Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Diplomová práce

Praktické využití Rich Internet Application

Bc. Petr Cihelka

Prohlašuji, že svou diplomovou práci „Praktické využití Rich Internet Application“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor vedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 30.3.2013

.....

Petr Cihelka

Rád bych touto cestou poděkoval vedoucímu práce Ing. Petru Bendovi za odborné vedení a cenné připomínky při zpracování této práce. Dále bych rád poděkoval Ing. Janu Vondrusovi za odbornou konzultaci a pomoc při orientaci v dané problematice.

Praktické využití Rich Internet Application

Practical use of Rich Internet Application

Souhrn

Námětem diplomové práce jsou moderní technologie pro tvorbu aplikací souhrnně označovaných pojmem Rich Internet Application (RIA). Hlavním cílem je návrh softwarové architektury na základě zvolené RIA platformy. Důraz je kladen na zdůvodnění volby dané RIA technologie a zejména na způsob návrhu optimální architektury aplikace. Také je podrobně představeno řešení kriticky důležitých částí navrhovaného řešení, zejména způsob zpracování náročných úloh pomocí časově plánovaných služeb. Nechybí celkové zhodnocení navrženého řešení. Dílčím cílem práce je představení pojmu RIA jako takového, společně se stručným uvedením historie tohoto pojmu a nechybí ani definice vlastností charakteristických pro tento druh aplikací. Na základě analýzy možných přístupů pro tvorbu RIA jsou definována vodící doporučení pro výběr vhodné stavební RIA technologie. Také jsou zhodnoceny jednotlivé RIA platformy, na základě kterých je RIA možné vytvářet.

Klíčová slova

RIA, Rich Internet Application, použitelnost, Uživatelská přívětivost, architektura aplikace, Adobe, Adobe Flash, Apache Flex, Adobe Air, Java, Spring Framework

Summary

Cutting edge technologies for the development of Rich Internet Applications are the main focus of this thesis. The primary objective is to develop a complete software framework based on a given RIA platform. A reasoning that led to choosing a particular RIA technology is offered as well as a method for an optimal architecture design. A solution to potential bottlenecks in its crucial segments is explained in detail especially a manner in which computationally demanding time sensitive services are handled. The chosen design is thoroughly evaluated and assessed. A definition and history of the term RIA is provided along with introducing characteristics of such applications. Guidelines for selecting a suitable RIA technology are offered based on a comprehensive analysis of possible approaches to the development of RIA. Finally, RIA platforms at the core of application development are assessed as well.

Keywords

RIA, Rich Internet Application, Usability, User Experience, application architecture, Adobe, Adobe Flash, Apache Flex, Adobe Air, Java, Spring Framework

Obsah

1 Úvod

První zmínku či vizi celosvětové počítačové sítě lze nalézt již v roce 1946, kdy byla v časopise Astounding Science Fiction publikována povídka *A Logic Named Joe* [15], jíž autorem byl Murray Leister, vlastním jménem William F. Jenkins. V této povídce Murray Leister představuje zařízení zvaná *Logic*, která svým charakterem odpovídají dnešním osobním počítačům. Tato zařízení jsou navzájem propojena a prostřednictvím distribuovaného systému serverů, zvaných *Tanks*, poskytují přístup ke komunikačním službám, zábavě, datům a komerčním službám. Je na místě připustit, že Murray Leister významně předběhl svoji dobu a možná i svojí vizí ovlivnil budoucí internetové evangelisty.

Samotné technologické počátky Internetu, chápaného jakožto celosvětovou počítačovou síť, lze datovat od roku 1960, kdy se vláda U.S.A vložila do vybudování distribuované a chybám odolné počítačové sítě. Toto úsilí, přesněji řečeno úsilí Ministerstva obrany U.S.A., bylo vyvoláno na základě úspěšného startu rakety Sputnik I, kterou dne 4. 10. 1957 na oběžnou dráhu vypustila SSSR. Jednou z okamžitých reakcí Spojených států bylo založení agentury ARPA (Advanced Research Project Agency), jež měla za úkol shromažďovat vědce různých oborů s cílem vyvinout moderní technologie, které by zajistily strategickou nadvládu Spojených sátů. Tato organizace se na základě požadavku vlády U.S.A. aktivně zapojila do budování decentralizované počítačové sítě a později dala vzniknout síti ARPANET, jež byla základním kamenem budoucí počítačové sítě, jak jí známe dnes. Za zmínku také stojí, že tato organizace po dobu necelého půl roku spravovala satelitní program U.S.A., než byla založena NASA (The National Aeronautic and Space Administration) [14].

První veřejný pokus, ve kterém došlo ke skutečnému odeslání datové zprávy z laboratoří University of California Los Angeles (UCLA) do laboratoří Stanford Research Institute (SRI), byl proveden již roku 1950. Propojení ukázalo nejen možnosti počítačové, resp. síťové komunikace, ale také dalo vzniknout první páteřní síti Internetu. Komunikace byla dvoubodová a zprvu byla využívána pouze pro komunikaci mezi sálovými počítači či terminály. Rozmach a zejména výzkum síťové komunikace dává vzniknout sítím, jejichž nejznámějším zástupcem byla síť ARPANET. Vznik sítě ARPANET je datován od roku 1960 a za vznikem stojí výše zmíněná agentura ARPA (později DARPA - Defense Advanced Research Projects Agency). Významnost této sítě oproti ostatním je dána vývojem protokolů, které umožnily propojení více oddělených sítí do jednoho celku, neboli „sítě sítí“, a to při zachování decentralizace, čímž byla splněna výše uvedená podmínka. V prvopočátku byla síť ARPANET uzavřená

a byla využívána zejména pro univerzitní využití či vojenské účely armády U.S.A. a její velikost byla omezena na několik počítačů a směrovacích prvků, které řídily provoz síťové komunikace. Na konci roku 1971 bylo již do sítě ARPANET připojeno 15 datových center [13] [14].

Stejně jako velké události, bitvy či myšlenky mají své významné osobnosti, tak jsou i technologie, jež dopomohly ke zrodu současného Internetu spojovány s několika zvučnými jmény. Mezi ty první patří Vinton Gray Cerf a Bob Kahn, kteří navrhli a vytvořili vícevrstvý komunikační protokol TCP/IP, který měl zajišťovat, a dodnes zajišťuje provoz internetové sítě [13][14]. Další z důležitých jmen, podílejících se na rozvoji internetové sítě byli Paul Mockapetris a Jon Postel, jež v roce 1983 představili specifikaci protokolu DNS, který měl a má dodnes za úkol obousměrný překlad číselných IP adres na doménová jména.

Samotný vznik pojmu internet se datuje k prosinci roku 1974, kdy byla publikována specifikace protokolu TCP, jejímiž autory byli Vinton Gray Cerf, Yogen Dalal a Carl Sunshine. Ve specifikaci byl poprvé použit termín „internet“ jakožto zkrácenina termínu „internetworking“. V roce 1982 pak byl protokol TCP/IP označen za standard a koncept celosvětové sítě propojené pomocí protokolu TCP/IP byl oficiálně nazván jako Internet.

Rozvoj sítě a zejména její komercializace začala v roce 1981, kdy byla organizací National Science Foundation (NSF) představena Computer Science Network (CSNET). Jednalo se o první síť, jež nebyla součástí sítě ARPANET a sloužila k propojení výzkumných a univerzitních institucí. Rozvoj této sítě hrál velmi významnou roli, zejména v rozvoji povědomí o možnostech a přístupu k této síti. V průběhu roku 1983 se od sítě ARPANET oddělila její vojenská část a vznikla síť MILNET, určená výhradně pro armádní účely. V roce 1986 byl veřejnosti poprvé umožněn přístup k univerzitním a výzkumným superpočítačům připojeným do sítě CSNET. Samotná komercializace již ovšem započala v roce 1980, kdy začínají vznikat první internetoví poskytovatelé (ISP). V roce 1987 bylo do sítě připojeno více než 27 000 počítačů, avšak stále se jednalo především o univerzitní a vědecká pracoviště. Rok 1990 byl koncem jedné éry, kdy byla z provozu vyřazena síť ARPANET. Skutečnou komercializaci Internetu datujeme k roku 1995, kdy byla vyřazena i síť CSNET a byly odstraněny poslední restrikce, které zabraňovaly využití Internetu pro veřejné, a tedy i komerční účely.

Komerční rozvoj sítě, ve smyslu propojení jednotlivých počítačů a jejich následné připojení k Internetu, byl však pouze jednou částí. Chceme-li pojednávat o rozvoji Internetových aplikací a jejich využití, je nutné představit velmi významnou osobnost,

která tento rozvoj umožnila. Touto osobností je Tim Berners-Lee, který v roce 1989 představil koncept WWW, aby jej o dva roky později prakticky realizoval v laboratořích CERN. Uvedení protokolu WWW lze považovat za poslední technologický krok, který byl nutný pro rozvoj Internetu tak, jak jej známe dnes.

Zavedením WWW započala nová epocha Internetu. Do té doby byl Internet převážně využíván pro přenosy zpráv či jednoduché zobrazování textu. Koncept WWW poskytl uživatelům možnost publikovat své texty na síti s možností veřejného přístupu k informacím. I přes prvotní nedůvěru v toto médium začínaly vznikat první širokosáhlé prezentace. Jednalo se zprvu o velmi jednoduché internetové stránky, ve kterých převažoval text. Obsah byl přizpůsoben tehdejším datovým rychlostem, ale postupem času a zvyšováním průchodnosti datových spojů bylo nasazováno více multimediálních prvků. Zvyšování průchodnosti bylo zapříčiněno velkým zájmem uživatelů o toto nové médium. V počátečních letech komercializace internetu byl oproti očekávání roční nárůst uživatelů internetu dvakrát vyšší. V roce 1996 bylo k Internetu připojeno více než 56 milionů uživatelů, o 4 roky později to bylo již 250 milionů uživatelů! O další 2 roky později to bylo významných 600 milionů. V současné době se křivka počtu připojených uživatelů stále zvyšuje, pochopitelně již daleko pomaleji. Odhadovaný počet připojených uživatelů byl k červnu roku 2012 2,4 miliardy, což je přibližně 34% celosvětové populace. Pro představu lze ještě poznamenat, že v roce 1993 bylo pomocí Internetu přeneseno pouhé 1 % všech telekomunikačních informací, v roce 2000 to bylo 51 %, a v současné době je pomocí internetu přeneseno více než 97 % telekomunikačních informací.

Výše uvedený enormní zájem uživatelů o služby Internetu vyvolal velký tlak na možnosti internetových technologií. Původní, převážně statické textové stránky, přestávaly postačovat a to zejména díky zvyšujícím se nárokům uživatelů. Takovéto stránky byly postupně vytlačovány stránkami využívajícími dynamického generování obsahu (např. PHP, .NET). Samotný jazyk HTML byl doplněn o možnosti stylyzování pomocí CSS a skriptování pomocí JavaScriptu. Někteří výrobci však nebyli spokojeni s možnostmi, které poskytoval jazyk HTML, a započali s vývojem vlastních, proprietárních řešení.

Od internetových stránek byla vyžadována větší interaktivita a tím se zvyšovaly nároky na generování obsahu stránek dle požadavků či vstupů uživatele. Vývojáři na tyto požadavky reagovali vytvářením technologií a internetových aplikací, které poskytovaly uživatelům služby a komfort desktopových aplikací a v mnohých případech je i z hlediska funkčnosti překonávaly. Vývojáři a uživatelé si začali uvědomovat výhody aplikací běžících v rozhraní běžného internetového prohlížeče. Uživatelé mohli

s aplikací pracovat téměř kdekoli, kde byl dostupný internetový prohlížeč a připojení k síti. Díky centralizaci takových technologií měli vývojáři usnadněn vývoj, správu a distribuci nových verzí. Nový technologický směr dal vzniknout nové oblasti internetových aplikací, které jsou dnes známé pod označením RIA (Rich Internet Applications). Běžný uživatel si jejich existenci možná již ani neuvědomuje, ale každodenně je využívá. Jedná se například o aplikace typu YouTube, GMail, Microsoft Outlook Online, Microsoft Hotmail, FaceBook, MySpace atp.

2 Cíl práce a metodika

Hlavním cílem diplomové práce je provedení návrhu internetové aplikace na základě zvolené RIA platformy. Dílčím cílem práce bude charakteristika pojmu RIA a vymezení základních rysů těchto aplikací spojené s analýzou a zhodnocením možných přístupů jejich tvorby. Dále bude na základě syntézy požadavků na internetovou aplikaci definován optimální přístup pro návrh architektury vytvářené RIA. Taktéž bude provedeno zhodnocení navrženého technologického postupu.

Z počátku první části bude nejprve charakterizován a vysvětlen samotný pojem RIA. Budou představeny jednotlivé organizace a technologie, které se na rozvoji podílely a podpořily tak vznik tohoto komplexního a moderního odvětví.

Ve druhé části bude vymezena platnost RIA a budou definovány základní typy těchto aplikací. Poté bude provedena analýza všech kriterií, na které je nutné brát při návrhu RIA zřetel, zejména uživatelská přívětivost aplikací a jejich použitelnost. Na základě syntézy uvedených kriterií budou definovány charakteristické vlastnosti RIA. Dojde k představení výhod a nevýhod RIA technologií jako celku, ale také v rámci jednotlivých technologických směrů. Na závěr budou specifikována doporučení, která mohou pomoci při volbě vhodné technologie pro tvorbu RIA.

Ve třetí části bude představena široká škála technologií a platforem, které poskytují nezbytné nástroje pro tvorbu RIA. Jednotlivé technologie budou zhodnoceny podle kritérií, jež byly syntetizovány z charakteristických vlastností uvedených ve druhé části práce. Hodnocení úspěšnosti představených technologií bude doplněno o ekonomické zhodnocení, a to zejména ve smyslu jejich pozice na trhu. Na závěr bude provedeno celkové bodové zhodnocení vyjmenovaných technologií.

Ve čtvrté části budou definovány základní požadavky na aplikaci, jejíž návrh je hlavním cílem diplomové práce. Bude provedeno zdůvodnění volby platformy Adobe Flash a společně s definicí základních přístupů řídících vývoj aplikací, bude představena rámcová architektura navrhované aplikace. Blíže bude představena klientská strana aplikace, zejména platforma Adobe Flash s technologií Apache Flex a dále bude podrobně prezentován návrh architektury klientské strany. Následně bude prezentována architektura serverové strany aplikace a zmíněno bude vlastní řešení Perzistentní vrstvy. Taktéž bude diskutován způsob řešení časově náročných úloh. Na závěr bude celkový návrh aplikace zhodnocen a budou jednotně shrnuty výhody a nevýhody navrženého řešení.

V závěrečné části práce budou zhodnoceny dosud zavedené technologické postupy při vývoji RIA. Bude provedeno hodnocení uvedených cílů této práce. Současně bude uvedeno zamyšlení nad budoucím vývojem RIA technologií a nástrojových platforem.

3 RIA – vývoj a historie

3.1 Definice a vysvětlení pojmu

Stěžejní pojem RIA, neboli Rich Internet Applications, lze do češtiny volně přeložit jako „Bohaté Internetové Aplikace“. Bohaté jsou především ve smyslu souboru možností poskytujících uživateli komfort při práci s aplikací.

Samotný vznik RIA byl logickým vyústěním prvotní epochy exapanzivního rozvoje Internetu, který, jak již bylo osvětleno, nastal okolo roku 1990. Internetové médium se stalo dostupné široké veřejnosti a postupem času se běžné, převážně statické stránky, změnily na stránky dynamické přinášející uživateli jistou úroveň interaktivity. V průběhu této proměny se začaly objevovat takové druhy internetových aplikací, které splňovaly požadavky na dynamiku a interaktivitu do takové míry, že byly schopny plně nahradit běžné desktopové aplikace. Výsledkem bylo, že aplikace fungující v obyčejném internetovém prohlížeči nabízející funkčnost desktopových aplikací, se staly vyhledávaným druhem aplikací zejména díky jednoduché správě, integrovatelnosti, dostupnosti a multiplatformnosti.

Pojem RIA jako takový není bohužel možné přesně definovat. Neexistuje žádná norma či definice, popř. organizace, která by definovala, co by měla či neměla aplikace splňovat, aby mohla být označena jako RIA. Samotný pojem se navíc v čase vyvíjí a to společně s technologiemi, které jsou k tvorbě takovýchto aplikací využívány. Vlastnosti a charakteristiku RIA taktéž mění i chápání a nároky uživatelů, zejména pak ve smyslu nástupu mobilních technologií. Přesnou definici tedy nelze podat, lze však na základě analýzy požadavků na tyto aplikace definovat jisté charakteristické rysy.

3.2 Historie

Jedno lze však říci s naprostou jistotou, autorem termínu RIA je firma Macromedia (nyní Adobe Inc.), která jej uvedla v roce 2002. Nutno poznamenat, že koncept samotný není zcela unikátní a již dříve byl představen jinými firmami, např. Microsoft. Vzhledem k téměř stagnujícímu vývoji jazyka HTML, ke kterému ve stejné době došlo, bylo zřejmé, že soudobé HTML nebude v nejbližší době dostačovat stále se zvyšujícím požadavkům. Logickým vyústěním bylo představení technologií, které jako běhového prostředí využívaly prioprietárních prohlížečových modulů, jež měly za cíl odstranit limity statického HTML.

3.2.1 Společnost Macromedia/Adobe

Výsledkem tohoto úsilí byl v rámci firmy Macromedia produkt Macromedia Flash uvolněný v prosinci roku 1996 ve verzi 1.0. Uvedená prvotina byla zpočátku přijata velmi rezervovaně a poskytovala pouze nástroje pro animaci obrázků a vektorových objektů. V několika málo následujících verzích byl Macromedia Flash využíván převážně pro tvorbu grafických prezentací a reklamních bannerů. K jednomu ze zásadních zlomů došlo v roce 2002, kdy byla vydána klíčová verze produktu Macromedia Flash Player, která uvedla podporu protokolům pro komunikaci se vzdálenými službami (AMF, podpora pro SOAP), vysílání on-demand (dostupného na vyžádání)/živého videa a byla zavedena podpora komponent a sdílených knihoven. Zároveň s uvedením nové verze publikovala Macromedia dokument, ve kterém specifikovala základní požadavky na charakter a vývojové nástroje RIA aplikací. Firma Macromedia i nadále pokračovala s vývojem svého Macromedia Flash, jenž díky novým vlastnostem nabýval na oblibě. V roce 2003 byl představen Macromedia Flash ve verzi 7, který kromě jiného podporoval i jazyk Action Script 2.0, jenž po dlouhých očekáváních přinesl podporu objektově orientovaného programování. O dva roky později došlo k další významné události: krom vydání nové verze Macromedia Flash byla Macromedia dne 3. prosince 2005 odkoupena firmou Adobe Inc. Dále byl ve stejném roce vydán, stále ještě pod hlavičkou firmy Macromedia, nový produkt s názvem Macromedia Flex, který měl být primárně určen pro tvorbu RIA aplikací. O dva roky později, v roce 2007, byla vydána nová verze Flash Player 9.0, avšak to již pod názvem Adobe Flash. Kromě zlepšení integrovatelnosti s produkty Adobe přinesla také podporu pro Action Script 3.0. Ve stejném roce způsobila firma Adobe Inc. další revoluci v RIA aplikacích představením nového produktu Adobe AIR. Ten umožnil přenést aplikace z internetového prohlížeče na uživatelské počítače ve formě plnohodnotné aplikace, tzn. se všemi výhodami, kterými běžné aplikace disponují, jako přístup k souborovému systému, přístup ke zdrojům systému, a jiné [10][18][26].

K velkému zlomu v propagaci a využití technologie Adobe Flash došlo ke konci roku 2012, kdy společnost Adobe ukončila podporu prohlížečového modulu pro mobilní telefony a taktéž oznámila, že ukončuje vývoj produktu Adobe Flex a předává jej organizaci Apache.org k zajištění případného následného vývoje. Adobe Flash se od té doby zaměřuje spíše do herní oblasti, kam i firma Adobe Inc. směřuje většinu své propagační činnosti.

3.2.2 Java

Nebyla to však pouze firma Macromedia která se snažila o vytvoření prostředí pro vývoj RIA aplikací. Společně s Macromedia Flash existovaly i další proprietální moduly, jako např. Java Applets, avšak na poli RIA aplikací příliš neuspěly. Jednalo se o malé aplikace psané v jazyce Java, které byly vkládány do HTML stránek a pomocí virtuálního stroje JVM (Java Virtual Machine) spuštěny v prohlížeči. Nutnou podmínkou byla přítomnost JRE (Java Runtime Environment).

3.2.3 Microsoft

Firma Microsoft také nezůstala pozadu a ve snaze zmírnit dopad rozmachu Adobe Flash představila v roce 2007 vlastní technologii jménem Microsoft Silverlight 1.0. V současné době je k dispozici Microsoft Silverlight ve verzi 5.0. Produkt firmy Microsoft se těší veliké oblibě, zejména na poli vývojářů na platformě .NET, nese však s sebou břemeno plynoucí z menšího rozšíření mezi uživateli [21]. Stejně tak jako u firmy Adobe je zde velké riziko, že firma Microsoft opustí cestu vývoje vlastního proprietálního řešení a spíše využije nových možností HTML5.

3.2.4 HTML a DHTML

Vývojem prošel i samotný jazyk HTML, který ve spojení se skriptovacím jazykem JavaScript poskytoval a stále poskytuje dobré zázemí pro tvorbu RIA aplikací. Vývoj jazyka HTML ustrnul na verzi HTML 4.1 a v současné době se čeká na finální podobu budoucího HTML5, které by mělo nahradit současný HTML standard. Dalším krokem byl vznik DOM (Document Object Model), který představuje objektově orientovanou reprezentaci internetové stránky v podobě stromové struktury jejích HTML tagů. Tato struktura muže být dále modifikována skriptovacím jazykem JavaScript. Spojení technologií HTML, CSS, JavaScript a DOM dalo vzniknout DHTML, což je ve své podstatě pouze rozšířená verze jazyka HTML doplněná o možnosti změny obsahu a struktury stránek bez nutnosti nového přenosu vyžádaných stránek. Na základě tohoto vývoje bylo možné vytvořit aplikační rámce, moderní češtinou nazývané frameworky, např. ExtJS, jež poskytují bohaté kolekce pro tvorbu RIA aplikací a jsou tak schopné odstranit omezení plynoucí z jazyka HTML [2][34].

3.2.5 HTML5

Jazyk HTML5 je značkovací jazyk, vycházející z jazyka HTML (přesněji HTML 4.1) a je tedy stejně jako on určen pro zobrazování obsahu internetových stránek. Pořadové číslo v názvu této technologie značí, že jde o pátou revizi jazyka HTML. Práce na tomto novém standardu započaly již roku 2008. V současné době je HTML5 ve fázi pracovního návrhu a finální vydání je plánováno na konec roku 2014. K masivnímu nasazení HTML5 však došlo již v roce 2011 [9].

Vzhledem k tomu, že HTML5 vychází z HTML, jsou i jeho vlastnosti shodné. HTML5 přináší nové obsahové elementy (section, article, header, footer), jejichž úkolem je zlepšení sémantiky internetových stránek [9]. Některé elementy jsou naopak převzaty z původního HTML, avšak dostávají nový význam. Samotné HTML5 však stále nenabízí žádné možnosti dynamického obsahu a pro tento účel je využíván jazyk JavaScript. Pro definici vzhledu je shodne s HTML využíváno CSS, přesněji CSS verze 3 (CSS3).

HTML5 vzniklo jako odpověď na dominanci technologie Adobe Flash, a to zejména na poli multimediální zábavy, zejména podpora živého vysílání, her a živé komunikace. HTML5 již nabízí plnou podporu videa, a ve spojení s možnostmi JavaScriptu, i animace. Je vhodné uvést, že podpora HTML5 není mezi prohlížeči stejná a vývojáři musí mít při využití nových prvků tento stav na paměti.

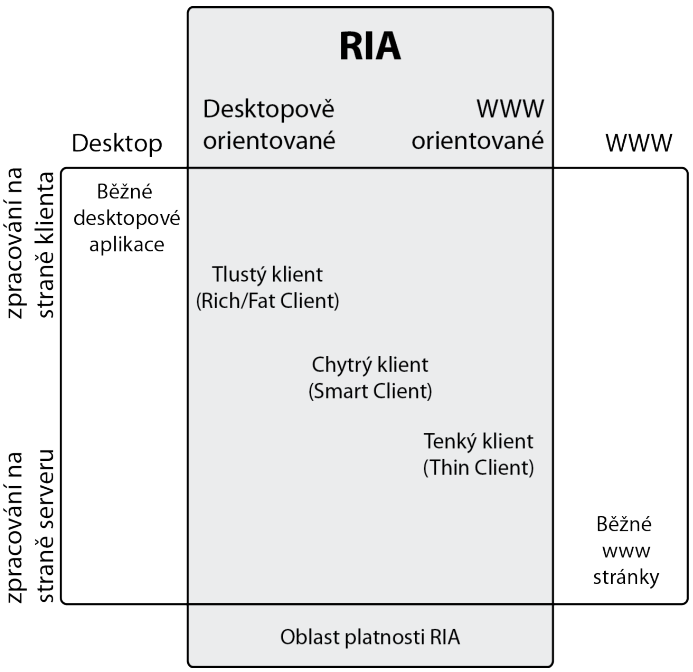
Rozvoj HTML5 měl a má velmi razantní dopad na rozvoj ostatních technologií pro tvorbu RIA, zejména na technologie Adobe Flash a Microsoft Silverlight. Firma Adobe v reakci na rapidní rozvoj HTML5 v roce 2011 přesouvá své zaměření pro RIA platformy právě na HTML5 a framework Adobe Flex je darován organizaci Apache Inc [43]. Firma Microsoft taktéž v reakci na rozvoj HTML5 oznámila, že po vydání Microsoft Silverligh verze 5 s dalším vývojem končí a taktéž se zaměřuje na vývoj technologií využívajících HTML5 [17].

4 Charakteristika RIA

Jak již bylo výše uvedeno, přesnou definici RIA nelze vyslovit, je však možné analýzou požadavků na konkrétní aplikace definovat jejich charakteristické vlastnosti.

4.1 Oblast RIA

Dříve než budou vysloveny jednotlivé požadavky a následně definovány charakteristiky RIA, jen nutné vymezit platnost RIA jako takových. Oblast aplikací je v dnešní době velice rozsáhlá a v některých případech může být pro čtenáře matoucí. Existuje mnoho technologií, které jsou pro tvorbu aplikací používány. Jedním z dělících aspektů, který lze pro vymezení platnosti RIA použít, je rozdělení funkčnosti a využití konceptu klient – server. Některé z technologií soustřeďují veškerou funkčnost aplikace na stranu klienta. Do této kategorie spadají kupříkladu běžné desktopové aplikace. Na opačné straně stojí druh aplikací, které naopak přesouvají veškerou funkčnost na stranu serveru, do této množiny spadají například běžné internetové stránky. Poslední množinou, jsou aplikace, které rozdělují funkčnost jak na stranu klienta, tak na stranu serveru. Do poslední uvedené množiny spadají právě RIA. Poměr v jakém je funkčnost aplikace rozdělena na stranu klienta a stranu serveru určuje druh RIA.



Obrázek 1: Oblast platnosti RIA [autor]

Schéma uvedené v Obrázku 1 znázorňuje množinu server-klient aplikací, ve které se RIA pohybují. Na levé straně jsou čistě desktopově orientované aplikace, které je nutné na operační systém nainstalovat. Klasickým představitelem takovýchto aplikací jsou například textové procesory jako Microsoft Word. Do této skupiny spadají taktéž desktopové aplikace, které ke své práci využívají připojení k síti, avšak jsou schopné pracovat i bez tohoto připojení ač s omezenou funkčností. Klasickým zástupci takovýchto aplikací jsou e-mailové programy, jako např. Mozilla Thunderbird či Microsoft Outlook.

Další skupinou aplikací jsou aplikace využívající tzv. tlustých klientů označovaných jako *Rich Clients* či *Fat Clients*. Tyto aplikace se vyznačují velkou podobností s desktopovými aplikacemi, většina jejich funkčnosti je zajišťována klientským systémem, avšak ke své funkčnosti již vyžadují připojení k serveru a v mnoha případech nejsou schopné bez tohoto připojení pracovat.

Naproti desktopovým aplikacím stojí klasické internetové stránky. Do této množiny lze zahrnout i jednodušší internetové aplikace a aplikace využívající k běhu tzv. tenkých klientů, *Thin Clien*“ aplikace. Aplikace ve většině případů nevyžadují instalaci na klientský systém a jsou doslova závislé na připojení k síti a mnohdy fungují v prostředí běžného internetového prohlížeče. Jako zástupce této skupiny lze jmenovat firemní intranetové portály.

RIA se pohybují mezi oběma výše popsanými skupinami aplikací, přičemž využívají výhod tradičních desktopových a webově orientovaných aplikací. Jejich funkčnost není pouze omezena na běhové prostředí internetového prohlížeče, ale díky technologiím jako je Adobe AIR, je možné tyto aplikace přesunout na operační systém uživatele. Jak již bylo uvedeno, RIA využívají výhod obou skupin, z desktopových aplikací přebírají vlastnosti jako je ovladatelnost, přizpůsobitelnost, jež jsou nejčastěji seskupovány pod označení *uživatelská přívětivost* neboli *Rich User Experience*. Z prostředí webových aplikací si RIA přebírají vlastnosti, jako jsou např. integrovatelnost, spravovatelnost, efektivnost, multiplatformnost, apod.

4.2 Pohled na vývoj

Již bylo uvedeno, že definice RIA je značně závislá na chápání aplikace, resp. na úhlu pohledu, pod kterým se na danou aplikaci či problematiku uživatel či vývojář dívá. Taktéž bylo uvedeno, že změna chápání RIA se může měnit v čase, a to zejména na základě vývoje nových technologií. Změnu chápání lze velmi dobře demonstrovat na výsledcích analýzy, kterou v roce 2007 publikoval v článku *A Rose By Any Other Name* Simon Morris [20], ve kterém se zamýšlí nad budoucím rozvojem RIA. V tomto článku definuje tři možná chápání či postoje, jak uživatelé či vývojáři nahlížejí na internetové aplikace, jsou to: *Browserism*; *Neo-Desktopism* a *Pragmatic Neo-Desktopism*.

- *Browserism* – je popisován jako víra ve fakt, že internetový prohlížeč a technologie které nabízí, jsou budoucností uživatelských aplikací. Toto tvrzení je založeno zejména na základě masivního využívání internetových prohlížečů, jakožto dominantního nástroje k přístupu na internet. Cílem vyznání je postupné zdokonalování současných technologií internetového prohlížeče do takové míry, aby mohl poskytovat funkčnost, kterou nabízejí tradiční desktopové aplikace.
- *Neo-Desktopism* – je protikladem Browserismu. Jedná se o předpoklad, že internetový prohlížeč není platformou budoucnosti pro uživatelské aplikace. V extrémním případě jde o evoluční *cul-de-sac* uživatelských aplikací. Lidé tohoto vyznání preferují klasické desktopové aplikace s důrazem na vývoj takových technologií, které umožní spouštět desktopové aplikace z prostředí internetu, ale bez použití internetového prohlížeče.
- *Pragmatic Neo-Desktopism* je třetím postojem, který je založen na pochopení, že je internetový prohlížeč, jakožto platforma pro uživatelské aplikace, mrtvou evoluční větví. Zároveň však chápou, že nelze výhody internetového prohlížeče zcela ignorovat. Výsledkem tohoto pochopení je využití výhod internetového prohlížeče ve spojení se zásuvným prohlížečovým modulem. Příkladem, který je výsledkem tohoto smýšlení je např. Adobe Flash, Microsoft Silverlight či JavaFX.

Označení, která ve svém zhodnocení Simon Morris použil, lze označit za lehce odvážná a samotné rozdělení do skupin za dosti vyhraněné, je však možné z uvedeného hodnocení vyvodit důsledek, že slepá víra v existující technologie jako takové, může způsobovat problémy, plynoucí z přehlížení jejich nevýhod. Správnou technologickou cestou je využití výhod desktopových a webových aplikací.

Na základě zhodnocení současné situace a situace panující v době vydání článku Simona Morrise lze velice dobře demonstrovat změnu chápání RIA. Při použití terminologie Simona Morrise je možné tvrdit, že směr zvaný *Neo-Desktopism* byl slepou evoluční větví vývoje uživatelských aplikací. Směr zvaný *Pragmatic Neo-Desktopism* a *Browserism* se naopak rozvíjí do dnešní doby. Rozvoj obou dvou směrů je v lehkém rozporu s tvrzením Simona Morrise, který naopak vyvodil ze své analýzy tvrzení, že i směr zvaný *Browserism* zanikne a jako jediný naopak přežije směr *Pragmagic Neo--Desktopism*, který lze označit jako průnik obou předchozích směrů [20].

Je nutné poznamenat, že *Browserism* byl opravdu na pokraji zániku, avšak iniciativa a propagace HTML5 jej v posledních letech vrátila mezi směry, se kterými je nutné při vývoji aplikací počítat. Oproti tomu směr *Pragmatic Neo-Desktopism* je na zdánlivém ústupu, a to zejména díky vysoké popularitě HTML5 v posledních letech.

4.3 Rich User Experience

Termín *Rich User Experience* (RUA) je možné volně přeložit jako *Bohatá uživatelská přívětivost*. Tento termín vychází z termínu *Uživatelská přívětivost* (User Experience – UX/UE) a použil jej roku 1993 Donald A. Norman, který je dodnes uznávaným odborníkem na problematiku uživatelského rozhraní. Pojem Uživatelská přívětivost lze definovat jako úroveň prožitku a spokojenosti, kterou uživatel získává při práci s aplikací. Nejedná se pouze o způsob interakce uživatele s aplikací, ale také o aspekty, které se tykají rozmístění prvku v aplikaci, samotné rozhraní aplikace, barevné provedení, ale také aspektů mimo aplikaci, jako je například podpora či provedení manuálu k aplikaci [29].

RUA kombinuje funkci aplikace s formou, stylem a uměním. Lze říci, že cílem RUA je tzv. „wow“ faktor, který vyvolá v uživateli pozitivní emoce. Jedná se o výjimečné a vysoce použitelné aplikace s robustní architekturou [35]. Na straně klienta jsou tyto aplikace ohromující, poutavé, lákavé, zábavné, vzrušující, živé, interaktivní, intuitivní, přizpůsobitelné, sympatické a krásné. Oproti tomu serverová řešení se vyznačují škálovatelností, spolehlivostí, přenositelností, flexibilní, rozšiřitelné, adaptabilní, lehce udržovatelné, znovu použitelností, upravitelností, pružností, robustností, efektivností a bezpečností [22]. Často také zahrnují sociální aspekty a poskytují velké možnosti personalizace.

Jsou-li výše uvedené charakteristiky aplikovány na běžné HTML stránky, lze tvrdit, že míra uživatelské přívětivosti je velmi nízká. Veškerá interakce s takovouto stránkou je limitován stránka-stránka (page-to-page) konceptem, tj. pokud požadujeme jakoukoli interakci, je nutné načíst požadovanou stránku či obnovit současnou. Zpracování a zejména zobrazení výsledků aplikačních formulářů je možné zobrazit pouze po předchozím odeslání na server a následném zobrazení stránky s výsledky. Veškeré zpracování požadavků a komunikace s aplikací je tedy synchronní. Uvedené limitující faktory však nelze označit jako chybu technologie, jak by se nabízelo. Jak již bylo výše uvedeno, původní smysl a zaměření webu byl pouze ve výměně dokumentu a jejich vzájemnému propojení, nikoliv pro interaktivní aplikace.

Protipólem běžných HTML stránek jsou desktopové aplikace. Takové aplikace těží z vysoce komfortní aplikační interaktivity. Formuláře jsou vyhodnocovány již při vyplňování a aplikace okamžitě nabízí zpětnou reakci, která může uživatele informovat o chybně zadaných položkách a nabízet řešení nastalého problému. Vzhledem k povaze aplikací, je možné neustále sledovat uživatelskou interakci s aplikací. Desktopové aplikace jsou mnohdy navrženy tak, aby byla zajištěna alespoň omezená funkčnost, není-li k dispozici připojení k síti. Desktopové aplikace disponují *událostmi řízenou architektururou* (EDA - Event Driven Architecture), která zdokonaluje uživatelskou interakci s aplikací. EDA naslouchá událostem v systému či aplikaci a disponuje schopností aktualizace uživatelské aplikace na základě těchto událostí, a to vše bez jakéhokoliv požadavku uživatele na tuto akci. Desktopové aplikace využívají zejména asynchronního volání a přístup ke zdrojům je díky absenci serverového protějšku daleko jednodušší.

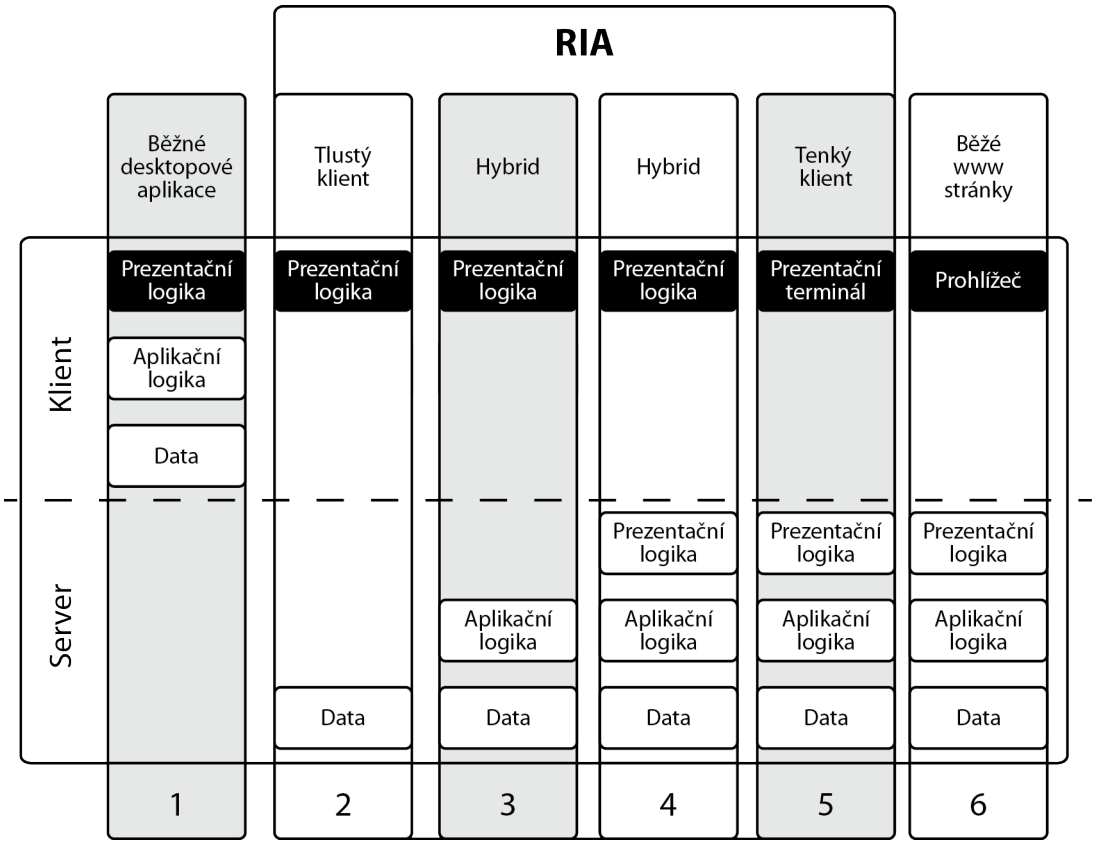
Řešením, které umožnilo zvýšit interaktivou a použití webových aplikací bylo využití technologie AJAX. S použitím této technologie bylo možné využít asynchronního volání požadavku. Je nutné podotknout, že synchronní model volání se nezměnil, pouze je díky této technologii skryt. Uživatelská interakce s aplikací je s využitím technologie AJAX lepší, a je možné využít vlastností desktopových aplikací, jako je například výše zmíněné potvrzení správnosti formulářových prvků, bez jejich předchozího odeslání. S přispěním technologií, které nabízejí proprietální moduly jako jsou Flash, Silverlight a Java je možné využít také podpory audia a videa.

Lze tvrdit, že právě uživatelská přívětivost je klíčovou vlastností a jedním z hlavních důvodů, proč se RIA používají.

4.4 Architektura RIA

Běžné internetové aplikace jsou založeny na principu architektury klient-server, kde klient je označován jako „tenký klient“, nejčastěji v roli internetového prohlížeče. V tomto případě je veškeré zpracování provedeno na straně serveru a klient je pouze využit k zobrazení statického či vygenerovaného výstupu odeslaného ze serveru, nejčastěji v podobě HTML. Nevýhodou modelu je fakt, že veškerá interakce s aplikací musí být provedena přes server, který k takové operaci vyžaduje data, na základě dat provede požadovanou operaci, následně odpoví a odešle data s odpovědí klientovi. Výsledkem výše popsané komunikace je omezení plynoucí z tzv. synchronní komunikace, tedy po odeslání požadavku na server je nutné počkat a není možné s aplikací pracovat, než se vrátí odpověď.

Jak již bylo zmíněno výše, v průběhu rozvoje internetu a internetových technologií došlo k vzniku technik či technologií, které výše popsaný problém řeší (např. AJAX). S použitím těchto technologií bylo možné vytvářet asynchronní volání požadavků na server, tj. odesílat požadavky na server bez nutnosti práce s danou aplikací. Takovéto aplikace však ještě nemůžeme označit jako RIA. RIA této asynchronní komunikace hojně využívají, avšak oproti běžným aplikacím zavádějí novou aplikační vrstvu, nejčastěji zvanou jako *Klientská vrstva* (Client Engine), jež je vsazena mezi klienta a server. Taková střední vrstva je nejčastěji zavedena a spuštěna při inicializaci aplikace. Zodpovědností vrstvy je vykreslování uživatelského rozhraní aplikace a komunikace se serverem, které probíhá ve většině případů asynchronně a nedochází k blokaci aplikace při požadavku na akci. Je-li však potřeba, může dojít k synchronním voláním. Další typickou vlastností využití *Client Engine,* je nahrávání dat, která jsou pro dané zobrazení na straně klienta skutečně potřeba, popř. automatické donahrávání (tzv. on-demand či lazy loading) dat při práci s aplikací. Na server je tedy pouze odeslán požadavek na data, server data odesílá, avšak zpracování a zobrazeni dat je již v kompetenci samotného klienta.



Obrázek 2: Rozdělení aplikací využívajících architektury klient - server [autor]

Popis jednotlivých druhů aplikací:

1. Tradiční desktopové aplikace, veškeré zpracování probíhá na straně klienta, serverový protějšek není. Jde o silně klientsky orientované aplikace.
2. Skupina aplikací využívající *Tlustého klienta* využívá serverový protějšek k ukládání dat. Veškerá funkční a prezentační logika je v režii klienta. Výhodou aplikací je vysoké přiblížení jejich rozhraní a funkčnosti klasickým desktopovým aplikacím. Nevýhodou jsou delší spouštěcí časy a mnohdy využití proprietálních modulů, které jsou nutné pro jejich běh.
3. Hybridní model aplikace přesouvá, oproti předchozímu případu, část funkční logiky na stranu serveru. Prezentační logika je stále ve výhradní režii klienta.
4. Další z hybridních modelů již zužuje odpovědnost klientské strany a funkční logiku přesouvá plně na stranu serveru. Ve své odpovědnosti si ponechává prezentační logiku. Lze ji označit za jistou formu *Tenkého klienta*.
5. V případě tohoto modelu dochází k využití *Tenkého klienta*, veškeré funkční a aplikační zpracování je prováděno na straně serveru. Server nově zpracovává i prezentační logiku a klient slouží pouze k zobrazení rozhraní aplikace. Výhodou přístupu jsou kratší startovací časy aplikace a jednoduchost. Nevý-

hodou jsou delší reakční časy aplikace, protože každá uživatelská interakce z aplikací požaduje odpověď od serveru.

6. Tradiční statické internetové stránky, povětšinou bez větší interakce, povětšinou pouze synchronní.

Jak výše uvedené schéma a rozdělení napovídá, míra odpovědnosti *Klientské vrstvy* (Client Engine) je závislá na poměru rozdělení funkčnosti aplikace mezi klienta a server.

Vzhledem k vývoji v posledních letech je nutné poznamenat, že již není možné RIA definovat pouze jako aplikace běžící v běžném internetovém prohlížeči, ale díky technologiím jako je Adobe AIR je nyní možné tyto aplikace transformovat do podoby desktopových aplikací.

4.5 Nezávislost na platformě

Značnou výhodou prohlížečově orientovaných aplikací je jejich nezávislost na operačním systému, označovaná mnohdy jako multiplaformnost. Optimalizace aplikace je prováděna oproti běžným internetovým prohlížečům, u kterých je předpoklad, že respektují předepsané standardy HTML. Reálný stav je bohužel jiný, některé prohlížeče implementují předepsanou funkčnost dle vlastních standardů. Popsaný neduh byl velmi markantní při využití technologie AJAX, kdy různé prohlížeče implementovaly způsob vzdáleného volání jinak. V současné době je toto volání již sjednocené, přesto však existují malé rozdíly, které jsou však řešeny využitím knihoven třetích stran, jež toto rozdílné chování či volání funkcí řeší. Zástupcem uvedených knihoven je například v dnešní době velmi populární JavaScriptová knihovna jQuery.

Při použití prohlížečových technologií, které ke svému běhu využívají zásuvných modulů, tj. Adobe Flash, lze tvrdit, že je multiplatformnost zajištěna také.

Nezávislost na operačním systému či platformně je pro RIA důležitým předpokladem. Jak již bylo uvedeno, RIA by měly přebírat to nejlepší ze světa desktopových a webových aplikací avšak s tím, že je bude možné provozovat na běžném operačním systému, bez jakéhokoli dalšího upravování klientské stanice. Pravdou je, že zmíněný předpoklad neplatí u technologií, které využívají ke svému běhu prohlížečových zásuvných modulů (např. Adobe Flash), avšak u nich je předpoklad, že je jejich rozšíření tak masivní, že jsou běžně dostupné na klientských stanicích.

Jak bylo také poznamenáno, RIA již nejsou pouze doménou klientských stanic, ale přesouvají se i na mobilní zařízení. Platnost výše uvedeného pravidla lze tedy rozšířit i na požadavek nezávislosti nejen na operačním systému, ale také jako nezávislost na zařízení.

4.6 Komunikace – SOA a EDA

V předchozím textu byl již zmíněn termín Event Driven Architecture (EDA), neboli událostmi řízená funkčnost či architektura aplikace. V návaznosti na tomto pojmu, je nutné představit i koncept architektury orientované na služby, nebo-li Service Oriented Architecture - označované jako SOA.

SOA je obecným funkčním modelem, který lze velmi dobře aplikovat pro potřeby IT. Je postaven na principu modularity a zapouzdření jednotlivých komponent, předpokládá slabé vazby mezi jednotlivými komponentami a službami, jejich bezstavovost a nezávislost. Komunikační model je ve většině případů postaven na synchronní komunikaci. SOA využívá žádost-odpověd (request-response) modelu, ve kterém se klient dotazuje serveru a posléze čeká na jeho odpověď. Dá se konstatovat, že SOA model je téměř shodný s funkčností běžných internetových stránek, což potvrzuje skutečnost, že tento koncept není nikterak nový, protože stejně tak jako internetové stránky tak i SOA ke své funkčnosti využívá protokolu HTTP [16][41].

Přístup událostmi řízené architektury (EDA) je oproti SOA asynchronní, což umožňuje daleko efektivnější komunikaci a vyřizování požadavků. Stejně jako u SOA, je u EDA komunikace rozdělena mezi klienta a server. EDA oproti SOA však zavádí funkci či vrstvu manažera. Odpovědností tohoto manažera je evidence klientů, kteří naslouchají na dané události. Události mohou být vyvolány uživatelským požadavkem, ale také pouhým stavem, vyvolaným nezávisle na uživatelské interakci. Při výskytu takovéto události je odpovědností manažera informovat o této akci klienta, který dané události naslouchá. EDA mění chápání rolí a oproti SOA, která využívá modelu request-response, zavádí model publisher-subscriber. EDA je oproti SOA dominantou desktopových aplikací [16][41].

Rozdíl mezi SOA a EDA přístupem je tedy ve způsobu komunikace. Tuto rozdílnost komunikace definují dva komunikační modely sloužící pro distribuci informací či dat: *push* a *pull* model. SOA architektura využívá *pull* model, kdy je uživatelem iniciován požadavek (request), ve kterém od serveru požaduje dodání dat (pull). Server v tomto případě není schopen jakkoli notifikovat uživatele na nastalé události bez předchozího

požadavku či akce uživatele. Uživatel se tedy musí na nová data či stavy aktivně dotazovat. Naopak EDA model využívá *push* modelu, kdy je možné uživatele informovat o změnách bez jeho předchozího přičinění. Server je tedy schopen uživateli poskytovat data nezávisle na požadavcích uživatele [16][41].

Jak již bylo poznamenáno, EDA přístup je dominantou desktopových aplikací, avšak s rozvojem technologií bylo možné tento přístup přenést i do prostředí internetových aplikací. Komplikací přenosu však byl samotný protokol HTTP. Protokol HTTP, jež je ze své podstaty bezstavový, a EDA pro svou funkčnost potřebují uchovávat informaci o stavech. Řešením popsané situace bylo využití tzv. Cookies, které jsou schopné tyto informace uchovávat.

Dalším problémem, který bylo nutné při adaptaci EDA do prostředí Internetu vyřešit, byl samotný *pull* model. Jelikož je HTTP protokol bezstavový a po vyřízení požadavku uživatele je spojení uzavřeno, server nemá možnost uživatele dodatečně kontaktovat uživatele s informací o nové události. Tento problém byl vyřešen periodickým voláním či obnovou HTML stránky. Řešení dalo vzniknout pojmům jako je *Pooling* či *Long pooling*. *Pooling* v tomto případě znamená periodické dotazování serveru klientem na nové události, čímž je simulován *pull* model.

Jako technologické řešení popsaného problému byla využita již dříve zmiňovaná technologie AJAX, která díky možnosti asynchronního volání poskytla prostředky k efektivnímu využití poolingu.

4.7 Interakční model

Prozatím byly předloženy argumenty týkající se zejména pohledu, kterým se na RIA dívá vývojář. Byla představena obecná architektura aplikací, komunikační model a technologie. Důležitým aspektem, na který je nutné při návrhu aplikace brát ohled, je způsob jakým pobíhá interakce mezi aplikací a uživatelem. V tomto smyslu mluvíme právě o Interakčním modelu.

Matthias Müller-Prove a Frank Ludolph [24][25] definovali interakční model následovně: „*Interakční model popisuje způsob, jak uživatelské akce ovlivňují změny v systému a naopak. Jedná se o nepřetržitý tok informací mezi uživatelem a strojem, jenž představuje komunikaci s PC.*“

4.7.1 Jednoduché vs. dvojité kliknutí

Při návrhu aplikace je třeba porozumět návykům uživatelů, kteří budou s aplikací pracovat. Interakční model desktopových aplikací je oproti webovým aplikacím ve většině případů odlišný. Jako příklad lze uvést rozdíl mezi dvojitým a jednoduchým kliknutím myši. V desktopovém interakčním modelu, který může zahrnovat jak operační systém jako takový, ale i desktopové aplikace, je jednoduché stlačení myši používáno k výběru objektu (výběr ikony, výběr položky v menu) či k určení pozice kurzoru. Dvojité kliknutí je především využíván ke spouštění příkazů či programů. Oproti tomu internetové aplikace jednoduché kliknutí využívají ke spuštění příkazů (kliknutí na odkaz způsobí změnu stránky či odeslání formuláře). Taktéž se využívá k určení pozice v upravitelných polích aplikací. Dvojité klinutí je v případě webových aplikací, které jsou postaveny na technologiích HTML, nevyužito. Technologie využívající zásuvných prohlížečových modulů (Adobe Flash, MS Silverlight) disponují podporou dvojitého kliknutí a umožňují tak plně zastoupit chování desktopové aplikace, navíc umožňují změnit chováni jednoduchého kliknutí [24][25].

4.7.2 Označování textu

Dalším velice důležitým aspektem je označování textu. V případě webových aplikací jsou uživatelé zvyklí označovat veškerý text, který je zobrazen prohlížečem. Nutno však poznamenat, že v případě využití technologií jako je Adobe Flash či MS Silverlight toto pravidlo zcela neplatí. Prostředí těchto zásuvných modulů neumožňuje libovolný výběr jakéhokoli zobrazeného textu, musí se explicitně povolovat.

4.7.3 Přímá manipulace

K zajímavým závěrům lze dojít při zamyšlením nad interakčním modelem ve smyslu *přímé datové manipulace* (Direct manipulation). Termín byl představen v roce 1983, autorem byl Ben Shneiderman, který přímou manipulaci popisuje [36]: „*Základní ideou přímé manipulace je viditelnost pouze těch objektů a akcí, které jsou předmětem zájmu a nahrazení ručně psaných příkazů, sloužících k manipulaci s objektem, sadou nástrojů, které umožní přímo manipulovat s těmito objekty pouhým ukázáním (kliknutím)*“. Přímá manipulace je tedy snaha o virtualizaci objektů, prezentovaných aplikací, přenesení jejich reprezentace do reálného světa a nabídnutí takových nástrojů, které umožní uživateli s těmito objekty přímo manipulovat. Přímá manipulace s objekty se v hojné míře využívá v grafických programech, jako je Adobe Photosthop či Gimp, či v operačních systémech, např. při provádění akce *Drag&Drop* [36][37][38].

Při aplikaci výše uvedených tvrzení ohledně HTML webových aplikací je nutné prohlásit, že nelze zcela splnit všechny kladné požadavky. I aplikace obohacené o JavaScript jsou schopné plnit požadavky přímé manipulace pouze do jisté míry a dalo by se tvrdit, že i značně obtížně. V případě aplikací postavených na Adobe Flash či MS Silverlight, je splnění těchto požadavků, oproti HTML aplikacím, zvládnuto ve vyšší míře. Oba přístupy však trpí vlastnostmi plynoucími z architektury klient-server; zásadním problémem je odezva a rychlost zpracování, jež jsou velmi důležité. Aby nedošlo ke zmatení uživatele, musí být reakce aplikace rychlá, jinak by uživatel mohl nabýt dojmu, že jím vyvolaná akce nemá žádnou odpověď. V případě časově náročných operací, lze uživateli zobrazit informaci o provádění této operace, a pokud to prováděná operace a technologie umožňuje, zobrazit i detailnější informace o jejím průběhu.

4.7.4 Drag&Drop

V předchozím textu byla zmíněna akce *Drag&Drop*, která jedním z charakteristických představitelů přímé manipulace. Vzhledem k popularitě této akce v operačních systémech či desktopových aplikacích jí nelze zcela přehlížet a převod této akce do světa webových aplikací může pro uživatele znamenat zásadní výhody. V případě webových aplikací je však nutné rozdělit tuto akci na dva druhy: Drag&Drop v rámci aplikace a na Drag&Drop z vnějšku do aplikace, či naopak.

V případě Drag&Drop v rámci webové aplikace či stránky lze prohlásit, že s rozvojem technologií a JavaScriptových frameworků, se stal Drag&Drop běžnou součástí webových aplikací. Oproti tomu Drag&Drop mimo aplikaci, tj. mimo rozhraní internetového prohlížeče, nemůže být plně implementován. Lze namítnout, že některé prohlížeče umožňují přetažení některých elementů stránky mimo prohlížeč (např. obrázků), toto je však specifická vlastnost daného prohlížeče a aplikace jako taková nemůže tuto vlastnost ovlivnit. Druhým aspektem je přenos objektů z vnějšku do webové aplikace. V tomto případě je bohužel opět nutné prohlásit, že jej nelze zcela a spolehlivě implementovat. Některé prohlížeče nabízejí tuto funkci, ale jde opět o specifickou vlastnost daného prohlížeče.

4.7.5 Akce Zpět

Zajímavou vlastností interakčního modelu je vlastnost kroku *Zpět* (Undo). Alan Cooper [7] definuje vlastnost Zpět jako „*záchranu pro uživatele v potížích*“. Pro běžného uživatele se jedná o velmi důležitou vlastnost aplikace a mnohdy velmi využívanou, a to do takové míry, že si její využití uživatel ani neuvědomuje. V závislosti na zkušenostech uživatele s obsluhou počítače se chápání této funkce mění. Začátečník může funkci *Zpět* chápat jako záchrannou brzdu v případě potíží, kdežto zkušenější uživatel chápe tuto funkci jako seznam provedených akcí, které lze vrátit zpět. Uvedené rozdílné chápání funkce *Zpět* klade na implementaci v systému velké nároky, všechny uživatelem provedené akce a stavy aplikace samotné je nutné zaznamenávat. Alan Cooper [7] v této věci poznamenává, že nejlepším řešením je navržení globální akce *Zpět*, která při požadavku provede zneplatnění předchozí operace, bez ohledu na to, zda byla akce provedena prostřednictvím přímé manipulace s objektem nebo pomocí dialogu aplikace [6][7].

V případě webových aplikací, běžících v internetovém prohlížeči, je implementace funkce *Zpět* velmi špatná. Výše uvedené tvrzení v případě internetového prohlížeče zcela selhává, protože bez ohledu na implementaci akce zpět v aplikaci nabízí prohlížeč vlastní implementací této funkce, která má navíc prioritu před implementaci v aplikaci. V komunikačním kontextu klient-server je využití funkce *Zpět* velmi problematické, lze implementovat funkci *Zpět* v rámci aplikace, která provedená serverová volání označí jako neplatná, avšak dochází zde ke zdvojení implementace funkce *Zpět*. Jedna implementace je v rámci webové aplikace a druhá v rámci prohlížeče. Rozlišení těchto rozdílných implementací a zejména jejich pochopení klade zvýšené nároky na uživatele a může vést k jeho zmatení.

Problematiku akce *Zpět* lze demonstrovat na výsledcích empirické studie, kterou provedl Paul Pop [33]. Zkoumání studie porovnává využití funkce *Zpět* v desktopové aplikaci a webové aplikaci. Přestože má tato studie jiné zaměření než si klade za cíl tato diplomová práce, lze z ní přesto získat jeden zajímavý závěr. Využití funkce *Zpět* v rámci internetového prohlížeče vykazovalo v rámci studie velkou chybovost. Tato chybovost byla způsobena chybným pochopením funkce *Zpět* testované aplikace a rozhraní prohlížeče. Uživatelé, podílející se na dané studii, často zaměňovali tlačítko *Zpět* (Back) se samotnou akcí *Zpět* (Undo) a analogicky zaměňovali tlačítko *Dopředu* (Forward) s akcí *Znova* (Redo). Lze tedy prohlásit, že stav internetového prohlížeče a stav aplikace, běžící v internetovém prohlížeči, je navzájem konfliktní.

4.7.6 Notifikace/Upozornění

Přestože se může na první pohled zdát zobrazování upozornění a notifikací jako nepříliš důležitý problém, opak je pravdou. Uživatelé jsou zvyklí, že je aplikace upozorní na událost (například konání schůzky), a to i v případě, že s ní aktivně nepracují. V případě desktopových aplikací zodpovídá za upozornění proces dané aplikace, který běží na pozadí a v případě potřeby uživatele viditelně upozorní. Jako příklad lze uvést přijetí nového e-mailu, událost v kalendáři, informace o nedostatku volného místa na pevném disku. Implementace těchto upozornění je silně závislá na systému a na druhu použitého upozornění. Může jít od zobrazení samostatných oken po problikávání aplikace [6][7][33].

V případě webových aplikací je situace s upozorněním uživatele komplikovaná. Internetové prohlížeče nemají takové možnosti, jako desktopové aplikace. Jsou schopné uživatele upozornit vytvořením nového okna (pop-up) avšak díky častému zneužívání bývá tato možnost blokována samotným prohlížečem a je nutné ji explicitně povolit.

Zobrazování upozornění v rámci aplikace je vhodné, avšak v případě, že uživatel s aplikací aktivně pracuje. Je-li okno s webovou aplikací překryto jinou aplikací, upozornění je sice zobrazeno, avšak uživatel není jakkoli upozorněn. Dalším problémem je případné nechtěné zavření prohlížeče a tím i ukončení práce s aplikací, aplikace již dále nemá možnost uživatele jakkoli upozornit.

4.7.7 Zhodnocení

Z výše uvedených argumentů je nutno vyvodit důležitost zamyšlení se nad interakčním modelem aplikace jako jednoho ze zásadních kroků při výběru vhodné technologie. To, jak uživatel chápe možnosti a ovládání aplikace, je velmi důležitým aspektem při návrhu funkčnosti aplikace, což bylo demonstrováno na příkladu akce *Zpět*. I přes technologie, které velmi markantně snižují rozdíl mezi desktopovými a webovými aplikacemi, lze tvrdit, že interakční model je stále rozdílný.

4.8 Použitelnost

4.8.1 Myšlenkový model

Pracuje-li uživatel s aplikací, vytváří si svou vlastní představu, jak aplikace funguje, jak reaguje na jeho požadavky a jaký dopad mají tyto požadavky na aplikaci. Uživatelská představa funkčnosti aplikace se nazývá *myšlenkový model* (Mental model) či konceptuální model a slouží uživateli k lepšímu pochopení systému. Chápání funkčnosti systému uživatelem se mnohdy od skutečné funkčnosti systému liší. Model, který popisuje skutečnou funkčnost aplikace, je označován jako *implementační model* (Implementation model) a je ve většině případů výrazně složitější než myšlenkový model. Rozdílnému chápání funkčnosti aplikace je třeba věnovat pozornost, protože hrozí riziko, že si uživatel některou funkčnost systému vysvětí chybně či nedokáže díky chybnému myšlenkovému modelu efektivně využít všech poskytovaných nástrojů. Na druhé straně nelze uživatele nutit k pochopení implementačního modelu, který může být z jeho pohledu příliš složitý. Cílem je tedy nalézt takový model, který nenutí uživatele vytvářet si vlastní představu o funkčnosti aplikace, ale naopak poskytnout jednoduší způsob pochopení aplikace a zefektivnit tak uživatelskou činnost. Takový model označujeme jako *reprezentační model* - je maximálně přiblížen myšlenkovému modelu uživatele a je oproštěn od složitostí implementačního modelu. Alan Cooper definuje [7] reprezentační model jako model, pomocí kterého je uživateli vysvětlována funkčnost aplikace.

Při návrhu rozhraní RIA je nutné myslet na výše uvedené aspekty a zejména zvyklosti uživatelů. Uživatel si při práci s počítačem automaticky vytváří svůj myšlenkový model, ve kterém si zjednodušeně vysvětluje, jak systém funguje, čímž si vytváří zvyklosti. Tyto návyky se týkají základních ovládacích prvků desktopových systémů, jedná se o okna, ikony, nabídky (menu) a kurzor (myš). V podstatě každý uživatel, který pracoval s počítačem, musel práci s těmito základními elementy zvládnout a nabyl také povědomí o možných akcích, které lze s těmito objekty provádět, tj. na objekty lze kliknout, lze je přesunout, upravit či smazat. Nutno připomenout, že se vlastně jedná o koncept výše popsané *přímé manipulace* (Direct manipulation), tj. možnost vyvolání akce (klik) na objektu zájmu, např. kliknutí na ikonu programu a její následné přesunutí. Dalším specifikem je uvědomění si perzistence desktopového systému, tj. objekty zůstávají tam, kam je uživatel umístil. Nabízí se možná otázka, kde se nachází interakční model systému. Interakční model desktopového systému je skryt, za grafickým rozhraním, které v tomto případě plní roli reprezentačního modelu. Veškeré akce, které

uživatel provádí, jsou pomocí grafického rozhraní předávány implementačnímu modelu, který provádí potřebné akce na pozadí. Pro představu lze jako příklad uvést běžné kopírování souboru z jedné složky do druhé. V případě využití reprezentačního modelu, tedy pomocí grafického rozhraní systému, zkopírujeme soubor tak, že klikneme na ikonu souboru a přetáhneme jej do požadované složky. Z hlediska uživatele velmi jednoduchá akce. Pokud by měla být splněna stejná úloha s využitím interakčního modelu, který si lze představit jako příkazovou řádku systému, tak složitost operace pro běžného uživatele neúměrně roste. Bylo by nutné příkazovou řádku spustit, vyhledat požadovaný adresář a poté napsat příkaz k samotnému kopírování souboru a to vše ještě za předpokladu, že je uživateli známa syntaxe příkazu ke kopírování.

Prezentace jednotlivých modelů je v souvislosti s webovými aplikacemi oproti desktopovým systémům v mnohých aspektech rozdílná. Internetový prohlížeč lze v rámci desktopového systému chápat jako komponentu reprezentačního modelu. Internetový prohlížeč je vlastně pouze zobrazovacím zařízením, jež slouží k zobrazení internetové stránky neboli dokumentu. Zde je možné stanovit milník, kdy je ještě chápání desktopových a webových modelů shodné. Rozdíly jednotlivých modelů vyniknou při změně chápání internetové stránky, jakožto pouhého dokumentu. Funkčnost a bohatost HTML stránek byla navýšena do takové míry, aby se internetový prohlížeč stal pouhým prostředkem, který slouží uživateli pouze jako přístupový bod k aplikacím a informacím. Lze tvrdit, že se v podstatě jedná o nový svět, který má svá pravidla a specifika.

Markantní rozdíly jsou evidentní již v rozdílném chápání základních ovládacích prvků, jako jsou již výše zmíněná okna, ikony a nabídky. Lze prohlásit, že tyto elementy v mentálním modelu webové aplikace buď zcela chybí, nebo je změněno jejich chápání. V případě použití ikon je rozdíl nejjasnější. V případě desktopového systému slouží jako reprezentace objektů, se kterými lze provádět akce. Reprezentují soubory, programy, adresáře. V případě webových aplikací jsou ikony použity v naprosto jiném kontextu. Slouží jako náhrada textu, jako loga či jako vizuální podpora odkazů.

Na základě výše popsaných rozdílu v chápání jednotlivých modelů lze vyvodit závěr, že je při analýze a návrhu RIA vhodné myslet na uživatelské zvyklosti, nabyté prací v desktopovém prostředí. Pokud bude zvolena taková technologie, která umožní maximální přiblížení reprezentačního modelu aplikace desktopovému reprezentačnímu modelu, hrozí zde riziko, že uživatel bude aplikovat své zvyklosti při ovládání aplikace, čímž může dojít k jejímu chybnému či neefektivnímu využití. Naopak zcela jiná logika ovládání aplikace je spojena s vyšší potřebou na zaškolení a vyšší mírou počáteční chybovosti. Je si nutné uvědomit, že i při maximálním přiblížení obou modelů stále existují technologické překážky, které je budou v mnoha aspektech odlišovat.

4.8.2 Souborový koncept

Problematika souborového konceptu nabízí zajímavý pohled na použitelnost RIA a nemusí být na první pohled patrné, proč představuje problém. Pod pojmem souborový koncept si lze představit princip práce se soubory v operačním systému či aplikaci, jejich vytvoření a práci s nimi.

Souborový koncept lze velmi dobře popsat na činnostech s textovým dokumentem. Většina uživatelů na počítači s textovým dokumentem běžně pracuje. Je-li dokument upraven a uživatel se pokusí zavřít aplikaci, je dotázán, zda se mají provedené změny uložit. U nezkušeného uživatele může v tomto případě dojít k nepochopení. Funkce implementačního modelu požaduje po uživateli informaci, zda a kam má být dokument na disku uložen či zapsán, což může kolidovat s mentálním modelem uživatele. Ten si dokument představuje jako popsaný list papíru a nikoliv jako data, která jsou bez uložení dokumentu držena pouze v paměti, a jejich neuložením dojde k jejich nenávratné ztrátě. Problém v chápání mohou představovat i příkazy *Uložit* a *Uložit jako*, kdy uživatel vytvoří kopii dokumentu a pokračuje dále v úpravě originálu, který chtěl původně zachovat. Dle tvrzení Alana Coopera [6] je mentální model dokumentu chápán uživatelem tak, že existuje pouze jeden dokument, který náleží uživateli a může s ním libovolně manipulovat. Dojde-li však k otevření dokumentu, tak z hlediska implementačního modelu dochází k rozdílu v chápání, protože vzniká druhá kopie dokumentu, která je uložena v paměti, a jsou do ní zapisovány uživatelem provedené změny. Tato pracovní verze dokumentu již nenáleží uživateli, ale aplikaci, pomocí níž je editace prováděna.

Na poli webových aplikací není situace jiná. K uložení dokumentu je stále vyžadována akce uživatele a automatické ukládání zcela chybí. Zásadním rozdílem je však využití databáze jakožto média pro ukládání dat. Alan Cooper [6] vidí využití databází jako výhodu a zároveň jako možnost vypustit implementační model používaný v rámci desktopových systémů. Při úpravě informace není tedy nutné vytvářet kopii dokumentu a nedochází tak k narušení mentálního modelu uživatele.

4.9 Paradigma internetového prohlížeče

Tim Berners-Lee [4], jakožto představitel konceptu WWW, zamýšlel využít Internetu jako nástroje pro publikování dokumentů a jejich vzájemné propojení. Jeho myšlenka se opravdu stala skutečností a lze tvrdit, že internetový prohlížeč je již od roku 1991 primárním nástrojem pro přístupu k informacím na Internetu. Rozvoj internetových stránek a aplikací dal vzniknout několika druhům internetových prezentací či aplikací. Alan Cooper [6] rozdělil tyto aplikace do několika kategorií.

První kategorií jsou **Informační internetové stránky**. Jde o vůbec první kategorii stránek, která vznikla po představení konceptu WWW. Tyto internetové stránky jsou sestaveny z jednotlivých stránek či dokumentů, nejčastěji v následné či hierarchické struktuře. Jde o jednoduché stránky bez hlubší logiky, důraz je kladen zejména na obsah. Lze prohlásit, že pro tento typ stránek bylo přizpůsobeno grafické rozhraní prohlížeče a prohlížeč samotný. Funkčnost ovládacích prvků prohlížeče souhlasí s mentálním modelem uživatele, dopředná i zpětná navigace funguje dle předpokladů, historie navštívených stránek je platná, obnovení či zastavení nahrávání stránky nemá zásadní dopad na zobrazovanou stránku. Každá ze zobrazovaných stránek má ve většině případů unikátní URI (Uniform Resource Identifier). Menu prohlížeče taktéž souhlasí s mentálním modelem uživatele, protože plně ovládá zobrazovanou stránku. Lze prohlásit, že souborový koncept je v jisté úrovni splněn taktéž, stránku lze uložit či vytisknout s očekávaným výsledkem.

Další kategorií jsou **Webové aplikace**, jejichž cílem je přiblížení k desktopovým aplikacím. Jedná se o silně interaktivní aplikace s dynamickým grafickým rozhraním. Souborový model se u těchto aplikací vytrácí. Interakce a přechod mezi jednotlivými stránkami je povětšinou prováděn asynchronně a zobrazovaná stránka může obsahovat více stavů, které jsou závislé na aktuálním stavu aplikace. Navigace internetového prohlížeče v případě těchto aplikací již neslouží k navigaci a vzhledem k masivnímu využití asynchronního volání je spíše ke škodě. Mentální model u ovládacích prvků „Zpět“ či „Dále“, kdy je při jejich využití předpokládán návrat provedené akce zpět či posun na předchozí provedenou akci, je v tomto případě velkým nebezpečím, neboť při chybném využití zmíněných ovládacích prvků může dojít ke zmatení aplikace a vyvolání chyb. Taktéž možnost „Zpět“, kterou lze nalézt ve standardní nabídce prohlížečů, neplní požadovanou funkčnost. Dalo by se očekávat, že při jejím použití dojde k navrácení stavu aplikace či předchozí provedené akce, tak jak je uživatel zvyklý u desktopové aplikace. Velmi kritickou záležitostí může byt použití tlačítka „Obnovit“ (Reload), sloužícího pro obnovení obsahu stránky, kdy může dojít ke ztrátě neulože-

né práce uživatele či kompletnímu zmatení grafického rozhraní aplikace a může tak dojít k neočekávaným chybám. Velmi problematickou záležitostí je porušení platnosti URI, u kterého se předpokládá, že každá zobrazená stránka či obsah, má jednoznačný ukazatel. Toto pravidlo je porušeno díky využití asynchronního volání a vnitřních stavů jednotlivých stránek. Popsaná situace má dopad na využití „Záložek" prohlížeče. Lze namítnout, že zkušenější uživatelé těchto aplikací chápou rozdíl mezi běžnou internetovou stránkou a aplikací a tím i rozumí rozdílům ve změně ovládání těchto stránek. Tato námitka je platná, avšak vzhledem k tomu, že vývojář nemá možnost jakkoli ovlivnit chování standardních ovládacích prvků prohlížeče (Zpět, Další, Obnovit, atp.), je nutné s výše popsaným chováním počítat při návrhu aplikace.

Třetím druhem aplikací jsou **Aplikace s přístupem k internetu**. Jde o aplikace, které neběží v základním rámci internetového prohlížeče, nýbrž se jedná o samostatné aplikace, běžící na klientském systému, avšak ke své funkčnosti vyžadují připojení k internetu. Nabízejí vyšší uživatelský komfort, snaží se o maximální přiblížení k desktopovým aplikacím a ve velké míře těží z faktu, že jsou nainstalovány přímo na klientské stanici (vyšší výkon, Drag&Drop, přístup k zdrojům). Definují si vlastní ovládací prvky či vlastní navigační menu a nehrozí tak zmatení uživatele, jako tomu bylo u předchozího typu aplikací.

Může se zdát, že tyto aplikace jsou řešením na výše uvedené problémy, avšak není to zcela pravda. Jak Alan Cooper [6] uvádí, výhodou aplikací přístupných pomocí rozhraní prohlížeče je zejména jejich dostupnost, bez nutnosti instalovat software na klientský počítač. Uživatel má takovou aplikaci dostupnou kdekoliv a lze tvrdit, že i z jakéhokoliv počítače připojeného k internetu [6][33].

Zajímavý pohled na problematiku navigace poskytování obsahu v rámci prohlížeče má Jakob Nielsen. V článku *Death of Web Browsers* [4] vyslovuje hypotézu, že klíčovým elementem budoucnosti by měl být odklon od používání internetových prohlížečů. Nutno podotknout, že Jakob Nielsen zaměřuje spíše svou kritiku na princip přístupu k informacím jako takovému a nikoliv na webové aplikace. Jakob Nielsen si bere na mušku zejména fakt, že dostupnost informacím by neměla záviset na jejich umístění. Uživatel by se k nim měl dostat pomocí jednotného rozhraní a to bez ohledu na to, zda je informace umístěna lokálně či na vzdáleném severu. Internetové prohlížeče poskytují přístup pouze k informacím umístěným vzdáleně, prezentují dokumenty jako objekty a umožňují navigaci mezi nimi. Dle Jakoba Nielsena, by měla být zodpovědnost rozdělena, navigace by měla být více obecná a měla by být zodpovědností operačního systému a měla by poskytovat obecný a unifikovaný přístup k historii, záložkám a vyhledávání a to opět bez ohledu na umístění informace [27][28].

Na základě výše uvedených problémů při přístupu k informacím a aplikacím pomocí internetového prohlížeče lze tvrdit, že je role prohlížečů přeceňována. Internetové prohlížeče jsou mnohdy využívány k plnění úkolů, na které nebyly navrženy, z čehož plynou problémy s použitelností, nebezpečí při chybném využití navigačních prvků prohlížeče, atp.

Řešením by bylo přesunout RIA mimo rozhraní prohlížeče. Takový přístup by umožňoval vlastní definici navigačních prvků. V tomto směru je nutné zmínit projekt firmy Mozilla, zvaný „Prism“. Jde v podstatě o minimalizovaný internetový prohlížeč Mozilla Firefox, nabízející pouze prostředí základního aplikačního rámce, zcela chybí běžná navigační lišta (zpět, další), nabídky menu (Soubor, Upravit) a vstupní řádek pro vložení URI adresy. Přístup k webové aplikaci je umožněn pomocí zástupce na ploše, který jako parametr obsahuje URI adresu, na kterou chce uživatel po spuštění přejít. Tuto adresu nelze po spuštění uživatelsky upravit. Přítomnost tlačítek „Zpět“ a „Dále“ zcela chybí. Bohatost aplikace je stále omezena vlastnostmi internetového prohlížeče, avšak díky transparentnosti prohlížeče je navigace plně v kompetenci webové aplikace a je tak možné maximální možné přiblížení mentálnímu modelu uživatele. Nutno podotknout, že projekt je v současné době neaktivní a jeho vývoj již neprobíhá. Firma Mozilla však zahájila nový projekt nazvaný Chromeless, ideově vycházejícího z projektu Prism, jehož cílem by mělo být odstranění omezení plynouchcích pro aplikace běžící v internetovém prohlížeči (např. přístup k souborovému systému).

4.10 Vzhled

Jak již bylo výše zmíněno, RIA se snaží o maximální uživatelskou přívětivost a ovladatelnost. Aspekt vzhledu aplikace se těchto vlastností přímo týká. Uživatelé jsou zvyklí, že webové stránky obsahují, oproti desktopovým aplikacím, daleko více interaktivních prvků či animací. Dalo by se říci, že jsou značně stylově zaměřené. Každá tradiční webová stránka má svůj vlastní vzhled (design), kterým definuje i svou identitu, či propaguje svou značku. Oproti tomu styl rozhraní tradičních desktopových aplikací je mnohdy dán operačním systémem, kdy je možné změnit barevné schéma, avšak styl prvků je povětšinou pevně spjat s operačním systémem. Lze tvrdit, že webové stránky a aplikace mají velký potenciál z hlediska individuality, oproti tomu desktopové aplikace poskytují standardní rozhraní a ovládací prvky.

Vzhledem k silnému spojení webových aplikací s webovými stránkami, je uživateli u těchto aplikací očekávána i vyšší míra jedinečnosti. Styl stránek a barevné schéma musejí však být navrženy tak, aby uživatele neobtěžovaly či nenarušovaly jeho sou-

středění k práci, nebo ho dokonce nemátli při využívání aplikace. Dalším aspektem, na který je nutné myslet při návrhu grafického rozhraní aplikace, jsou zrakově postižení uživatelé, pro které může být silně stylově orientované rozhraní nepoužitelné.

S individualitou aplikace a jejím grafickým rozhraní je úzce spjat pojem téma, často označovaný jako *skin*. Témata či skiny jsou prostředkem, který jednak mění vzhled prvků, ale také umožňuje změnit jejich chování. Tato možnost změny se mnohdy nazývá *Look and Feels*, což je možné volně přeložit jako *Vzhled a pocit*. Změna grafického rozhraní aplikace pomocí témat je spíše předmětem prvního pojmu „Vzhled" a je možné jej definovat jako grafickou reprezentaci prvku aplikace, kdežto „Pocit" popisuje způsob chování dané grafické reprezentace a jejích komponent.

Využívání skinů či témat je důsledkem jak uživatelských požadavků na individualitu aplikace, tak různých názorů uživatelů směrem k líbiví grafické reprezentaci. Uživatelé chtějí pracovat s aplikací, jejíž rozhraní se jim líbí, usnadňuje jim práci s aplikací a poskytuje přehlednou navigaci. Chápání těchto parametrů je však uživatel od uživatele různé, proto aplikace umožňují změnu vhledu a chování pomocí témat. Některé aplikace dokonce poskytují uživatelům nástroje, pomocí kterých může uživatel upravit styl aplikace, popř. vytvořit přímo vlastní téma, které může posléze sdílet s jinými uživateli.

S tématy či skiny souvisí i využití vizuálních efektů. Vizuálními efekty jsou myšleny animace ovládacích prvků či dynamické přechody mezi jednotlivými stavy aplikace. Rozumné využití vizuálních efektů může zpříjemnit uživateli práci a zvýšit estetičnost grafického rozhraní. Chet Haase a Romain Guy poznamenávají [8], že vizuální efekty by měly být využity „efektivně" a nemělo by docházet ke stavu, kdy vizuální efekty vytvářejí z aplikace hororové představení. Jennifer Tidwell uvádí [39], že animace ovládacích prvků mohou sloužit jako podpora uživatelské interakce s aplikací a tím usnadnit uživateli orientaci ve virtuálním prostředí aplikace. Animované přechody mezi jednotlivými stavy mohou sloužit jako názorná ukázka změny v aplikaci a usnadnit tak uživateli pochopení dané funkčnosti. Jennifer Tidwell však zároveň upozorňuje, že reakční doba efektů a animací by měla být rychlá, protože při delší časové prodlevě hrozí ztráta zájmu uživatele.

Na základě výše uvedených tvrzení lze konstatovat důležitost využití vizuálních efektů jako nástrojů pro zvýšení atraktivity a použitelnosti aplikace. Je nutné však věnovat pozornost faktu, že efekty musí být využity „efektivně", tj. musí být respektováno vhodné trvání a uživatele nesmí zdržovat od práce či jej dokonce otravovat [8][39].

4.11 Charakteristické vlastnosti

V předchozích kapitolách byly analyzovány různé aspekty, na které je nutní při návrhu RIA přihlížet, popřípadě ovlivňují výběr technologie, která bude zvolena pro tvorbu RIA. Na základě těchto tvrzení lze definovat charakteristické rysy RIA:

- Klient – server architektura;
- Spojení výhod desktopových a webových aplikací;
- Asynchronní komunikace;
- Síťová efektivnost;
- Offline provoz;
- Snadná dostupnost;
- Multiplatformnost;
- Snadná údržba, správa a distribuce;
- Interaktivní a bohaté uživatelské rozhraní;
- Multimediální obsah.

4.12 Výhody a nevýhody

Nespornou výhodou aplikací, které běží v běžném internetovém prohlížeči, je jejich dostupnost. Vzhled je stažen ze vzdáleného serveru a není tedy nutná instalace vlastní desktopové aplikace. K běhu je potřeba pouze internetový prohlížeč, který má v dnešní době předinstalován každý operační systém. Díky serverové centralizaci se výrazně zjednodušuje vývoj a správa takovýchto aplikací a není potřeba další systém, který by zajišťoval distribuci nových verzí. Jako další výhodu lze označit snahu jednotlivých RIA platforem maximálně se přiblížit vzhledu a chování klasických aplikací, což výrazně ulehčuje uživatelům pochopit základní princip práce s takovýmto systémem.

Na základě informací, které zde byly doposud předloženy, je možné tvrdit, že RIA aplikace přinášejí pouze výhody. Z pohledu uživatele představují RIA aplikace problém v přístupnosti. V mnoha aplikacích nefungují klávesové zkratky, na které jsou uživatelé zvyklí, nefunguje tlačítko zpět a historie navštívených stránek se v takovýchto aplikacích mnohdy neukládá. Běžně lze narazit na problémy s indexací obsahu pro vyhledávače. Je-li na RIA nahlédnuto z pohledu vývojáře, lze opět narazit na problémy, tentokrát spojené se samotným vývojem aplikace. Přestože platformy pro tvorbu RIA poskytují bohaté kolekce funkcí, často vzniká problém v podobě přílišné složitosti a s tím spojeným vývojem, laděním a nasazením do produkčního prostředí. Dalším negativním faktorem je nemožnost pracovat v režimu Offline, tj. ve stavu, kdy je

počítač odpojen od sítě. Mnohé aplikace jsou závislé na nepřetržitém připojení k síti, a pokud není tento stav ošetřen, tak jejich odpojení může způsobit problémy [35][22].

Další výhody a nevýhody již nelze aplikovat dostatečně obecně, budou tedy vysvětleny dle výše zmíněného rozdělení a to dle aplikací:

- využívajících proprietárních zásuvných modulů:

Výhody	Nevýhody
Prohlížečový modul, na kterém je běh aplikace závislý, pochází od jednoho autoritativního dodavatele, čímž odpadá ladění aplikací pro různé prohlížeče	Pro provoz aplikace je potřeba zásuvný prohlížečový modul, bez tohoto modulu nebude aplikace fungovat
Výkon aplikací, které využívají vlastních modulů je mnohdy daleko vyšší než u JavaScriptového interpretu prohlížeče	Ačkoli se aplikace tváří, že běží v prohlížeči, častokrát lze narazit na problémy s ovládáním prohlížeče. Dochází k nefunkčnosti klávesových zkratek prohlížeče, není možné použít tlačítko *Zpět*, formulářová pole si nepamatují předchozí hodnoty. Také není možné využít prohlížečový modul na zapamatování přihlašovacích údajů. Také může dojít ke zmatení uživatele, pokud dojde k nahrání aplikace v nezašifrovaném spojení, avšak aplikace nadále komunikuje pomocí zabezpečených kanálů
Odpadají veškerá technologická omezení, která s sebou přináší spojení HTML/CSS/JavaScript	
Vývoj aplikací pomocí těchto zásuvných modulů bývá často efektivnější, protože jde o technologie vytvořené na míru vývojářům	
K dispozici jsou bohaté kolekce nástrojů, které simulují rozhraní operačního systému	
Přítomnost kvalitních vývojářských nástrojů s podporou kontroly syntaxe a ladění	

Tabulka 1: Přehled výhod a nevýhodřešení využívajících proprietálních zásuvných modulů [autor]

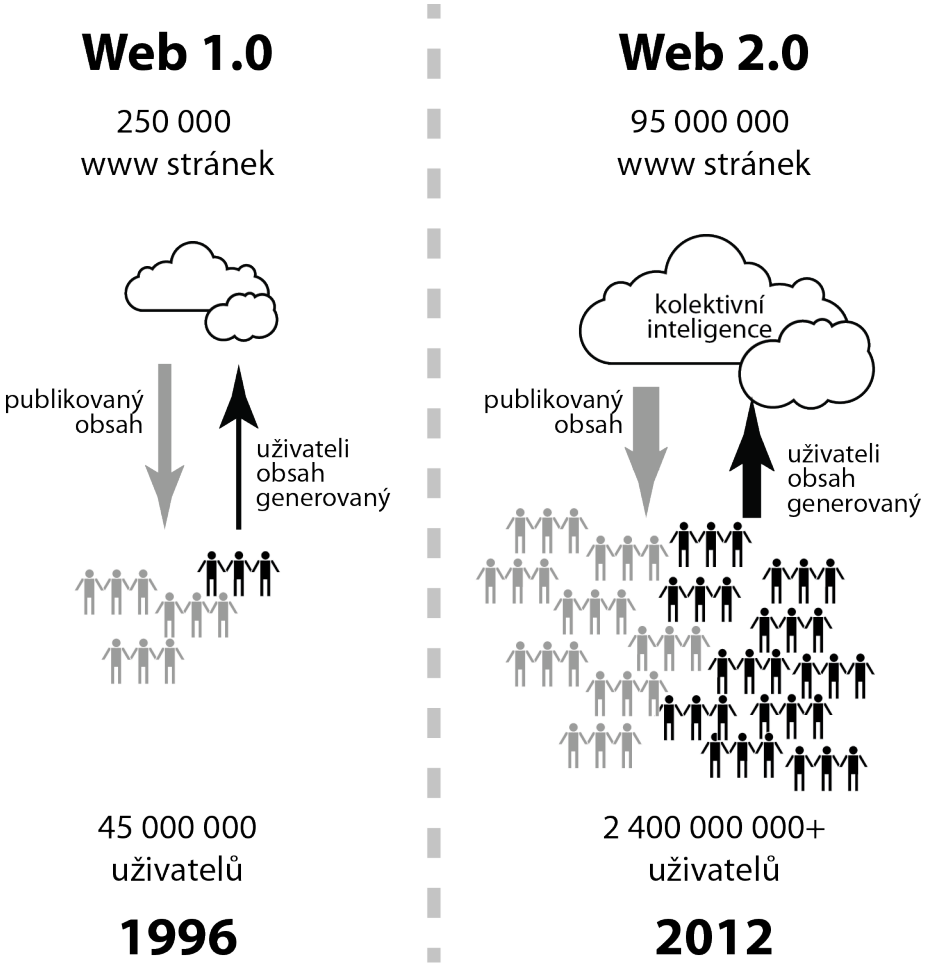
- využívajících technologií internetového prohlížeče

Výhody	Nevýhody
Multiplatformnost – lze tvrdit, že aplikace, které využívají běžných technologií poskytovaných prohlížečem, fungují na téměř všech operačních systémech	Problémy s odladěním aplikace – vzhledem k velkému počtu prohlížečů, je velice náročné aplikace odladit tak, aby byl vzhled všude jednotný
Jednoduchost údržby a vývoje aplikací	Problémy s výkonností JavaScriptu. Technologie využívající zásuvných prohlížečových modulů jsou řádově výkonnější
Aplikace není nutno kompilovat, vzhled aplikace je přenášen do internetového prohlížeče, který následně vykresluje grafiku aplikace	Technologická omezenost HTML/CSS. Tento problém vychází z historického zaměření těchto technologií
	Pomalý vývoj standardu, které by pomohly odstranit limity současných HTML/CSS technologií
	Slabá podpora AJAX vývoje ve vývojářských nástrojích

Tabulka 2: Přehled výhod a nevýhod řešení využívajících proprietálních zásuvných modulů [autor]

4.13 Web 2.0 a RIA

Ačkoli uživateli Internetu může připadat vymezení pojmu Web 2.0 velmi jednoduché, není tomu tak. Web 2.0 není technologií a není to ani přesně definovaný termín; lze říci, že jeho definice nemá ustálený charakter. Samotný pojem může vyvolávat představu, že se jedná o novou verzi WWW, avšak název nevychází z žádného vývoje technických specifikací, jde spíše o změnu přístupu k současným internetovým technologiím. Vznikl z názvu konference O'Reilly Media Web 2.0, pořádané v roce 2004, kde měl odkazovat na návrat internetového podnikání po krachu pádu internetových společností v roce 2001 [29].



Obrázek 3: Znázornění přínosu Webu 2.0

Tvůrcem tohoto pojmu je Tim O'Reilly, který se pokusil definovat Web 2.0 takto: *Web 2.0 je revoluce podnikání v počítačovém průmyslu způsobená přesunem k chápání webu jako platformy a pokus porozumět pravidlům vedoucím k úspěchu na této nové platformě. Klíčovým mezi těmito pravidly je toto: tvořte aplikace, které budou díky síťovému efektu s přibývajícím počtem uživatelů stále lepší. (Což jsem jinde nazval „zapřažením kolektivní inteligence".)* [32].

Tim Berners Lee, tvůrce WWW popsal pojem Web 2.0 následovně: „*Nikdo ve skutečnosti neví, co znamená .... Pokud pro vás Web 2.0 znamená blogy a wiki, tak to jsou lidé lidem. Což byla již původní myšlenka WWW.*" [11]. Uvedený citát naráží na skutečnost, že veškeré technologie, které Web 2.0 využívá, byly dostupné již od prvopočátků WWW a nepřináší tak žádné technologické vylepšení a socializaci internetových aplikací vytvářejí především samotní uživatelé, nikoli technologie.

Web 2.0 tedy znamená především změnu přístupu k využívání internetových technologií, snaží se o zpřístupnění a sdílení informací mezi uživateli a poskytuje jim nástroje, pomocí kterých se mohou spolupodílet na tvorbě obsahu a vytvářet tak sociální sítě.

Vztah Webu 2.0 a RIA je těsný, protože RIA technologie umožňují vytvářet aplikace pro tvorbu obsahu, sdílení informací mezi uživateli a poskytují tak nástroje pro tvorbu sociálních sítí. Může docházet k mylnému dojmu, že RIA aplikace jsou Webem 2.0, popř. vznikly na základě potřeb Webu 2.0. Není tomu tak. Stejně jako technologie WWW i technologie RIA zde byla dříve, než byl pojem Web 2.0 vymezen a definován. RIA jsou prostředkem pro vytváření aplikací, které lze nazvat aplikacemi Webu 2.0.

4.14 Zhodnocení

Je patrné, jak jsou oba přístupy (směr využívající proprietárních zásuvných modulů a směr využívající technologií internetového prohlížeče), ač patřící pod společný termín RIA, v poměrně ostrém kontrastu. Při tvorbě RIA aplikací je nutné zamyšlení a stanovení cílů, které má aplikace v budoucnu splňovat. Je potřebné zvážit pro a proti jednotlivých technologií, a dle toho vybrat vhodnou cestu pro budoucí aplikaci, popřípadě se vydat cestou sloučení. Tato cesta bude pravděpodobně složitější, avšak umožní vývojářům využít maximum výhod z obou technologických směrů. Toto rozhodnutí nemusí být vždy jednoduché a na první pohled jasné. Jako podporu při rozhodování lze využít jistých pravidel:

- technologický směr proprietárních zásuvných modulů použijeme za předpokladu, že aplikace je:
  - natolik specifická, že lze oželet ztrátu některých uživatelů, kteří nebudou ochotni zásuvný modul do svého internetového prohlížeče instalovat;
  - použita v intranetovém prostředí, ve kterém jsme schopni zajistit přítomnost zvoleného zásuvného modulu;
  - natolik komplikovaná, že by použití AJAXu bylo příliš složité nebo nákladné;
  - náročná na výkon.
- technologický směr využívající technologií internetového prohlížeče použijeme za předpokladu, že aplikace je:
  - použita pro širokou klientelu, ve které si nemůžeme dovolit ztrátu uživatelů;
  - jednoduššího charakteru, pouze s jednoduchou aplikační logikou;
  - natolik specifická, že prioritou je běh v jakémkoli prohlížeči.

Uvedená doporučení vycházejí ze zkušeností autora s vývojem aplikací a studiem literatury, která se tématem zabývá. Na poli RIA aplikací lze najít výjimky, které se uvedeným doporučením vymykají. Je třeba si uvědomit několik pravidel, která platí pro vývoj aplikací všeobecně. I pro jednoduchou aplikaci s jedním formulářem lze vybrat robustní technologii, kterou například poskytuje Adobe Flex, avšak v tomto případě si bereme do ruky pomyslný „kanón na vrabce“. Technologie Adobe Flex je pro takto jednoduché aplikace příliš robustní a vývoj takovýchto aplikací se nemusí vyplatit, a to jak z časového, tak z finančního hlediska. Nadruhou stranu, lze napsat komplexní a složitou RIA aplikaci, stavějící na základech technologií, poskytovaných internetovým prohlížečem. Za touto volbou musíme hledat kvalitní analýzu aplikace a vskutku dobrý aplikační návrh.

5 Technologie

V této kapitole budou představeny a zhodnoceny technologie a platformy využívané pro tvorbu RIA. Výsledné hodnocení bude provedeno na základě přiřazení bodů, podle níže uvedených kritérií [19]:

- **uživatelská přívětivost** – hodnotí se úroveň uživatelské přívětivosti, které je úzce spjata s interakčním modelem. V potaz se též bere použitelnost aplikace a míra splnění prohlížečového paradigma;
- **nasazení** – kritérium označuje míru náročnosti v případě implementace či nasazení aplikace a její následnou správu;
- **dostupnost** – jsou hodnoceny minimální požadavky pro běh aplikace, tj. zda je možné aplikaci spustit v prostředí internetového prohlížeče, popř. zda je nutné na klientský systém instalovat dodatečný software potřebný pro běh aplikace;
- **technologie** – zhodnocení jejich využití. Sleduje se, zda je k dispozici pouze jedna technologie či nelze-li technologie kombinovat;
- **délka spuštění** – čas potřebný na stažení a výchozí inicializaci aplikace;
- **klient-server zpracování** – je zhodnocena míra komunikace mezi klientem a serverem, její zpracování a efektivnost, zda je možné využít „pull“ či „push“ volání;
- **nezávislost na platformě** – šířka využití aplikace napříč operačními systémy;
- **podpora pro vývoj** – míra složitosti a vývojářské podpory dané technologie;
- **bohatost grafického rozhraní** – hodnoceny jsou možnosti úpravy vzhledu aplikace, míra přiblížení se desktopovým aplikacím a složitost úpravy vzhledu aplikace;
- **náklady** – ekonomické aspekty zvolené technologie, cena za vývojové nástroje;
- **unikátní vlastnosti** – nebo také zajímavé, tedy takové, které je žádoucí vyzdvihnout;
- **budoucnost** – zamyšlení se nad směrem a smyslem vývoje dané technologie.

5.1 Prohlížeč jako RIA platforma

5.1.1 HTML/XHTML

HTML (HyperText Markup Language) je značkovací jazyk využívaný pro tvorbu struktury obsahu internetových stránek a jejich vzájemného propojení pomocí odkazů. XHTML (eXtensible HTML) je reformovaný jazyk HTML, jehož cílem bylo zajistit soulad se standardem XML a dovolit tak interakci se systémy založenými na XML jazyku.

	Podpora	Omezení	Ostatní
Uživatelská přívětivost	Velmi dobrá interakce s konceptem prohlížeče. Očekávaná funkčnost ovládacích prvku (Zpět, Dále, Stop, Obnovit)	Povětšinou omezené na statické obsahově orientované stránky mnohdy bez vyšší míry interakce	Malá velikost, rychlé nahrávání
Nasazení	Nasazení je velmi jednoduché, není potřeba žádná instalace na klientské stanici		
Dostupnost	Díky standardizaci HTML/ XHTML je dostupnost zajištěna na všech prohlížečích		
Technologie		Možné problémy při rozdílné implementaci v prohlížečích	
Délka spuštění	Velmi rychlé nahrávání		

	Podpora	Omezení	Ostatní
Klient-server zpracování		Velká míra interakce mezi klientem a serverem. Každý požadavek klienta vyvolá opětovné načtení zobrazené stránky. Bez podpory pull.	
Nezávislost na platformě	Zcela nezávislé		
Podpora pro vývoj	Velmi jednoduše naučitelné. Velké množství nástrojů pro tvorbu HTML/XHTML stránek	Velmi nízká míra funkcionality, omezené možnosti skriptování.	
Bohatost grafického rozhraní		Velmi omezené možnosti stylování komponent, není možné vytvářet vlastní komponenty. Hrozí riziko rozdílné interpretace napříč prohlížeči	
Náklady	Vzhledem k nízké komplexnosti technologie jsou náklady velmi nízké		
Unikátní vlastnosti	Nasazení formou překopírování souborů, zdrojové kódy stránky jsou snadno zobrazitelné		
Budoucnost	Budoucnost této technologie je v přechodu na technologii HTML5. Již se neočekává výrazné nasazení HTML/XHTML na poli RIA.		

Tabulka 3: Hodnocení technologie HTML/XHTML [autor][19]

5.1.2 DHTML (HTML, JavaScript, CSS)

DHTML je výsledkem kombinace více technologií, konkrétně HTML, JavaScriptu a CSS. JavaScript je na platformě nezávislý objektově orientovaný skriptovací jazyk postavený na ECMA standardu. I přes tento standard hrozí riziko rozdílné interpretace v různých prohlížečích. JavaScript je využíván zejména k manipulaci s HTML prvky uvnitř stránky. CSS je nástrojem pro definici vzhledu jednotlivých HTML komponent.

Vzhledem k tomu, že DHTML je postaveno na základech HTML, platí zde všechny aspekty, které byly vyjmenovány výše.

	Podpora	Omezení	Ostatní
Uživatelská přívětivost	Podpora Drag&Drop, podpora témat, animací a vizuálních efektů.	Bez podpory audia a videa	
Nasazení	Nasazení je velmi jednoduché, není potřeba žádná instalace na klientské stanici	Vykonávání JavaScriptu může být uživatelem explicitně zákázáno	
Dostupnost	-	-	-
Technologie		Možné problémy při rozdílné implementaci JavaScriptu v prohlížečích	
Délka spuštění	Vzhledem k využití více technologií je spuštění pomalejší nežli u HTML, avšak stále relativně rychlé	Při masivním využití JavaScriptu hrozí navýšení datového objemu	
Klient-server zpracování		Velká míra interakce mezi klientem a serverem. Každý požadavek klienta vyvolá opětovné načtení zobrazené stránky. Bez podpory pull.	

	Podpora	Omezení	Ostatní
Nezávislost na platformě	Zcela platformě nezávislé	Riziko rozdílné interpretace JavaScriptu a CSS	
Podpora pro vývoj	Velmi jednoduše naučitelné. Velké množství nástrojů pro tvorbu HTML/ JavaScriptu/CSS	Na vývojáře jsou kladeny vyšší nároky, vzhledem k hrozící rozdílné implementaci JavaScriptu	
Bohatost grafického rozhraní	Oddělení vzhledu od obsahu díky využití CSS, jednoduší podpora pro témata.	Složitá implementace „Bohatých vlastností“ jako je např. Drag&Drop	Možno využít nástrojů jako je Dojo či jQuery
Náklady	Vzhledem k vyutiží JavaScriptu a CSS jsou nároky vyšší	Rozdílné vlastnosti zvyšují náklady na vývoj	
Budoucnost	Vyhlídky této technologie jsou shodné s předchozím hodnocením. Očekává se přechod na HTML5.		

Tabulka 4: Hodnocení technologie DHTML [autor][19]

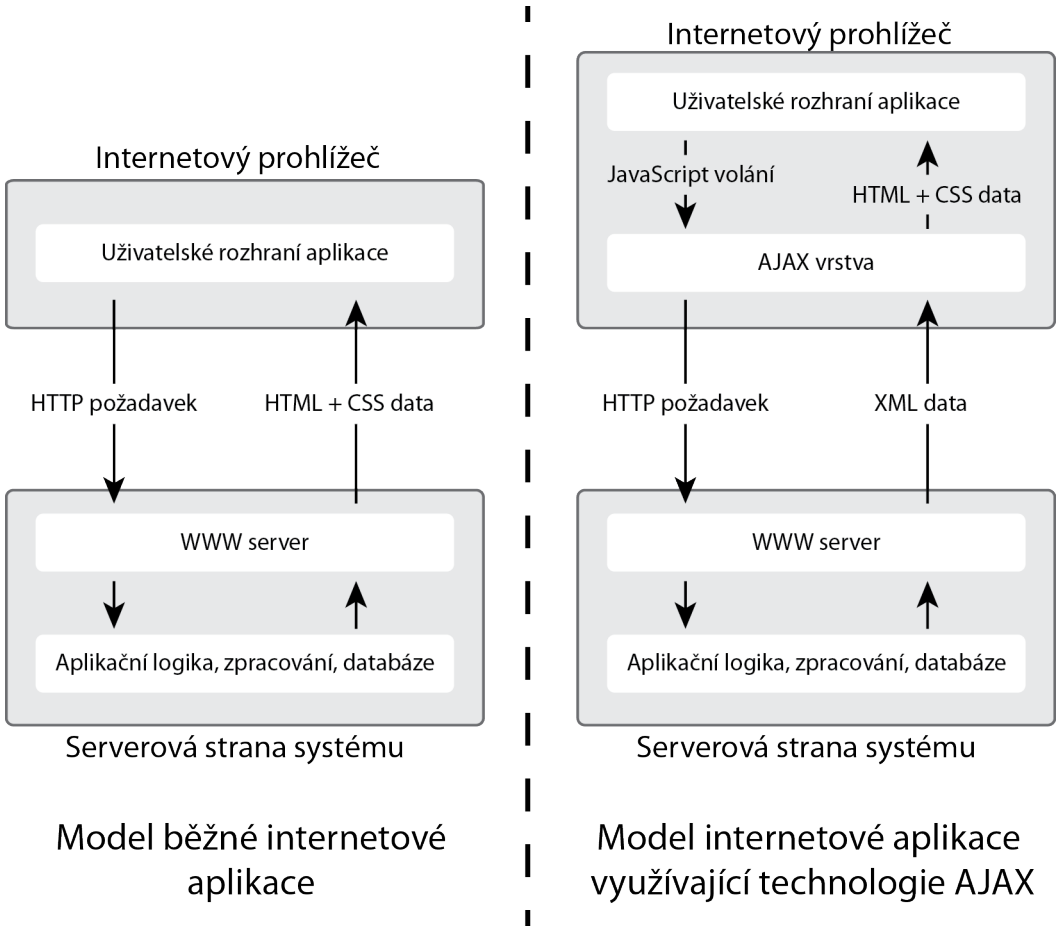
5.1.3 AJAX

AJAX je zkratkou pro termín *Asynchronous JavaScript and XML*. Přestože tato technologie vychází z JavaScriptu a je využívána HTML stránkami, je pro svou významnost oddělena. Významnost této technologie spočívá v nalezení způsobu, jak běžné synchronní volání, které využívají běžné internetové stránky, přeměnit na volání asynchronní a tím zvýšit interaktivitu aplikace. AJAX není ve skutečnosti technologií nebo softwarovým produktem, jde spíše o koncept, resp. návrhový vzor pro RIA aplikace, ale nejen pro ně. Dalo by se říci, že jde o shluk různých technologií a to:

- Document Object Model (DOM);
- XMLHttpRequest (JavaScriptová kolekce funkcí);
- HTML, CSS a JavaScript.

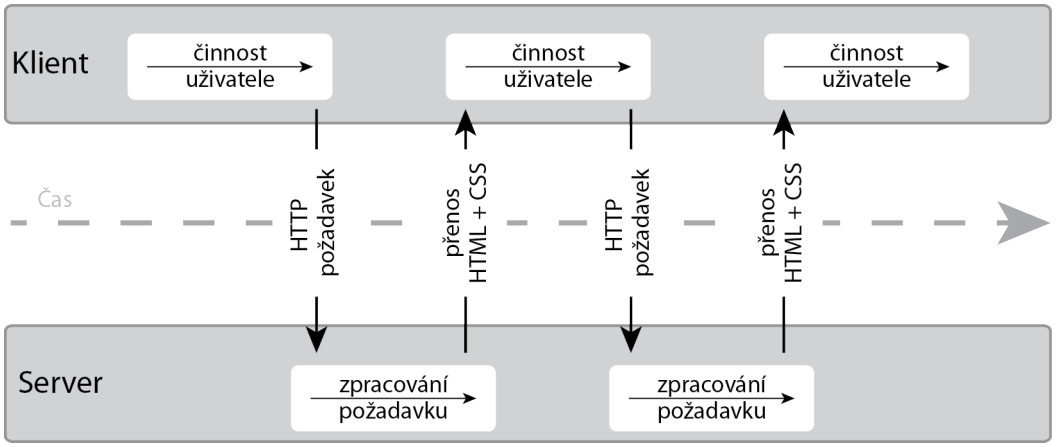
Při obecném pohledu na využití AJAXu je možno tvrdit, že celá jeho funkčnost (a tím i kouzlo) spočívá ve využití kolekce XMLHttpRequest, kterou poskytuje skriptovací jazyk JavaScript.

Orborná i laická veřejnost si může položit otázku co je na AJAXu tak revolučního a proč je řešením problému, který způsobuje protokol HTTP. Využití AJAXu totiž poskytuje možnost asynchronního volání serveru a zpracování odpovědi aniž by bylo nutné překreslovat celou stránku, která akci vyvolala. Samotná technologie je znázorněna na Obrázku č. 4.



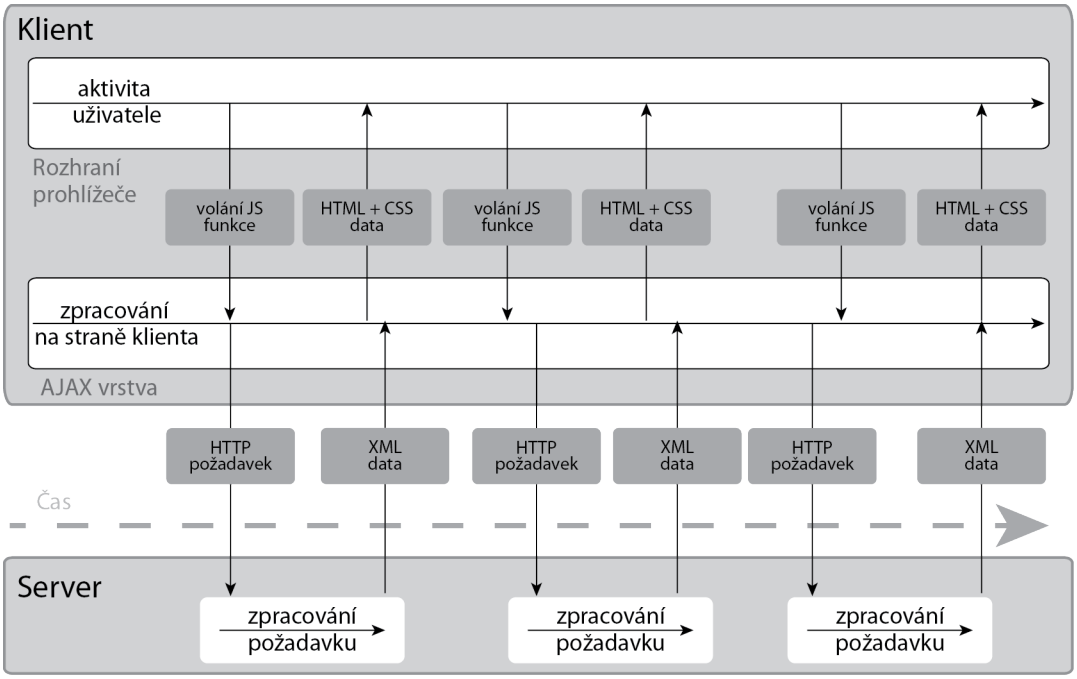
Obrázek 4: Rozdíl mezi využitím klasického modelu a AJAX modelu [autor]

Vysvětlení lze znázornit na příkladu. Vývojář potřebuje vytvořit jednoduchý formulář s několika prvky a tlačítko, které vyvolá akci, jež odešle formulářová data na server. Pokud bude realizovat formulář pomocí běžných nástrojů (tj. bez využití AJAXu) bude samotné odesílání formuláře probíhat takto. Po stisku tlačítka jsou data odeslána jako požadavek na server. Server data zpracuje a požadavek vyřídí, celá operace skončí zasláním kompletního uživatelského rozhraní včetně dat. Všechny kroky tohoto procesu jsou vzájemně synchronizovány.



Obrázek 5: Synchronní zpracování požadavku bez použití AJAXu [autor]

Pokud bude odeslání formuláře realizováno pomocí AJAXu, bude samotný proces odeslání velmi podobný předchozímu řešení, avšak s tím rozdílem, že samotné rozhraní formuláře může zůstat po dobu zpracování požadavku beze změn. Po stisknutí tlačítka, které slouží k odeslání formuláře na server, je vyvolána funkce či metoda AJAXového jádra, která vygeneruje požadavek a pomocí *XMLHttpRequest* jej odešle na server. Server požadavek zpracuje a vrátí odpověď. AJAXové jádro, na základě odpovědi ze serveru provede patřičné změny uživatelského rozhraní. Celé zpracování formuláře proběhlo v tomto případě asynchronně. V tomto případě je dokonce možné, po stisknutí tlačítka, vyslat na server požadavků několik a následně je zpracovat zcelaa odděleně.



Obrázek 6: Asynchronní zpracování požadavku s použitím AJAXu [autor]

	Podpora	Omezení	Ostatní
Uživatelská přívětivost	Klient-server interakce, při požadavku na server není nutné obnovit celou stránku, okamžitá reakce při validaci vstupních polí formulářů	Možné problémy při použitelnosti, interakční model aplikace neodpovídá mentálnímu modelu uživatele, riziko při použití ovládacích prvku prohlížeče ve spojení s AJAXem.	
Nasazení	Nasazení je velmi jednoduché, není potřeba žádná instalace na klientské stanici	Vykonávání JavaScriptu může být uživatelem explicitně zákázáno	
Dostupnost	Široce dostupné, není potřeba jakékoli doinstalace modulů na klientskou stanici.		
Technologie	Efektivní propojení technologií (HTML, JS, CSS)	Možné problémy při rozdílné implementaci v prohlížečích	
Délka spuštění	Vzhledem k využití více technologií je spuštění pomalejší nežli u HTML, avšak stále relativně rychlé	Při masivním využití JavaScriptu hrozí navýšení datového objemu	
Klient-server zpracování	Využití asynchronního volání, možná redukce počtu volání serveru. Možnost lazy-loadingu.	Není možné permanentní propojení, pouze simulace pull	
Nezávislost na platformě	Zcela platformě nezávislé	Možné problémy při rozdílné implementaci v prohlížečích	
Podpora pro vývoj	Velmi jednoduše naučitelné. Velké množství nástrojů	Na vývojáře jsou kladeny vyšší nároky, vzhledem k hrozící rozdílné implementaci JavaScriptu	

	Podpora	Omezení	Ostatní
Bohatost grafického rozhraní	Oddělení vzhledu od obsahu díky využití CSS, jednoduchá podpora pro témata	Složitá implementace „Bohatých vlastností“ jako je např. Drag&Drop	Možno využít nástrojů jako je Dojo či jQuery
Náklady		Vzhledem k využití asynchronní komunikace jsou na vývojáře kladeny daleko vyšší nároky	
Unikátní vlastnosti	Velmi dobré propojení s internetovým prohlížečem, široká dostupnost		
Budoucnost	S masivním rozvojem HTML5 je budoucí využití technologie AJAX zajištěné, zejména jako komunikační vrstvy v rámci různých JavaScript frameworků (Angular, jQuery, Dojo, GoogleGears).		

Tabulka 5: Hodnocení technologie AJAX [autor][19]

5.1.4 HTML5

Přestože jazyk HTML5 vychází z jazyka HTML, je mu vzhledem k jeho významu věnováno vlastní zhodnocení. Je to dáno tím, že je na HTML5 nahlíženo jako na budoucnost RIA a oproti předchozímu HTML nabízí nové možnosti, které jsou zaměřeny právě pro oblast tvorby RIA. HTML5 nevychází pouze z jazyka HTML, ale spíše se pod tímto označením kombinuje více technologií. Jedná se především o HTML, JavaScript, AJAX a CSS3. Tato kombinace je dána faktem, že HTML5 jako takové stále nenabízí žádné možnosti dynamického obsahu a stejně jako u HTML/DHTML je jeho dynamičnost zajišťována pomocí jazyka JavaScript. Je dobré poznamenat, že HTML5 je prozatím ve fázi rozvoje a nejedná se o ustanovený standard. S přihlédnutím ke všem platformám lze říci, že se HTML5 pravděpodobně stane budoucností RIA.

	Podpora	Omezení	Ostatní
Uživatelská přívětivost	Díky využití technologie AJAX a JavaScriptu je dostupný Drag&Drop (pouze v rámci aplikace) a dynamické nahrávání obsahu	Možné problémy při použitelnosti, interakční model aplikace neodpovídá mentálnímu modelu uživatele, riziko při použití ovládacích prvků prohlížeče ve spojení s AJAXem.	
Nasazení	Nasazení je velmi jednoduché, není potřeba žádná instalace na klientské stanici	Vykonávání JavaScriptu může být uživatelem explicitně zákázáno	
Dostupnost	Široce dostupné, není potřeba jakékoli doinstalace modulů na klientskou stanici.		
Technologie	Efektivní propojení technologií (HTML, JS, CSS3)	Možné problémy při rozdílné implementaci v prohlížečích, nedokončený standard HTML5	
Délka spuštění	Vzhledem k využití více technologií je spuštění pomalejší nežli u HTML, avšak stále relativně rychlé.	Při masivním využití JavaScriptu hrozí navýšení datového objemu	
Klient-server zpracování	Využití asynchronního volání, redukce počtu volání serveru. Možnost lazy-loadingu.	Není možné permanentní propojení, pouze simulace pull	
Nezávislost na platformě	Zcela platformě nezávislé	Možné problémy při rozdílné implementaci v prohlížečích	
Podpora pro vývoj	Velmi jednoduše naučitelné. Velké množství nástrojů	Na vývojáře jsou kladeny vyšší nároky, vzhledem k hrozící rozdílné implementaci JavaScriptu	

	Podpora	Omezení	Ostatní
Bohatost grafického rozhraní	Oddělení vzhledu od obsahu díky využití CSS3, jednoduchá podpora pro témata	Složitá implementace „Bohatýchvlastností“ jako je např. drag&drop	Možno využít nástrojů jako je Dojo či jQuery
Náklady		Vzhledem k využití asynchronní komunikace jsou na vývojáře kladeny daleko vyšší nároky	
Unikátní vlastnosti	Velmi dobré propojení s internetovým prohlížečem, široká dostupnost		
Budoucnost	Vzhledem k masivnímu využití v současné době lze tvrdit, že je budoucnost HTML5 zajištěná. Na ustanovení standardu HTML5 se stále pracuje a vzhledem k tomu, že se na procesu standardizace podílejí firmy zvučných jmen (IBM, Oracle, Microsoft, Adobe), lze očekávat, že se bude HTML5 i nadále rozvíjet.		

Tabulka 6: Hodnocení technologie HTML5 [autor][19]

5.1.5 Flash Platform

Adobe Flash je běhové prostředí jež poskytuje uživatelům vysokou míru interaktivity, přívětivosti a komfortu pro práci. Podporuje integraci různých druhů medií a nabízí možnosti dynamického a rychle reagujícího (responsivního) obsahu. Na straně klienta využívá Flash Platform běhového prostředí zvaného Flash Player, který je ve většině případů instalován do internetového prohlížeče jakožto zásuvný modul. S aplikacemi psanými pro Flash Plaform se lze setkat i mimo prostředí internetového prohlížeče, s využitím technologie Adobe AIR je možné tyto aplikace spouštět jako běžné desktopové programy. Flash Platform používá programovací jazyk zvaný Action Script3 jež je, podobně jako jazyk JavaScript, založen na standardu ECMAScript.

Na straně serveru lze využít několik možných způsobů, na základě kterých Flash aplikace komunikují. Nejsnadnějším z možností je jednoduché stažení datového souboru ze serveru a jeho zpracování na straně klienta. V tomto případě však nelze hovořit o příliš velké interakci a tento přístup je spíše vhodný pro malé aplikace bez větší interakce, reklamní bannery či aplikace bez stálého přístupu k síti. Pravdou je, že tento

způsob komunikace byl zprvu pro Flash zcela dostačující a až s nástupem stále častějšího využívání této platformy pro vývoj RIA, byly poskytnuty nástroje a technologie, které umožnily lepší využití a propojení aplikace se serverem.

Jedním z takovýchto nástrojů, který vznikl díky sílícímu tlaku vývojářů je Apache Flex (dříve Adobe Flex). Jde o framework, který vývojářům poskytuje podporu pro interakci se serverovými řešeními založenými na protokolu HTTP (HyperText Transfer Protocol) a to pomocí různých protokolů, jako AMF, (Action Message Format), RTMP (Real Time Messaging Protocol), SOAP (Simple Object Access Protocol), REST (Representational State Transfer) či XML (eXtended Markup Language). Samotné serverové řešení může být postaveno na téměř libovolném programovacím jazyku, jediným požadavkem je jeho podpora pro výše uvedené komunikační protokoly. Popřípadě je možné využít již hotových serverových řešení, jako jsou Adobe BlazeDS, Adobe LiveCycle, které navíc nabízejí podporu pro přenos videa v reálném čase, správu transakcí, zasílání zpráv a integrované řešení zabezpečení aplikace. Architektura frameworku Adobe Flex nabízí možnosti komponentové orientovaného vývoje aplikací. Adobe Flex dále nabízí možnost tvorby grafického rozhraní pomocí jazyka MXML (Magic eXtensible Markup Language), které je založené na jazyce XML.

Aplikace psané s využitím Adobe Flex jsou primárně určeny pro běh v běžném internetovém prohlížeči, na kterém je nainstalován zásuvný modul Adobe Flash. Toto tvrzení však narušuje technologie Adobe AIR (Adobe Integrated Runtime), pomocí které je možné přenést tyto aplikace do prostředí operačního systému. Díky Adobe AIR se aplikace chovají jako běžné desktopové aplikace. Nutno poznamenat, že Adobe AIR neposkytuje běhové prostředí pouze pro aplikace vytvořené v Adobe Flash a Apache Flex, avšak poskytuje podporu i pro aplikace psané v jazyce HTML. Adobe AIR nabízí aplikacím vyšší míru integrace s operačním systémem uživatele, k dispozici je podpora drag&drop, komunikace s ostatními aplikacemi, přístup k systémové schránce (Clipboard) a vylepšená úroveň upozornění na události.

Velmi zajímavou alternativou k Adobe Flex je OpenSource nástroj OpenLaszlo. Tento framework využívá pro popis aplikací jazyk zvaný LZX, vycházející z XML. Pro zápis chování klientské aplikace se používá jazyk JavaScript. Výhodou OpenLaszlo je možnost výsledné kompilace aplikace jak pro zásuvný modul Adobe Flash tak i jako pouhé HTML/JavaScript aplikace [12].

	Podpora	Omezení	Ostatní
Uživatelská přívětivost	Flash Player poskytuje vyšší míru uživatelského komfortu. Adobe AIR nabízí, oproti aplikacím běžícím v internetovém prohlížeči, výší míru integrace s uživatelským systémem.		
Nasazení	Velmi snadné nasazení na server, existence již hotových serverových řešení		
Dostupnost	Velmi dobrá dostupnost, zejména díky velkému rozšíření v internetových prohlížečích (až 97%)		
Technologie	Aplikace je tvořena pomocí jedné technologie.	Omezené možnosti interakce s ostatními technologiemi	
Délka spuštění	Relativně rychlé spuštění; možnost postupného donáhrávání za běhu aplikace; modulární řešení		Rychlost spuštění aplikace je závislá na velikosti aplikace
Klient-server zpracování	Vysoká redukce zatížení serveru, aplikace přenášení pouze potřebná data, nedochází k redundantním přenosům dat		
Nezávislost na platformě	Běhové prostředí Adobe Flash je dostupné na většině operačních systémů		
Podpora pro vývoj	Vysoká podpora vývojových nástrojů (Adobe Flash, Adobe Flash Builder, InteliIDEA)		

	Podpora	Omezení	Ostatní
Bohatost grafického rozhraní	Vysoké možnosti stylování aplikace, podpora animací; standardizovaná podpora témat	Díky vysokým možnostem úprav vzhledu hrozí riziko zmatení uživatele	
Náklady	Základní nástroje potřebné pro tvorbu aplikací jsou k dispozici zdarma, Adobe Flex je OpenSource framework	Nástroje pro vývoj poskytované firmou Adobe patří do dražší kategorie softwaru, zejména serverová řešení	
Unikátní vlastnosti	Skvělá podpora pro audio video		
Budoucnost	Využití této technologie na poli RIA v současné době lehce klesá a to zejména díky nástupu HTML5. V rozmezí následujících 5 až 7 let lze označit budoucnost této technologie za dobrou, avšak postupný okdlon od této technologie směrem k HTML5 se zdá nevyhnutelný.		

Tabulka 7: Hodnocení technologie Flash Platform [autor][19]

5.1.6 Microsoft Silverlight

Přestože je platforma Microsoft Silverlight na ústupu, je vhodné jí pro její účast na rozvoji RIA zmínit. Hlavní platformou firmy Microsoft pro vývoj softwaru (včetně RIA) je framework .NET. Microsoft Silverlight je součástí tohoto frameworku. Stejně tak jako Adobe Flash využívá Microsoft Silverlight pro svůj běh proprietálního prohlížečového modulu. Programovací jazyk není omezen pouze na jeden daný typ, ale jak je u .NET frameworku zvykem, lze zvolit jeden z podporovaných, tj. Visual Basic, .NET, C#, Python či Ruby. Grafické rozhraní je definováno pomocí jazyka XAML který, stejně jako jazyk MXML, vychází z jazyka XML.

Silverlight je oproti Adobe Flash výhradně prohlížečový modul, přenos dané aplikace na desktopový systém je možný pouze v rámci kompilace. Takto zkompilovaná aplikace je však spustitelná pouze na systémech firmy Microsoft. Na obranu platformy Silverlight lze zmínit, že jejím cílem nebyla možnost transformace aplikace na klasickou desktopovou aplikaci, jako je tomu u Adobe AIR. O to zajímavější však muže být fakt, že původ Microsoft Silverlight má původ v desktopově orientovaném frameworku .NET.

	Podpora	Omezení	Ostatní
Uživatelská přívětivost	Je poskytována vyšší míra uživatelského komfortu než u HTML/ AJAX aplikací; shodné s Adobe Flash	Nižší rozšíření mezi internetovými prohlížeči	
Nasazení	Velmi snadné nasazení na server, existence již hotových serverových řešení		XAML je i po kompilaci aplikace stále čitelný, není převáděn do binární podoby jako je tomu u Adobe Flash a Java FX
Dostupnost	Velmi dobrá dostupnost	Nižší rozšíření mezi interentovými prohlížeči	Na operačních systémech firmy Microsoft bývá přeinstalován
Technologie	Logiku a chování aplikace lze definovat ve více programovacích jazycích	Omezené možnosti interakce s ostatními technologiemi	
Délka spuštění	Relativně rychlé spuštění, možnost postupného donáhrávání za běhu aplikace, modulární řešení		Rychlost spuštění aplikace je závislá na velikosti aplikace
Klient-server zpracování	Vysoká redukce zatížení serveru, aplikace přenášení pouze potřebná data, nedochází k redundantním přenosům dat		

	Podpora	Omezení	Ostatní
Nezávislost na platformě		Omezená dostupnost, primárně dostupné na operačních systémech firmy Microsoft a Apple	
Podpora pro vývoj	Vysoká podpora vývojových nástrojů, výběr z několika programovacích jazyků	Menší dostupnost nástrojů zdarma	
Bohatost grafického rozhraní	Vysoké možnosti stylování aplikace, podpora animací; standardizovaná podpora témat		
Náklady	Základní nástroje potřebné pro tvorbu aplikací jsou k dispozici zdarma	Nástroje pro vývoj je nutné zakoupit	
Unikátní vlastnosti	Skvělá podpora pro audio video, hardwarová akcelerace pomocí rozhraní Direct3D		
Budoucnost	Budoucnost této technologie je velmi malá, zejména z důvodu ukončení dalšího vývoje.		

Tabulka 8: Hodnocení technologie Microsoft Silverlight [autor][19]

5.1.7 Java Platform

Java je technologií, jejíž jméno je spojeno s programovacím jazykem Java a běhovým prostředím Java, přesněji označovaného jako Java Runtime Environment (JRE) či Java Development Kit (JDK). Využití technologie Java je pro tvorbu RIA možné několika způsoby:

- Java klient (Java Rich Client) – jde o klasickou desktopovou aplikaci avšak s tím rozdílem, že na straně klienta je obsluhována pouze prezentační a aplikační vrstva, veškeré zpracování a perzistence je prováděna v rámci severu.
- Java Web Start – aplikace tohoto typu ze spustit přímo z operačního systému či internetového prohlížeče. Veškeré datové soubory jsou uchovávány na serveru, odkud jsou v případě spuštění aplikace staženy. Tento přístup poskytuje jednodušší správu aplikace, protože klient vždy obdrží poslední verzi aplikace.
- JavaFX – jde o nejnovější z technologií patřící do skupiny Java. Jedná se o zjednodušený jazyk Java, který byl vyvinut pro potřeby RIA.

	Podpora	Omezení	Ostatní
Uživatelská přívětivost	Podpora jak desktopových tak prohlížečových řešení	Delší startovací časy	
Nasazení	Velmi snadné nasazení na server, potřebná data jsou stahována ze serveru		
Dostupnost	Velmi dobrá dostupnost, JRE/JDK je běžně nainstalováno na osobním počítači		
Technologie	Využití pouze jednoho programovacího jazyku, jedno běhové prostředí pro všechny operační systémy		

	Podpora	Omezení	Ostatní
Délka spuštění	Možnost spouštění aplikací z osobního počítače	Delší startovací časy, některé aplikace vyžadují před samotným spuštěním stažení programových částí	
Klient-server zpracování	Možnost využití aplikace i bez přístupu k síti. Asynchronní volání.		Vyšší nároky běhového prostředí JRE/JDK na klientskou stanici, zejména ve smyslu výkonu
Nezávislost na platformě	Nezávislost na platformě je velmi dobrá, běhové prostředí je dostupné na všech hlavních operačních systémech		
Podpora pro vývoj	Velmi silná základna vývojářů, velké množství nástrojů pro vývoj		
Bohatost grafického rozhraní	Možnost využití komponent, které jsou schopné převzít vzhled z prvků operačního systému		
Náklady	Veškeré nástroje, které jsou potřebné pro vývoj a spuštění aplikací jsou zdarma		
Budoucnost	Využití technologie Java jakožto nástroje pro tvorbu klientského rozhraní RIA není příliš rozšířené, kritizované jsou především startovací časy těchto aplikací. Technologie Java se však velmi hojně používá pro stranu serveru, kde poskytuje velmi robustní řešení pro RIA.		

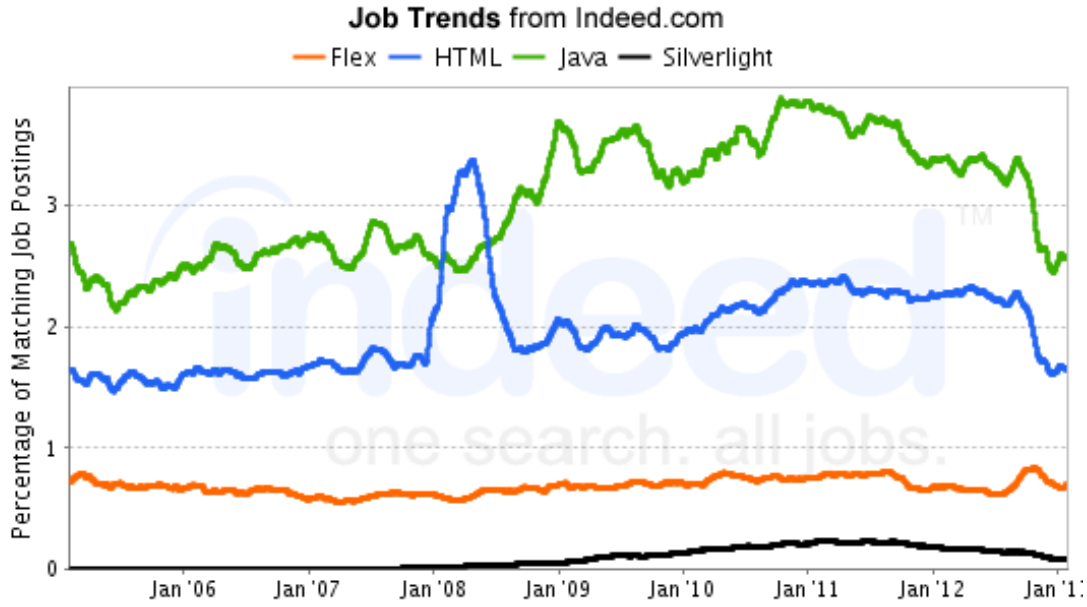
Tabulka 9: Hodnocení technologie Java Platform [autor][19]

5.2 Analýza trhu

Velmi důležitým aspektem, při výběru vhodné RIA platformy, je pohled na ekonomickou stránku věci. Pro zhodnocení úspěšnosti technologií, a tím i jejich zastoupení na trhu, lze využít ukazatel poptávky firem na zaplnění pozic vývojářů daných technologií. Zhodnocení bude provedeno na trhu Spojených státu, který je v tomto směru nejvýznamnějším.

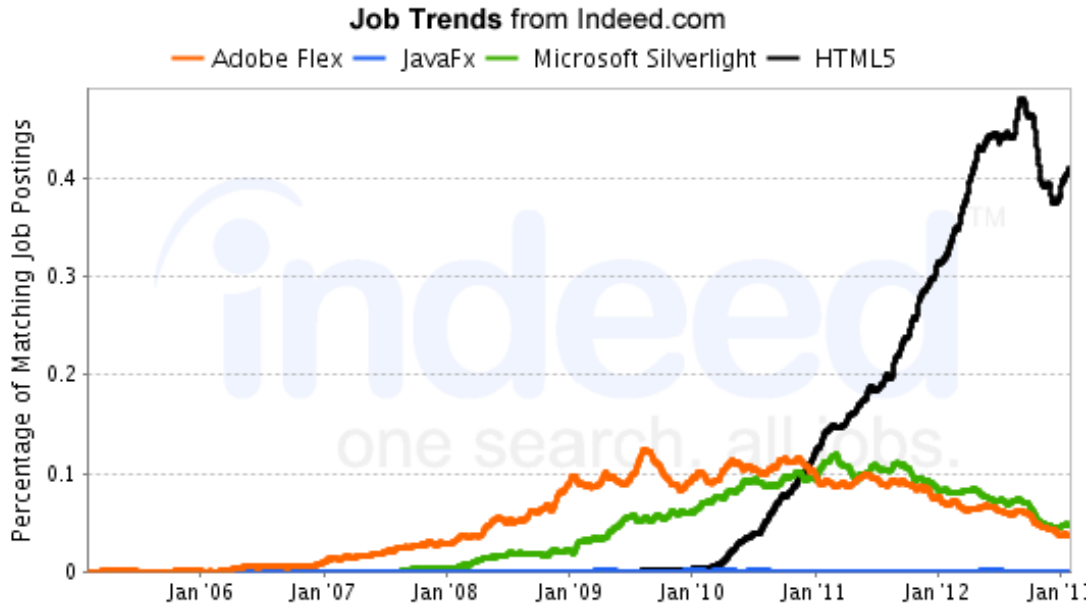
Analýza zastoupení technologií na trhu bude provedena na základě dat dostupných z portálu Indeed.com zaměřeným na vyhledávání pracovních míst. Hodnota zastoupení dané technologie je vždy zanesena na souřadnici „y“; změna v čase je zanesena na ose „x“. Sběr dat byl proveden ke dni 24. Března 2013.

V prvním případě (Obrázek 7) jsou porovnávány hlavní platformy pro tvorbu RIA s omezením na využití zásuvných prohlížečových modulů. Na grafu je patrná dominance Adobe Flex těsně následovaná Microsoft Silverlight. Lze si však velmi dobře všimnout klesající tendence u obou technologií, jež byla mimo jiné způsobena rapidním nástupem HTML5.



Obrázek 7: Trend technologií využívajících zásuvného prohlížečového modulu [indeed.com]

Další graf (Obrázek 8) je doplněn o přítomnost HTML5 a jeho nástup od počátku roku 2010, kdy bylo HTML5 představeno a uvolněno k použití, je velmi dominantní.



Obrázek 8: Trend technologií pro tvorbu RIA [indeed.com]

Následující graf (Obrázek 9) byl doplněn o přítomnost jazyka JavaScript. Lze si všimnout zjevné dominance nad všemi technologiemi a i přes soustavnou rostoucí tendenci je patrný rapidní vzrůst na počátku roku 2010, který souvisí s příchodem HTML5. Nutno podotknout, že současná dominance jazyka JavaScript je dána faktem, že v poslední době dochází k masivnímu využití tohoto jazyka i mimo prostředí klienta (resp. prohlížeče) a je možné se s ním setkat v podobě skriptovacího jazyka na straně serveru.

Obrázek 9: Trend základních technologií pro tvorbu RIA doplněný o JavaScript [indeed.com]

Poslední graf (Obrázek 10) nabízí pohled na zhodnocení všech platforem používaných pro tvorbu RIA. Platforma využívající technologie HTML či HTML5 je zastoupena pod označením AJAX. Tento přístup byl zvolen z toho důvodu, že technologie AJAX je v případě RIA využívána téměř vždy a dále že při použití samotného klíčového slova HTML či HTML5 by došlo k značnému zkreslení grafu, protože tyto jazyky se využívají i k tvorbě běžných internetových stránek. Při pohledu na vývoj lze konstatovat, že stále dominuje technologie JavaScript, následovaná technologií Adobe Flash a Microsoft Silverlight. Java dominuje celému grafu, avšak tato dominance je způsobena díky jejímu využití i na poli desktopových aplikací a proto je její význam v tomto grafu lehce nadhodnocen.

Obrázek 10: Trend technologií pro tvorbu RIA s účastí technologie AJAX [indeed.com]

5.3 Zhodnocení

Na základě výše uvedených skutečností bude provedeno celkové bodové zhodnocení jednotlivých technologií. Pro přehlednost bude platforma HTML/HTML5 hodnocena jako celek pod označením HTML/AJAX.

Bodové hodnocení jednotlivých kritérií bude provedeno na stupnici 1 až 5, kdy 1 znamená nízkou podporu či zastoupení. Za předpokladu, že u dané technologie nelze dané kritérium hodnotit, bude toto kritérium ohodnoceno hodnotou 0.

	HTML/AJAX	Flash	Silverlight	Java
Uživatelská přívětivost	3	4	3	4
	Díky využití technologie AJAX je uživatelská přívětivost aplikací postavených na HTML o mnoho lepší, ale stále trpí paradigmem internetového prohlížeče. Silverlight, shodně jako Flash, poskytuje velmi vysokou míru uživatelské přívětivosti, avšak není příliš rozšířen. Java a Flash je možné využít pro aplikace běžící jak v internetovém prohlížeči, tak v rámci operačního systému.			
Nasazení	4	4	4	3
	Všechny technologie nabízejí možnost centralizovaného nasazení aplikace, které je mnohdy řešeno pouhým nahráním aplikačních souborů na server. Výhodou technologie HTML/AJAX je její integrace v rámci internetového prohlížeče. Ostatní technologie je nutné do operačního systému uživatele doinstalovat.			
Dostupnost	5	4	1	4
	HTML/AJAX je integrován do internetového prohlížeče a jeho dostupnost je vynikající. Flash je dostupný v 97% procentech prohlížečů. Navíc bývá ve většině případů již předinstalován, stejně tak jako je tomu v případě technologie Java a Silverlight.			
Technologie	1	3	4	4
	HTML/AJAX trpí různou implementací standardu napříč internetovými prohlížeči a interakce s jinými technologiemi je velmi nízká. Flash, Silverlight a Java využívají jednotného běhového prostředí, které zajišťuje stejné chování ve všech internetových prohlížečích. V případě technologie Silverlight, lze navíc zvolit programovací jazyk, ve kterém bude logika aplikace popisována.			

	HTML/AJAX	Flash	Silverlight	Java
Délka spuštění	5	5	4	2
	Technologie HTML/AJAX v hodnocení tohoto kritéria těží ze skutečnosti, že jsou pevně integrovány do prostředí internetového prohlížeče a spuštění aplikace je tak téměř okamžité. Spouštěcí doba u technologie Flash a Silverlight je relativně dobrá, avšak závisí na velikosti aplikace a způsobu její prvotní inicializace. Java nabízí spuštění aplikací jak z prostředí prohlížeče tak i desktopového systému, avšak startovací časy jsou dlouhé.			
Klient-server zpracování	3	5	3	5
	V případě technologie HTML/AJAX je díky využití asynchronní komunikace možné využít efektivnější komunikace klienta se serverem, zatím chybí podpora soketů, které je nahrazována „poolingem“. Flash i Silverlight poskytují vestavěnou podporu pro přímou komunikaci se serverem, je možné využít soketů a tím i opačného způsobu komunikace.			
Nezávislost na platformě	4	4	2	5
	HTML/AJAX je díky standardizaci jazyka HTML zcela nezávislý na cílové platformě. Flash a Java jsou dostupné na nejpoužívanějších operačních systémech. Silverlight je podporovaný na operačním systému firmy Microsoft a Apple.			
Podpora pro vývoj	3	4	3	4
	HTML/AJAX se těší velmi dobré podpoře vývojářů, k dispozici je velmi mnoho nástrojů, které jsou poskytovány zdarma. Je zde však riziko rozdílně funkční implementace, zejména v případě jazyka HTML5 či JavaScript mezi internetovými prohlížeči. Flash a Java nabízejí početnou množinu vývojových nástrojů, mezi kterými lze nalézt i nástroje poskytované zdarma. Adobe Flex je navíc uvolněn pod licencí OpenSource. Silverlight nabízí využití programovacího prostředí Visual Studio, to je však poskytováno zdarma pouze v základní verzi.			

	HTML/AJAX	Flash	Silverlight	Java
Bohatost grafického rozhraní	3	5	4	4
	HTML/AJAX poskytuje možnost oddělení aplikační logiky a definice vzhledu pomocí CSS/CSS3 avšak oproti ostatním technologiím nedosahuje vysoké míry stylování. Flash a Silverligh těží z vysoké míry možností stylování, interakce a animací grafického rozhraní aplikace. Java poskytuje taktéž velmi dobré možnosti stylování, navíc nabízí velmi dobrou interakci i s rozhraním desktopové stanice.			
Náklady	3	2	2	3
	Rozdílné implementace HTML/AJAX, mezi internetovými prohlížeči, mohou způsobovat navýšení ceny při vývoji aplikace. Běhová prostředí pro Flash, Silverligh a Java jsou dostupná zdarma, základní nástroje jsou ve většině případů poskytovány taktéž zdarma. Při využití již hotových řešení u technologií Flash a Silverlight jsou finanční nároky velmi vysoké.			
Budoucnost	5	3	1	3
	Budoucnost HTML/AJAX je velmi dobrá a díky nástupu HTML5 lze očekávat, že se na poli RIA stane primární platformou pro tvorbu těchto aplikací. Budoucnost technologie Java a Flash je velmi dobrá, oproti HTML5 mají tyto technologie stále co nabídnout, avšak rozdíl mezi nimi se velmi rychle zmenšuje. Budoucnost Silverlight jako platformy pro RIA je velmi malá.			
Celkem	40	43	27	41

Tabulka 10: Vzájemné hodnocení platforem pro tvorbu RIA [autor][19]

Porovnání všech technologií navzájem poskytuje přehled o rozdílnosti daných hodnotících kriterií u daných technologií. Je dobré poznamenat, že výše uvedená hodnotící kriteria nemají stejnou váhu vzhledem k důležitosti, naopak, jejich důležitost je velmi rozdílná a tím i významnost přidělených bodů není stejná. Obodování těchto kriterií je navíc silně individuální záležitostí. I přes tyto výtky lze z celkového hodnocení vypozorovat jistou tendenci vývoje RIA. Ačkoli z celkového součtu bodů vychází technologie Adobe Flash jako nejlepší, je nutné si všimnout velmi nízkého rozdílu mezi technologiemi AJAX/HTML a Java. Zejména technologie HTML/AJAX v posledních letech stoupá na popularitě, což lze přisuzovat přechodu na HTML5. Naopak rapidní snížení popularity Adobe Flash je krom nesporného nástupu HTML5 způsobeno také velmi špatně provedenou marketingovou strategií a PR, jež od této platformy odlákalo mnoho vývojářů a firem.

6 Praktická část – Adobe Flash

6.1 Popis a zadání řešené aplikace

V rámci diplomové práce bude řeše návrh architektury a provedení aplikace, jež bude mít za úkol správu a kompletní zpracování životního cyklu diplomových či bakalářských prací na vysoké škole. Dalším požadavkem na aplikaci bude možnost správy a organizace komisí státních zkoušek a následná správa výsledků s návazností na organizaci promocí. Prostřednictvím aplikace bude probíhat nahrávání (odevzdávání) digitálních verzí absolventských prací a jejich následná kontrola na obsahovou shodu ve spolupráci s portálem theses.cz.

Aplikace bude určena pro studenty a zaměstnance univerzity, čímž upřesňuje svůj charakter uzavřené aplikace, tj. intranetové aplikace. Předpokládá se její rovnoměrné zatížení s možným nárůstem požadavků v období výběrů témat prací a v období jejich odevzdávání.

Požadovaná funkčnost aplikace:

- systém práv na základě rolí (např. pedagog, editor, vedoucí katedry, děkan);
- správa jednotlivých témat prací, ze kterých je možné vypisovat samotné diplomové a bakalářské práce;
- správa výběrových řízení na jednotlivá témata formou soutěže/konkurzu, kdy studenti na základě předložených materiálů k určitému předmětu dané téma obdrží či nikoliv;
- správa samotných záznamů bakalářských a diplomových prací, evidence změn v těchto záznamech;
- možnost odevzdání digitální verze práce do systému;
- kontrola obsahové shody bakalářských a diplomových prací na portálu theses.cz;
- podpora pro tvorbu posudkových formulářů na bakalářské a diplomové práce;
- správa oponentů bakalářských a diplomových prací;
- systém na rozesílání zpráv s požadavkem na vyhotovení oponentského posudku;
- tvorba a správa státnicových komisí;
- tvorba a správa promočních kolegií.

6.2 Zdůvodnění volby

Pro praktickou část, ve které bude představena tvorba aplikace, byla zvolena pro klientskou stranu technologie Adobe Flash s využitím frameworku Apache Flex. Pro serverovou stranu byla zvolena technologie Java s podporou Spring Framework+ BlazeDS.

Volba technologie klientské strany byla provedena na základě výše uvedeného zhodnocení technologií, ze které Adobe Flash vyšlo jako jedno z velmi dobrých řešení pro tvorbu RIA. Z hlediska pohledu uživatele byla tato technologie zvolena díky bohatým možnostem návrhu a úpravy uživatelského rozhraní a také díky možnému způsobu atraktivního provedení. Lze namítnout, že vzhledem k vysokému počtu uživatelů je volba technologie, vyžadující zásuvný modul prohlížeče špatná. Vyřčená námitka je věcná a byla zvážena, avšak vzhledem k vysokému zastoupení Adobe Flash mezi prohlížeči bylo možné riziko zhodnoceno jako přijatelné. K tomuto výsledku přispěl taktéž fakt, že se jedná o aplikaci, jež je svou povahou spíše intranetového typu, tj. není přístupná široké veřejnosti a pracovat s ní mohou pouze studenti, kteří mají již zadanou práci či mají nárok si téma práce vybrat. V případě, že by tato aplikace byla určena pro širokou veřejnost, bylo by nutné velmi dobře zhodnotit zmíněné riziko, a to zejména na základě cílových skupin uživatelů, kteří budou s aplikací pracovat.

Z hlediska vývoje aplikace byla technologie zvolena díky plně objektovému přístupu jazyka ActionScript3, prototypovému a komponentovému přístupu tvorby aplikace a dále na základě snadnosti návrhu grafického rozhraní s pomocí jazyka MXML. Objektovost jazyka ActionScript 3 poskytuje vývojáři velmi dobrou rozšiřitelnost a možnosti správy aplikace. Prototypový vývoj je velmi výhodný pro rapidní vývoj aplikace, kdy je možné klientovi představit zjednodušený prototyp aplikace, na kterém lze demonstrovat požadované funkčnosti. Prototypový model je možné rychle upravit a případné připomínky zákazníka lze velmi rychle zapracovat a následně přistoupit k samotnému programování aplikace. Komponentový přístup tvorby aplikace přináší výhody ve znuvupoužitelnosti prvků aplikace.

Technologie Java, jež bude zajišťovat provoz aplikace na straně serveru, byla zvolena na základě její robustnosti. Svou povahou spadá aplikace pod takzvané Enterprise Applications, které jsou součástí celkového informačního systému organizace. Od podobných aplikací se očekává výkonnost při zpracování požadavků, rozšiřitelnost o nové vlastnosti či moduly a robustnost, která zajišťuje odolnost proti chybám. Právě technologie Java EE byla pro požadované využití navržena. Stejně tak jako jazyk ActionScript3, je jazyk Java plně objektový. Má velmi dobré možnosti škálovatelnosti výkonu a díky objektovosti poskytuje velmi dobrou rozšiřitelnost a odolnost proti chybám.

Jako aplikační framework pro stranu serveru byl zvolen Spring Framework. Jedná se o framework vyvíjený firmou VMware, známou především z prostředí virtualizací serverů a koncových stanic. Spring Framework je v komunitě vývojářů velmi dobře hodnoceným aplikačním frameworkem a lze říci, že jde o ověřený etalon na poli enterprise aplikací. Vybrán byl na základě skutečnosti, že disponuje modulem, poskytujícím nástroje pro přímou komunikaci s aplikacemi postavenými na základech Flash Platform. Nutno poznamenat, že tento zásuvný modul vychází a používá knihovny ze serverového řešení firmy Adobe a to konkrétně z OpenSource varianty komerčního serveru, zvaného BlazeDS. Spring Framework je v podstatě souborem knihoven, které lze použít samostatně, či využít jejich vzájemné integrace a vytvořit s jejich pomocí velmi robustní aplikační framework.

Spojení Apache Flex, na straně klienta, a Javy, ve spojení se Spring Framework a zásuvným modulem BlazeDS na straně serveru, umožňuje využití přímého přenosu objektů mezi jednotlivými stranami, tj. není nutné požadavek jakkoli připravovat pro přenos a zároveň není nutné jakoukoli odpověď od serveru či klienta převádět na srozumitelnou formu. Protistrany totiž poskytují komunikační rozhraní, které tento převod zajistí a do samotného programového kódu aplikace je předán platný objekt. Transfer objektů funguje oboustranně.

6.3 Architektura aplikace

Jak již bylo výše uvedeno, aplikace bude rozdělena na dvě části; na stranu klienta (client side) a stranu serveru (server side). Klient bude v tomto případě plnit roli Tenkého klienta, tj. jeho zodpovědností bude pouze prezentace a vykreslení dat uživateli. Z povahy aplikace a ušetření zdrojů může v některých případech dojít k přesunu jednoduché aplikační logiky na stranu klienta, avšak v případě kritických požadavků budou data na straně serveru vždy ověřena. Je možné namítnout, že navrženým postupem dochází k zbytečnému zdvojení shodné aplikační logiky. Námitka je opodstatněná, a proto bude zdvojení využito pouze v méně důležitých případech, jakým jsou například přepočet známek pro prezentaci výsledků. Veškerá aplikační logika bude obsluhována serverem. Je-li použito rozdělení aplikací, které bylo vyjmenováno v kapitole 4.4, půjde o hybridní RIA, konkrétně typ (3), tj. prezentační logika a jednodušší aplikační logika v rámci klientské strany a aplikační logika a datová perzistence v rámci serveru.

Taktéž je možné namítnout, proč není více využita strana klienta ke složitější aplikační logice, zejména díky výhodám, které plynou z využití technologie Adobe Flash. Přesun složitější aplikační logiky na stranu klienta by skutečně byl možným řešením, avšak vzhledem k povaze aplikace, jež je zapojena do celého aplikačního ekosystému univerzity, je očekávána příprava API (Application Protocol Interface), pomocí kterého budou ostatní systémy univerzity přistupovat k jejím zdrojům. Některé prvky navrhovaného rozhraní mohou využívat služeb aplikační logiky, která by však byla v tomto případě dostupná pouze na klientské straně. Aby bylo možné této aplikační logiky využít pro API, bylo by nutné požadované procesy implementovat i na straně serveru, což by vedlo k zdvojování aplikační logiky a narostly by i nároky na správu systému, protože by bylo nutné udržovat kód aplikační logiky na více místech, tj. jak na straně klienta, tak na straně serveru.

Dalším faktorem, jež přispěl k výběru zvoleného typu aplikace byl fakt, že datům, které přicházejí od klienta na server nelze důvěřovat. Prakticky řečeno je nutné příchozí data ověřit vždy tam, kde nebyla zadána na zpracovávajícím stroji, jelikož jakákoliv (bez)drátová komunikace může být zkompromitována. To tedy znamená, že centrální server příchozí data zkontroluje a až následně uloží. Logicky výše uvedeného vyplývá, že navržený systém vyžaduje aplikační logiku na straně serveru.

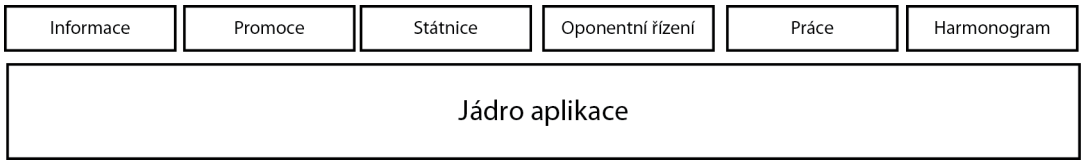
Neméně významnou výhodou přesunu veškeré logiky na stranu serveru je možnost výměny klientské strany, a to bez zásadnějšího dopadu na aplikaci jako takovou. Taková výměna by byla možná právě díky skutečnosti, že server využívá klienta pouze k prezentaci předložených dat. Vzhledem k tomu, že komunikace je prováděna pomocí vzdálených volání a především pomocí standardizovaného protokolu, je možné vytvořit jiného klienta, který tohoto aplikačního rozhraní využívá. Lze namítnout, že jde povětšinou o binární protokol a jeho zpracování by v obyčejné HTML5/JavaScript aplikaci bylo velmi obtížné. Je-li servisní vrstva aplikace navržena dobře, je možné využít objektových principů jako je dědičnost a polymorfismus a vytvořit v podstatě obal současného rozhraní a poskytovat výsledky volání v jiné formě, např. XML, JSON, atp.

Na základě požadavku na kontrolu obsahové shody odevzdaných bakalářských a diplomových prací oproti portálu theses.cz a dále odesílání požadavků na oponentské posudky odevzdaných prací, bude nutné zvolit vhodný model pro zpracování těchto úloh. Velmi dobrým a použitelným modelem je způsob zpracování požadavků pomocí fronty či seznamů úloh. Tento způsob chrání aplikaci před nepříznivým scénářem v případě vysoké zátěže a taktéž umožňuje vyšší míru kontroly a škálovatelnosti při zpracovávání požadavků. Dále je možné v případě potřeby přesunout zpracovávání úloh na zcela jiný server a při dobrém návrhu datové struktury lze uvažovat o možnosti využití distribuovaného zpracování úloh.

6.3.1 Rozdělení na moduly

Z požadavků na aplikaci plyne, že se bude skládat z několika modulů, avšak než bude provedeno jednotlivé rozdělení, je dobré uvést modul aplikace, který nemusí z požadavků zcela plynout. Jedná se o modul, který bude nazván „Jádro“ (Core). Jedná se v podstatě o základní stavební kámen aplikace, jehož odpovědností bude poskytovat služby týkající se základních životních cyklů a procesů, např. autentifikace a autorizace uživatele, kontrola oprávnění při přístupu k objektům aplikace, správa databázového připojení, správa datového úložiště a zejména bude zajišťovat prostředí pro nahrávání dalších modulů. Tento modul bude zastoupen jak na straně serveru tak na straně klienta.

Rozdělení aplikace na moduly:



Obrázek 11: Rozdělení aplikace na moduly [autor]

6.3.2 Technologické postupy

Při tvorbě aplikace budou použity postupy a principy, jež mají za úkol zajistit vysokou kvalitu aplikace, konzistenci programového kódu, snadnou udržitelnost a integrovatelnost s ostatními systémy.

Jedním z postupů vývoje softwaru je princip **DRY** (**D**on’t **R**epeat **Y**ourself), který je zaměřen na snižování duplicitních informací v kódu aplikace. Andy Hunt a Dave Thomas definují DRY princip následovně: „*Každá znalost či pouhá část znalosti musí mít v rámci systému jednotné, jednoznačné a autoritativní zastoupení*“ [42]. Při návrhu a vývoji aplikace je tedy nutné dbát na kvalitu kódu, zbytečně neopakovat kontextově shodné bloky kódu a hlavně mít na mysli, že jakékoli zdvojení informace klade zvýšené nároky na následnou údržbu a rozvoj aplikace. Pravidlo lze aplikovat nejen na kód aplikace, ale také na databázové schéma, testy ale také na dokumentaci k aplikaci. Při důkladně dodržovaném DRY principu lze upravit jeden element aplikace bez nutnosti úpravy elementů navazujících. Je nutné dodat, že ne vždy je možné toto pravidlo dodržet, avšak jeho záměrné porušování vede ke snížení čitelnosti kódu, spravovatelnosti aplikace a taktéž se zvyšuje riziko chyby v aplikaci.

Dalším z postupů vývoje aplikací je **KISS** (**K**eep **I**t **S**imple and **S**traithforward), jehož filosofií je zachování jednoduchosti řešených úloh či problémů. Lze jej definovat následovně: „*Návrh řešení by měl byt tak jednoduchý, jak je možné, avšak ne jednodušší*“. Cílem pravidla je upozornit na skutečnost, že lidský mozek má mnohdy sklony k umělému zesložiťování problému a přitom řešení mnohdy tkví v jednoduchosti, a to zejména v rozdělení řešeného problému na malé části, které jsou funkčně jednoduché a lze je řešit samostatně. Princip pravidla vyslovil Kelly Johnson, jež byl hlavním leteckým konstruktérem ve firmě Lockheed Skunk Works, v souvislosti s požadavkem, že návrh konstrukce letadla musí být maximálně jednoduchý, aby jej bylo možné opravit běžně dostupnými nástroji, a to i v případě boje. Citované tvrzení lze velmi dobře převést do prostředí softwarového vývoje, kdy je příliš složitý kód velmi obtížně čitelný, udržovatelný. Dále je vhodné myslet na životnost aplikace a na skutečnost, že aplikační kód bude upravován i jinými vývojáři, kteří mohou být časově, finančně či intelektuálně limitovaní. Naopak aplikační kód, který je rozdělen do jednoduchých, samostatných a samostatně fungujících logických celků je velmi přehledný a dobře udržovatelný.

Při vývoji aplikace bude použito **komponentově orientovaného návrhu aplikace** (Component Oriented Design). V tomto případě se spíše než o princip jedná o návrhový vzor, který poukazuje na maximální znovupoužitelnost komponent aplikace. Komponentou může být například knihovna, webová služba, prvek grafického rozraní či dokonce celý modul v rámci aplikace. Jde v podstatě o maximální separaci kontextově souvisejících prvků, které mezi sebou komunikují na základě jasně definovaného rozhraní. Tento přístup tak poskytuje vývojáři možnost vyměnit danou komponentu bez rizika porušení funkčnosti aplikace. Taktéž je zvýšena míra udržitelnosti a spravovatelnosti aplikace.

Dále bude v obou frameworcích, jež budou použity jak na straně serveru (Spring Framework) tak na straně klienta (Apache Flex + Parsley), využito návrhového vzoru **Dependency Injection** (DI), který poskytuje možnost vyhnout se přímo definovaným závislostem v kódu aplikace. Jde o konfiguraci či inicializaci objektu, při které jsou potřebné objekty a spolupracující komponenty inicializovány pomocí externí entity, v tomto případě DI kontejneru. Přiřazování je prováděno na základě typu daného objektu či předem definovaného jednoznačného identifikátoru. Výhodou tohoto návrhového vzoru je redukce zbytečného kódu v inicializačních strukturách aplikace, flexibilita konfigurace a velmi dobrá testovatelnost.

Dalším prvkem, který bude při návrhu aplikace použit, jsou **anotace**. Anotace budou využity jak na straně klienta, tak na straně serveru. Jde o dodatečné metainformace ve zdrojovém kódu, jež poskytují kompilátoru dodatečné informace o chování dané proměnné, metody či objektu. Anotacemi lze velmi značně ovlivnit chování aplikace, popř. toto chování rozšířit. Anotací například v hojné míře využívá výše zmíněný návrhový vzor Dependency Injection.

Poslední a zřejmě nejzajímavější technologií bude využití **Aspektů**. Aspekty budou využity pouze na straně serveru. Aspektem se myslí část programu, objekt či metoda, jejíž funkčnost je napojena na ostatní části systému, avšak tyto části systému nejsou s tímto objektem či metodou přímo spojeny. Princip funkčnosti aspektu lze představit na příkladu logovacího modulu, jehož úkolem je logování všech událostí a volání metod napříč aplikací, avšak metody o existenci tohoto logovacího modulu vůbec nevědí, nejsou s ním jakkoli propojeny. Tento přístup vývoje softwaru se nazývá Aspektově Orientované Programování (AOP – **A**spect **O**riented **P**rogramming), jehož cílem je maximální oddělení částí systému a zvýšení modularity aplikace.

6.4 Adobe Flash Platform - architektura klienta

Jak již bylo uvedeno, klient aplikace bude postaven na platformě Adobe Flash a jako nástroj, pro tvorbu aplikace samotné bude sloužit framework Apache Flex s podporou frameworku Parsley. Zde je na místě vysvětlit vzájemný vztah těchto dvou frameworků. Framework Apache Flex poskytuje prostředky pro základní vývoj aplikace, jako jsou jednotné formulářové prvky a jiné komponenty potřebné pro tvorbu grafického rozhraní aplikace, dále základní model řízený událostmi (Event Driven Model), systémem pro lokalizaci aplikace (i18n) a řešením pro implementaci témat (skinů) aplikace. Nevýhodou je nepřítomnost pokročilejších aplikačních funkcí, jako je například Dependency Injection (DI) a rozšířený model řízený událostmi, který dovoluje řízení událostí v rámci aplikace pomocí anotací. Právě pro tyto účely je využito nástrojů, které poskytuje framework Parsley. V podstatě lze říci, že framework Parsley je vybudován nad Apache Flex frameworkem, navzájem se tedy doplňují a díky tomuto spojení lze využít pokročilých aplikačních funkcí.

6.4.1 Apache Flex

Apache Flex je aplikačním frameworkem vytvořeným původně firmou Macromedia, později firmou Adobe Inc., a nyní jeho vývoj zaštiťuje organizace Apache. Framework byl představen již v roce 2004. V současné době je dostupný ve verzi 4.9 (1.4.2013) a je distribuován pomocí balíčku zvaného Apache Flex SDK (Software Development Kit), který obsahuje kompletní zdrojové soubory frameworku a dále soubory potřebné pro jeho kompilaci.

Apache Flex využívá jako programovacího jazyku jazyk ActionScript verze 3, původně vyvinutého firmou Macromedia. Jazyk vychází z dialektu ECMAScript, jehož sémantika a syntaxe je založena na jazyku JavaScript. Je určen primárně pro vývoj aplikací běžících na Adobe Flash platformě. Specifikace jazyka samotného je dostupná pod open-source licencí. Jedná se o moderní objektový jazyk, který podporuje poslední objektové přístupy:

- silná typově orientovaná architektura – jsou dostupné jak primitivní datové typy (int, boolean), tak i jejich objektové nástavby (Integer, Number, String, Boolean);
- podpora objektových přístupů, jako je dědičnost či polymorfismus;
- podpora balíčků, jmenných prostorů a regulárních výrazů;
- kompilace programového kódu do bytecode;
- událostmi řízená architektura;
- plná podpora ECMASrcipt pro XML zaručující integrované zpracování XML dokumentu;
- podpora pro vykreslování 3D objektů.

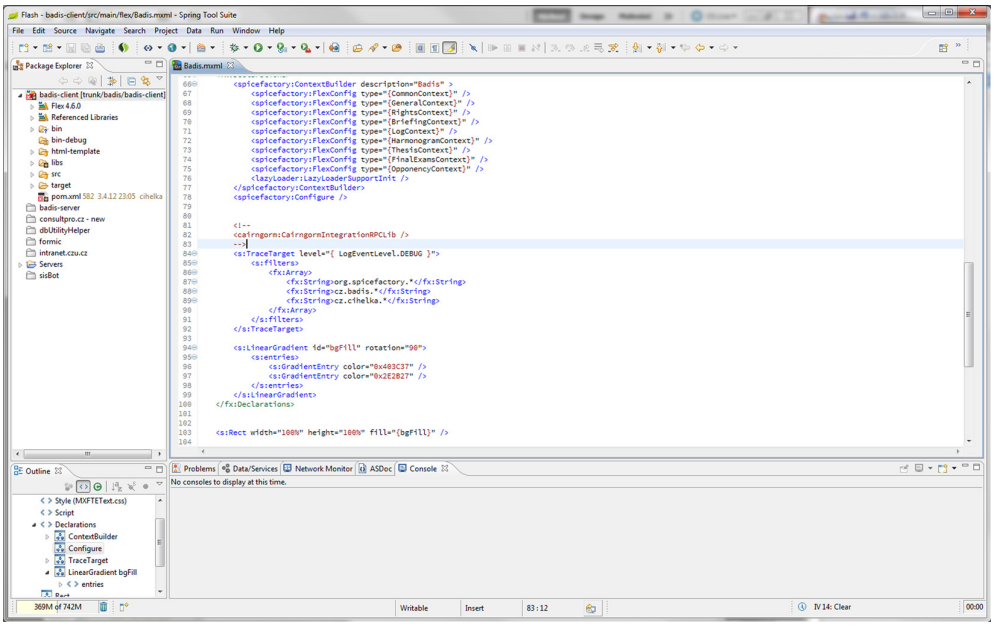
Pro zápis vzhledu využívá framework Apache Flex značkovacího jazyka MXML, který je založený na základu jazyka XML. V zápisu vhledu aplikace (GUI) je možné kombinovat jazyk MXML s jazykem ActionScript, popř. komponenty jazyka MXML zapisovat programově přímo v jazyce ActionScript. V praxi se však využívá kombinace těchto přístupů, kdy je jazyk MXML použit k tvorbě vzhledu a jazyk ActionScript je využit pro ovládání prvků. Tento přístup i zápis kódu je velmi shodný s využitím HTML a JavaScriptu v případě běžných internetových stránek.

6.4.2 Vývojové prostředí Adobe Flash Builder

Jedná se o komerční vývojové prostředí pro vývoj aplikací v Apache Flex. Pro běh využívá obecného prostředí Eclipse, které je vyvíjeno neziskovou společností Eclipse Foundation, jejímiž členy jsou firmy jako IBM, Oracle, Sun, Zend, SAP, Iona, Nokia [3].

Adobe Flash Builder poskytuje editor ActionScript kódu s podporou zvýraznění syntaxe jazyka. Psaný kód je průběžně kontrolován na správnost syntaxe a v případě nalezení chyby je vývojář okamžitě upozorněn.

Vývojové prostředí obsahuje nástroje pro odhalování chyb (debugger), k dispozici je zde také Profiler, který umožňuje odhalit paměťové chyby vznikající při běhu aplikace.

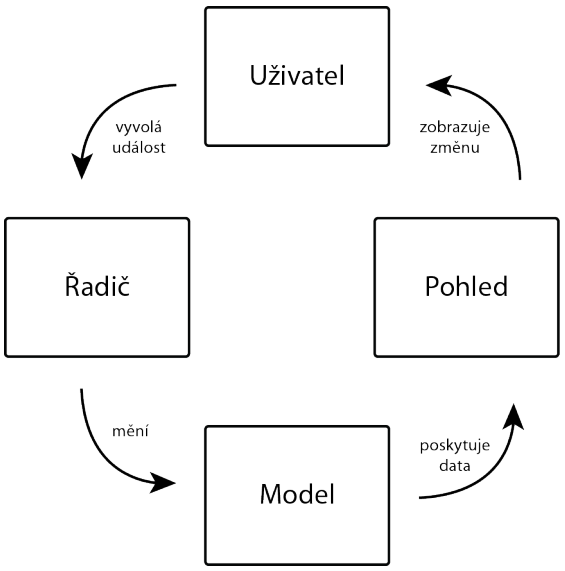


Obrázek 12: Rozhraní Adobe Flash Builder [autor]

6.4.3 Architektura klienta

Architektura klienta bude postavena na základě softwarového vzoru MVC (**M**odel – **V**iew – **C**ontroller). Cílem tohoto vzoru je oddělení informací a datových struktur od uživatelské interakce. Jak již z názvu plyne, veškerá interakce v rámci aplikace je rozdělena do tří částí: *Model* (Model), *Pohled* (View) a *Řadič* (Controller).

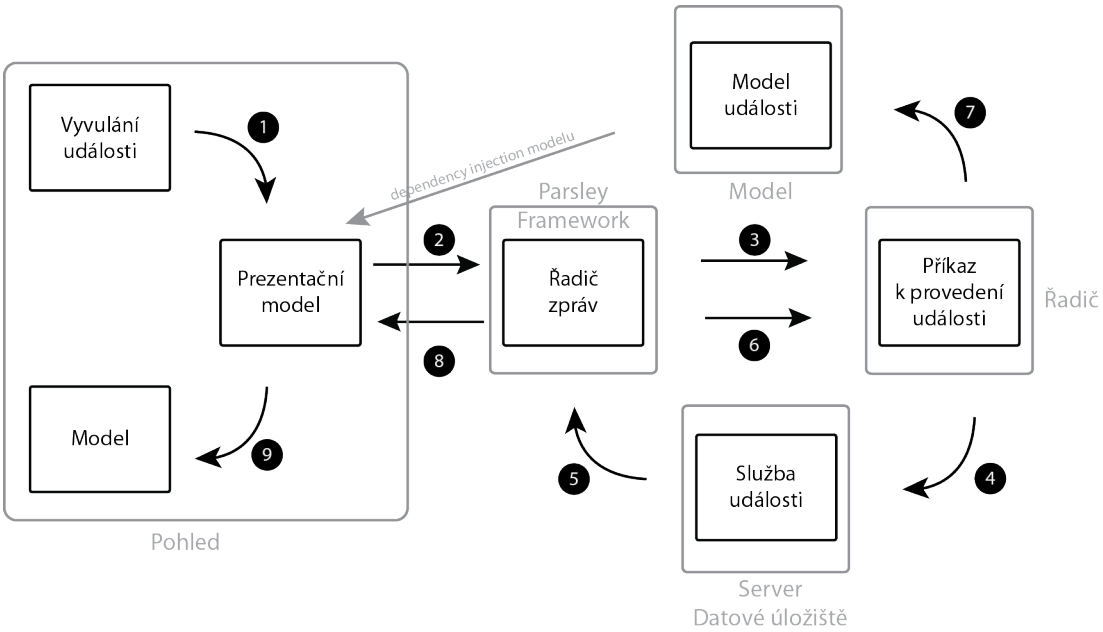
- Řadič (Controller) – reaguje na události v aplikaci a v reakci na tyto události zajišťuje změnu v modelu či pohledu. Tyto události jsou povětšinou vyvolány akcí uživatele (např. kliknutí na tlačítko);
- Model – jde o doménově specifickou reprezentaci informací, s nimiž aplikace pracuje, například seznam vedených prací daného pedagoga;
- Pohled (View) – převádí data v modelu na grafickou reprezentaci a na základě uživatelské interakce vyvolává požadavky na Řadič.



Obrázek 13: Softwarový vzor MVC [autor]

Výhodou použitého MVC vzoru je oddělení prezentační logiky od aplikační logiky. Implementace MVC se může zdát na první pohled složitá a v mnoha případech tomu tak je. Přednosti MVC lze spatřovat v následné správě aplikace, protože je vývojáři umožněna změna části aplikace bez nutnosti měnit další prvky. Současně se zvyšuje čitelnost a rozšiřitelnost jednotlivých prvků aplikace díky jasnému rozdělení zodpovědnosti.

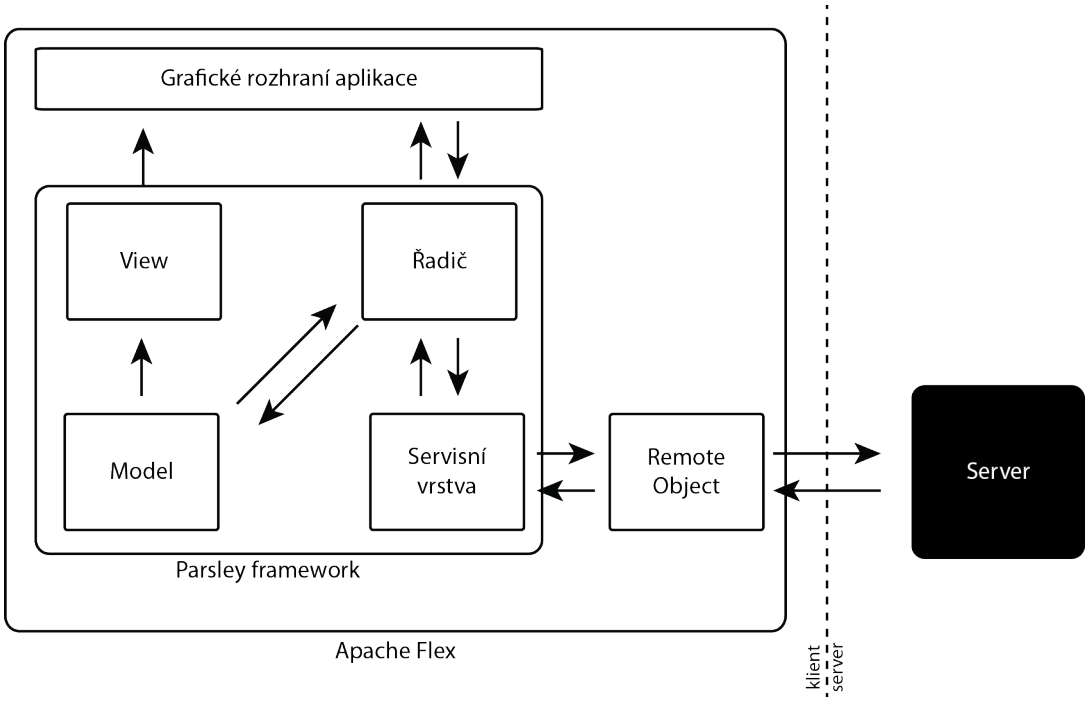
V případě využití MVC vzoru v prostředí Parsley frameworku dochází k malé změně. *Pohled* (View) rozšiřuje o svůj vlastní model, zvaný *Prezentační Model* (Presentation Model). Pohled pak reaguje na změny v tomto *Prezentačním Modelu*. Zavedení prezentačního modelu je z důvodu principiální funkčnosti MVC v Parsley frameworku, jež interně závisí na zasílání asynchronních zpráv mezi jednotlivými komponentami. Právě tohoto zasílání zpráv je využito i v MVC Parsley frameworku. Využití zasílání zpráv má bohužel negativní důsledek v tom, že není možné vyvolat odeslání zprávy přímo z pohledu, ale je nutné tuto zprávu s uživatelskou interakcí volat prostřednictvím *Prezentačního Modelu*.



Obrázek 14: Parsley MVC schéma [autor]

Uvedené schéma (Obrázek 14) znázorňuje průběh volání jednotlivých komponent při vyvolání uživatelské akce na systém. Z uvedeného schématu je dobře patrná skutečnost, že Parsley framework není jakkoli provázán s komponentami MVC modelu. Funguje pouze jako jakýsi zprostředkovatel komunikace mezi jednotlivými komponentami, avšak žádným způsobem do jejich vzájemné interakce nevstupuje. Oddělení je dosaženo právě díky využití zpráv a skutečnosti, že celý Apache Flex framework je řízen událostmi. Nízká provázanost Parsley frameworku a jednotlivých komponent poskytuje vývojáři možnost vyměnit celý MVC systém za jiný a to bez razantního dopadu na ostatní komponenty systému. Tohoto nízkého provázání jednotlivých komponent je dosaženo díky návrhovému vzoru Dependency Injection, jež byl představen výše.

Aplikační logika klientské aplikace je prováděna *Servisní vrstvou*, která má ve své kompetenci volání vzdálených procedur. Samotné zpracování vzdáleného příkazu a navrácení výsledku je v pravomoci vrstvy zvané *RemoteObject*.



Obrázek 15: Schéma architektury klientské strany aplikace [autor]

Samotná servisní vrstva je složena z jednotlivých příkazů, které jsou definovány v kontextovém konfiguračním souboru frameworku Parsley, jež vypadá následovně:

```
<parsley:DynamicCommand type="{ PraceFillCommand }" selector="{
PraceEvent.FILL}" />
<parsley:DynamicCommand type="{ PraceCreateCommand }"
selector="{ PraceEvent.CREATE}" />
<parsley:DynamicCommand type="{ PraceUpdateCommand }"
selector="{ PraceEvent.UPDATE}" />

<s:RemoteObject id="praceService" destination="remote-práce-
service" />
```

Příklad 1: Konfigurace řadiče [autor]

Uvedené tři definice určují, jaký příkaz se má při vyvolání dané události vykonat. V případě požadavku na vytvoření nové práce je vyvolána událost *PraceEvent*.CREATE. MVC provede vyhledání a spuštění příkazu, který této události náleží, tj. je vykonán příkaz *PraceCreateCommand*.

Jednotlivé příkazy jsou zapsány pomocí jazyka ActionScript a jejich struktura vypadá následovně:

```
package cz.czu.oikt.infrastructure {

public class PraceCreateCommand {

    [Inject(id="praceService")]
    public var praceService:RemoteObject;

    public function execute ( event:PraceEvent ): AsyncToken {
        return praceService.createItem( event.prace );
   }
    public function result ( data:ResultEvent, event:PraceEvent
):void
    {
      /* obsluha vysledku */
    }

    public function error ( fault:Fault ): void {
        /* obsluha chyby */
    }
  }
}
```

Příklad 2: Struktura příkazové třídy [autor]

Každá třída příkazu musí obsahovat alespoň dvě metody a to metodu *execute()* a metodu *result()*. Metoda *error()* je zde uvedena z důvodů kompletní představy o struktuře třídy, metoda sama obsahuje řešení nastalé chyby, jež byla vrácena serverem. V praxi se využívá globálního řešení chyb, popř. je využito dědičnosti a metoda je extrahována do nadřazené třídy, ze které ostatní příkazové třídy dědí funkčnost. Na struktuře příkazu je dále patrno využití Dependency Injection pomocí anotace *[Inject]*, jež frameworku sděluje, že má v případě vytvoření instance třídy *PraceCreateCommand* vložit do proměnné *praceService* instanci objektu, jehož jedinečné id je *praceService*. Instance třídy *praceService* je definována v kontextovém konfiguračním souboru.

Je dobré poznamenat, že kontextový konfigurační soubor je jediným prvkem, který významněji zasahuje do vzájemné interakce mezi jednotlivými komponentami frameworku. Jedná se v podstatě o jakýsi registr, ze kterého Parsley framework čerpá informace o komponentách systému. Komponenty systému a popis jejich vzájemné interakce lze popisovat i přímo v kódu a to pomocí anotací:

```
[Event(name="createPrace",type="cz.czu.oikt.application.
PraceEvent")]
[ManagedEvents("createPrace")]
public class PraceService extends EventDispatcher {
    private function handleCallResult ( prace:PraceVO ) : void {
        dispatchEvent(new PraceEvent("createPrace", práce ));
    }
}

.....

public class PraceManager {
    [MessageHandler(selektor="createPrace")]
    private function handleResult ( event:PraceEvent ) : void {
      [. . . . ]
    }
}
```

Příklad 3: Konfigurace zasílání a příjímání zpráv pomocí anotací [autor]

Výše uvedený příklad využívá anotací ke konfiguraci objektů. První třída *PraceService* poskytuje pomocí anotace *[Event]* informaci, že bude vyvolávat událost s identifikátorem *createPrace*, která je typu *PraceEvent*. Definice *[Event]* je standardní součástí frameworku Apache Flex. Anotace *[ManagedEvents]* je již anotací frameworku Parsley a říká, že událost *createPrace* má být spravována frameworkem. Pokud by nebyla anotace *[ManagedEvents]* použita, framework Parsley by její volání ignoroval.

V případě druhé třídy *PraceManager* je použita anotace *[MessageHandler]* jež podává frameworku Parsley informaci, že metoda *handleResult* naslouchá na vyvolání události, jež je typu *PraceEvent*. Vzhledem k tomu, že událost typu *PraceEvent* může obsahovat kromě identifikátoru *createPrace* i jiné identifikátory, je v tomto případě využito parametru *selector*, který říká, že daná metoda bude spuštěna pouze v případě, že byla vyvolána událost *PraceEvent* s identifikátorem *praceCreate*. Pokud by nebyl parametr *selector* použit, byla by metoda *handleResult* spuštěna při jakémkoli volání události, jež je typu *PraceEvent*.

Využití zasílání událostí jakožto nástroje pro interakci mezi objekty je mocným nástrojem, pomocí kterého lze vybudovat velmi složité programové struktury. Při volání jedné události je možné vyvolat spuštění několika jiných metod, které volání naslouchají. Při použití dědičnosti objektů je dokonce možné naslouchat i větší množině událostí, jež mají stejného předka. Framework Parsley poskytuje nástroje pro určení pořadí, ve kterém budou metody volány a vytvářet tak sekvenční struktury zpracování událostí. Vlastní kapitolou je využití tzv. *MessageInterceptors*, jež mají schopnost vstoupit mezi zpracování vyvolané události a spuštění metody, která na vyvolanou událost naslouchá.

```
public class PraceCreateInterceptor {
      [MessageHandler]
      public function praceInterceptorHandler (
event:PraceEvent, procesor:MessageProcessor ):void {
            processor.suspend();
            if( event.prace.id == 1) {
                  processor.resume();
            } else {
                  processor.cancel();
            }
      }
}
```

Příklad 4: Ukázka třídy MessageInterceptor [autor]

Uvedený interceptor *PraceCreateInterceptor* narušuje volání události *PraceEvent*. Při vyvolání této události je spuštěna metoda *praceInterceptorHandler*, která vykonáním metody *processor.suspend()* pozastaví odeslání zprávy a tím i spuštění metod, které události naslouchají. V případě, že je identifikátor práce roven jedné, je povoleno pomocí metody *processor.resume()* odeslání zprávy. V opačném případě je jakékoli další zpracování pomocí příkazu *processor.cancel()* zrušeno. Použití interceptorů je dalším z mocných nástrojů frameworku Parsley, avšak jde o jediný případ, kdy dochází k silnému provázání s frameworkem. Uvedená závislost je způsobena předaným parametrem *processor:MessageProcessor*, jež je objektem frameworku. Proto je vhodné využití interceptorů dobře dokumentovat.

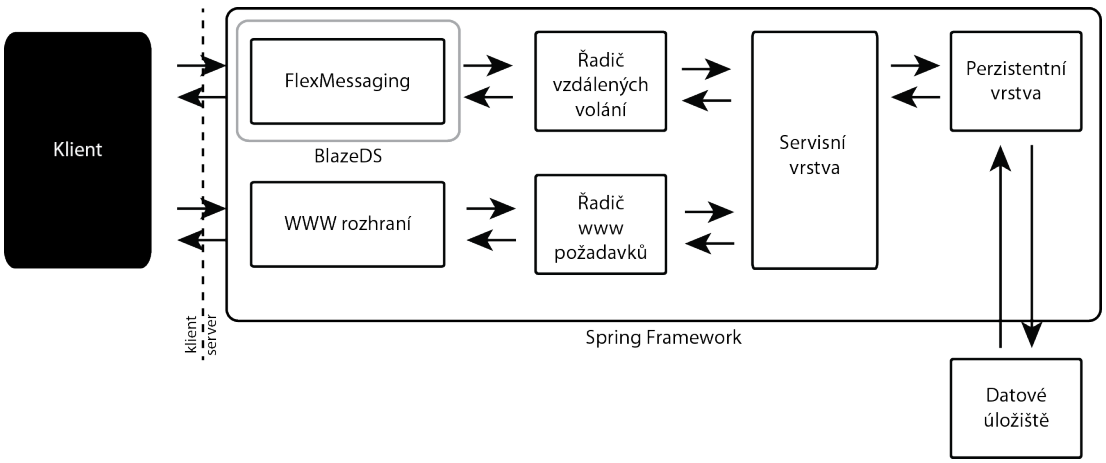
6.5 Java – architektura serveru

Architektura serverové části aplikace je velmi podobná klientské části. I v tomto případě je využito MVC vzoru, pouze je v tomto případě rozděleno přijímání požadavku mezi dva druhy řadičů. Jedním z řadičů je řadič vzdálených volání, který přijímá požadavky vzdálených volání, jež byly doručeny do vrstvy zvané *FlexMessaging*. Uvedená vrstva úzce spolupracuje s *RemoteObject* vrstvou klientské strany a provádí převod přijatých požadavků na platné objekty aplikace. Již převedené volání je pak dále předáno *Řadiči vzdálených volání*, který na základě požadavku vykoná pomocí *Servisní vrstvy* požadované akce.

Druhým z řadičů je *Řadič www požadavku*, jež je standardním MVC řadičem, který přijímá požadavky vykonané například pomocí internetového prohlížeče. Tento řadič je použit zejména z důvodů poskytnutí API rozhraní pro ostatní aplikace, které jsou součástí informačního systému univerzity.

Jednotlivé řadiče fungují pouze jako tranzitní objekty, které neobsahují žádnou aplikační logiku. Veškerá aplikační logika je provedená v rámci *Servisní vrstvy*, jež je také jako jediná schopna volat prvky *Perzistentní vrstvy*.

Spring Framework shodně s Parsley frameworkem volí přístup, ve kterém jsou prvky systému maximálně odděleny od vlastností samotného frameworku. Shodná je i konfigurace, která je provedena formou XML či anotací. Pro interakci jednotlivých částí systému je ve velké míře využito Dependency Injection.



Obrázek 16: Schéma architektury serverové části [autor]

Perzistentní vrstva využívá návrhového vzoru DAO (**D**ata **A**ccess **O**bject), jehož cílem je izolace aplikačních požadavků na datové úložiště a samotným vykonáním těchto požadavků. V praxi to znamená, že *Servisní vrstva* volá rozhraní *Perzistentní vrstvy*, ale již se nestará, odkud a jak *Perzistentní vrstva* data získá. Způsob získání dat je právě v kompetenci *Perzistentní vrstvy* a v případě, že je jako *Datové úložiště* použita databáze, tak je *Perzistentní vrstva* jediné místo v aplikaci, kde mohou být umístěny SQL dotazy. Poslední odpovědností *Perzistentní vrstvy* je převod získaných dat z *Datového úložiště* na platné objekty aplikace, zvané *Value Object* (VO), které jsou pak předány jako výsledek požadavku *Servisní vrstvě*.

Pro potřeby *Perzistentní vrstvy* bylo vyvinuto vlastní řešení. K tomuto kroku bylo přistoupeno z důvodu potřeby automatizovat převod získaných dat z databáze na platné objekty aplikace. Převod je realizován pomocí anotačního mapování na jednotlivé prvky databáze.

Příklad třídy aplikace s nastaveným mapováním:

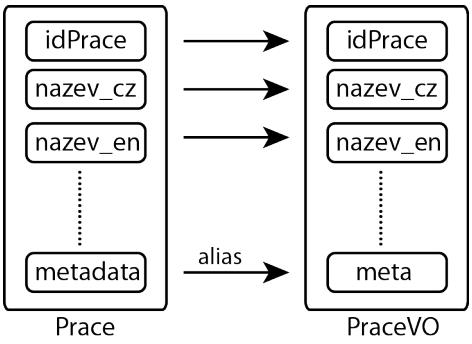
```
@BadisValueAnnotation(name="table", value="Prace")
public class PraceVO extends InteligentVO implements
IValueObject, IInteligentVO {
      @BadisAnnotation(name=BadisAnnotationKeys.PRIMARY_KEY)
      public int idPrace;
      public String nazev_cz;
      public String nazev_en;

      @BadisAlias(value="metadata")
      public String meta
      ...
      ...
}
```

Příklad 5: Nastavení mapování [autor]

V uvedeném přikladu je pomocí anotace *@BadisValueAnnotation* řečeno, že daná třída je mapována z tabulky („name=table“) jejíž jméno je *Prace*. Samotná třída *PraceVO* využívá rozhraní *IValujeObject* a *IInteligentVO*, jež jsou rozhraními poskytovanými uvedeným vlastním řešením. Tato rozhraní popisují podpůrné metody (např. toString, hashMap), které jsou pro jejich obecnou platnost implementovány v třídě *InteligentVO* jež je předkem třídy *PraceVO*. Dále je ve struktuře vlastností třídy využito anotace *@BadisAnnotation*, jež určuje, že vlastnost *idPrace* je primárním klíčem tabulky *Práce*. Ostatní parametry jako *nazev_cz* jsou mapovány na základě shodného názvu databázového sloupce a vlastnosti dané třídy. V případě, že se názvy

neshodují, je možné využít anotace *@BadisAlias* jež má jako parametr uveden název databázového sloupce *metadata*, na který je vlastnost *meta* mapována.



Obrázek 17: Mapování parametrů [autor]

Další ze zajímavých vlastností daného řešení je schopnost definice SQL dotazů pomocí spojování jednotlivých objektů aplikace. Z tohoto spojení je pak vygenerován samotný SQL dotaz, který je odeslán do databáze. Při návratu z databáze jsou dané záznamy automaticky převedeny na objekty a zároveň je provedeno automatické zanoření těchto objektů. Problematiku lze demonstrovat na příkladu, kdy je potřeba získat záznam absolventské práce s požadavkem na přítomnost prvku vedoucího práce. V tomto případě jsou definovány třídy:

```
@BadisValueAnnotation(name="table", value="Prace")
public class PraceVO extends InteligentVO implements
IValueObject, IInteligentVO {
@BadisAnnotation(name=BadisAnnotationKeys.PRIMARY_KEY)
      public int idPrace;
      public String nazev_cz;
      public String nazev_en;

      @BadisNotDbField
      public VedouciPraceVO vedouci;
      public int uicVedouci;

      @BadisAlias(value="metadata")
      public String meta

      ....
      ....
}
```

```
@BadisValueAnnotation(name="table", value="VedouciPrace")
public class VedouciPraceVO extends InteligentVO implements
IValueObject, IInteligentVO {
      @BadisAnnotation(name=BadisAnnotationKeys.PRIMARY_KEY)
      public int uicVedouci;
      public String jmeno;
      public String prijmeni;

      ....
      ....
}
```

Příklad 6: Konfigurace zanoření objektů aplikace [autor]

V kódu třídy *PraceVO* je uvedena nová anotace *@BadisNotDbField*, jež říká, že daná vlastnost třídy nemá reprezentaci v databázové tabulce a nemá být tedy zahrnuta do SQL dotazů. Naopak je tato vlastnost určena pro vložení objektu *VedouciPraceVO*.

Definice propojení těchto dotazů:

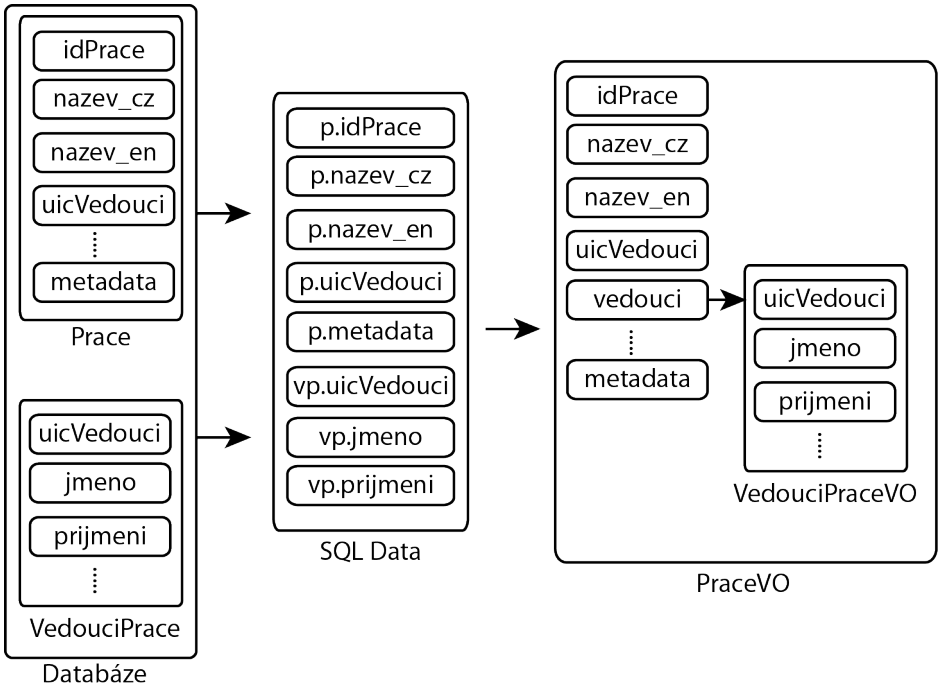
```
...
FillQuery fq = new FillQuery().select().from( new
FillQueryTable( PraceVO.class, "p") ).join( new FillQueryTable(
VedouciPraceVO.class, "vp").join("INNER JOIN").on("o.uicVedouci
= vp.uicVedouci").injectInto("p", "vedouci");
...
```

Příklad 7: Definice propojení objektů aplikace [autor]

Výsledkem uvedeného propojení je SQL dotaz:

```
SELECT p.idPrace, p.nazev_cz, p.nazev_en, p.uicVedouci,
p.metadata, vp.uicVedouci, vp.jmeno, vp.prijmeni FROM Prace AS p
INNER JOIN VedouciPrace AS vp ON p.uicVedouci = vp.uicVedouci;
```

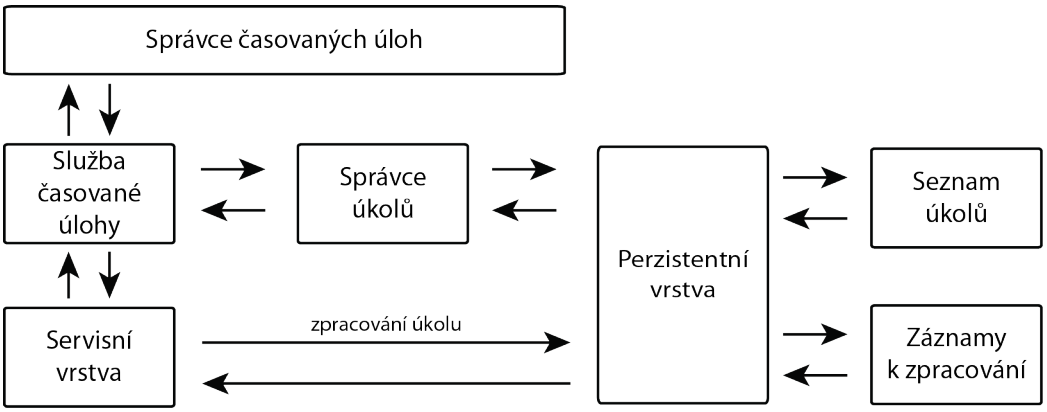
Příklad 8: Příklad výsledného SQL příkazu [autor]



Obrázek 18: Diagram mapování vnořených parametrů [autor]

6.6 Architektura časovaného zpracování úloh

Složité a časově náročné operace aplikace jsou prováděny systémem pro časované zpracování úloh. Jde v podstatě o služby, které jsou v daných časových intervalech spouštěny *Správcem časovaných úloh*. Každá, ze spouštěných časových služeb má k dispozici vlastního *Správce úkolů*, do kterého aplikace či jiné časově spouštěné služby ukládají úlohy připravené pro zapracování časovou službou. Jde v podstatě o FIFO (**F**irst **I**n **F**irst **O**ut) frontu úloh. Časované úlohy obsahují aplikační logiku potřebnou pro splnění úkolu, ale v maximální možné míře využívají *Servisní vrstvy*.



Obrázek 19: Schéma časového zpracování úloh [autor]

Aby bylo dosaženo maximální jednoduchosti daných služeb, jsou některé operace rozděleny do několika časovaných služeb, které spolu spolupracují, popř. si předávají navzájem úkoly. Jemné rozdělení služeb lze demonstrovat na případu kolekce služeb, jež mají za úkol odesílat odevzdané absolventské práce do portálu theses.cz zajišťující kontrolu prací na obsahovou shodu (kontrola plagiátorství).

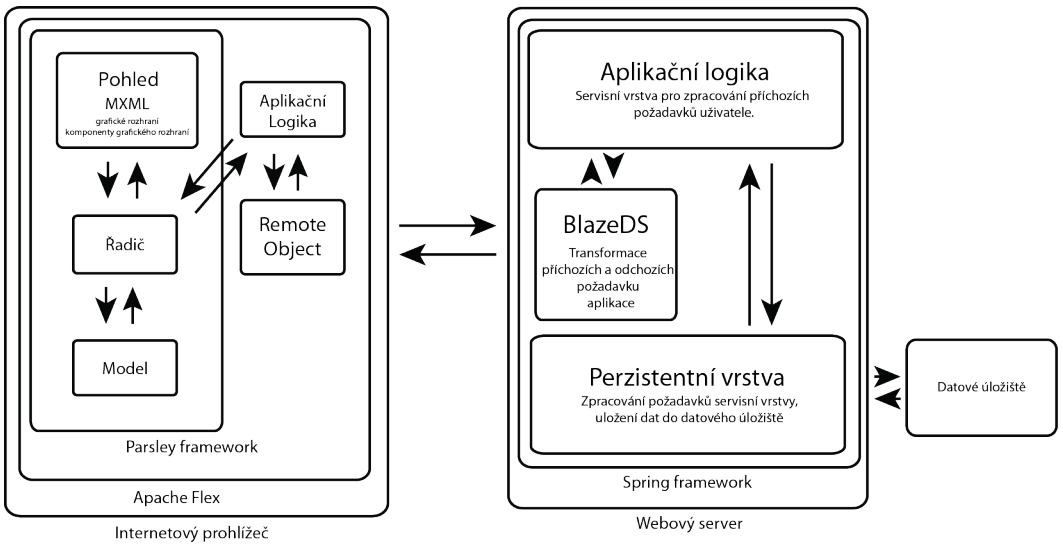
Služba	Popis činnosti
Vyhledání BP/DP prací pro odeslání	Vyhledá odevzdané práce a vytvoří úkoly pro následující služby
Odeslání BP/DP prací do portálu theses	Služba provede odeslání prací do portálu theses.cz
Kontrola stažených prací portálem theses.cz	Služba kontroluje, zda portál theses.cz vyzvedl práce určené ke kontrole. V případě, že k vyzvednutí došlo, připraví služba úkol pro kontrolu výsledků.
Získání výsledků z portálu theses.cz	Služba splní zadané úkoly a provede zapsání výsledků.
Přepočet výsledků na obsahovou kontrolu	Služba vyhledá nově zapsané výsledky kontroly a provede jejich zpracování a následné propsání do výsledkových tabulek.

Tabulka 11: Členění časové služby [autor]

Z přehledu je patrné, že jedna operace je rozdělena do 5 samostatných služeb, jež mají přesně určené oblasti své působnosti a některé mezi sebou spolupracují. Ostatní hlídají změny v databázi a na základě těchto změn provádějí určené akce.

Princip časového zpracování služeb poskytuje možnost velmi dobrého škálování celé aplikace a díky jasně dané zodpovědnosti daných služeb je zvýšena i míra spravovatelnosti a rozšiřitelnosti aplikace. Časované zpracování úloh je velmi ohleduplné k celkovému zatížení systému a v případě vysokého zatížení aplikace je možné celý systém časovaných úloh přesunout na jiný server. Dále jde o velmi odolný způsob proti případným nehodám, například při pádu aplikačního serveru. Veškeré úlohy jsou zachovávány v databázi a v případě chyby či přerušení systém pokračuje od místa, kde došlo k přerušení. V případě potřeby by bylo možné využít i paralelního zpracování úloh na více serverech.

6.7 Celková architektura aplikace



Obrázek 20: Celková architektura aplikace [autor]

Obrázek 20 znázorňuje rozdělení architektury aplikace na dva základní bloky, stranu klienta a stranu serveru. Každá z částí je složena z několika vrstev, jež spolu úzce spolupracují. Uživateli je prezentováno grafické rozhraní (Pohled - View), které přijímá požadavky uživatele a následně je předává řadiči událostí. *Řadič* provede předání požadavku aplikační vrstvě, jež daný požadavek zpracuje a výsledky propíše do modelu. V případě požadavku na vzdálené volání předá aplikační vrstva požadavek na volání vrstvě *RemoteObject* a čeká na odpověď. *RemoteObject* vykoná vzdálené volání a navrátí výsledek do vrstvy aplikační logiky, která provede zpracování a zobrazení přijatých dat.

Strana serveru přijímá vzdálená volání klienta pomocí vrstvy *BlazeDS*, jež úzce spolupracuje s klientskou vrstvou *RemoteObject*. Obě tyto vrstvy zajišťují přenos objektů aplikace mezi klientem a serverem. V případě přijatého požadavku, ze strany klienta, provede vrstva *BlazeDS* předání do řadiče serverové části aplikace. *Řadič* aplikace předá volání aplikační vrstvě; ta vykoná požadované operace. Je-li při zpracování požadavku nutná komunikace s databází, je za tímto účelem volána *Perzistentní vrstva*, jež zajistí dodání požadovaných dat z *Datového úložiště*. Po zpracování obdrženého požadavku předá *Aplikační vrstva* výsledek volání *Řadiči*, který odešle prostřednictvím vrstvy *BlazeDS* odpověď klientské straně.

6.8 Zhodnocení

Architektura aplikace byla navržena s ohledem na splnění požadavků podnikových aplikací (Enterprise Applications) a charakteristických rysů RIA. V návrhu je využito moderních technologických principů (DRY, KISS) a postupů (MVC, DI, Aspekty). Začlenění těchto přístupů poskytuje výrazně lepší správu a čitelnost struktury aplikace. Taktéž je kladen důraz na možnost opakovaného použití aplikačních komponent a důsledného oddělení jednotlivých vrstev aplikace, jež umožňují efektivní správu aplikace a zejména její další rozvoj. Z uvedených závěrů lze vypozorovat, že mnoho použitých postupů se zaměřuje na následnou efektivní údržbu aplikace či následný rozvoj. Lze tvrdit, že tomu tak skutečně je. Již od vytvoření první části aplikace dochází k neustálé údržbě již hotového kódu a je tedy vhodné využít takových přístupů, které zajistí, že čas strávený správou a údržbou kódu aplikace bude efektivní.

Dalším z cílu optimálního návrhu bylo efektivní využití zdrojů, zejména v případě vysokého zatížení serverové části. Při využití architektury klient-server je nutné zajistit co nejrychlejší odezvu serverové části na požadavky klienta. Proto bylo pro potřeby aplikace vytvořeno řešení, které umožňuje časové náročné úlohy řadit do front a ty následně zpracovávat. Dalším způsobem, který může dopomoci k rychlosti zpracování požadavků na serveru je využití dočasných úložišť (Cache). Dočasná úložiště obsahují předpřipravené informace, jejichž výpočet je náročný, a to jak časově či výkonově. V případě požadavku klienta na takovýto druh informace je aplikační logikou automaticky navrácena hodnota z dočasného úložiště. Dočasná úložiště vždy obsahují jen ty hodnoty, které je možné získat či vypočítat z hodnot dostupných v datovém úložišti. Aktualizace dočasných úložišť je zajištěna pomocí Aspektů, které naslouchají na změny v aplikaci a automaticky aktualizují obsah dočasných úložišť.

Dalším z prvků, který usnadnil vývoj aplikace je využití vlastního řešení pro potřeby Perzistentní vrstvy. Uvedené řešení minimalizuje nutnost psaní základních SQL dotazů a odstraňuje nutnost vlastního převodu získaných databázových dat na objekty aplikace. Vývojář má tedy více času na rozvoj logiky aplikace.

7 Závěr

Od počátku spuštění do dnešních dní prošel Internet a technologie podílející se na jeho fungovaní, velice rychlým rozvojem. Nebyl to však jen Internet, kdo prošel komplexními změnami. K zásadnímu obratu došlo i v množství uživatelů Internetu, u jehož zrodu jich bylo několik málo tisíc a nyní se jejich počet přibližným odhadem rozrostl na 2,4 miliardy. Chápání Internetu a služeb, které Internet nabízí, se změnilo a nevyhnutelně i měnit bude. Tlak uživatelů s požadavkem na vyšší interaktivitu a intuitivnost webových aplikací dal vzniknout novému odvětví a pojmu RIA.

RIA zahrnují v dnešní době jak nástroje pro správu, tak i internetové portály a nově i desktopové aplikace. Zejména v oblasti administračních aplikací došlo k mohutnému rozvoji. Firmy si uvědomují výhody těchto aplikací plynoucí z faktu, že samotný internetový prohlížeč, který je v dnešní době k dispozici na téměř každém počítači, může fungovat i jako základ aplikací, ve kterých lze kupříkladu spravovat finance firmy, evidovat skladové zásoby či monitorovat pohyby na burze.

S příchodem pojmu Web 2.0 nabyly RIA obliby i u internetových portálů, pro které byly technologie RIA odpovědí na změny v chápání a nakládání s informacemi dostupnými na Internetu. Uživatelé v době sociálních sítí vyžadují vyšší míru interaktivity, chtějí sdílet informace a společně nové informace vytvářet. Na všechny tyto požadavky jsou RIA technologie odpovědí.

Nelze však hovořit pouze o kladech, které RIA přinášejí. Zvolení špatného vývojového směru může vést k problémům, které na první pohled nebudou patrné. Mohou například vzniknout problémy s přístupností aplikace či v odlišnosti s ovládáním. Může se také stát, že technologie zvolená bez ohledu na pravděpodobný budoucí vývoj v této oblasti přestane být výrobcem podporována, což povede k problémům se správou aplikací a jejich rozšiřováním. Při analýze a návrhu budoucí RIA je tedy potřeba klást důraz na účel a funkci, a dle těchto kritérií volit vhodný technologický směr, popř. zvolit určitou kombinaci.

RIA se s rozvojem Internetu a především webových aplikací, staly dobrým obchodním artiklem. Společnosti, které na tomto trhu podnikají, nabízejí kvalitní aplikace, a s využitím dostupných RIA technologií dokáží plnit různorodá přání klientů. Bohužel ani RIA, jakožto symbol interaktivity, přívětivosti a moderních technologických přístupů, se nedokázaly ubránit zneužití. Označení RIA mnohdy využívají aplikace, které ani zdaleka nedosahují základních požadavků kladených na RIA. Zmíněné zneužívání komerčně úspěšných značek, jež je v posledních letech populární, RIA škodí a je pouze

na tvůrcích aplikací, aby uživatelům dokázali poskytnout dostatek vodítek k rozpoznání opravdových RIA od náhražek, které se tímto označením pouze chlubí a zdaleka nedosahují takové kvality.

Cílem diplomové práce byl komplexní návrh aplikace na základě zvolené RIA platformy. Dílčím cílem bylo uvedení pojmu RIA a vymezení základních rysů těchto aplikací spojený s analýzou a zhodnocením možných přístupů jejich tvorby. Z výsledků byly syntetizovány závěry, na základě kterých byl definován optimální přístup pro návrh architektury vytvářené RIA.

V první části práce byl pojem RIA představen a co nejsmysluplněji definován. Také zde byl věnován prostor historii vzniku zpracovávaného pojmu. Byly představeny jednotlivé organizace a technologie, které ke vzniku RIA přispěly.

Dále byl proveden rozbor základních rysů RIA a vymezena oblast platnosti těchto aplikací. Byl nabídnut pohled na vývoj RIA z hlediska použitelnosti, uživatelské přívětivosti a rozdílnosti interakčního modelu RIA a desktopových aplikací. Představeny a popsány byly jednotlivé druhy RIA, se kterými se může uživatel setkat. Dále byl proveden rozbor mezních rysů a představena charakteristika RIA s uvedením dvou technologických směrů či přístupů, na základě kterých lze aplikace vytvářet. Taktéž byly vyjmenovány výhody a nevýhody RIA technologií jako celku, a byly porovnány i jednotlivé technologické směry. Byla provedena analýza a rozbor volby vhodného technologického směru pro RIA aplikace. V poslední kapitole první části bylo provedeno zamyšlení nad vztahem RIA aplikací a pojmu Web 2.0. Následovalo představení samotného pojmu Web 2.0 a k jeho kritickému zhodnocení.

Dalším z cílů diplomové práce bylo představení široké škály technologií a platforem, pomocí kterých lze RIA vytvářet. Jednotlivé technologie byly podrobně popsány a zhodnoceny podle kriterií, jež byly syntetizovány z charakteristických vlastností uvedených ve druhé části práce. Byla provedena analýza zastoupení daných platforem na trhu a byl vysloven předpoklad jejich budoucího vývoje. Závěrem byla, na základě hodnotících kritérií, každá z platforem obodována a bylo provedeno jejich vzájemné porovnání.

Hlavním cílem práce bylo představení praktického postupu při realizaci RIA. Bylo provedeno zadání aplikace, v jehož rámci došlo k odůvodnění volby technologie Adobe Flash a Java. Na základě požadavků na funkčnost aplikace byla navržena její architektura. Představeny byly hlavní technologické postupy, které byly při realizaci použity. Detailně byla popsána architektura klienta realizovaného pomocí frameworku Apache Flex s podporou frameworku Parsley. Taktéž bylo charakterizováno vývojové

prostředí Adobe Flash Builder s popisem základních vlastností tohoto nástroje. Dále bylo představeno řešení na straně serveru s využitím programovacího jazyka Java a frameworku SpringSource. Shodně s architekturou klientské strany byl uveden princip funkčnosti MVC modelu. Dále bylo popsáno vlastní řešení Perzistentní vrstvy vyvinuté pro tento účel. V závěru praktické části byla představena architektura časovaného zpracování úloh a možnosti jejího rozšíření v případě nárůstu počtu uživatelů.

Navržená architektura aplikace, jež byla cílem diplomové práce, je splněna s ohledem na požadavky podnikových aplikací (Enterprise Applications) a RIA. Při návrhu jsou použity moderní technologické principy a postupy, jež umožňují vysokou míru udržitelnosti kódu, ale i aplikace jako celku. Důraz je kladen na nízkou provázanost jednotlivých vrstev aplikace, díky které je možné aplikaci efektivně spravovat a rozšiřovat.

Představená architektura klientské části využívá model jednotlivých aplikačních vrstev, které spolu komunikují na základě událostí. Taktéž je ve velké míře využito komponentově orientovaného návrhu aplikace, který je zaměřen zejména na znovupoužitelnost. Rozhraní aplikace je navrženo s ohledem na vysokou míru uživatelského komfortu a přívětivosti, ohled byl brán i na rozdílné chápání aplikace uživatelem, zejména ve smyslu jejího ovládání.

Architektura serverové části je zaměřena na poskytnutí dobré výkonnosti i při vysokém zatíženi, zejména díky možnosti přesunutí náročných úloh (časově i zdrojově) do systému front, jež je možné zpracovávat dávkově a díky navrženému řešení i zcela oddělit od aplikačního serveru. Představené vlastní řešení perzistentní vrstvy poskytuje nástroje k efektivní správě a manipulaci s objekty aplikace a jejich následnému ukládání do datového úložiště.

Přestože byl návrh aplikace proveden v souladu s moderními principy a postupy, je nutné poukázat na některá omezení, která navrhované řešení obsahuje. Použití anotací ve zdrojových kódech je při špatné organizaci kódu velmi nebezpečné, zejména z toho důvodu, že anotace samotné lze velmi jednoduše přehlédnout. Vzhledem ke skutečnosti, že anotace mohou značně ovlivnit chování aplikace, může mít toto opomenutí fatální následky v podobě nekontrolovatelného chování aplikace a v nejhorším scénáři k poškození dat. V případě klientské strany je nutné uvést potíže s MVC modelem aplikace. Díky zavedení Prezentačního modelu do struktury MVC dochází k nárůstu kódu aplikace a počtu zdrojových souborů, což zvyšuje nároky na jejich vhodnou organizaci. Samotný prezentační model pohledu má z definice zakázánu komunikaci s pohledem (View), což při složitější funkční logice daného prezentačního modelu

může vést k nárůstu složitosti pohledu, a tím i k delší prodlevě při zobrazení pohledu v aplikaci. Další oblastí je samotné zpracování příkazů, které je ve většině případů asynchronní. Při využití asynchronního volání vzdálených služeb je nutné počítat s možnou časovou prodlevou při jejich vykonání a přizpůsobit tomu funkční logiku aplikace. Na straně serveru je nutné upozornit na rizika spojená s využitím aspektů. Shodně jako u využití anotací, lze přítomnost aspektů přehlédnout, a to zejména díky faktu, že aspekty jsou od hlavního programového kódu aplikace zcela odděleny a pouze naslouchají volání metod. Proto je nutná jejich velmi dobrá organizace a dokumentace.

Z uvedených charakteristických vlastností a zejména možností jednotlivých technologií a platforem lze vydedukovat skutečnost, že RIA adoptují nejmodernější technologie, jež jsou na trhu dostupné. Lze tvrdit, že mnoho moderních technologií bylo pro potřeby RIA speciálně navrženo (Apache Flex, Microsoft Silverlight). Uvedený stav je výsledkem velkého tlaku společností na tvorbu a využití těchto aplikací, a to zejména díky skutečnosti, že RIA využívají běžný internetový prohlížeč jako základního běhového prostředí. Do budoucna lze předpokládat velmi podobný trend, kterému napomáhá rozvoj RIA platforem postavených na technologii HTML5 (např. AngularJS). Posledním trendem, který bude pravděpodobně pokračovat je přechod uživatelů od stolních počitačů na mobilní zařízení (telefony, tablety). Na tento trend reagují i RIA a ač jde mnohdy o poměrně složité aplikace, lze se s nimi setkat i na mobilních zařízeních.

8 Slovník pojmů

ActionScript

Objektově orientovaný programovací jazyk využitelný v aplikacích vyvíjených pro platformu Adobe Flash. Vychází ze standardizované verze jazyka ECMAScript.

AJAX

Asynchronous JavaScript And XML – obecné označení pro technologii vývoje interaktivních webových aplikací, které mění obsah svých stránek bez nutnosti jejich celkového znovunačítání.

AMF

Action Message Format – binární formát založený na protokolu SOAP. Je primárně určen pro výměnu dat mezi aplikacemi psanými na platformě Adobe Flash a databází.

API

Application Programming Interface – rozhraní pro programování aplikací. Jedná se o soubor procedur, funkcí či tříd určité knihovny, které muže programátor využívat. API určuje, jakým způsobem se funkce knihovny mají volat ze zdrojového kódu programu.

ARPA

Advanced Research Projects Agency – Agentura pro výzkum pokročilých projektů - Vznik agentury odsouhlasil roku 1958 preziden Spojených státu Dwight D. Eisenhower. Ukolem agentury bylo vytvořit a udržet technologický náskok ozbrojených sil Spojených států. Název této agentury byl později změněn, viz. DARPA.

ARPANET

Advanced Research Projects Agency Network – počítačová síť jež byla zárodkem sítě, kterou dnes chápeme pod názvem Internet. Spuštěna v roce 1969 pod hlavičkou organizace ARPA, později DARPA.

CSN

Computer Science Network v počítačová síť založená v roce 1981. Účelem sítě bylo propojení akademických a výzkumných pracovišť Spojených států, jež nemohly být, díky finančním či právním omezením, připojeny do sítě ARPANET.

CSS

Cascading Style Sheets – jazyk pro popis způsobu zobrazení stránek napsaných ve značkovacích jazycích. Je primárně navržen k tomu, aby umožnil oddělení obsahu dokumentu od vzhledové prezentace. Ve většině případů je používán pro popis vzhledu webových stránek psaných v HTML a XML, ale může být taktéž aplikován na jakýkoli dokument založený na standardu XML, včetně SVG a XUL.

DARPA

Defense Advanced Research Projects Agency – Agentura pro výzkum pokročilých obranných projektů - Jde o agenturu pod hlavičkou amerického ministerstva obrany, původním názvem ARPA (přejmenováno roku 1972). Primárním úkolem agentury je výzkum a vývoj nových technologií se zaměřením na jejich vojenské využití. Narozdíl od ostatních vyzkumných organizací Spojených států, je organizace DARPA nezávislá, disponuje vlastním rozpočtem (cca 2,8 miliardy USD) a zodpovídá se přímo vedení ministerstva obrany.

DHTML

Dynamic HTML – Pojmenován pro kombinaci technologií používaných ke tvorbě dynamických a interaktivních webových stránek. Těmito technologiemi se obvykle myslí HTML, JavaScript, CSS, DOM.

DOM

Document Object Model – objektově orientovaná reprezentace XML nebo HTML dokumentu. DOM je API umožňující přístup či modifikaci obsahu, struktury, nebo stylu dokumentu, či jeho částí.

ECMA

Ecma International – mezinárodní soukromá nezisková organizace, založená roku 1961 s cílem standardizace informačních a komunikačních systémů v Evropě.. Dříve také známá pod názvem European Computer Manufacturers Association. Její název byl v roce 1994 změněn, aby zdůraznil mezinárodní zaměření organizace.

ECMAScript

Skriptovací jazyk, standardizovaný asociací Ecma International ve specifikaci ECMA-262. Jazyk je především určen pro použití v internetových stránkách, kde jej většina uživatelů zná pod názvem JavaScript.

EDA

Event-driven architecture – Architektura řízená událostmi - jde o softwarový vzor popisující tvorbu, příjem a zpracování úkolů v reakci na události v aplikaci.

Framework

Softwarová struktura, která slouží jako podpora při programování, vývoji a organizaci jiných softwarových projektů. Cílem je převzetí typických problémů dané oblasti, čímž se usnadní vývoj tak, aby se návrháři a vývojáři mohli soustředit pouze na své zadání.

IP

Internet Protocol – základní protokol síťové vrstvy.

ISP

Internet service provider – Poskytovatel internetového připojení - firma nebo organizace zprostředkovávající přístup k Internetu.

JavaScript

multiplatformní, objektově orientovaný skriptovací jazyk. Zpravidla je používán jako interpretovaný programovací jazyk pro internetové stránky. Syntaxí patří do rodiny jazyků C/C++/Java. Slovo Java bylo použito v názvu pouze z marketingových důvodů a s programovacím jazykem Java jej vedle názvu spojuje pouze podobná syntaxe. Roku 1997 byl standardizován asociací ECMA.

JVM

Sada počítačových programů a datových struktur, která využívá modul virtuálního stroje ke spuštění dalších počítačových programů a skriptů vytvořených v jazyce Java.

JSON

JavaScript Object Notation – způsob zápisu dat nezávislý na počítačové platformě, určený pro přenos dat, která mohou být organizována v polích nebo objektech.

HTML

HyperText Markup Language – jde o značkovací jazyk pro internetové stránky. Poskytuje prostředky pro popis struktury textových informací v dokumentu. Je psán ve formě značek (tagů), jež jsou uzavřeny do úhlových závorek („<“ a „>“).

MILNET

Military Network – počítačová síť armády Spojených států, vznikla v roce 1983 oddělením od sítě ARPANET.

NASA

National Aeronautics and Space Administration – Národní úřad pro letectví a kosmonautiku - americká vládní agentura zodpovědná za americký kosmický program a všeobecný výzkum v oblasti letectví. Založena roku 1958 na základě zákona „National Aeronatics and Space Act“, jež podepsal prezident Eisenhower. Oproti ARPA, byl úkolem NASA vyzkum a vývoj převážně nevojenských prostředků.

NSF

National Science Foundation – Národní vědecká nadace - nezávislá vládní agentura Spojených států, zodpovědná za podporu základního vědeckého vyzkumu.

Pull

Způsob síťové komunikace, ve kterém je počáteční požadavek na data vyvolán na straně klienta a následně zodpovězen serverem. Jde o opak metody Push.

Push

Způsob síťové komunikace, kdy je požadavek vyvolán na akci vyvolán vzdáleným serverem a zodpovězen klientem. Opak Pull metody.

RIA

Rich Internet Application – internetové/intranetové aplikace s charakterem běžných desktopových aplikací.

SVG

Scalable Vector Graphics – značkovací jazyk a formát souboru, který popisuje dvojrozměrnou vektorovou grafiku pomocí XML.

SOA

Service Oriented Architecture – Architektura orientovaná na služby - sada principů a technologií, jež doporučuje skádat složité aplikace a jiné systémy ze skupin na sobě nezávislých komponent poskytujících služby.

SOAP

Simple Object Access Protocol – protokol pro výměnu zpráv založených na XML. Tvoří základní vrstvu komunikace mezi webovými službami a poskytuje prostředí pro tvorbu složitější komunikace.

TCP/IP

Transmission Control Procotol / Internet Protocol – sada protokolů pro komunikaci v počítačové síťi.

téma/skin/skinování

Pojem „skin“ je možné do českého jazyka přeložit jako kůže. Je to vlastní volitelný grafický vzhled aplikace nebo webové stránky. Skin má vliv pouze na vzhled, obsah zůstává nezměněn.

UI

User Interface – rozhraní, pomocí kterého uživatel komunikuje s aplikací a aplikace s uživatelem. Toto prostředí umožňuje uživateli manipulaci s aplikací (např. zadávání příkazů) a aplikaci na tyto příkazy reagovat.

URI

Uniform Resource Identifier – Jenodný Identifikátor Zdroje - Význam je shodný s URL, avšak URI je používáno s cílem jednoznačné identifikace informací či objektů na Internetu. Neposkytuje tak pouze informaci o jeho umístění, ale slouží také k jeho jednoznačné identifikaci.

URL

Uniform Resource Locator – Jenodný Ukazatel Zdrojů - jde o řetězec znaků s definovanou strukturou, který slouží k přesné specifikaci umístění zdrojů informací na Internetu.

XHTML

Extensible HyperText Markup Language – jde o značkovací jazyk, vycházející z HTML. Důraz je kladen na správnou (validní) syntaxi jazyka XML.

XML

eXtensible Markup Language – obecný značkovací jazyk, který byl vyvinut a standardizován konsorciem W3C. Umožňuje snadné vytváření konkrétních značkovacích jazyků pro různé účely a je vhodný pro komunikaci a výměnu dat mezi aplikacemi. Jazyk je určen pro popis struktury dokumentu z hlediska věcného obsahu jednotlivých částí, nezabývá se sám o sobě vzhledem dokumentu nebo jeho částí.

9 Literatura

[1] CIHELKA, Petr. *Rich Internet Applications*. Praha, 2009. Bakalářská práce. Česká zemědělská univerzita v Praze, Provozně ekonomická fakulta, Katedra informačních technologií. Vedoucí práce Ing. Petr Benda.

[2] ALAVI, Ali, et al. *HTML/DHTML web interface system and method*. U.S. Patent No 6,691,100, 2004.

[3] All Eclipse Foundation Members. Eclipse.org [online]. 2013 [cit. 14.3.2013]. Dostupné z: http://www.eclipse.org/membership/showAllMembers.php

[4] BERNERS-LEE, Tim, *The World Wide Web: Past, Present and Future,* August 1996, [cit. 14.3.2013]. Dostupné z: http://www.w3.org/People/Berners-Lee/1996/ppf.html

[5] CHAMBERS, Mike, DURA, Daniel, HOYT, Kevin. *Adobe integrated runtime (AIR) for JavaScript developers pocket guide*. O'Reilly, 2007.

[6] COOPER, Alan, CRONIN, Dave, REIMANN, Robert. *About face 3: the essentials of interaction design*. [3rd ed.], Completely rev. Indianapolis, IN: Wiley Pub., 2007, xxxv, ISBN 04-700-8411-1.

[7] COOPER, Alan. *About face 2.0: the essentials of interaction design*. Indianapolis, IN: Wiley, 2003, xxxiv, ISBN 07-645-2641-3.

[8] HAASE, Chet, GUY, Romain. *Filthy rich clients: developing animated and graphical effects for desktop Java applications*. Upper Saddle River: Prentice Hall, 2008, xxvii, the Java series. ISBN 978-0-13-241393-0.

[9] HICKSON, Ian, HYATT, David. *Html5. W3C Working Draft* WD-html5-20110525, May, 2011.

[10] KAZOUN, Chafic, Joey, LOTT. *Programming Flex 2: The Comprehensive Guide to Creating Rich Internet Applications with Adobe Flex*. Adobe Dev Library, 2008.

[11] LANINGHAM, Scott. *Interviews: Tim Berners-Lee*. DEVELOPERWORKS. DeveloperWorks [online]. 22.9.2006 [cit. 20.30.2013]. Dostupné z: http://www.ibm.com/developerworks/podcast/dwi/cm-int082206txt.html

[12] LASZLO SYSTEMS, Inc. OpenLaszlo [online]. 2007 [cit. 14.3.2013]. Dostupné z: http://www.openlaszlo.org/

[13] Leiner, B. M., et al. (1997). *The past and future history of the Internet*. Communications of the ACM, 40(2), pp. 102-108.

[14] Leiner, B. M., et al. (2009). *A brief history of the Internet*. Computer communication review, 39(5), 22.

[15] LEINSTER, Murray. *A logic named Joe.* New York: Pocket Books, 2005. ISBN 978-074-3499-101.

[16] MARÉCHAUX, Jean-Louis. *Combining service-oriented architecture and event-driven architecture using an enterprise service bus*. IBM Developer Works, 2006, 1269-1275.

[17] MARY JO, Foley. Will there be a Silverlight 6 (and does it matter)?. ZDNet [online]. Cambridge, MA: CNET Networks, Inc, 2011 [cit.14.3.2013]. Dostupné z: http://www.zdnet.com/blog/microsoft/will-there-be-a-silverlight--6-and-does-it-matter/11180

[18] MCCUNE, Doug; SUBRAMANIAM, Deepa. *Adobe Flex 3.0 for Dummies*. For Dummies, 2009.

[19] MORITZ, Florian. *Rich Internet Applications (RIA): A Convergence of User Interface Paradigms of Web and Desktop Exemplified by JavaFX*. Zweibrücken, Germany, 2008. Diploma thesis. University of Applied Science Kaiserslautern Fachhochschule Kaiserslautern Germany, Informatik und Mikrosystemtechnik. Vedoucí práce Prof. Hendrik Speck.

[20] MORRIS, Simon. A Rose By Any Other Name. Java.net [online]. 2007 [cit. 14.3.2013].Dostupné z: http://weblogs.java.net/blog/javakiddy/archive/2007/06/a_rose_by_any_o.html

[21] MORONEY, Laurence. *Introducing Microsoft® Silverlight (TM) 2*. Microsoft Press, 2008.

[22] MORVILLE, Peter. *User Experience Design*. [online]. 2004 [cit. 5.3.2013]. Dostupné z: http://semanticstudios.com/publications/semantics/000029.php

[23] MOSCHOVITIS, Christos J., POOLE, Hilary, SENFT, Theresa M. *History of the Internet: A Chronology, 1843 to the Present.* AB C-CLIO, Incorporated, 1999.

[24] MÜLLER-PROVE, Matthias. *Vision and reality of hypertext and graphical user interfaces*. Univ., Bibliothek des Fachbereichs Informatik, 2002.

[25] MÜLLER-PROVE, Matthias, LUDOLPH, Frank. *Dueling interaction models of personal-computing and web-computing*. In: Proceedings of the Workshop on Methodic and Didactic Challenges of the History of Informatics (MEDICHI). Österreichische Computer Gesellschaft. 2007.

[26] MULLET, Kevin, TEAM, Macromedia Experience Design. T*he essence of effective rich internet applications*. SF, CA: Macromedia White Paper, 2003, pp. 4-27.

[27] NIELSEN, J. *Usability engineering.* Vyd. 1. Boston: AP Professional, 1993. ISBN 01-251-8406-9.

[28] NIELSEN, Jakob. *Designing web usability*. Indianapolis: New Riders, 2000, xiii, 419 s. ISBN 15-620-5810-X.

[29] NORMA, Donald A. *The Life Cycle of a Technology: Why it is so difficult for large companies to innovate*. [online]. [cit. 5.3.2013]. Dostupné z: http://www.jnd.org/dn.mss/the_life_cycle_of_a_.html

[30] NORMAN, Donald A. *The invisible computer: why good products can fail, the personal computer is so complex, and information appliances are the solution*. Cambridge, Mass.: MIT Press, c1998, xii, 302 p. ISBN 02-621-4065-9.

[31] NOVÁK, Jiří. *Analýza RIA metod a technik*. Brno, 2009. Diplomová práce. Masarykova univerzita, Fakulta Informatiky. Vedoucí práce RNDr. Jan Pavlovič.

[32] O'REILLY, Tim, *Web 2.0 Compact Definitiv: Trying Again* [online] 12.10.2006 [cit. 26.4.2006] Dostupné z: http://radar.oreilly.com/archives/2006/12/web-20-compact.html

[33] POP, Paul. *Comparing Web Applications with Desktop Applications: An Empirical Study*, Dept. of Computer and Information Science, Linköping University, Sweden, 2000, http://www.ida.liu.se/labs/eslab/publications/pap/db/hci.pdf, Access: 1st October 2007

[34] POWELL, Thomas A. *HTML: the complete reference*. McGraw-Hill Professional, 1999.

[35] SAAD, David. *The Anatomy of Rich User Experience*. In: 2010. Dostupné z: http://www.slideshare.net/Luristic/the-anatomy-of-rich-user-experience

[36] SHNEIDERMAN, Ben. *1.1 direct manipulation: a step beyond programming languages*. Sparks of Innovation in Human-Computer Interaction, 1993, p.17.

[37] SHNEIDERMAN, Ben. *Direct manipulation for comprehensible, predictable and controllable user interfaces*. In: Proceedings of the 2nd international conference on Intelligent user interfaces. ACM, 1997. pp. 33-39.

[38] SHNEIDERMAN, Ben. *The future of interactive systems and the emergence of direct manipulation*. In: Behaviour & Information Technology, 1982, 1.3: pp. 237-256.

[39] TIDWELL, Jenifer. *Designing interfaces*. 1st ed. Sebastopol, CA: O'Reilly, 2006, xx, ISBN 05-960-0803-1.

[40] TRETOLA, Rich. *Beginning Adobe AIR: building applications for the Adobe integrated runtime*. Wrox, 2008.

[41] VAN HOOF, Jack. *How eda extends soa and why it is important.* 2006.

[42] VENNERS, Bill. Orthogonality and the DRY Principle: *A Conversation with Andy Hunt and Dave Thomas*, Part II. [online]. 10.3.2003. [cit. 23.3.2013]. Dostupné z: http://www.artima.com/intv/dry.html

[43] WINOKUR, Danny. Flash to Focus on PC Browsing and Mobile Apps;: Adobe to More Aggressively Contribute to HTML5. ADOBE INC. *Adobe's Conversations Blog* [online]. 2011 [cit. 20.3.2013]. Dostupné z: http://blogs.adobe.com/digitalmedia/2011/11/flash-to-focus-on-pc-browsing-and-mobile-apps-adobe-to-more-aggressively-contribute-to-html5/

10 Seznam tabulek

11 Seznam obrázků

12 Seznam příkladů