotit přihlašování napříč moduly a vstup do určitého modulu pak povolit pouze po kontrole práv přihlášeného uživatele.
- správa chybových hlášek
  - jedná se hlavně o chybu 404, která se může vyskytnout v kterémkoliv z modulů aplikace, o chybu při vstupu neoprávněného uživatele k určité akci napříč všemi moduly aplikace a o chybu 500 – Internal server Error.
- Gateway pro vzdálené volání objektů – přes něj prochází veškerá komunikace z flexového rozhraní. Jeho úkolem je především nalézt volanou službu a tuto službu spustit. O Gateway a službách se více rozepisuji v kapitole 5.2.1.
- Browser služeb – nemá význam pro chod aplikace, ale je vhodný pro debugging.

1.5.1. AOP – aspektově orientované programování pro ověřování práv

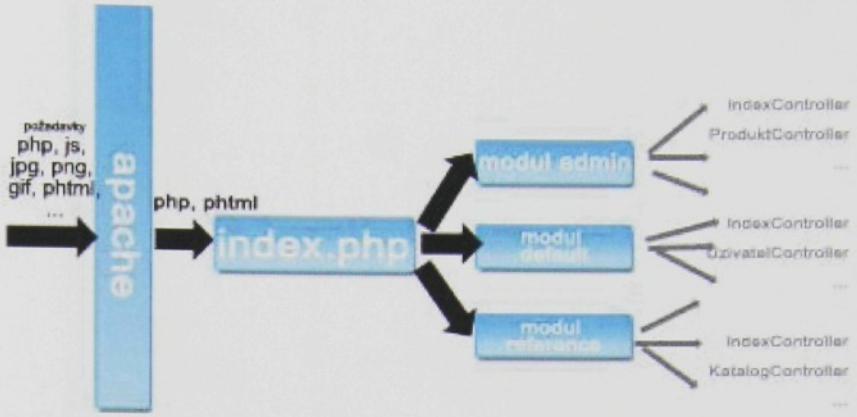
Pro společné části kódu napříč všemi třídami a metodami byla použita technika AOP – aspektově orientovaného programování. AOP je evoluční krok, jakési vylepšení objektově orientovaného programování, a to pro oblasti, které OOP nezvládá řešit efektivním způsobem. Některé věci lze velmi dobře zapouzdřit do objektů, s jinými to však jde velmi těžko. Odborně se jim říká průřezové koncerny a jak název odpovídá, jedná se o části kódu opakující se napříč metodami nebo skupinami metod v aplikaci. Nově vzniklý konstrukt zvaný „aspekt" tyto průřezové koncerny sdružuje do jednoho místa. Typickým představitelem je ověřování práv k určité akci. Přihlášený uživatel spouští akce v rámci aplikace a každá tato akce musí ověřit, zda má uživatel k jejímu vykonání právo. V případě, že ne, akci neprovede a zobrazí uživateli hlášku o nedostatečných právech. Bylo by tedy dobré nemuset na začátku každé takové metody volat např. metodu _checkRights, ale nějak toto automatizovat.

V aplikaci je proto zaregistrován plugin, který při provádění každé akce zkoumá, jaká má uživatel práva a porovnává je s právy potřebnými pro vykonání této akce. Minimální přístupová práva jsem vytvořila jak pro modul, controller, tak pro konkrétní akci. Nejdříve musí mít uživatel právo přístupu do modulu obecně. Poté je možné v každém controlleru definovat minimální právo pro jakoukoliv jeho akci a následně lze definovat právo pro jednu

konkrétní akci. Tyto kontroly není nutno zadávat v každé akci zvlášť, provádějí se při každém požadavku na akci.

1.6. Zavedení aplikace - Bootstrap, Application, Konfigurační soubor

V jediném zvenčí přístupném adresáři public nalezneme zaváděcí soubor: index.php. Přes tento soubor přechází všechny požadavky, které přijdou na base url, tj. url domény, případně se složkou, ve které se aplikace nachází. Získáváme tak možnost konfigurace aplikace z jednoho místa pro jakýkoliv požadavek, který na aplikaci přijde. Nastavení vyžaduje spolupráci Apache serveru s rozšířením mod_rewrite, kdy si v souboru .htaccess definujeme pravidlo, jaké požadavky mají být na index.php přesměrovány. Pravděpodobně budeme chtít vyloučit obrázky, javascripty atd. a naopak zahrnout veškeré php soubory.



Obrázek 39 - zavádění aplikace

Jak již bylo řečeno, v bootstrap souboru se zavádí konfigurace celé aplikace. Jelikož jsem chtěla docílit nejvyšší přehlednosti, jediným, co bootstrap soubor v naší aplikaci dělá, je vytvoření instance třídy Application, která se pak stará o jednotlivá nastavení v jejích metodách.

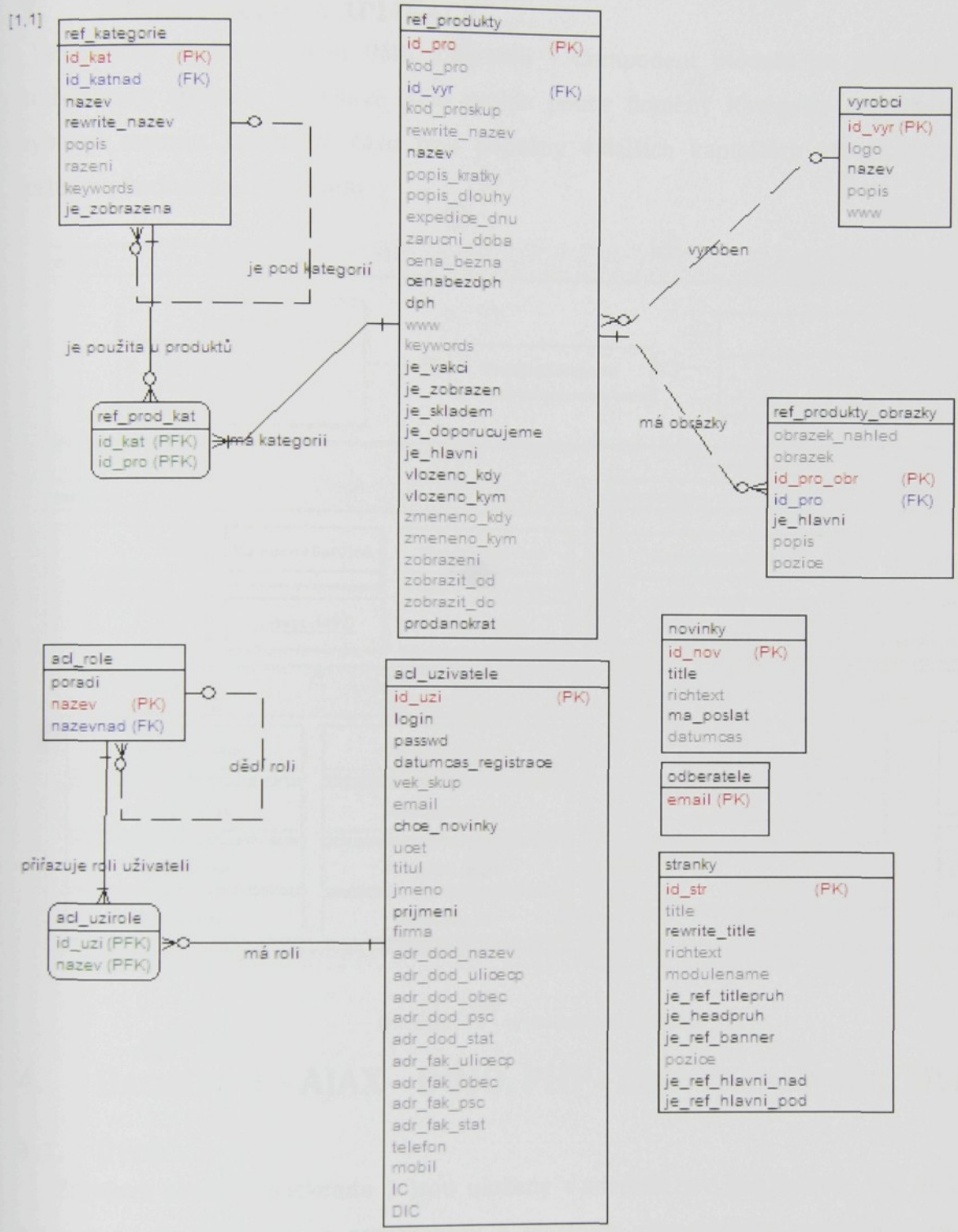
```
$app = new Application();
$app->bootstrap();
```

Metoda bootstrap provede následující: inicializaci aplikace (nastavení cest, base url, view, modulů, kontrolerů.

Dále provede setup rout, databáze, SMTP, ACL (přihlášený uživatel) a registruje příslušné pluginy. Pak již spustí požadovanou akci.

Konfigurační ini soubor aplikace app_config.ini je též přebírán třídou Application (při její inicializaci) a obsahuje definice pro nastavení: base url, e-mailů, databázové připojení, verze aplikace a minimální požadované právo přihlášeného uživatele v rámci modulů aplikace.

2. ER DIAGRAM DATABÁZE

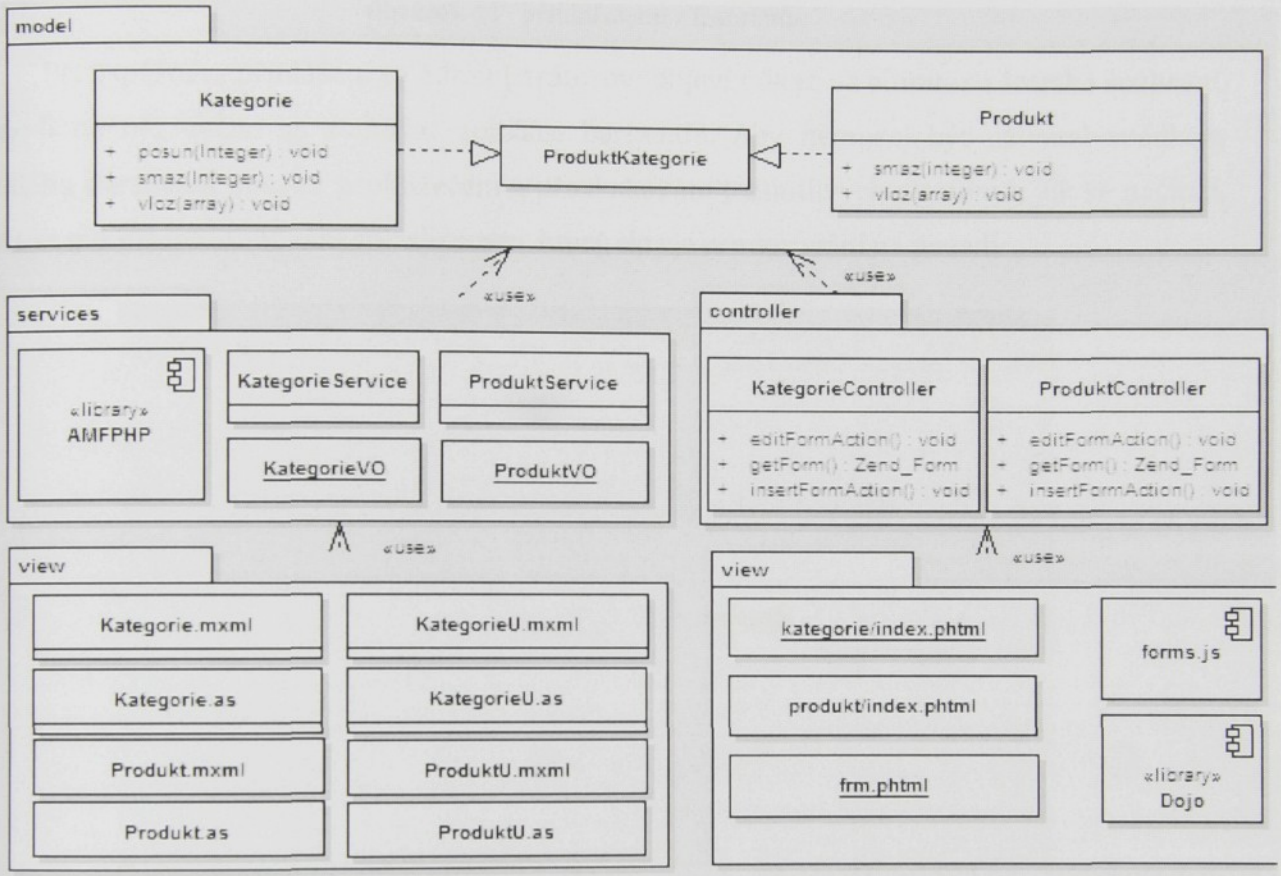


Obrázek 40 - ER diagram MySQL databáze

Entity bez prefixu jsou obecně použitelné, entity s prefixem acl_ se používají pro autentifikaci a autorizaci uživatelů, tabulky s prefixem ref_ (reference) jsou použitelné pro modul správa referencí. Detailní popis struktury databáze, referenční integrity, indexů, klíčů a datových typů lze nalézt na přiloženém CD.

3. ARCHITEKTURA APLIKACE

Ve funkčně zjednodušeném UML diagramu - Component modelu lze přehledně zobrazit architekturu aplikace. Modelově jsem zvolila pouze domény Kategorie a Produkty a jejich vybrané metody. Jednotlivé části jsou popsány v dalších kapitolách věnujících se detailně každému backendu a frontendu.



Obrázek 41 - architektura aplikace

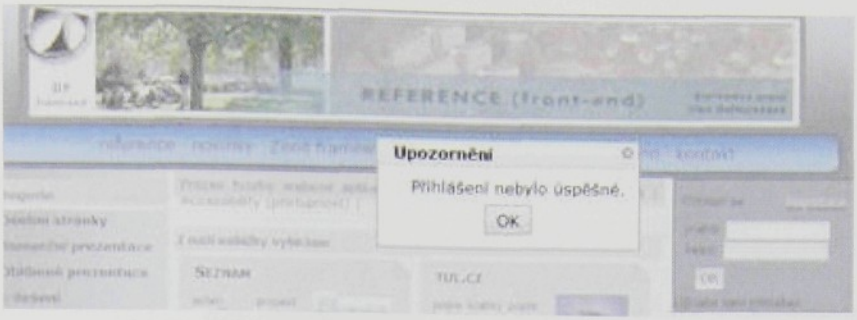
4. BACKEND 1 – AJAX – DOJO, PHP+ZEND FRAMEWORK, MYSQL

4.1. ÚVOD

Zdrojové soubory backendu 1 jsou uloženy v adresáři src_php/admin. Pro lepší ilustraci nejprve uvedu informace a screenshoty o vlastním uživatelském rozhraní a způsob jeho implementace ponechám na následující kapitoly.

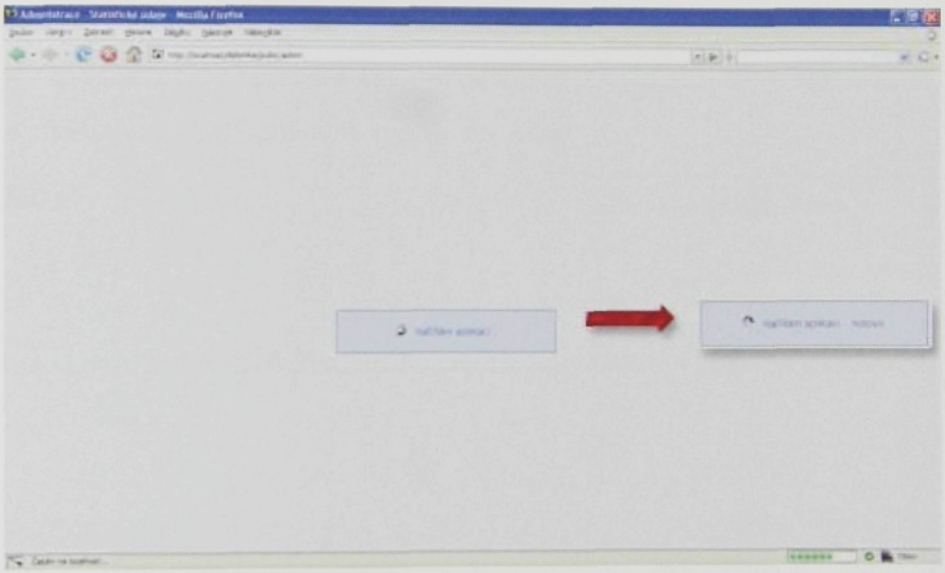
4.2. UŽIVATELSKÉ ROZHRANÍ

Jak již bylo řešeno, je modul pro přihlašování uživatelů společný, proto je možné přihlásit se do aplikace například i frontendu aplikace (modulu reference):



Obrázek 42 - přihlašování z frontendu

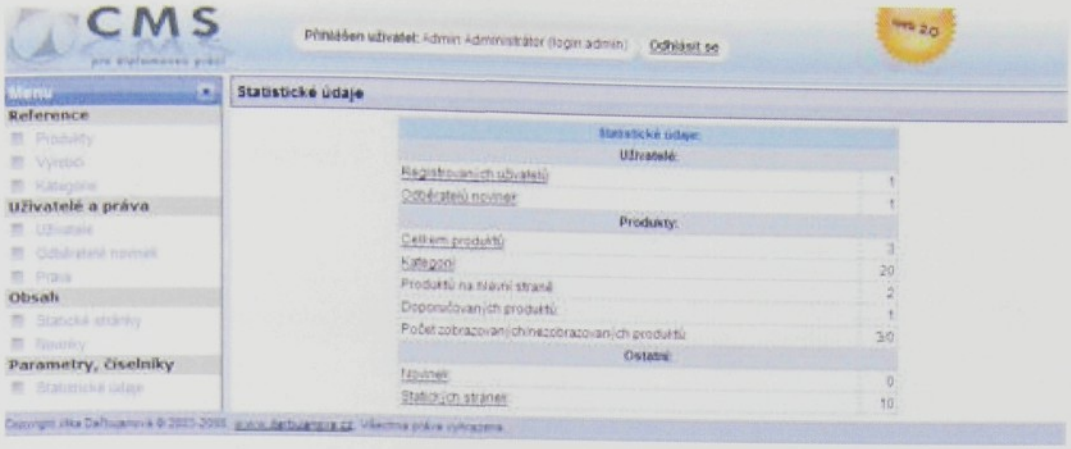
Při úspěšném přihlášení se administrátorovi objeví odkaz na administrátorské rozhraní. Klikne-li na něj, začne se stahovat aplikace backendu. Aby nemusel být uživatel svědkem pomalého parsování obsahu prohlížečem a přeskakování jednotlivých elementů, jak se načítají, objeví se mu v době stahování splashscreen, který skryje renderování na pozadí:



Obrázek 43 - stahování aplikace – splashscreen

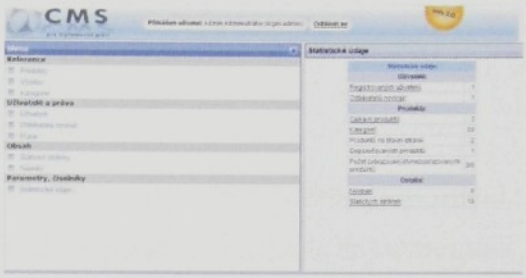
Stahování trvá zhruba 300ms, renderování pak 3s a stahuje se 600KB dat.

Po stažení se uživateli objeví úvodní okno se statistikou o uložených datech:



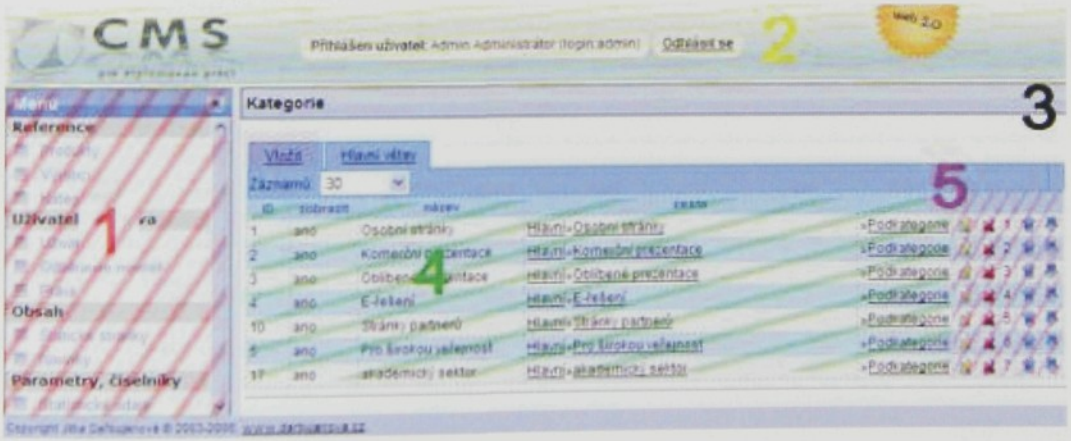
Obrázek 44 - úvodní okno - statistika uložených údajů

Od této chvíle je již veškeré volání serveru asynchronní, aplikace se překresluje po částech (Single Web Page Application). V tuto chvíli layout aplikace umí sledovat zvětšování, zmenšování prohlížeče, uživatel může zvětšovat či zmenšovat menu na úkor obsahu (SplitBar).



Obrázek 45 - splitBar

Popišme si nyní jednotlivé části aplikace tak, jak je bude vnímat její uživatel.

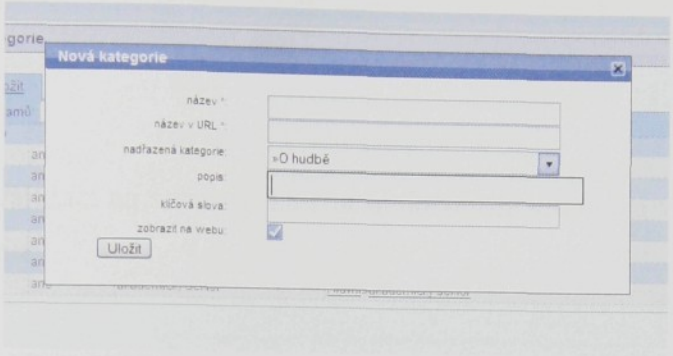


Obrázek 46 - backend 1 - jednotlivé části

Část 1 obsahuje nabídku vstupu do jednotlivých spravovaných částí. Nabídka se vždy zobrazí v části 3. Volání je asynchronní JSON. Část 2 je hlavička zobrazující přihlášeného uživatele s odkazem na jeho odhlášení. V části 3 pak uživatel stráví nejvíce času. Každá akce má definován svůj pohled, jakým dovoluje uživateli pracovat. Většinou však obsahuje grid hodnot z databáze, tlačítko pro vložení, editaci a smazání záznamu. Případně pak může umožňovat

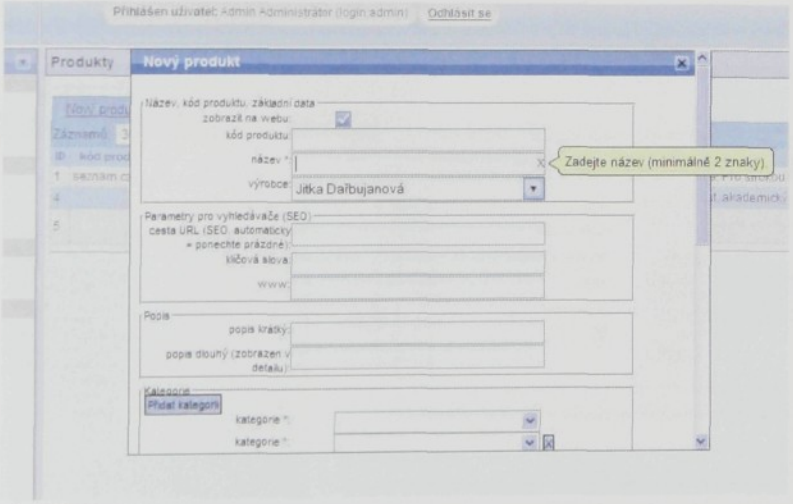
přesun záznamu a další. Obsahuje též stránkování záznamů, které způsobí též asynchronní volání.

Při požadavku na vložení či editaci záznamu se otevře modální okno „ala Windows" a zbytek stránky zešedne:



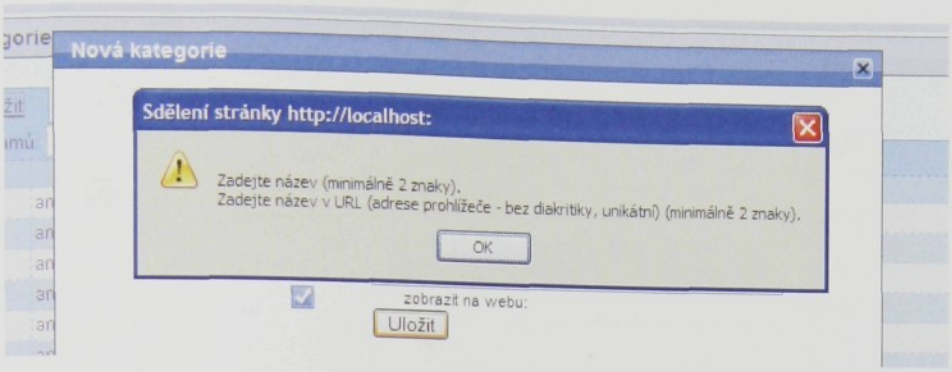
Obrázek 47 - nové okno kategorie

Přeteče-li okno výšku obrazovky díky své složitosti, zobrazí se spolu s postranní lištou. Bylo nutno dosáhnout vzájemné kompatibility mezi prohlížeči díky jejich rozdílné implementaci kaskádových stylů. Aby aplikace byla přístupná, je zde funkční klávesová zkratka pro zavření formuláře z klávesnice.



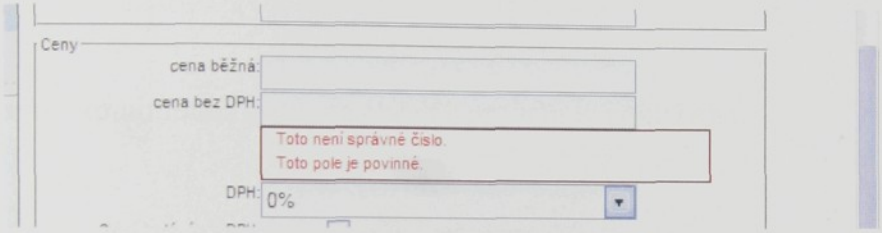
Obrázek 48 - nové velké okno produkty

Na obrázku 45 je též možno vidět hint, který napovídá, co je požadováno vyplnit. Tato validace je generována z jednoho místa – tam, kde se definuje Zend_Form se též přidávají regulární výrazy pro klientskou validaci. Neprojde- li formulář při odesílání validací na straně klienta, zobrazí se klasická hláška nad modálním oknem, která napovídá uživateli, co vše je třeba vyplnit:



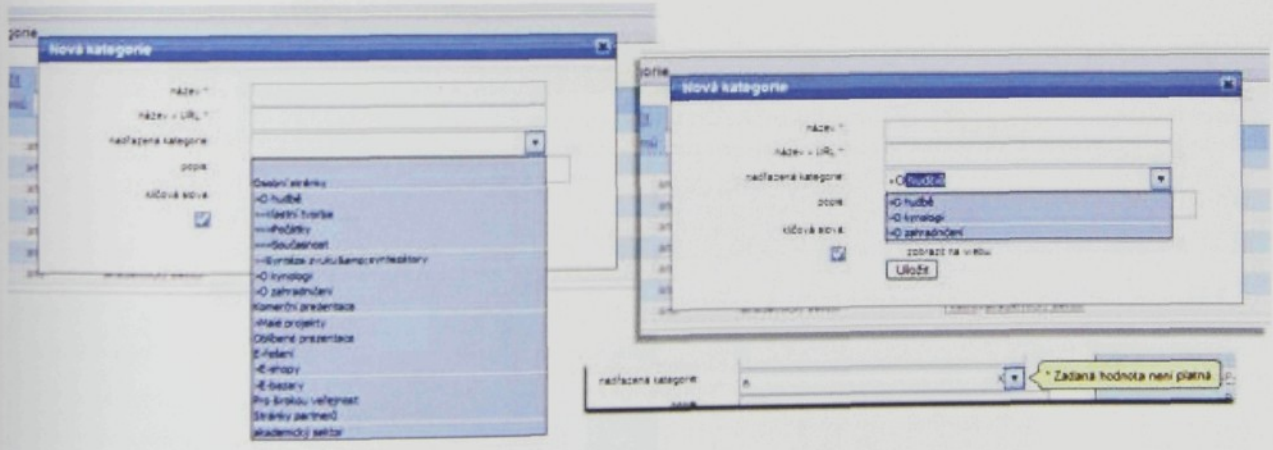
Obrázek 49 - klientská validace

Neprojde- li formulář validací na serveru, kterou neodchytila klientská validace, pak se zobrazí hláška u příslušného pole:



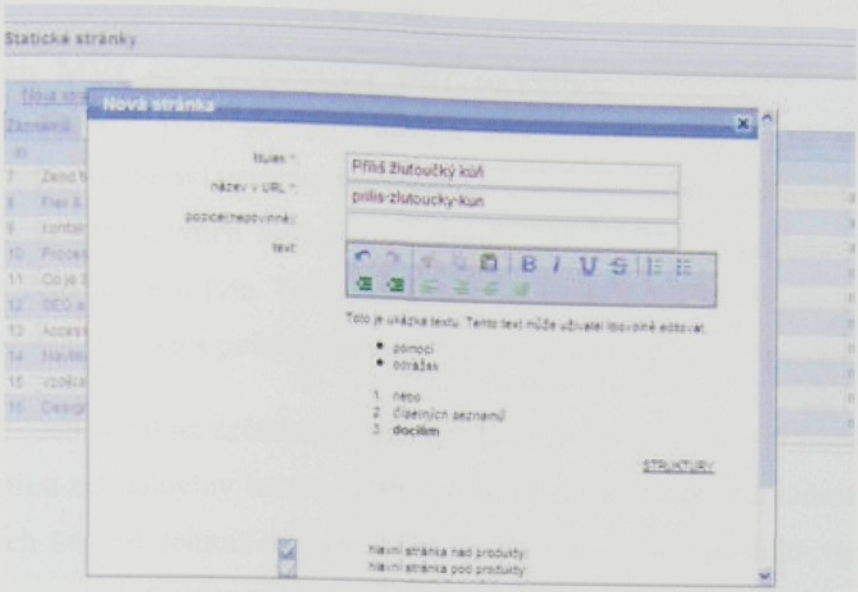
Obrázek 50 - validace na serveru

V rámci formulářů jsou použity prvky, které uživateli zpříjemňují práci s aplikací jako např. tento combobox s našeptáváním:



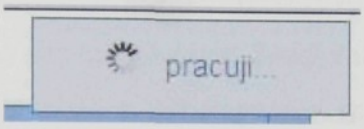
Obrázek 51 - combobox s našeptáváním

Nebo automatické vytvoření SEO URL z názvu a wysiwyg editor:



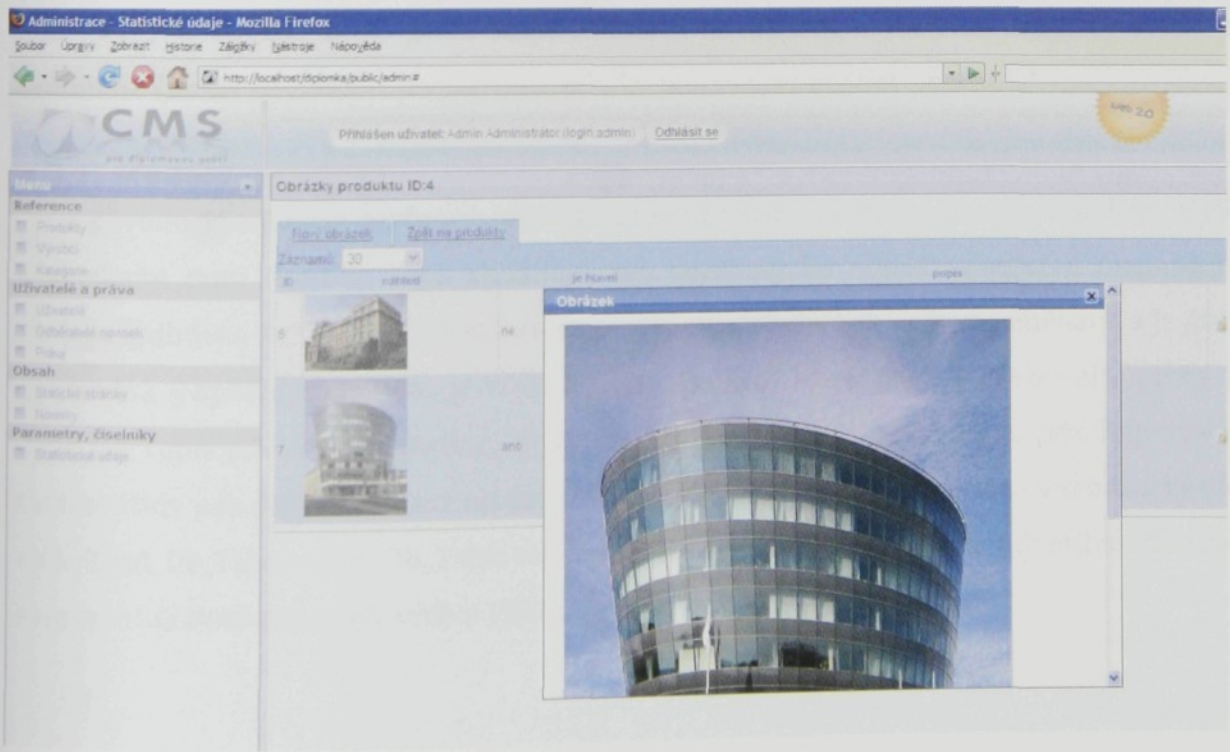
Obrázek 52 - wysiwyg editor

Asynchronní komunikaci indikuje prvek v pravém horním rohu:



Obrázek 53 - indikace asynchronního volání

Při vkládání obrázků se automaticky vytváří jeho zmenšený náhled i jeho zmenšená podoba, přesahuje-li originál určité meze:



Obrázek 54 - obrázky k produktu

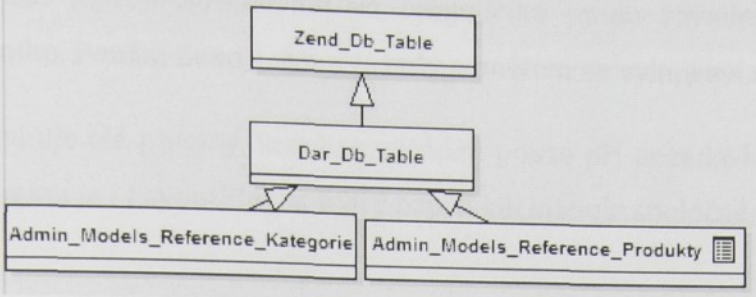
4.3. SERVER, MVC ARCHITEKTURA, PRG PATTERN

Na straně serveru jsem implementovala MVC architekturu Zend frameworku. Je tak oddělena prezentační (view) vrstva od aplikační (controller) a datové (model). Ve výsledku jsme získali třívrstvou architekturu aplikace na straně serveru. Každý požadavek typu POST dále implementuje PRG pattern (viz. Teoretická část, Kapitola 2.7.), takže po jeho vykonání dojde k přesměrování na stránku s požadavkem GET. Důvody jsou vysvětleny ve zmíněné kapitole.

Vývoj této části začal na začátku prosince, kdy byla aktuální verze frameworku 1.0. Vývojáři však nezaháleli a od poloviny ledna začali vydávat beta a release candidate verze ZF 1.5, a to zhruba každých 14 dnů. Jelikož mi bylo jasné, že má-li být práce po jejím vydání aktuální, bude nutné sledovat vývoj frameworku, byla jsem nucena sledovat změny ve všech vydaných meziverzích. To mělo za následek také to, že jsem si otestovala novinku, s kterou přišla verze 1.5, a která byla až na pár chyb dobře použitelná. Byli jí formuláře Zend_Form – knihovna pro serverově generované formulářové prvky se serverovou validací. Do té doby bylo nutné formuláře ručně psát ve View a pracně přidávat Zend_Validators. Jelikož však k této knihovně nebylo dostatečné množství materiálů a dokumentace, musela jsem vymýšlet způsoby, jak se použije. Navštěvovala jsem též webináře, které proklamovaly příchod Zend_Form a nějaké doporučované postupy zde byly uvedeny (ačkoliv se samozřejmě před vydáním ostré verze několikrát změnily). Výsledkem však je architektura, kterou náhodou nyní doporučuje samotný Zend a která přidává též některé nové možnosti oproti současné dokumentaci ZF 1.5 (vyšel na konci března 2008). Např. generovaná validace na klientu a možnost obsluhy array prvku formuláře (končí []) pomocí javascriptu, nikoliv pouze přes atribut „multiple".

4.3.1. Modely, sessions

Z důvodu všeobecné podpory na webhostinzích jsem se rozhodla pro databázi MySQL 5 s tabulkami typu InnoDB. Jako knihovnu pro automatické objektově-relační mapování byla zvolena knihovna Zend_Db. Tato knihovna prošla za poslední rok velkými změnami a je již dobře použitelná v aplikacích. Chybí jí však některé funkce, které jsem potřebovala, proto je od Zend_Db_Table odvozena třída Dar_Db_Table přidávající další funkcionalitu (viz. Kapitola 1.4.3). Od této třídy pak dědí jednotlivé modely. V terminologii, která byla popsána v teoretické části, je třída Zend_Db_Table a Dar_Db_Table součástí Data access vrstvy (DAL) a jednotlivé třídy modelu reprezentují Business Logic vrstvu (BLL).

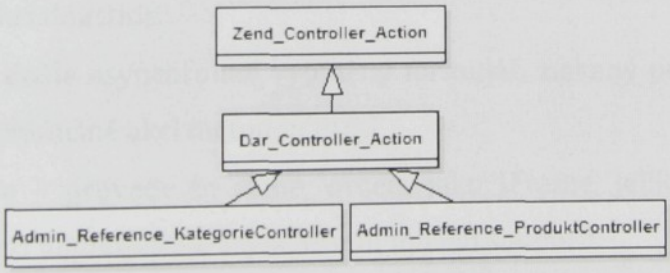


Obrázek 55 - diagram dědičnosti tříd modelu

Jednotlivé modely jsou implementovány jako singletony a mohou iniciovat vlastní session namespace, do kterého se pak předávají data související pouze s daným objektem.

4.3.2. Controller

Řadiče aplikace jsou odvozeny od třídy Dar_Controller_Action. Každý, který implementuje Dar_IAjaxForm, pak obsahuje kromě jednotlivých akcí obsluhujících požadavky též funkci getForm(), která navrací objekt typu Zend_Form. Tato funkce stačí k tomu, aby se vygeneroval vzhled, chování, serverová i klientská validace formuláře. Každé view má svůj vlastní řadič (KategorieController, ProduktController,...), ale řadič může též obsluhovat požadavky, které vlastní view nemají. Řadič je vstupní místo pro zpracování požadavku v rámci vlastní aplikace a volá dále modely, ve kterých vykonává příslušné akce. Model požadavky validuje a v případě chybného požadavku (chybějící požadovaný argument atp.) vyhodí výjimku, kterou controller dále zpracovává, většinou hlášku zobrazí uživateli.



Obrázek 56 - diagram dědičnosti tříd controlleru

4.3.3. View, Pluginy

View neboli pohledy/šablony slouží jako prezentační vrstva aplikace. Pro generování view byl použit jazyk PHP a knihovna Zend_View. Každá akce kontroléru vrátí nějaké vlastní vyrenderované view, které se předá do helperu Dar_View_Helper_Content. Pomocí třídy Dar_Controller_Action_Helper_ViewRenderer se pak obsah obalí jednotným layoutem dle požadavku (jiný pro ajax, jiný pro obyčejný http požadavek, jindy zcela bez obalu layoutem. Nutno dodat, že Zend framework 1.5 přišel s novou knihovnou Zend_Layout, která řeší zcela to

samé, ovšem nebyl čas k přeprogramování již fungujícího mnou zavedeného způsobu do současného standardního. Použití Zend_Layout je tedy námětem na vylepšení aplikace.

Každý modul obsahuje též pluginy, které se zavádějí pouze při požadavku na tento modul. Jedním z takových pluginu je i LayoutPlugin, který například iniciuje společné proměnné layoutu modulu.

4.4. KLIENT, ASYNCHRONNÍ KOMUNIKACE

Jako ajaxový framework byla použita knihovna Dojo, a to po výběru z mnoha widgetových frameworků. Důvody výběru byly následující: přístupnost, interaktivita, vzhled, mnoho použití, doporučováno též firmou Zend na jejích webinářích o tvorbě RIA aplikací. Další volba padla na framework ExtJS nebo YUI, ale raději jsem dala na doporučení firmy Zend.

Tím, že veškerá validace jednotlivých prvků formulářů je generována ze serveru, nebylo nutné psát mnoho javascriptového kódu. Napsala jsem pouze 2 další javascriptové soubory: common.js a forms.js. Common.js kontroluje jednotlivé prvky formulářů, ptá se jich, zda jsou validní a pokud ne, zobrazí uživateli hlášku.

Forms.js obsahuje následující funkce:

- doaction (url) – provede asynchronní volání typu GET na dané URL
- postform(url, formName) – provede asynchronní odeslání formuláře metodou POST
- form_smaz – požaduje potvrzení akce smazání záznamu, v případě, že je potvrzena, provede funkci doaction
- form_vloz – pošle asynchronně vyplněný formulář, získaný pomocí asynchronního požadavku, příslušné akci metodou POST
- form_vlozFile – provede to samé, ovšem jako IFrame, jelikož Javascript ve své současné verzi nepodporuje asynchronní upload souborů.
- form_otevri – otevře modální okno s asynchronně přijatým formulářem
- page_nacti – načte obsahovou část stránky backendu asynchronním voláním
- ... a další odpůrné funkce, které naleznete na CD

Princip asynchronního volání je jednoduchý (o to složitější je jeho realizace):

1. uživatel vygeneruje událost na klientovi, která ústí k zavolání javascriptové funkce.
2. Javascriptová funkce vytvoří a nakonfiguruje objekt XMLHttpRequest na klientovi a specifikuje funkci pro callback
3. XMLHttpRequest objekt zavolá server – asynchronní http požadavek.

4. Webový server zpracuje požadavek a vrátí odpověď ve formátu TEXT/HTML nebo JSON
5. XMLHttpObject zavolá callback funkci a přijme odpověď ze serveru
6. Klient obnoví HTML DOM reprezentující stránku s novými daty.

5. Backend 2 - Flex - Flex, AMFPHP1.9, PHP+Zend framework, MySQL

5.1. Úvod

5.1.1. Získávání dat ze serveru pomocí Flexu

Pro technologii Flex bylo zapotřebí zvolit způsob komunikace klienta se serverem, jelikož se jedná o komunikaci asynchronní, založenou na SOA architektuře (service oriented architecture). Klient pak využívá techniku RPC volání serveru, ať již se jedná o PHP, ASP.NET, ColdFusion a další. Ve Flexu je možné získávat data ze serveru třemi způsoby:

1. **Webové služby založené na REST** (HTTP GET, POST, HEAD, OPTIONS, PUT, TRACE a DELETE) *pomocí knihovny* HTTPService (`<mx:HTTPService>`)
   - Jejich výhodou je jejich jednoduchost. Není zapotřebí zvláštního API, na straně serveru může být jakákoliv technologie, která umí přijmout http požadavek a odeslat XML odpověď. Nevýhodou je nemožnost extrakce hlavičky od těla zprávy. Dá se však rozpoznat chybové volání od úspěšného. Dále není možné odeslat multipart post požadavek.
2. **Webové služby založené na SOAP** pomocí knihovny WebService (`<mx:WebService>`)
   - Serverová technologie je použita jako webová služba mající svůj standardizovaný WSDL popis. Její míra abstrakce je velká, proto se používá především ke komunikaci mezi nezávislými aplikacemi (B2B). Nevýhodou je velký přenos dat po síti a velké nároky na paměť na klientovi.
3. **AMF (Action Message Format) remoting services** pomocí knihovny RemoteObject
   - RemoteObject, neboli vzdálený či distribuovaný objekt je objekt, který lze volat z distribuované sítě. Komunikace probíhá prostřednictvím předávání zpráv.
   - V Javě například takový objekt musí implementovat rozhraní `java.rmi.Remote` a jak napovídá jméno balíčku, volá se přes Remote method invocation. Objekty jsou zavedeny do RMI registrů a volány

požadavkem: rmi://*hostname:port/remoteObjectName* přes protokoly JRMP nebo IIOP.

- Téměř každá technologie podporuje takové vzdálené objekty, a to ať svou vlastní knihovnou: (např. Remote .Net Object, CORBA, ColdFusion komponenty) nebo prostřednictvím software třetí strany – často opensource (AMFPHP, SabreAMF, Midnight Coder WebORB, Fluorine)

Jednotlivé způsoby se liší především v rychlosti odezvy uživateli.

5.1.2. Srovnání způsobů volání dat Flexem ze serveru a zvolený způsob

Provedla jsem následující test způsobů volání dle předchozí podkapitoly: opakované spouštění určitých funkcí ve smyčce o určitém počtu průchodů.

Výsledky dopadly takto:

1.test – jeden argument typu string posílaný funkci, která jej pouze vrátí, počet opakování 100:

Webové služby SOAP	Webové služby REST	Distribuované objekty
13062 ms	11907 ms	**4266 ms**

2.test – jeden argument typu Objekt (odvozená třída) posílaný funkci, která jej pouze vrátí, počet opakování 100:

Webové služby SOAP	Webové služby REST	Distribuované objekty
13563 ms	13547 ms	**4696 ms**

3.test – 1 argument – pole 1000 tříd z testu č.2 posílaný funkci, která jej pouze vrátí, počet opakování 5:

Webové služby SOAP	Webové služby REST	Distribuované objekty
69297 ms	48719 ms	**9343 ms**

Z výsledků vyplývá, že nejrychlejším způsobem komunikace Flex klienta se serverem je použití RemoteObject přes protokol AMF3. Proto byl v práci zvolen tento způsob.

Dále bylo nutné vybrat ze dvou způsobů předávání parametrů vzdálené funkci: **explicitní předávání parametrů a automatický binding**. Zvolila jsem automatický binding, který je vhodnější v případě užití Flex komponent, a ty aplikace využívá.

5.1.3. AMFPHP

AMFPHP je RPC toolkit pro PHP. Umožňuje PHP komunikovat s Flashem/Flexem, AJAXem s JSON a XML s XML-RPC. Hlavním posláním knihovny je:

1. Deserializace požadavku
2. Nalezení příslušné vzdálené třídy
3. Vytvoření instance nalezené třídy
4. Bezpečnostní ověření
5. Zavolání vzdálené metody s příslušnými deserializovanými argumenty
6. Serializace vrácených dat metody

Veškerá volání jdou přes soubor gateway.php, ve kterém je potřeba nastavit cestu k value objektům, adresář služeb a kódování. Hlavním důvodem pro použití beta verze 1.9 je jeho podpora protokolu AMF3 a podpora knihovny Zend_Db.

5.2. SERVER

Knihovna AMFPHP z předchozí kapitoly je jedna složka tříd, která ve svém hlavním adresáři obsahuje soubor gateway.php, přes který musí procházet všechny požadavky. Obsahuje též složku „services", ve které by se měly nacházet všechny remoting služby. To se však neshoduje s routováním MVC frameworku Zend.

První ideou bylo, že pro tuto klientskou část využiji stávajících modelů z předchozího backendu, abych tak simulovala výhody použití MVC architektury - znuvupoužitelnost kódu modelu při použití jiného klientského rozhraní, jednotná podoba URL a jednotné načítání balíčků funkcí autoload.

Tento cíl jsem nehodlala opustit, proto bylo potřeba vymyslet způsob implementace AMFPHP do MVC struktury Zend frameworku tak, aby se brána chovala jako controller.

Vytvořila jsem tedy GatewayController (který se vždy z klienta volá s příslušnými parametry), který nastavuje vlastnosti gateway (kódování atd.) a vytváří její instanci. K tomu bylo zapotřebí vytvořit router, který pozná jakou gateway (v jakém modulu) voláme. Další nutnou podmínkou je nastavení systémových cest- zatímco pro všechny předchozí balíčky ji bylo možné zadat relativně, kvůli balíčku amfphp se musí zadat absolutně. Pak se teprve totiž může využít autoloadingu modelů dle jejich jména (Admin_Model_Reference_Kategorie se ve skutečnosti nachází v admin/model/reference/Kategorie.php, to samé platí pro value objects).

5.2.1. Služby a value objects

Samotné služby, na které přes Gateway přicházejí požadavky volání metod s danými parametry, jsou uloženy v každém modelu ve složce „services". Každá složka „service" má ještě podsložku „vo" – value objects, kde jsou uloženy objekty pro vzdálený přenos. Tyto objekty se svojí strukturou i pořadím jednotlivých členských proměnných musí přesně shodovat se strukturou value objects v ActionScriptu. Zde vidíme nevýhodu nutnosti užití jiné technologie na straně klienta a na straně serveru – co se valueobjects týče, vzniká duplicitní kód. V této oblasti očekávám změnu po nástupu klienta SilverLight 2 +klientské CLR se serverem s .NET frameworkem. Definice value objects pak bude moci být psána v jednom jazyce (C#) pro klienta i server.

Navržené řešení – efektivní spojení technologií, je přínosem této práce v oblasti RIA aplikací, založených na PHP a Flexu a je možné jej nalézt na přiloženém CD. Jako bonus lze brát též implementaci prohlížeče služeb do BrowserController v rámci struktury Zend Frameworku. Tento prohlížeč je napsán ve Flexu a je dodáván též jako součást AMFPHP.

5.3. Komunikace Flex klienta se serverem AMFPHP

Na straně serveru máme gateway – bránu, která čeká na jednotlivé požadavky, ty díky AMFPHP nasměruje na danou službu a její metodu. Metoda se provede (často se jedná o jednořádkové zavolání modelu – zde je vidět výhoda znuvupoužitelnosti kódu) a vrátí výsledek. Z principu RPC vyplývá, že během práce serveru je klient blokován a nepovolí jiné akce. Klientovi odpoví buď úspěšným voláním s předanými výsledky volané metody, nebo (a to i po uplynutí timeoutu) chybovou hláškou („service unavailable"). Na straně klienta jsou pro samotnou komunikaci vytvořeny dva důležité soubory:

- Services-config.xml – v kořenu aplikace klienta, struktura definována technologií Flex
- RemoteConnection.as – ve vlastní knihovně „library" (bude o ní zmínka v další kapitole), vlastní statická třída, která je použitá pro vzdálené volání napříč klientskými MXML a AS soubory.

5.3.1. services-config.xml

V tomto souboru se nastavuje konečné url pro volání služeb. Definovat se můžou kanály pro AMF, pro WebService a pro Httpservice. Kompilátoru Flexu je třeba říci, aby nastavení použil direktivou při kompilaci: - services „service-config.xml". Náš soubor vypadá takto:

```
<?xml version="1.0" encoding="UTF-8"?>
<services-config>
  <services>
    <service id="amfphp-flashremoting-service" class="flex.messaging.services.RemotingService"
     messageTypes="flex.messaging.messages.RemotingMessage">
      <destination id="amfgatewaydefault">
        <channels>
          <channel ref="channel-amf" />
```

```
    </channels>
    <properties>
      <scope>session</scope><source>*</source>
    </properties>
  </destination>
 </service>
</services>
<channels>
  <channel-definition id="channel-amf" class="mx.messaging.channels.AMFChannel">
    <endpoint uri="http://localhost:80/diplomka/public/gateway/index/modul/admin"
   class="flex.messaging.endpoints.AMFEndpoint" />
 </channel-definition>
</channels>
</services-config>
```

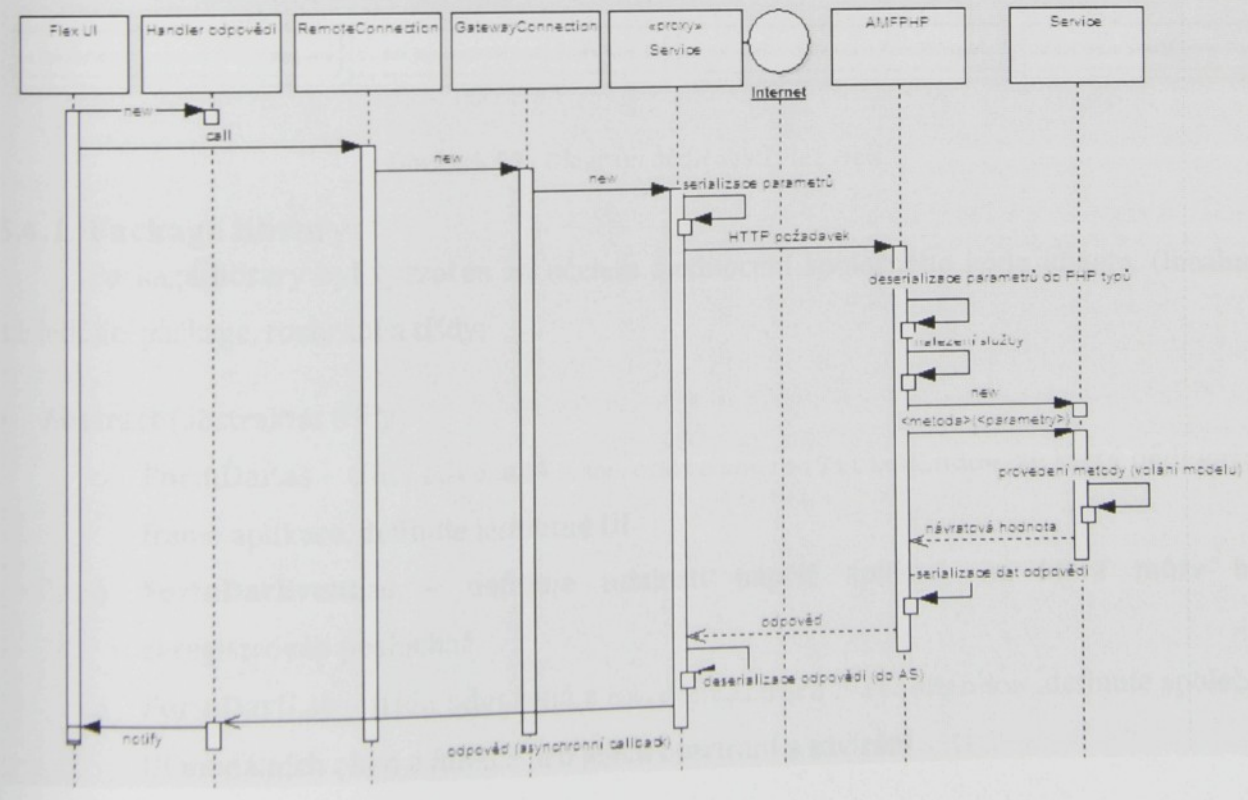
5.3.2. RemoteConnection.as

Třída remoteConnection má 2 metody. Hlavní je definována takto:

```
public static function call(service:String, metoda:String,
callBack:Function, nameVO:Object=null):void;
```

Nejdříve vytváří mx.rpc.remoting.mxml.RemoteObject s cílem amfgatewaydefault. Jemu pak nastaví odkaz na funkci pro nezdařené volání (pouze zobrazuje Alert) a argument „metoda", který přetypuje na typ mx.rpc.remoting.mxml.Operation. Po úspěšném volání zaregistruje parametrem předanou funkci callback. Dále již odesílá požadavek na server, a to buď s parametrem nameVO-jméno value objektu, nebo bez parametrů.

5.3.3. schéma komunikace

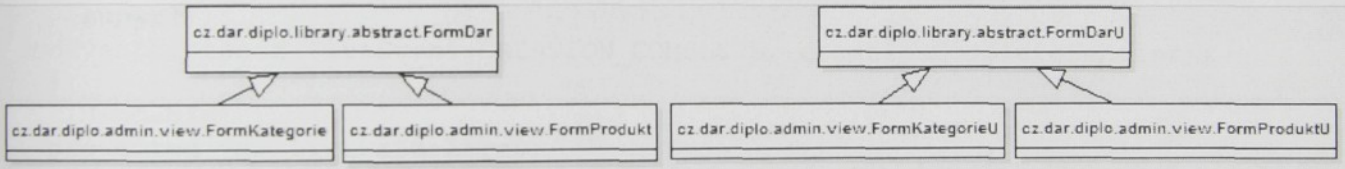


Obrázek 57 - sekvenční diagram Flex remotingu s AMFPHP

5.4. KLIENT

Struktura MXML a AS tříd odpovídá balíčkům tak, jak je zná např. Java. Celý klient je umístěn v package cz.dar.diplo.*. Opět zde najdeme tři moduly: admin, reference, common (default nelze v package použít, jelikož jde o vyhrazené slovo). V každé složce je opět struktura controller, model, view. Nutno zmínit, že pro opravdovou implementaci MCVC ve Flexu existuje oficiální framework CairnGorm, ten však ještě není dostupný pro verzi 3.0, která vyšla na konci února a je použita v této práci.

Model obsahuje value objekty na straně klienta. Každý value object se v třídě odkazuje na svého protějška na serveru definicí `[RemoteClass(alias="Admin_Services_Vo_OdbNovinekVO")]`. Ve View nalezneme MXML a AS soubory definující uživatelské rozhraní. Pro oddělení kódu a definice UI je použita technika „code behind", tedy GUI v MXML a reakce na události v ActionScriptu3.0, propojení je definováno direktivou `[Bindable]`. Jednotlivá View dědí hlavní funkcionalitu od třídy `cz.dar.diplo.library.abstract.FormDar` (frame) a `cz.dar.diplo.library.abstract.FormDarU` (modální okna). Tím jsme se dostali k nutnosti popisu package library před samotným popisem view.



Obrázek 58 - Diagram dědičnosti Flex view

5.4.1. Package library

Package library byl vytvořen za účelem sjednocení společného kódu klienta. Obsahuje následující package, rozhraní a třídy:

- **Abstract** (abstraktní třídy)
  - **FormDar**.as – třída odvozená z `mx.containers.TitleWindow`, ze které dědí každý frame aplikace, definuje jednotné UI
  - **FormDarEvent**.as – definuje události napříč aplikací, na které může být zaregistrován posluchač
  - **FormDarU**.as – třída odvozená z `mx.containers.TitleWindow` ,definuje společné UI modálních oken a funkce pro jejich otevírání a zavírání
  - **IFormDarU**.as
  - **Shared**.as – pomocné funkce např. vytvoření SEO Url z nadpisu
- **Assets** (gif soubory – nový, editace, smazání,...)

- **Components** (komponenty třetích stran např. toolbar a skiny)
- **Renderers** (pomocníci pro referování gridu)
    - **ItemRendererBooleanCZ**.mxml – nahradí boolean hodnotu za ANO/NE
    - **ItemRendererNahled**.mxml – vytvoří obrázek namísto URL k obrázku
- **Service**
    - **RemoteConnection**.as (bylo popsáno v kapitole XXX.).

5.4.2. views

Jednotlivá view zde popisovat nebudu, je možné je nalézt na přiloženém CD. Pro každý frame existují 2 soubory, jeden MXML a druhý AS, to samé pro každé modální okno, pomocí kterého se přidává a edituje záznam. Nejjednodušší frame (ve své ořezané verzi) vypadá takto:

ActionScript 3.0:

```
package cz.dar.diplo.admin.view
{
  import cz.dar.diplo.admin.model.vo.VyrobceVO;
  import ...
  public class FormVyrobci extends z.dar.diplo.library.abstract.FormDar{
    public var btnNovy:Button;
    public var ...
    public function FormVyrobci() {
      super();
  addEventListener(FlexEvent.CREATION_COMPLETE, creationCompleteHandler);
    }
     [Bindable]
    protected var listData:Array;
    private function creationCompleteHandler (event:FlexEvent):void {
      getData();
    }
    protected function setDataVybrana(e:Event):void {
      btnEdit.enabled = (dgMain.selectedIndex != -1);
      ...
    }

    protected function getData():void     {
      RemoteConnection.call("VyrobciService","getList",
                function(event:ResultEvent):void {
                            listData = event.result as Array;
                            setDataVybrana(event);

      });
    }

    protected function openForInsert():void{
      var vWin:FormVyrobceU = new FormVyrobceU();
      ...
    }

  openForEdit, zeptejSmaz, ...
}
```

Zde je vidět způsob, jakým se volá metoda navracející seznam výrobců do gridu.

MXML:

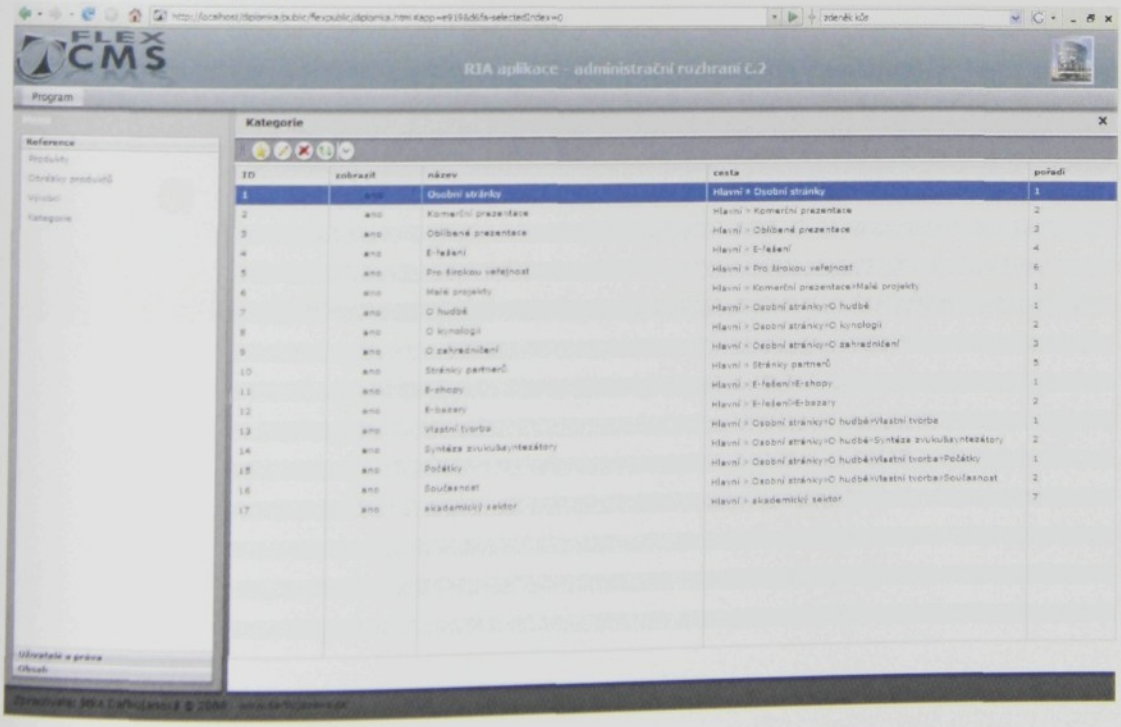
```
<?xml version="1.0" encoding="utf-8"?>
<dar:FormVyrobci title="Výrobci"
  xmlns:mx=http://www.adobe.com/2006/mxml"  xmlns:dar="cz.dar.diplo.admin.view.*"
  xmlns:fc="http://www.adobe.com/2006/fc">
  <fc:DockableToolBar>…</fc:DockableToolBar>
   <mx:DataGrid id="dgMain" change="this.setDataVybrana(event)"
 dataProvider="{this.listData}">
     <mx:columns>
      <mx:DataGridColumn headerText="ID" width="22" dataField="id_vyr"/> …
     </mx:columns>
 </mx:DataGrid>
</dar:FormVyrobci>
```

Definice modálního okna je v podobném stylu, např. volání vzdálené metody uložení výrobce vypadá takto:

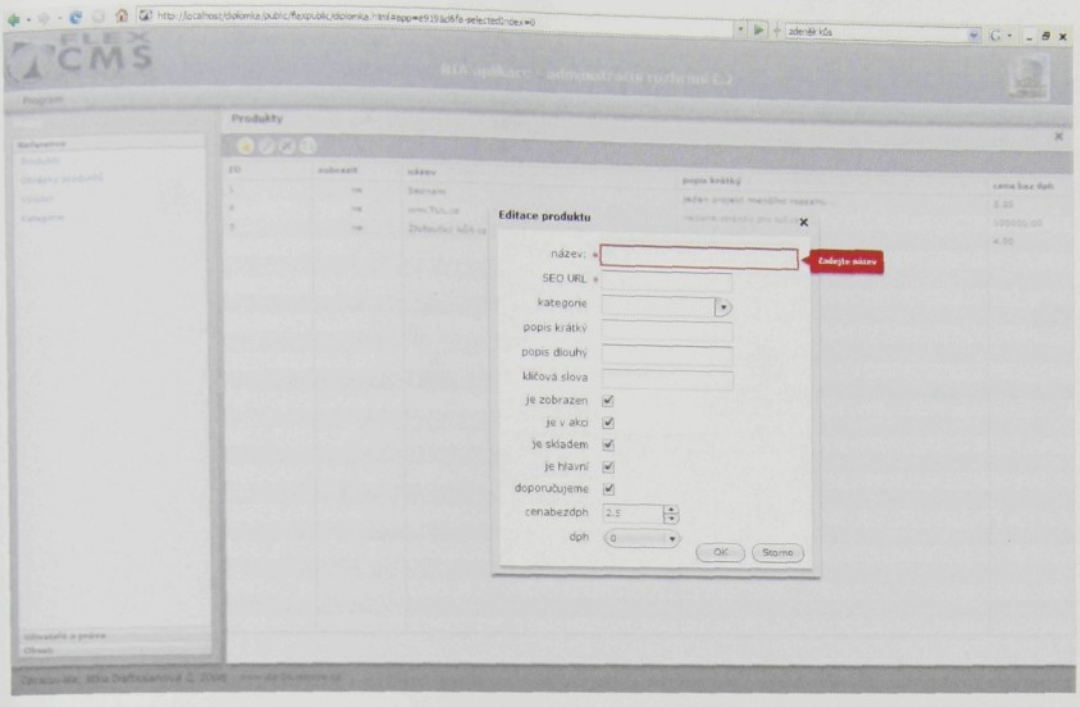
```
protected function saveVyrobce():void {
  var vVO:VyrobceVO = new VyrobceVO();
  vVO.nazev =  edtNazev.text; …
  RemoteConnection.call("VyrobciService","saveVyrobce",afterSave, vVO);
}
```

Všechny FormDar jsou dockovány do hlavní aplikace, která je definována soubor diplomka.as a diplomka.mxml.

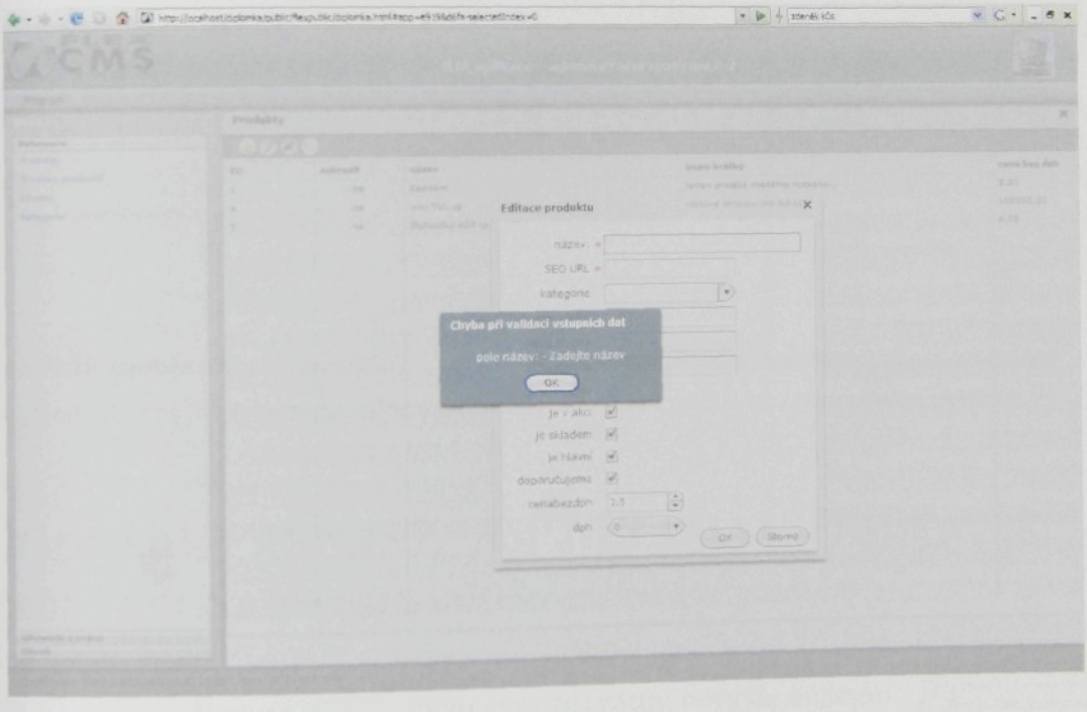
5.5. UKÁZKA APLIKACE



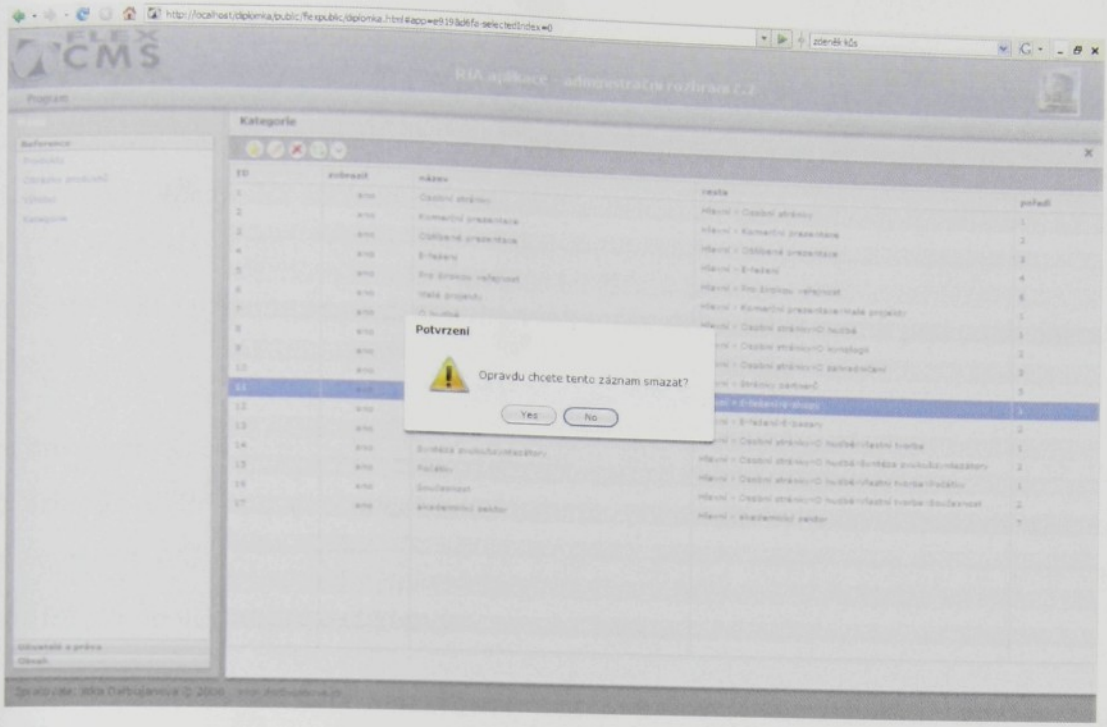
Obrázek 59 - vzhled backendu ve Flexu - grid kategorií přijatý RPC voláním



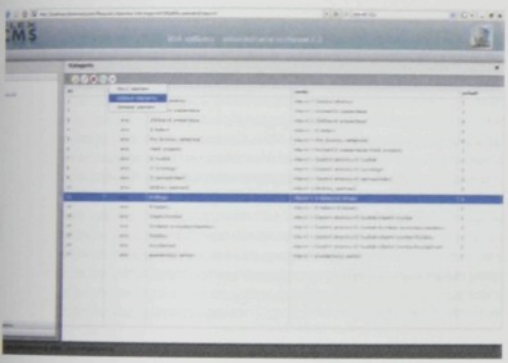
Obrázek 60 – Flex: modální okno pro vkládání produktu, nápověda hint



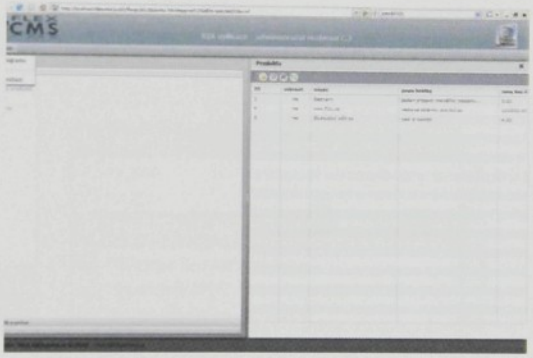
Obrázek 61 - Flex: klientská validace



Obrázek 62 - Flex: potvrzení smazání záznamu



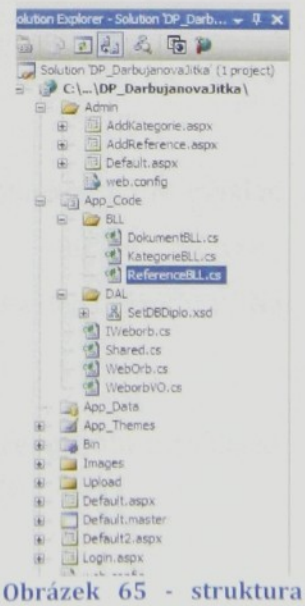
Obrázek 63 - Flex: popumenu v toolbaru



Obrázek 64 - Flex: menu a splitbar

6. Backend - ASP.NET, ASP.NET AJAX Framework, AJAX control Toolkit MS SQL 2005

Původně jsem měla v úmyslu srovnat též technologii ASP.NET pro tvorbu RIA aplikací, ačkoliv z počáteční analýzy prozatím nevyšla nejlépe. S vývojem jsem začala v říjnu, ovšem za 2 měsíce práce jsem rozhodnutí změnila. Důvod byl zřejmý: v únoru měl vyjít ASP.NET 3.5, který mnohé měnil (o SilverLightu 2 byly je nepatrné zmínky). Měla-li být práce aktuální, bylo by zapotřebí práce začít až po jeho vydání. Na to jsem však čekat nechtěla. Za dva měsíce práce s beta verzí ASP.NET AJAX framework jsem však vytvořila jakýsi prototyp aplikace, který je na přiloženém CD ponechán k možnému rozšiřování. Instalace AJAX frameworku nebyla zcela jednoduchou záležitostí, samotná práce s AJAXem byla o něco

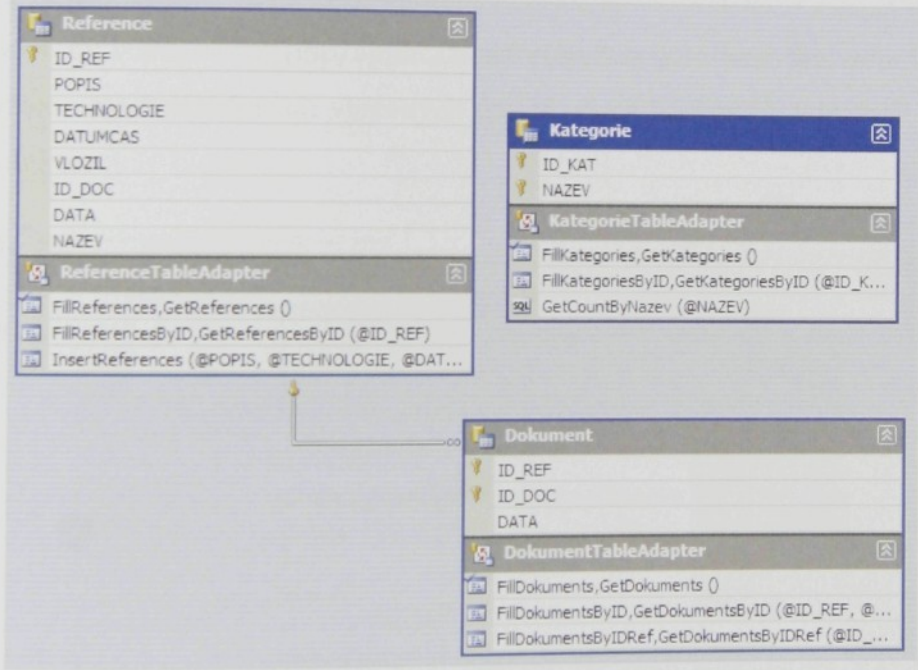


Obrázek 65 - struktura backendu ASP.NET

příjemnější než v PHP, za to však méně „průhledná" pro programátora. Aby stránka v ASP.NET věděla, že má použít asynchronního volání, je potřeba na každé prvek ScriptManager, který obsahuje jednotný skript pro asynchronní volání pro obnovu části stránky (updatePanel) a například volání služeb. Na stránku se pak pouze přidávají komponenty z balíku AJAX Control Toolkit (dll v adresáři bin), které mají předprogramované chování a volání serveru. Pro programátora velice příjemná práce, někoho však může děsit, když nevidí pod pokličku takovéto komunikace.

Dalším důvodem, proč jsem technologii opustila je jeho způsob dělení kódu – do jisté doby šlo kód od vzhledu dělit pouze pomocí code-behind. Microsoft byl pod nátlakem uživatelů již donucen vytvořit ASP.NET MVC framework, který vyšel na začátku března. Je však ještě dle ohlasů na internetu nestabilní a chybový.

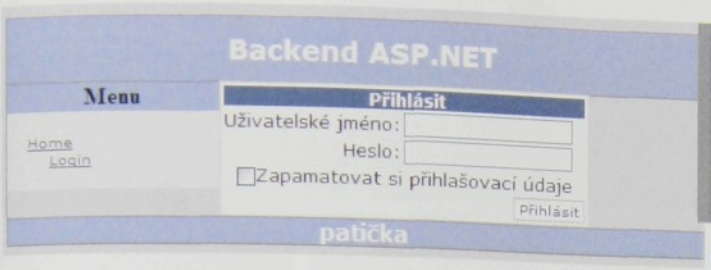
Na ASP.NET jsem vyzkoušela opravdové oddělení vrstev BLL a DAL, kdy vrstvu DAL představuje typový dataset ASP.NET (setDbDiplo.xsd):



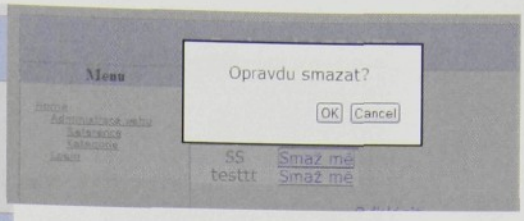
Obrázek 66 - DAL vrstva

a vrstvu BLL pak třída napsaná v C#, která definuje metody pro práci s objektem. Pro příklad uvedu `public class KategorieBLL` obsahuje metody `GetKategories`, `GetKategoriesByID`, `AddKategorie`, `UpdateKategorie`, `DeleteKategorie`, ... Na pozadí aplikace běží MS SQL SERVER 2005.

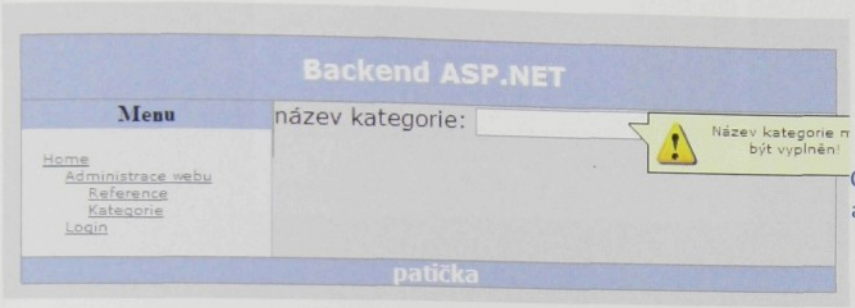
Dále se o tomto backendu rozepisovat nebudu, pouze uvedu několik screenshotů z aplikace, která je odproštěná od jakékoliv grafiky.



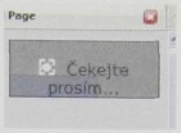
Obrázek 67 - přihlášení uživatele uloženého v DB



Obrázek 68 - dotaz před smazáním kategorie



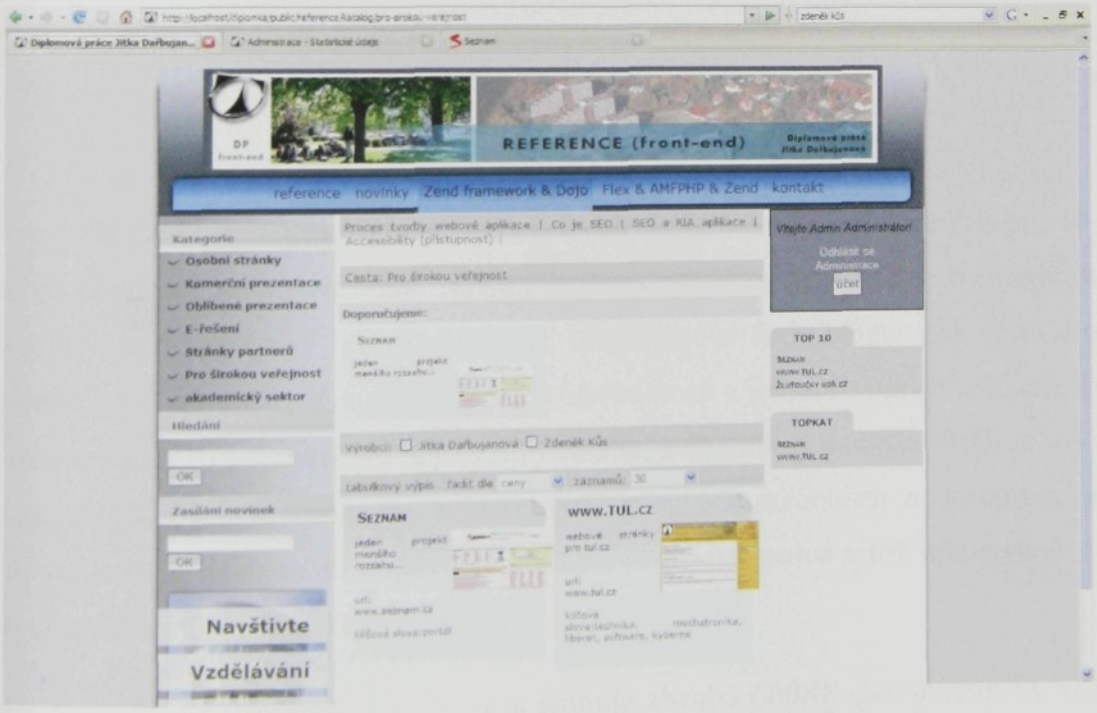
Obrázek 69 - pokus o vložení kategorie s prázdným názvem



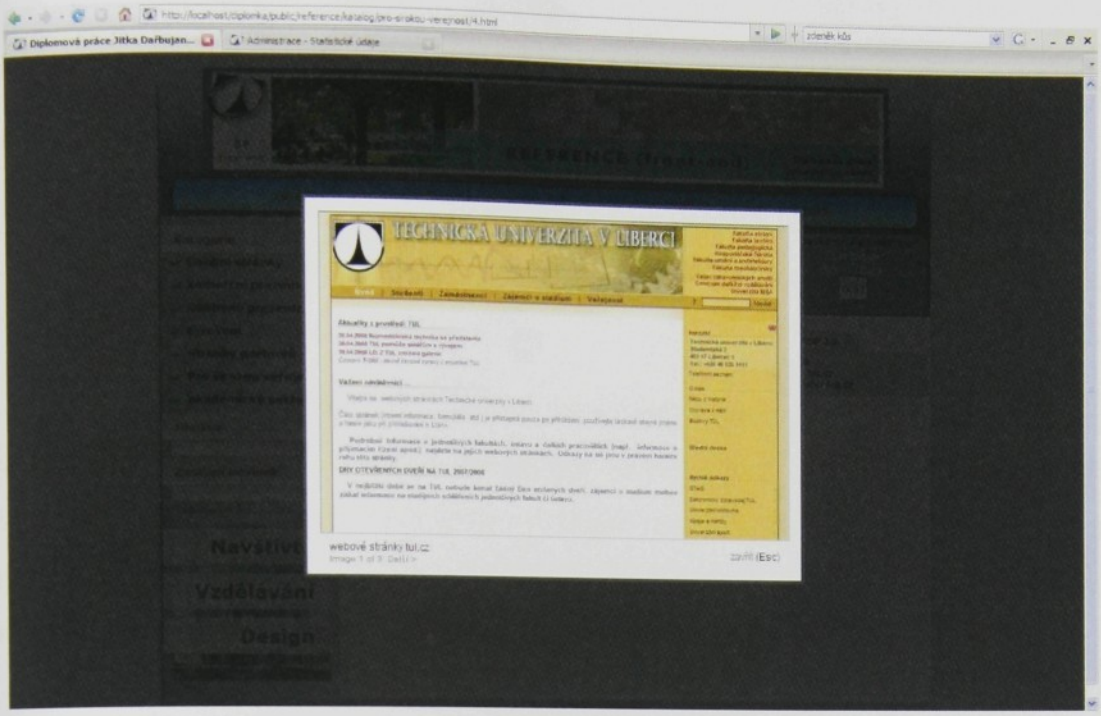
Obrázek 70 - indikace ajaxového volání

7. FRONTEND

Frontend, umístěný v složce src_php/application/reference, jsou klasické webové stránky, na kterých si můžeme otestovat formát vložených dat ze všech backendů. Zdrojové kódy jsou též pod MVC architekturou.



Obrázek 71 - frontend aplikace



Obrázek 72 - frontend - detail reference

ZÁVĚR

Vytvořila jsem sadu RIA aplikací, navzájem co nejefektivněji provázaných tak, aby bylo zapotřebí co nejmenší množství kódu. Navzájem jsem porovnávala zejména technologie AJAX s frameworkem DOJO a Flex. Je nutno zmínit, že pluginové technologie pro vývoj RIA jsou v dnešní době stále ještě v malém podvědomí programátorů a jedná se oproti technologii AJAX v podstatě o nováčka (OpenLaszlo je tu již velmi dlouho, avšak nikdy nedošlo většímu rozšíření). Technologie AJAX má své uplatnění zejména ve webových stránkách a veřejných malých webových aplikacích. Flex se dá doporučit zejména pro intranetové a též internetové typicky formulářové aplikace (CRUD aplikace).

Při AJAXovém backendu se na klienta stahuje zhruba 600KB dat. Můžou za to hlavně javascriptové soubory frameworku Dojo, následně obrázky a kaskádové styly. Pro backend Flex jsem naměřila množství stahovaných dat přibližně 2MB. To bylo docela překvapením, jelikož celý Flex framework má 120KB a většina grafiky je vektorová. Přisuzuji to použitým knihovnám

skinování - tedy bitmapám, případně použitým komponentám výrobců třetích stran. Kompilované SWF by totiž mělo menší velikost než HTML+CSS.

Naopak předpokládaných výsledků jsme dosáhli při měření a testování doby vykreslování na klientovi. Zde je a pravděpodobně i bude javascript za pluginovými technologiemi zaostávat.

Na práci jsem si vyzkoušela mnoho technologií, ať už se jednalo o serverové a klientské programovací jazyky, frameworky, přenosové protokoly a obecně různé techniky. AJAXový backend je novým přínosem této práce v oblasti generování formulářů Zend_Form, kterými pomocí odvozené knihovny generuji též klientskou validaci. Backend 2 Flex je novým přínosem hlavně integrací Flex serveru AMFPHP do MVC struktury Zend frameworku.

LITERATURA

1. **Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.** RFC 2616: Hypertext Transfer Protocol - HTTP/1.1. *IETF.* [Online] 1999. http://www.ietf.org/rfc/rfc2616.txt.

2. **Wikipedia contributors.** Comparison of Application Virtual Machines. *Wikipedia, the free encyclopedia.* [Online] [Citace: 1. 1 2008.] http://en.wikipedia.org/w/index.php?title=Comparison_of_Application_Virtual_Machines.

3. **Toth, David.** HyperMedia systems. *ČVUT.* [Online] červen 2007. http://webing.felk.cvut.cz/hs/.

4. **Erl, Thomas.** Introduction to the service oriented paradigm. *SOA Principles.* [Online] http://www.soaprinciples.com/.

5. **Kosek, Ing. Jiří.** Využití webových služeb a protokolu SOAP při komunikaci. *Inteligentní podpora navigace na WWW s využitím XML.* [Online] [Citace: 3. Duben 2008.] http://www.kosek.cz/diplomka/html/websluzby.html.

6. **W3C.** HTML 4.01 Specification. *W3C.* [Online] 1999. prosinec 24. http://www.w3.org/TR/html401/.

7. —. HTML 5 - W3C Working Draft. *W3C.* [Online] 22. Leden 2008. [Citace: 1. Duben 2008.] http://www.w3.org/TR/html5/.

8. **Koch, Peter-Paul.** W3C DOM Compatibility Tables. *QuirksMode.* [Online] 20. 3 2008. [Citace: 1. 4 2008.] http://www.quirksmode.org/dom/compatibility.html.

9. **Helmond, Anne.** Nova Spivack: "The Semantic Web as an open and less evil web". [Online] 3. Duben 2008. [Citace: 5. Duben 2008.] http://thenextweb.org/2008/04/03/nova-spivack-the-semantic-web-as-an-open-and-less-evil-web/.

10. **Garrett, Jesse James.** Ajax: A New Approach to Web Applications. *Adaptive Path.* [Online] 18. Únor 2005. [Citace: 1. Duben 2008.] http://www.adaptivepath.com/publications/essays/archives/000385.php.

11. **Adobe .** Adobe Flex Component Explorer. *Adobe Flex.* [Online] 2007. http://examples.adobe.com/flex3/componentexplorer/explorer.html.

12. **Gavrilov, Alexey.** Bubblemark animation test. *Balls animation test.* [Online] 28. říjen 2007. [Citace: 1. duben 2008.] http://bubblemark.com/.

13. **Ward, James.** Census Ria Loading BenchMarks. *Census.* [Online] 2008. http://www.jamesward.org/census/.

14. **Inkriti et all.** Design Patterns in RIA. *Inkriti.* [Online] http://www.inkriti.com/marketing/Design_Patterns_in_RIA.php.

15. **Dvořák, M.** Návrhové vzory (design paterns). *VŠE.* [Online] 2003. http://objekty.vse.cz/Objekty/Vzory.

16. **Scott Mitchell.** Creating a Data Access Layer. *ASP.NET.* [Online] Microsoft. [Citace: 1. 11 2007.] http://www.asp.net/learn/data-access/tutorial-01-vb.aspx.

17. **Chiaretta, Simone.** ORM vs HandCoded DAL: Adding a new field to a table. *Climbing the cliffs of c#.* [Online] 18. červen 2007. [Citace: 2008. Duben 2.] http://codeclimber.net.nz/archive/2007/07/18/ORM-vs-HandCoded-DAL-Adding-a-new-field-to-a.aspx.

18. Service-Oriented Architectures and Middleware. *IBM.* [Online] IBM, 2006. http://www.research.ibm.com/AEM/soa.html.

SEZNAM OBRÁZKŮ

PŘÍLOHY

OBSAH CD

- definice databází
  - MS SQL 2005
  - MySQL
- diplomová práce
  - docx
  - obrázky
  - pdf
  - zadání
- dokumentace
  - 01_PHP+ZF Dokumentace
  - 02_Flex backend
  - 03_model databáze
    - MS SQL 2005
    - MySQL 5.0
  - 04_UML
  - 05_technologie
- použité technologie
- video
- vlastní řešení

POTŘEBNÝ SOFTWARE

PHP, verze 5.1.4 nebo pozdější, doporučeno 5.2.3 a výše (testováno pod 5.2.4), URL: <http://php.net/>

Apache Server, verze 2.0 (testováno pod 2.0.59), URL: <http://www.apache.org/>

MySQL, verze 5.0, (testováno pod 5.0.45) URL: < http://dev.mysql.com/downloads/ >

Zend Framework, verze 1.5 , URL: <http://framework.zend.com/ >

Dojo, verze: 1.0, URL: http://dojotoolkit.org/

Flex Builder 3.0 (univerzity dostanou licenci zdarma), URL:
<http://www.adobe.com/products/flex/>

PODPOROVANÉ VERZE PROHLÍŽEČŮ

- Mozilla FireFox 2.0
- Internet Explorer 6
- Internet Explorer 7