VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

MOZILLA JAKO VÝVOJOVÁ PLATFORMA

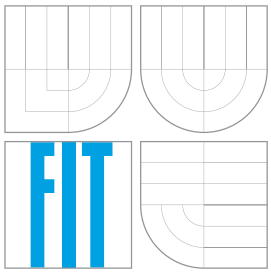
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN VÍDEŇSKÝ

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

MOZILLA JAKO VÝVOJOVÁ PLATFORMA

MOZILLA AS A DEVELOPMENT PLATFORM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN VÍDEŇSKÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAN SAMEK

BRNO 2008

Abstrakt

Tato práce se zabývá představením projektu Mozilla z pohledu jeho použití jako platformy pro vývoj aplikací. Je pomyslně rozdělena do čtyř částí. První tvoří teoretický úvod, který se zabývá architekturou, nejdůležitějšími technologiemi, nebo motivací proč použít platformu. Druhá část je koncipovaná jako první krůčky při tvorbě vlastního projektu. Třetí část je věnována ukázkové aplikaci Tester. Jedná se o projekt z e-learningového prostředí určený pro zjednodušení učení se zaměřením na slovní zásobu. Závěr práce se nese v duchu hodnocení platformy na základě praktických zkušeností.

Klíčová slova

ECMAScript, Firefox, Gecko, Mozilla, Platforma, RDF, XBL, XML, XPCOM, XPCONECT, XUL, XULRunner

Abstract

This thesis deals with introduction of Mozilla as a development platform. Thesis is divided into four parts. The first one consists of a theoretical introduction, which describes architecture, the most important technologies and motivation for usage of Mozilla as a development platform. The second part leads step by step threw making own project. The third part is dedicated to description of the example application Tester. Tester is an e-learning project designed for easier learning process with scope on vocabulary practise. In the conclusion of thesis, there is the evaluation of Mozilla platform based on practical experience.

Keywords

ECMAScript, Firefox, Gecko, Mozilla, Platform, RDF, XBL, XML, XPCOM, XPCONECT, XUL, XULRunner

Citace

Martin Vídeňský: Mozilla jako vývojová platforma, diplomová práce, Brno, FIT VUT v Brně, 2008

Mozilla jako vývojová platforma

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jana Samka.

. . . . . .
Martin Vídeňský
13. května 2008

© Martin Vídeňský, 2008.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

Kapitola 1

Úvod

Informační technologie se stále více uplatňují v širším spektru lidské činnosti. Softwarové projekty jsou čím dál tím rozsáhlejší, je vyžadována stále vyšší kvalita a bezpečnost. To se nutně projevuje na potřebě kvalifikovaných lidských zdrojů, kterých je omezené množství. Proto se zavádí do vývojového cyklu softwaru stále vyšší abstrakce. Ta dovoluje odpoutání od technických problémů směrem k vlastní logice projektů. Softwarový inženýři dostali CASE[1] nástroje pro zjednodušení návrhu, programátoři pak aplikační rámce nebo sofistikovanější vývojové platformy.

Vývojové platformy nebo aplikační rámce jsou souborem knihoven a nástrojů, které usnadňují programátorům práci a odvádí je od problémů, jako je multiplatformnost, hardware, grafické uživatelské prostředí a podobně. Lze tvrdit, že je to pomyslná vrstva mezi vlastní aplikací a operačním systémem. Mezi známé platformy patří například Java, Eclipse, .NET nebo Mozilla.

Poslední zmiňovanou platformou – Mozillou – se zabývá tato práce. Většina lidí si pod tímto označením představí pokročilý internetový prohlížeč nebo emailového klienta. Přesto to jsou pouze projekty postavené nad touto platformou.

Cílem této práce je ji prozkoumat: jaké poskytuje možnosti a na jakých technologiích je založena. A tyto poznatky zúročit v praktickém využití platformy.

Vlastní práce je pomyslně rozdělena do čtyř částí. První je ryze teoreticky orientovaná. Čtenáři poskytne, kromě dlouhé a zajímavé historie, rozbor architektury platformy, včetně představení nejdůležitějších technologii. Současně jsou na reálných aplikacích ukázané možnosti, které platforma poskytuje. Pro dokreslení slouží úvaha, jež rozvíjí toto téma pomocí příkladů možných aplikací. Pro uživatele, kteří se rozhodují, zda by jim platforma neposloužila pro jejich projekty, je přichystaný test, na jehož základě může padnout rozhodnutí.

Druhá část je koncipovaná jako první krůčky při tvorbě aplikace. Jsou v ní probrána témata, jak nainstalovat a používat platformu, vytvoření kostry projektu nebo distribuci výsledného produktu. Vynechány nejsou ani kapitoly, zabývající se například nástroji na podporu vývoje nebo radami pro vývojáře.

V předposlední části jsou probrané vědomosti zužitkované ve formě aplikace z e-learningového prostředí. Jedná se o program pro zjednodušení učení se zaměřením na slovní zásobu. Avšak návrh je poměrně flexibilní a aplikace může sloužit například pro vyplňování oblíbených kvízů.

Závěrečná část se věnuje hodnocení platformy na základě autorovy zkušenosti.

[1] Computer-Aided Software Engineering – počítačem podporované softwarové inženýrství

Kapitola 2

Základní pojmy

2.1 Platforma

Než se pustíme do vlastního výkladu o Mozille, tak si prvně definujeme vlastní pojem platforma. Slovník cizích slov definuje toto slovo jako základna, a to přesně vystihuje jeho podstatu, protože představuje pevné základy pro budování většího celku. Ve světě informačních technologií se výraz platforma používá ve více aspektech:

- Hardwarová platforma – pod tímto pojmem se myslí soustava komponent, která dohromady tvoří počítač. Je to například i86 od firmy Intel, nebo Sparc firmy SUN Microsystem.
- Software – který se dále dělí:
  - Operační systémy, představující základ pro spuštění programů. Pokud aplikace podporuje více OS, říká se jí multiplatformní.
  - Kolekce softwaru a nástrojů, s kterými může být aplikace vyvíjena a provozována [1].

Poslední definice představuje to, v jakém kontextu bude v této práci platforma chápaná, pokud nebude uvedeno jinak.

Softwarová platforma poskytuje pro vývojáře ulehčení práce při vývoji softwaru. Je toho docíleno pomocí vyšší abstrakce, díky které se vývojář nemusí zabývat problémy, které se ho netýkají, ale může se více zaměřit na vlastní logiku aplikace. Pro tyto účely poskytuje platforma velké množství knihoven (též nazývané rámce – viz další kapitola). Dalším charakteristickým rysem platformy je její schopnost provozovat (spouštět) aplikace v ní napsané. Tato vlastnost přináší uživateli jistotu, že spustí svůj produkt na libovolném zařízení (kombinaci hardwaru a softwaru), které je podporováno. Většina platforem používá pro vytváření aplikací jeden základní programovací jazyk. Avšak trendem v poslední době je nabídnout širší spektrum programovacích jazyků, které lze využít. Je to zejména způsobeno neochotou vývojářů učit se nové jazyky

Softwarových platforem existuje několik druhů, podle jejich komplexnosti a zaměření. Následující podkapitoly se zabývají nejdůležitějšími z nich.

2.1.1 Klasické platformy

Představují základní – nejobecnější skupinu platforem, tak jak byla definovaná. Následující výčet představuje dva zástupce z této skupiny:

JAVA – jedna z nejstarších platforem vyvinutých firmou Sun Microsystems. Poskytuje prostředky pro vývoj aplikací od vestavěných zařízení, přes klasické aplikace, až po aplikace webové. Javu tak můžeme najít například v mobilních telefonech, autech nebo bankovních informačních systémech. Svým rozsahem tak pokrývá skoro veškerá zákoutí informačních technologií. Platforma poskytuje pro vývoj aplikací stejnojmenný objektově orientovaný jazyk. Kromě něj lze použít i další jazyky, jako je například Groovy [2] nebo JPython [3], které však nejsou oficiálně podporovány.

.NET – je to odpověď na úspěch Javy od společnosti Microsoft. Neposkytuje takový rozsah podporovaných zařízení jako její vzor. Firma Microsoft podporuje pouze svůj operační systém a tím je omezen i hardware. Tento nedostatek řeší projekt Mono [4], jež představuje svobodnou implementaci této platformy. Hlavním zaměřením platformy jsou klasické a webové aplikace. Nejvýznamnějším programovacím jazykem platformy je C# (C-Sharp), který ale není jediným oficiálním jazykem. Dalšími jsou například C++, Visual Basic nebo Java#. Kromě nich existuje také velké množství neoficiálních jazyků.

2.1.2 Rich client platform

Pro tuto kategorii zatím neexistuje ustálený český výraz, ale někdy se uvádí překlad „platforma chytrého klienta“. Oproti klasickým platformám je úzce zaměřená na vytváření klasických desktopových aplikací. Převládající zaměření spočívá v budování tenkých klientů, pro které není vhodné použít standardní zázemí poskytované internetovým prohlížečem. Nejvýznamnější představitelé jsou:

Eclipse RCP – Za vznikem tohoto produktu stojí firma IBM. Dnes jej spravuje sdružení čítající přes 130 členů a je vyvíjen pod otevřenou licencí. Ke svému běhu používá obecnější platformu Java. Platforma vznikla ze stejnojmenného vývojového prostředí, které již od začátku bylo vyvíjeno značně modulárně. V tomto prostředí se také provádí vlastní vývoj aplikací [5, 6].

NetBeans RCP – o něco mladší, stejně zaměřený produkt jako Eclipse RCP. Vznikl podobnou cestou, jen s tím rozdílem, že je založen na konkurenčním vývojovém prostředí NetBeans [7].

Spring RCP – podprojekt známého webového J2EE rámce Spring. Ze zde přestavovaných platforem je nejmladší. Jeho hlavní předností by měla být vysoká konfigurovatelnost a standardní, jednoduše použitelné GUI [8].

2.1.3 Rich Internet application

Obdobně jako v předchozí kapitole i pro RIA neexistuje ustálený český překlad. Tuto skupinu nelze zcela jednoznačně označit jako platformu, přesto splňuje většinu jejích rysů a proto je zde uvedena. Funkčně je velmi podobná RCP s tím rozdílem, že přenáší aplikace do prostředí internetu. Aplikace jsou rozděleny na dvě části – klientskou a serverovou. Klientská část zpravidla běží na hostitelově počítači v rámci internetového prohlížeče. Na serverové části se pak ukládají data a stav. Celý koncept je poměrně nový a na svoje objevení zatím čeká. Hlavní reprezentanti jsou:

Adobe AIR – nejdůležitější představitel, který stál na počátku formování této kategorie. Základ na technologii Flash a Flex. Pro programátory je zajímavé, že používá převážně standardizované webové technologie (XML, CSS, ECMAScript) [9].

Microsoft Silverlight – založen na platformě .NET, čímž je omezena jeho rozšiřitelnost. Svými vlastnostmi je velmi podobny Adobe AIR. Oproti němu však Microsoft používá svoje proprietární technologie. Výhodou naopak je, že lze použít rozmanitější výběr programovacích jazyků [10].

JavaFX – opět podobná technologie založena na platformě Java. V době psaní práce se jednalo spíše o technologické představení, než o použitelný produkt. Přesto má Sun Microsystems s tímto produktem velké plány. Počítá se například s nasazením ve vestavěných zařízeních (mobilní telefony, PDA, ...) [11].

Do RIA se dále řadí i aplikace postavené na webové technologii AJAX, ty již jsou od myšlenky platformy, tak jak je v této práci chápaná, velmi vzdálené, a tak zde nebudou rozebírány.

2.2 Rámce

Rámec (anglicky Framework) je softwarová struktura, která slouží jako podpora při programování, vývoji a organizaci jiných softwarových projektů. Může obsahovat podpůrné programy, knihovnu API[1], návrhové vzory nebo doporučené postupy při vývoji.

Cílem rámce je převzetí typických problémů dané oblasti, čímž se usnadní vývoj tak, aby se návrháři a vývojáři mohli soustředit pouze na své zadání. Je tak velmi podobný vlastnostem platformy a často se taky tyto dva pojmy nesprávně zaměňují. Rámec se oproti platformě zaměřuje na konkrétní oblasti. Další z vlastností, která mu chybí, je běhové prostředí pro spuštění aplikace. Rámce lze tedy spíše chápat jako doplněk platformy, tj. její podmnožinu. Java například obsahuje rámce pro práci ze zvukem, grafikou nebo multimédii [12].

[1] Application Programming Interface – rozhraní pro programování aplikací

Kapitola 3

Historie Mozilly

Putování do historie musíme začít u samých kořenů „World Wide Webu“, a to u prohlížeče Mosaic. Ten naprogramovali v roce 1992 Marc Andreessen a Eric Bin ve firmě NCSA. Tento prohlížeč je často neprávem považován za první webový prohlížeč. Prvním byl Nexus. Napsal ho samotný zakladatel hypertextu sir Tim Berners-Lee. Nexus však byl omezen pouze na platformu NEXTStep [13].

V roce 1994 Marc Andreessen a Eric Bin založili firmu Mosaic Communications Corporation s cílem vytvořit nový prohlížeč. Už v říjnu vyšla první verze pod názvem Mosaic Netscape. Kód tohoto prohlížeče byl postavený na původním Mosaicu, přidal však podporu pro nové prvky jazyka HTML. Poněvadž název kolidoval s ochranou značkou NCSA, došlo k přejmenování produktu na Netscape Navigator a firmy na Netscape Communications Company [14].

Hvězda Netscape velmi strmě stoupala i přes to, že se jednalo o placený produkt. V roce 1995 vstoupila firma úspěšně na burzu a nic nenasvědčovalo rychlému pádu, který měl zanedlouho nastat. O ten se postarala firma Microsoft se svým produktem Internet Explorer [15], která začala chápat, že budoucnost je v internetu. Již ve stejném roce vyšla verze 1.0, kterou vzápětí vystřídala verze 2.0.

Následující rok vyšly „trojkové“ verze obou prohlížečů, které již měly srovnatelné vlastnosti, a tak začala bitva o zákazníky (pro toto období se ujal název „Válka prohlížečů“). Pouhý prohlížeč internetových stránek už nebyl dostačující a tak obě společnosti přidaly další aplikace, jako například emailového klienta. V roce 1997 vyšla čtvrtá generace obou prohlížečů. Netscape změnil název na Netscape Communicator, aby zvýraznil rozšíření balíku o další aplikace. Technologicky vyspělejší Internet Explorer se stal součástí operačního systému Windows 98 [16], díky čemuž se začal rozšiřovat. Tímto krokem pomyslně láme Netscape krk. Proto se Netscape Communications Company odhodlalo k historickému kroku a počátkem roku 1998 vydali veškeré kódy jako opensource, čímž vznikl projekt Mozilla[1]. Pro správu kódů a pokračování vývoje byla založena skupina Mozilla Organization. Koncem roku 1998 koupil firmu Netscape komunikační gigant AOL. Tímto krokem prakticky skončila historie tohoto prohlížeče jako samostatného celku, další verze jsou již založeny na kódech Mozilly. Připravovaná verze 5.0 nikdy nevyšla.

Po založení projektu Mozilla se začalo postupně pracovat na první verzi prohlížeče. Vývojáři se rozhodli pro přepis drtivé většiny kódu. Dostali tak šanci vytvořit čistší návrh,

[1] Jméno Mozilla vzniklo spojením slov „Mozaic Killer“. Objevuje se již v projektu Netscape jako jeho interní kódové pojmenování. Název Mozilla se také objevuje v řetězci „USER AGENT“, kterým se prohlížeče identifikuji v protokolu http, a to nejen u Netscape ale i v Exploreru. Tato identifikace zůstala v prohlížečích dodnes [17].

kde hlavním požadavkem byla multiplatformnost. Díky tomu přestala být Mozilla pouze aplikací a stala se vývojovou platformou. První stabilní verze nové Mozilly vyšla koncem roku 2000 pod verzí 0.6. Na této verzi byl také založen Netscape 6.0. Tato verze však byla velmi chybová a u veřejnosti nevzbudila velkou odezvu. Po opravě velké řady chyb vyšla v červnu, roku 2002 Mozilla verze 1.0 a na ní založen Netscape 7.0. U uživatelů měl ale podobnou odezvu jako předešlá verze. Používala se převážně jen u operačních systému UNIXového typu, na majoritním OS MS Windows vládl neohroženě Internet Explorer 6.0.

V červnu roku 2003 přišel další důležitý zvrat: společnost AOL založila neziskovou organizaci Mozilla Foundation. Té svěřila značku Mozilla a vývoj dalších verzí, ponechala si však značku Netscape. Do začátku, pro rozběhnutí, ještě věnovala hardware, tři placené zaměstnance po dobu tří měsíců a 2 miliony dolarů na dva roky činnosti.

Nyní se vraťme trochu v čase, do září roku 2002, kdy vznikl projekt Pheonix. Jak bylo dříve naznačeno, Mozilla neobsahovala pouze prohlížeč, ale také emailového klienta a další aplikace, jako například editor stránek a IRC chat, což tvořilo poměrně velký balík softwaru. Projekt Pheonix měl za úkol vytvořit jednoduchý prohlížeč stránek, bez zbytečných vlastností navíc. Další vlastnosti se mohly přidávat pomocí rozšiřujících modulů, což přetrvává dodnes. Každý uživatel si tak může vytvořit prohlížeč podle svých požadavků. Postupem času vycházely další verze Mozilly a na jejím kódu odvozený Netscape a Pheonix, avšak stále bez výraznějšího zájmu u široké veřejnosti. V první polovině roku 2003 došlo k přejmenování Pheonixu na Firebird, protože název kolidoval s ochrannou známkou známého výrobce BIOSu. Tento název však nebyl zvolen nejšťastněji, jmenovala se tak již opensource databáze. Z tohoto důvodu došlo ještě k jednomu přejmenování, a to na Mozilla Firefox – starší pojmenování pro pandu červenou.

Mozilla Firefox verze 1.0 vyšla 9. listopadu roku 2004. Před vlastním vydáním se uspořádala nevídaná akce – sbírka na podporu propagace. Této akce se zúčastnilo přes 10 000 lidí, kteří podpořili akci 250 tisíci dolary. Z těchto peněz se zaplatila celostránková reklama v deníku New York Times. Reklama měla podobu loga Firefoxu, které bylo vytvořeno ze jmen lidí, kteří přispěli libovolnou částkou [18, 19]. Je škoda, že Firefoxu nezůstalo jméno Pheonix, protože jak tento bájný tvor, tak i Firefox zažil znovuzrození. Je těžké posoudit, co zapříčinilo vysokou oblibu tohoto prohlížeče u uživatelů, jestli to byly záložky, blokování vyskakujících oken nebo zásuvné moduly. Dalším faktem mohl být nezájem firmy Microsoft o rozšiřování nových funkcí ve značně zastaralém Internet Exploreru (poslední verze vyšla v roce 2001). V každém případě podíl Firefoxu na trhu s prohlížeči neustále stoupá již několik let. V době psaní této práce byl podíl na trhu kolem 30% [20] (v naší republice dokonce 36%).

Tento úspěch zapřičinil ukončení vývoje balíku Mozilla. Mozilla Foundation ze začala plně věnovat Firefoxu a Thunderbirdu (emailový klient založený na stejných principech jako Firefox). Mozilla dostala nový název SeeMonkey, o který se stará komunita uživatelů. Na kódu Firefoxu je založená i osmá verze Netscape [21].

Na sklonku roku 2007 byla vydaná devátá verze Netscape, založená na Firefoxu 2.0. Funkcionálně se však příliš neliší, hlavní rozdíl spočívá ve vzhledu. Ani tato verze nevzbudila u uživatelů zájem a v lednu následujícího roku firma AOL rozhodla o úplném ukončení projektu Netscape[2]. Stejnojmennou ochranou známku si však nadále ponechává.

Odkaz Mozilly tak dnes spočívá pouze na bedrech prohlížeče Firefox.

[2]Na stránkách archive.netscape.com lze stáhnout všechny verze Netscape, na kterých se podílela společnost AOL.

Kapitola 4

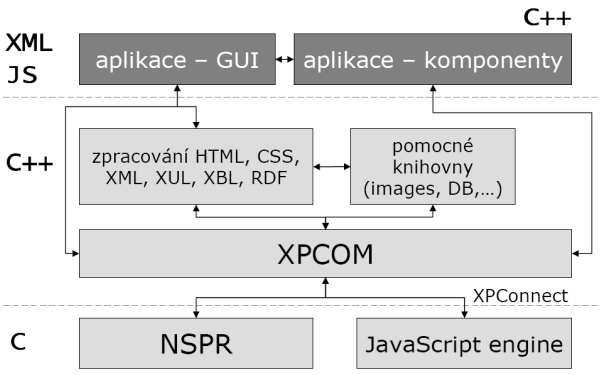
Mozilla se představuje

Tato kapitola slouží k prvotnímu seznámení s platformou. Je zde představena architektura platformy. Výklad pokračuje podkapitolou o aplikacích, které jsou na ní postavené. Zbývající část je věnována výčtem kladných a záporných vlastností se zamyšlením nad nejlepším využitím platformy.

4.1 Architektura

U Mozilly myšlenka platformy nevznikla na samém počátku, ale postupně v průběhu času. Programátoři se totiž při vývoji prohlížeče neustále potýkali s problémy spojenými s přenositelností přes různé hardwarovými a softwarovými platformami. Proto vznikla myšlenka rozdělit aplikaci na několik úrovní podle závislosti na hostitelském systému.

Rozdělení bylo provedeno do tří úrovní, které jsou znázorněny na schématu 4.1. Nejnižší vrstva se zaměřuje na vytvoření mostu mezi vyššími úrovněmi a hostitelským prostředím. Je psaná v jazyce C kvůli jeho dobré přenositelnosti. Druhou úroveň tvoří jádro platformy. Jeho převážná část je implementována v jazyce C++ (detailnější popis jednotlivých technologii je obsahem následující kapitoly). Poslední, nejvyšší stupeň tvoří samotná aplikace a její podpůrné knihovny. Zde je nejčastějším jazykem Javascript [22].



Obrázek 4.1: Architektura platformy [22].

O této architektuře se dá říci, že je již platformou, i když ne zcela ideální. Vlastní vývoj byl podřízen potřebám prohlížeče. Další nevýhodou je nemožnost společně sdílet instanci

platformy mezi aplikacemi.

První projekt, který tuto situaci řešil, byl Gecko Runtime Environment (česky běhové prostředí Gecka), zkráceně GRE. Pokoušel se řešit i problém se sdílením instance. Ale kvůli nevhodnému návrhu byla tato vlastnost prakticky funkční pouze na systému Windows. I zde se ale potykal s celou řadou problémů.

V březnu roku 2005 přišel Benjamin Smedberg s další myšlenkou řešení [23]. A dal tak podklad pro vznik XULRunneru. Z pohledu zde prezentované architektury je tvořen spodní a střední vrstvou. Celý projekt je koncipován pro vývoj různorodých aplikací, už se tedy jedná o plnohodnotnou platformu. Bohužel je XULRunner stále ve vývoji a termín vydání stabilní verze se neustále oddaluje. I přes tento fakt už existuje poměrně velký zástup aplikací, který tuto platformu využívají.

První vlaštovkou ze stáje Mozilla Foundation, která bude nad XULRunnerem postavena, bude prohlížeč Firefox 3. Spolu s ním by měla vyjít i stabilní verze platformy, ale v době psaní práce už bylo jisté, že se to nestihne. Firefox tak bude postaven na privátní verzi a oficiální vyjde až o něco později. V této práci se budu nadále převážně zabývat XULRunnerem [24, 25].

4.2 Zařazení platformy

První kapitola této práce se zabývala klasifikací platforem a jejich členěním. XULRunner lze zařadit do kategorie RCP. U některých zdrojů se lze setkat se zařazením i do kategorii RIA. S čímž se dá souhlasit jen částečně, protože jak je v následující podkapitole ukázáno, nad platformou lze vybudovat podstatně složitější aplikace. Přičemž jejich hlavní funkcionalita není nutně spojená s prostředím internetu.

4.3 Ukázka aplikací

Než přejdeme k vlastnostem platformy, tak pro motivaci, proč si zvolit tuto platformu, je zde přehled několika aplikací. Bezesporu nejznámější jsou internetový prohlížeč Firefox a poštovní program Thunderbird. Proto by byl jejich bližší popis neúčelný. Přehled je zaměřený na méně známé projekty, které i svým zaměřením vybočují z běžného předpokladu, že se platforma hodí pouze na internetové aplikace.

SongBird

Adresa: `http://www.songbirdnest.com`.
SongBird je multimediální přehrávač podobný iTunes. Dokáže, podobně jako jeho kolega, elegantně spravovat multimediální data v počítači pomocí knihovny. Jeho hlavními rysy jsou:

- Multiplatformnost.
- Široká podpora formátů – záleží na hostitelském OS, například na Linuxu nejsou podporovány skladby chráněné pomocí DRM[1].
- Pokročilé operace s knihovnou.
- Opensource.

[1] DRM – Digital Rights Management, lze přeložit jako Správa digitálních práv, obecně jsou touto zkratkou označována multimediální díla která jsou chráněná proti zneužití [26].

- Změna vzhledu.
- Rozšiřitelnost pomocí modulů.
- Plnohodnotný internetový prohlížeč.

Obrázek 4.2: Obrazovka z aplikace SongBird.

Miro

Adresa: http://www.getmiro.com.
Multimediální přehrávač, který je oproti SongBirdu zaměřený na video. Vlastnosti:

- Přehrávání široké palety videoformátů.
- Sledování internetové televize – standardně obsahuje tisíce kanálů.
- Stahování pořadu z komunitních serverů, jako jsou například YouTube nebo Google Video.
- Stahování přes výměnnou síť Bittorrent.
- Podpora HD videa.
- Nepodporuje rozšiřování pomocí modulů.
- Multiplatformnost.
- Opensource.

JOOST

Adresa: http://www.joost.com.
Joost je další internetová televize. Oproti projektu Miro se jedná opravdu o internetovou televizi. Nesází tak na videoklipy z komunitních serverů, ale na vlastní síť, která je postavena na myšlence p2p[2]. Výhodou tohoto přístupu je podstatně lepší kvalita obrazu a pořadu. Nevýhodou je malý počet kanálů, které jsou zatím pouze v anglickém jazyce.

Zakladatelé projektu jsou Niklas Zennström a Janus Friis – autoři výměnné sítě Kazaa a internetového telefonování Skype. Oba projekty ve své době způsobily revoluci a Joost má tento potenciál také. Jestli se tak stane, ukáže až čas.

[2]peer to peer označuje se tak architektura počítačové sítě, kde komunikují klienti mezi sebou bez centrálního serveru [27].

Obrázek 4.3: Obrazovka z aplikace Miro.

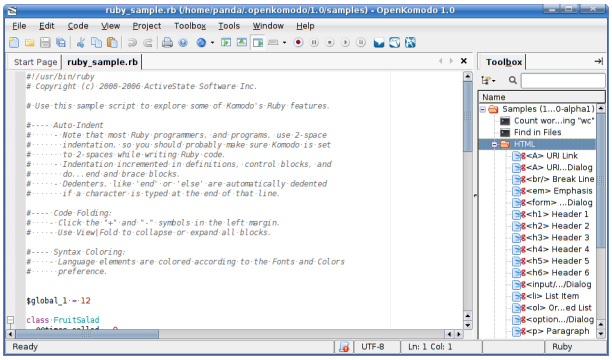
Obrázek 4.4: Obrazovka z aplikace Joost.

Komodo IDE

Adresa: http://www.activestate.com.
Komodo je integrované vývojové prostředí určené pro vývoj aplikací v dynamických jazycích, jako je například PHP nebo Python. Původně bylo vyvíjeno jako komerční produkt, ale později byla uvolněna komunitní verze, která však neobsahuje všechny nástroje z placených verzí. Kvalita IDE je na velmi vysoké úrovni a ničím nezaostává za dnešním standardem. Díky podpoře zásuvných modulů a otevřeného API lze možnosti editorů lehce rozšiřovat. Základní vlastnosti editoru:

- Zvýraznění syntaxe.
- Automatické doplňování kódu.
- Kontrola syntaxe.
- Správa projektů.
- Podpora různých jazykových kódování.

- Podpora klávesových zkratek – podporuje zkratky z Emacs a Vi.
- Velké množství podporovaných jazyků.
- Další vlastnosti lze najít na adrese stránek.

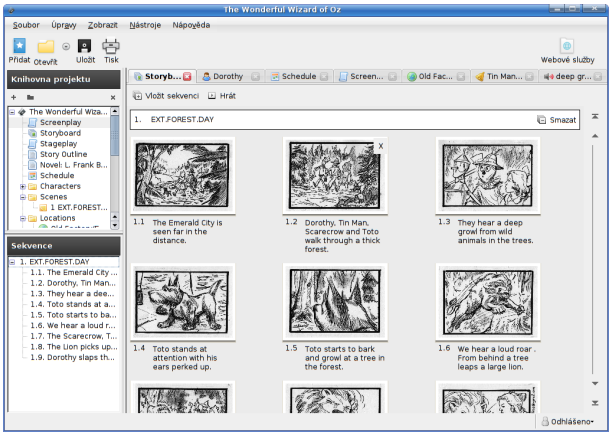


Obrázek 4.5: Obrazovka z aplikace Komodo IDE.

Celtx

Adresa: http://www.celtx.com.
Celtx je opensource aplikace pro podporu psaní scénářů a pre-produkce filmu, divadla a dalších medií [28]. Základní vlastnosti:

- Textový editor s kontrolou pravopisu.
- Editor postav, herců.
- Editor scén.
- Podpora pro psaní scénáře s podporou vkládání multimediálního obsahu.
- Plánování – časový harmonogram.



Obrázek 4.6: Obrazovka z aplikace Celtx.

Celtixem přehled aplikací končí. Další významné aplikace lze najít v přehledu na adrese: http://developer.mozilla.org/en/docs/XULRunner_Hall_of_Fame. Za povšimnutí stojí, že na seznamu lze najít i projekty známých firem jako je Google, Yahoo! nebo IBM.

4.4 Vlastnosti platformy

Předchozí kapitola o aplikacích měla čtenáři poskytnout základní představu, co vše lze s použitím platformy vytvořit. Tato kapitola se podrobněji zabývá jejími vlastnostmi, tj. jejími klady a zápory tak, aby se čtenář mohl lépe rozhodnout, zda je tato platforma pro jeho účely vhodná. Pro jednodušší rozhodnutí je na konci kapitoly test. Po jeho vyplnění, podle zamýšleného projektu, dostane čtenář hodnocení vhodnosti použití.

4.4.1 Klady

Výhody jsou shrnuty v následujících bodech:

Použití známých technologii – jedna z největších výhod po vývojáře je paralela s webovými technologiemi. Lze tvrdit, že vývojáři, kterému nejsou cizí pojmi XML, HTML, CSS nebo Javascript a orientuje se v problematice Webu 2.0, nebude činit vývoj problémy. S tím je spojeno i poměrně jednoduché vytváření aplikací a rychlá reakce na změnu zadání.

Multiplatformnost – pro většinu aplikací je požadováno, aby bylo možné jejich fungování na více systémech, nebo aby toho bylo možné v budoucnu jednoduše dosáhnout. Mozilla tuto podmínku bezproblémově splňuje. Projekty v ní napsané lze provozovat na všech systémech, pro které existuje běhové prostředí. Podpora třech nejvýznamnějších zástupců – MS Windows, Apple MacOS X a Linux – je samozřejmostí.

Lokalizovatelnost – programy je velmi jednoduché lokalizovat do libovolného jazyka. Podporované jsou i jazykové zvyklosti jednotlivých zemí.

Práce se sítí – podpora soketů, kanálů, stahování souborů. Nechybí ani webové služby, které jsou z pohledu moderních aplikací nejdůležitější, protože umožňují propojení s mnoha nabízenými službami na internetu. Implementované protokoly jsou SOAP a XML-RFC.

Práce s XML – obsahuje funkce pro práci s tímto formátem: DOM, SAX nebo dotazovací jazyk XPath.

Korektor pravopisu – do aplikací se dá jednoduše přidat kontrola překlepů v textu. Algoritmus používá ke kontrole slovníky. Pro použití je připraveno přes 50 jazyků a jejich dialektů.

Doplňky – do aplikace lze zabudovat mechanizmy pro správu doplňků. Ty dokáží libovolně rozšiřovat možnosti aplikace.

Změna vzhledu – pro aplikace je jednoduché vytvořit několik vzhledů a nechat výběr na uživateli. Ten má také možnost upravit nebo vytvořit vlastní styl.

Otevřenost vývoje – všechny zdrojové kódy jsou vyvíjeny otevřeně, kdokoliv tak může opravit chyby, rozšířit vlastnosti pro svoje potřeby a podobně.

4.4.2 Zápory

Rozsáhlé aplikace – platforma je nevhodná pro velké aplikace. U těch začne vadit používání Javascriptu – problém s výkonem. Ten lze částečně řešit přepsáním klíčových částí pomocí jazyka C++ do komponent XPCOM.

Vývojářské nástroje – pro vývoj neexistují sofistikované vývojové nástroje. Zejména chybí vizuální návrhář uživatelského rozhraní.

Dokumentace – Největším zdrojem informací je portál organizace Mozilly. Některé méně používané oblasti nejsou vůbec popsané a jediná možnost, jak se s nimi seznámit, je studium zdrojových kódů. Poměrně dobrou vstupní branou jsou dvě knížky, které byly o vývoji na platformě napsané. Jsou však poměrně staršího data.

Absence silné organizace – za vývojem Mozilly nestojí žádná velká organizace. To u větších projektů významně ovlivňuje výběr cílové platformy, protože vedení potřebuje jistotu, že vývoj platformy nebude nenadále ukončen.

Otevřenost vývoje – veškeré zdrojové kódy napsané v Javascriptu může kdokoliv vidět a nelze tomu úspěšně zabránit. Není možné tak ochránit svoje duševní vlastnictví.

4.4.3 Test

Pro zjednodušení rozhodnutí, zda použit Mozillu pro vývoj své aplikace, je zde test. Je převzatý z prezentace Davida Majli a upravený [22]. V testu projděte všechny otázky a v případě kladné odpovědi si připočtěte body v závorce.

- Potřebujete multiplatformnost? (+1 bod)
- Potřebujete pracovat s HTML či XML? (+1 bod)
- Potřebujete pracovat se sítí? (+1 bod)
- Potřebujete lokalizovatelnost? (+1 bod)
- Chcete umožnit rozšiřování aplikace pomocí doplňků? (+1 bod)
- Chcete rychle vytvářet a očekáváte časté změny v zadání? (+1 bod)
- Máte zkušenosti ve vývoji webových aplikací? (+ 0.5 bodu)
- Výsledná aplikace bude opensource? (+ 0.5 bodů)
- Aplikace je velmi rozsáhlá? (-1 bod)
- Potřebujete náročnou grafiku? (-1 bod)
- Potřebujete hrubý výkon? (-1 bod)
- K vývoji upřednostňujete sofistikované vývojové nástroje? (-1 bod)
- Potřebujete za vývojem platformy silnou organizaci? (-1 bod)
- Potřebujete nativní vzhled (GUI)? (- 0.5 bodu)

Vyhodnocení testu

Více než 5 bodů – o použití platformy byste měli vážně uvažovat. Zcela jistě se na vaše požadavky hodí.

3 – 5 bodů – zamyslete se, jestli nepřevažují výhody platformy nad jejími negativy. Analyzujte klíčové vlastnosti vašeho projektu a znovu promyslete, jestli by vám platforma s jejich implementací nepomohla.

Méně než 3 – platforma pro vás není vhodná. Poohlédněte se po jiném řešení.

4.4.4 Příklady použití

Z předchozích informací vyplývá, že Mozilla se nejlépe hodí na menší až střední projekty, které mají návaznost na služby spojené se sítí internet. V této podkapitole bych chtěl popsat pomocí pár modelových příkladů ideální použití platformy.

V roce 2007 nastal obrovský boom ve stěhování aplikací do prostředí webového prostředí, zapřičiněný technologii AJAX. Příkladem může být jedna z prvních služeb – Google Gmail. Tato aplikace se snaží svým komfortem plně nahradit klasického desktopového emailového klienta. Výhodou tohoto řešení je dostupnost z jakéhokoliv počítače připojeného k celosvětové síti. Nevýhodou pak, že přes poměrně propracované uživatelské prostředí nedosahuje komfort svých vzorů. Tento koncept také nevyhovuje některým uživatelům, kteří preferují „tlusté klienty“. Tento problém by šel poměrně elegantně vyřešit aplikaci, která by tvořila most mezi pojetím klasické klientské a internetové aplikace. Tento hybrid by obsahoval klasické menu, stavový řádek a jiné podobné uživatelské prvky, které by byly napojené na již zmíněnou internetovou aplikaci. Jednalo by se tedy o specializovaný prohlížeč určený na jednu určitou službu s přidanou hodnotou v podobě vylepšeného uživatelského komfortu.

Další možnou aplikaci, je vytvoření takzvaného „off-line klienta“. Tento typ aplikace funguje tak, že si uživatel na svém desktopu připraví data, která se následně odešlou do nějaké specifické aplikace na internetu. Jako přenosový protokol se ve většině případů používá nějaká forma webových služeb. Praktickým příkladem může být klient pro fotosběrnu. Uživatel si na domácím počítači připraví fotografie, které chce dát vytisknout. U každé si vybere formát, velikost, druh papíru, popřípadě provede poslední grafické úpravy fotky. Takto vybrané fotografie se pak odešlou najednou do fotosběrny. Výhoda řešení oproti internetové aplikaci je například v technickém omezení přenosu souboru, které neumožňuje hromadné nahrání fotografii.

Podobných aplikací lze vymyslet celou řadu, záleží jen na představivosti. Z možných aplikací ještě zmíním například editor pro psaní článku pro blogovací systém nebo klienta na posílaní SMS.

Některé zde představované nápady už mají své první vlaštovky v podobě existujících projektů. V případě prvního nápadu – specializovaného prohlížeče – se jedná o projekt Flock [29], který se specializuje na sociální sítě (např. Facebook nebo YouTube). O druhý nápad se postarala firma Yahoo!, která vytvořila aplikaci pro nahrávání fotek do své internetové galerie Flicker.

Kapitola 5

Klíčové technologie platformy

V této ryze teoretické kapitole jsou postupně představené nejdůležitější technologie spjaté s platformou.

5.1 NSPR

Netscape Portable Runtime je nejnižší vrstva platformy, napsaná v jazyce C. Jejím hlavním smyslem je vytvořit API, které je nezávislé na hostitelském systému. Použitím knihovny lze pak aplikaci přeložit na všech OS a hardwarových platformách, které NSPR podporuje. Knihovna je vyvíjena nezávisle na projektu Mozilla a lze ji začlenit do libovolného projektu. Tomu nebrání ani licence – kód je šířen pod: MPL, GPL, LGPL [30].

Hlavní součásti knihovny:

- definice datových typů – nezávislé na použitém HW a SW,
- vlákna,
- synchronizace,
- vstupně/výstupní operace,
- základní síťové funkce,
- funkce pro práci s časem,
- správa paměti,
- dynamická správa knihoven – načítání/uvolňování.

5.2 GECKO

Srdce platformy Mozilla. Jeho hlavní náplní je rendrováni stránek. Kromě toho se stará o další funkce – například síťové. Jádro je napsáno jako komponenta XPCOM v jazyce C++. Dá se tak použít jako samostatná část. Některé projekty toho využívají – například JREX [31], což je UI (SWING) komponenta pro platformu Java. Gecko umí rendrovat a pracovat s následujícími technologiemi:

- HTML 4.01, XHTML 1.1,

- CSS 1, 2.1, (částečně 3) a proprietární rozšíření,
- MathML,
- SVG 1.1, png, jpeg, gif,
- XSLT, XPath, jednoduchý XLinks, XPointer,
- XUL,
- Webové služby (XML-RPC, SOAP),
- DOM 1 a 2,
- ECMAScript/JavaScript,
- RDF,
- HTTP a SSL.

Původně bylo jádro navrhováno jako jednoduchá a kompaktní část. Postupem času ale narůstalo a stal se z něho moloch, který je velmi často odbornou veřejností kritizován. Přesto patří k nejlepším jádrům s nejširší podporou standardů [32].

5.3 JavaScript (ECMAScript)

JavaScript (dále pod zkratkou JS) je objektově orientovaný, beztřídní, skriptovací jazyk. Byl vymyšlen Brendnem Erichem ve společnosti Netscape. Slovo Java je ve jménu produktu pouze z marketingových důvodů, protože v době, kdy byl JS vymyšlen, byla Java velmi populární a předpovídala se jí skvělá budoucnost. Syntakticky je JS tvořen mixem několika jazyků.

Jazyk byl vytvořen za účelem přidání dynamických efektů do webových stránek, přičemž k interpretaci kódu dochází až u klienta. V průběhu let se význam JS měnil, první vlnu rozšíření zažil ve „válce prohlížečů“, bohužel se v ní jednotlivé firmy rozešly v implementaci. Klíčové vlastnosti jazyka, jako jsou například události, definuje každý prohlížeč jinak. Tento rozkol zatím stále přetrvává. Kvůli tomu také nebyl JS u programátorů moc oblíbený, až do chvíle, kdy se začal používat na manipulaci s DOM a na asynchronní načítáni dat – základy technologie AJAX a nového pojetí internetových stránek, které dostalo jméno Web 2.0.

Pro odstranění různorodé implementace JS vznikl standart pod hlavičkou mezinárodního standardizačního sdružení ECMA. Standard dostal název ECMAScript a Mozilla ho splňuje. V příloze B je seznam nejdůležitějších nových podporovaných vlastností.

5.4 XPCOM

The Cross Platform Component Object Module (multiplatformní komponentový model) je aplikační rámec, který dovoluje programátorům rozdělit projekt do malých kousků – komponent [33].

Účel modulů je doplnit chybějící funkcionalitu nebo vytvoření funkcí pro zjednodušení práce. Příkladem pro první případ může být ovladač měřící sondy na počasí, který bude

dodávat měřené údaje. Rozhraní takovéhoto modulu pak bude mít jednoduché metody typu zjisti teplotu, sílu větru a podobně.

Koncept XPCOM je založen na podobné technologii jako je COM od firmy Microsoft. Obdobně jako COM definuje rozhraní komponenty v jazyce XPIDL (obdoba jazyka IDL). Komponenty pak toto rozhraní implementují.

Nejčastější implementace rozhraní je v jazyce C++. Součástí jazyka XPIDL jsou nástroje, které dokáží z definovaného rozhraní vytvořit hlavičkový soubor a kostru zdrojového kódu. Jinou možností je implementovat rozhraní v jazyce JavaScript, pomocí technologie XPConnect (viz dále). Tvorba komponent se ale neomezuje pouze na tyto jazyky, ale lze teoreticky použít libovolný jazyk, ve kterém bude vytvořena vazba na XPCOM rozhraní. V současné době existují například projekty JavaXPCOM [34] nebo plXPCOM [35]. Nejzajímavějším konektorem je však PyXPCOM [36], který se má stát oficiální součástí XULRunneru verze 1.9.

Volba jazyka pro tvorbu komponenty má dopad na výslednou multiplatformnost a rychlost. Platí zde přímá úměra – u vysokoúrovňových jazyků je jednodušší dodržet multiplatformnost a naopak. Při použití nízkoúrovňového jazyka se na oplátku dosáhne vyšší rychlost. Přepsáním klíčových komponent do jazyka C++ lze škálovat výkon aplikace.

XPCOM, podobně jako jeho vzor, definuje předka všech rozhraní *nsISupports* (v COM se nazývá *IUnknown*). Toto rozhraní tak musí všechna rozhraní implementovat. Definice rozhraní je následující:

```
nsISupports {
    nsrefcnt AddRef();
    void QueryInterface(in nsIIDRef aIID, out nsQIResult aResult);
    nsrefcnt Release();
}
```

Metody *AddRef* a *Relase* slouží ke zvyšování a snižování počítadla referencí (XPCOM komponenty jsou založeny na této jednoduché správě paměti [37]). Metoda *QueryInterface* slouží ke zjištění, zda komponenta poskytuje požadované rozhraní. Pokud ano, tak na něj vrátí ukazatel. Komponentový model neposkytuje pouze podporu pro vývoj komponent, ale také potřebnou funkcionalitu, jako je například:

- správa komponent,
- abstrakce souboru,
- přeposílání zpráv mezi objekty,
- správu paměti.

5.4.1 XPConnect

Cross-platform Connect znamená česky multiplatformní propojení. Tvoří transparentní most mezi nízkoúrovňovými XPCOM objekty a vysokoúrovňovým JS. Dovoluje použít XPCOM komponentu z JS kódu a ovlivnit JS objekt z vnitřku XPCOM komponenty [38, 32]. Zejména má tyto úkoly:

- Zpřístupnění hierarchie objektu Mozilly v JS.
- Ošetření volání metod implementovaných v různých jazycích s ohledem na volací konvence a typy parametru.

- Umožnění tvorby nových komponent v JS, včetně odvození z již existujících.

Nejčastějším použitím je volání XPCOM objektů z JS kódu.

5.5 RDF

Je rámec na popis zdrojů (**R**esource **D**escription **F**ramework). Poskytuje obecný mechanizmus pro zápis metadat. Je součástí standardů vydaných konsorciem W3C [39, 40].

Rámec RDF poskytuje jednoduchý model pro popis zdrojů, který není závislý na konkrétní implementaci. Datový model RDF, zjednodušeně řečeno, umožní specifikovat trojici: zdroj, vlastnost a hodnota vlastnosti s významem: „daný *zdroj* má danou *hodnotu* dané *vlastnosti*“ [41].

RDF poskytuje následující:windows manager interface

- Možnost zpracování a výměny metadat mezi různými aplikacemi (interoperabilitu).
- Strojům srozumitelnou sémantiku metadat.
- Větší přesnost ve vyhledávání zdrojů než full-textové vyhledávání.

Pro reprezentace RDF se většinou používá XML ze syntaxi RDF, označuje se pak XML/RDF. Využití XML/RDF lze nejlépe ukázat na jednoduchém příkladě:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns\#"
         xmlns:vcard="http://www.imc.org/vcard/3.0/"
         xmlns:dc="http://purl.org/dc/elements/1.1/">
   <rdf:Description
      about="http://www.sport.cz/fotbal/2003/12/04/spartachelsea.html">
      <dc:Title>Sparta  Chelsea 0:1</dc:Title>
      <dc:creator rdf:resource="http://www.sport.cz/authors/PetrMatulik"/>
      <dc:date>2003-12-04</dc:date>
   </rdf:Description>
   <rdf:Description about="http://www.sport.cz/authors/PetrMatulik">
      <vcard:fn>Petr Matulík</vcard:fn>
      <vcard:email>petramatulik@email.cz</vcard:email>
   </rdf:Description>
</rdf:RDF>
```

Příklad popisuje článek publikovaný na internetu a jeho autora. K popisu jsou použity dva jmenné prostory:

- vcard – slouží pro popis osob,
- dc – standard pro popis digitálních objektů (včetně WWW stránek).

Z popisu je zcela jasné, kde se článek nachází, kdo ho vytvořil a kdy.

5.6 XUL

Je značkovací jazyk pro popis uživatelského rozhraní, založen na jazyku XML. Zkratka pochází z anglického XUL[1] (**X**ML **U**ser Interface **L**anguage) a vyslovuje se „zůl“. Byl navržen pro zjednodušení portace mezi podporovanými operačními systémy a hardwarovými platformami. Pro programátory je používaní XULu velmi příjemné, protože se velmi

[1] Popravdě je jméno XUL odkazem na film Krotitelé duchů, v kterém se tak jmenoval duch Sumerského boha. Odkazů na tento film se v Mozille najde celá řada.

podobá tvorbě webových stránek. Hlavní rozdíl spočívá v rozdílných jménech značek. Pro vzhled a definici chování se shodně u obou používají kaskádové styly a JS [42].

Použitím kaskádových stylů se oddělil vzhled od vlastních uživatelských prvků. Aplikacím se tak dá velice snadno měnit vzhled. Dříve měla Mozilla v sobě definovaný standardní vzhled, který vypadal na všech systémech stejně, což se ukázalo jako nevhodné, protože většina uživatelů je zvyklá na prostředí svého operačního systému a aplikace tak vypadala jako cizí. Proto byla do jádra Gecka zapracována podpora pro nativní vzhled z operačního systému. Dnes jsou podporovány OS Windows XP/Vista, Mac OS X a systémy podporující knihovnu GTK. Výběr stylu je tak plně na vkusu uživatele, pokud shledá, že mu nevyhovuje ani jeden, tak si může vyrobit styl vlastní.

Množina uživatelských prvků, kterou XUL nabízí, ničím nezaostává za běžným standardem, který poskytují sofistikované knihovny pro tvorbu rozhraní. Přehled nejdůležitějších[2] komponent je v tabulce 5.1.

Vstupní políčko	
Zaškrtávací políčko	Default Checked
Přepínač	Selected Normal
Rolovací nabídka	Normal / Normal / Selected / Disabled / Checkbox
Tlačítko	6 3
Ukazatel činnosti	
Tabulka	Name / Attribute / Cosmo / Round / Fergus / Long / Clint / Young
Záložky	Default Tab Orientation / Tab
Stromový výběr	Cats / Female / Gray / Pearl / White / Cos... / Orange

Tabulka 5.1: Seznam prvků

Jazyk XUL není omezen pouze na platformu Mozilla, ale existují portace do dalších jazyků a rámců. Příkladem může být například jazyk Java (projekt Luxor) [43].

5.7 XBL

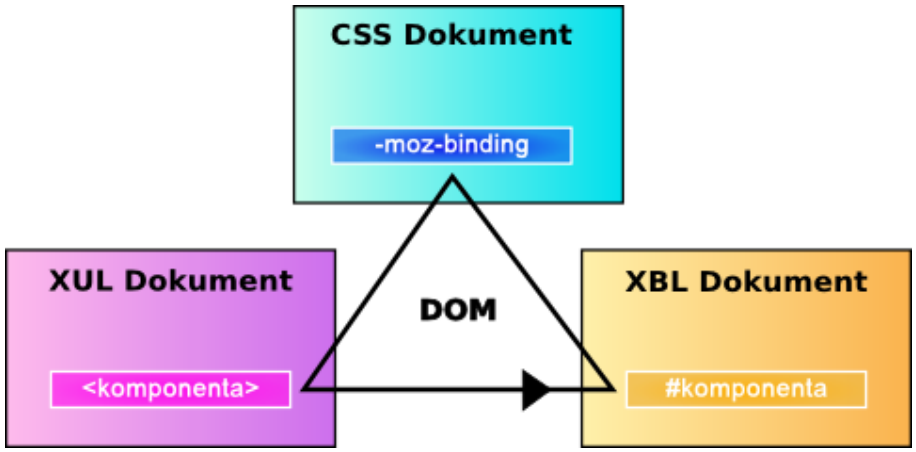
XML **B**inding **L**anguage (někdy také nazývaný Extensible Bindings Language) je jazyk pro popis vazeb, kterými mohou být připojeny elementy v jiných dokumentech. Element, který je takto připojen, se nazývá vázaný element a získává nové chování specifikované vazbou.

[2]Seznam všech prvků lze nalézt na adrese: http://developer.mozilla.org/en/docs/XUL_Reference.

Specifikaci XBL 1.0 vytvořila Mozilla a není nijak standardizovaná, což se snaží napravit verze druhá. Ta už je pod standardizačním procesem organizace W3C. V současnosti je ve stavu pracovního návrhu.

XBL je vlastně systém pro tvorbu nových prvků GUI. Většina vestavěných ovládacích prvků je tak implementována. XUL sice dovoluje vytvoření nového elementu, ale ten nemá sám o sobě přílišnou hodnotu, protože lze pouze nastavit vzhled, ale je to stále boxový prvek bez chování. XBL proti tomu nabízí vytvoření komplexního ovládacího prvku spolu s chováním. XBL nevytváří nový element, vytváří pouze vazbu. Pro navázání komponenty na konkrétní element se používá proprietární CSS vlastnost *-moz-binding*. Na vytvoření nové komponenty lze s výhodou využít dědičnosti a založit ji tak na již existující [44].

Na vytvoření nové komponenty lze s výhodou využít dědičnosti a založit ji tak na již existující. Pro lepší představu je vztah mezi jednotlivými technologiemi zobrazen na diagramu 5.1



Obrázek 5.1: Diagram znázorňující vytvoření nového prvku uživatelského rozhraní.

Následující příklad demonstruje vytvoření jednoduché komponenty. Ta obsahuje panel s dvěma tlačítky – ano a ne. Po kliknutí se zobrazí dialog informující o vybrané možnosti. Součástí příkladu je použití vytvořené komponenty v XUL. Výsledek zobrazuje ilustrace 5.2

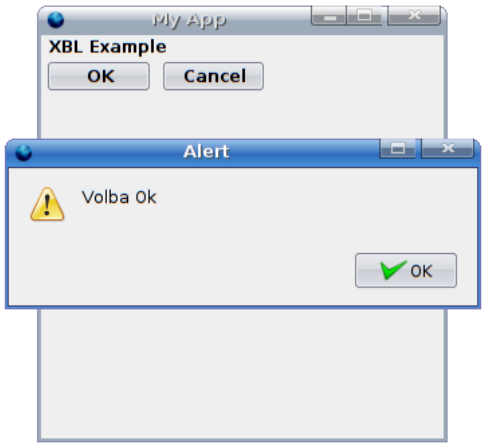
Soubor **main.xul** – obsahuje XUL dokument, který zobrazí komponentu.

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<?xml-stylesheet href="style.css" type="text/css"?>

<window id="main" title="My App"
   width="300" height="300"
   xmlns=
   "http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
    <label value="XBL Example"/>
    <box class="komponenta"/>
</window>
```

Soubor **style.css** – stylopis pro main.xul, obsahuje návazní komponenty panel na box.

```
label {
    font-weight: bold;
}
box.komponenta {
    -moz-binding: url('komponenta.xml#panel');
}
```



Obrázek 5.2: Diagram znázorňující vytvoření nového prvku uživatelského rozhraní.

Soubor **komponenta.xml** – obsahuje vlastní komponentu.

```
<?xml version="1.0"?>
<bindings xmlns="http://www.mozilla.org/xbl"
  xmlns:xul=
  "http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
    <binding id="panel">
       <content>
          <xul:button label="OK" oncommand="alert('Volba Ok');"/>
          <xul:button label="Cancel" oncommand="alert('volba Cancel');"/>
       </content>
    </binding>
</bindings>
```

Kořenový element <bindings> značí prostor pro vkládání nových komponent. Vlastní komponenta se uvozuje elementem <binding>. Jméno se definuje atributem id. V tabulce 5.2 jsou vyjmenovány nejdůležitější vlastnosti, které lze v komponentě definovat.

resource	Slouží pro definování vzhledu komponenty.
content	Vlastní tělo komponenty.
field	Definuje datové atributy komponenty.
property	Vlastnost, bezpečný přístup k atributům, který podporuje myšlenku zapouzdření. Používá se podobně jako v jazyku C#.
method	Definování metody tak jak známé s OOP jazyků.
constructor	Volá se v okamžiku, kdy je komponenta připojovaná k elementu.
destructor	Volá se v případě odpojení komponenty od elementu.
handlers	Slouží k připojení událostí, na které bude komponenta reagovat.

Tabulka 5.2: Přehled nejdůležitějších vlastností komponent.

5.8 HTML, XHTML a XML technologie

Díky původu platformy v prohlížeči internetových stránek, se pro vytváření uživatelského rozhraní dají použít technologie z této oblasti. Zejména se to týká nových značek z jazyka HTML 5. Přestože je tato verze ve fázi návrhu, bylo již do jádra Gecka implementováno několik vlastností.

Nejzajímavější nový element je <canvas/>. Ten umožňuje vytvořit kreslící plochu a pomocí JS do ní kreslit. Dalšími novými elementy jsou <video/> a <audio/>. Ty přinášejí podporu těchto multimédii přímo do jádra prohlížeče. Výhoda oproti stávajícímu stavu je vtom, že se nemusí spoléhat na služby třetích stran. Stav implementace těchto dvou značek je v době psaní práce v raném stadiu a není jisté, kdy se budou moci plnohodnotně použít [45].

Na jazyku XML jsou založeny dva formáty SVG a MathML, které Gecko podporuje. První zmiňovaný slouží k popisu vektorové grafiky. Druhý pak k popisu matematických vzorců [46].

5.9 Rozšíření (moduly)

Jednou z hlavních divizí prohlížeče Firefox jsou doplňky. Ty uživatelům umožňují přidávat novou funkcionalitu. Jejich možnosti jsou prakticky neomezené. Umožňují například měnit/předefinovat uživatelské rozhraní nebo měnit, popřípadě vytvářet, nové chování. To je umožněno architekturou platformy.

Bohužel tato volnost přináší i komplikace. Jednou je negativní dopad na rychlost aplikace, pokud je modul chybně napsaný. Ještě važnější je rozbroj mezi více moduly, který může nastat, pokud se snaží modifikovat to samé. Dalším neduhem je bezpečnost, protože rozšíření může obsahovat škodlivý kód.

Ideální způsob, jak zabráni uživateli, aby si takovýto nežádoucí modul instaloval, neexistuje. Částečné řešení spočívá v osvětě a umožnění instalovat pouze digitálně podepsané rozšíření s platným certifikátem.

Pro vývoj aplikací je příjemné, že do nich lze jednoduše vložit mechanizmus pro obsluhu modulů. A umožnit tak zákazníkům tvorbu rozšíření pro vaši aplikaci.

Kapitola 6

Mozilla prakticky

Po teoretických kapitolách přichází na řadu ryze praktická, která poskytuje lehký úvod do problematiky vytvoření aplikace. Smyslem kapitoly je poskytnout čtenáři odrazový můstek pro vlastní vývoj aplikací. Postupně se probírají témata: jak nainstalovat běhové prostředí, vytvoření nového projektu, jeho vlastní programování a typy pro vývoj. Kapitolu uzavírá jednoduchý demonstrační příklad – hra Šibenice.

6.1 Nástroje pro podporu programování

Asi jednou z nejméně příjemných skutečností při vývoji aplikace je absence sofistikovaného vývojového prostředí. Zajisté lze použít klasické programátorské editory typu Vim, Emacs nebo PSPad, ale většina vývojářů je zvyklá na komfort, který poskytují vývojové prostředí. Naštěstí existuje alespoň několik nástrojů s částečnou podporou.

6.1.1 Komodo IDE

Toto vývojové prostředí se v této práci již jednou vyskytlo – v přehledu aplikací postavených nad platformou. Editor nepodporuje přímo vytváření XULRunner projektů, ale pouze rozšiřujících modulů pro prohlížeč Firefox. Lze ho tak alespoň využít ke správě zdrojových kódů, zvýrazňování syntaxe (XUL, JS, ...) a částečně podporuje inteligentní doplňování kódu.

6.1.2 Eclipse

Pro toto oblíbené vývojové prostředí existuje zásuvný modul **XulBooster**, který přidává doplňovaní kódu (XUL, XBL, SVG), průvodce pro vytvoření rozšíření a pár dalších maličkostí. Bohužel je zatím ve velmi rané verzi a ne vše funguje k plné spokojenosti. Pro editování JavaScriptového kódu je vhodné doinstalovat zásuvný modul **Aptana**, který oproti standardnímu editoru obsahuje inteligentnější doplňování kódu a podporuje programovou dokumentaci podobnou JavaDoc.

Prostředí Eclipse je momentálně asi nejlepší volbou pro vývoj aplikací a dá se předpokládat, že jeho vlastnosti se nadále budou zlepšovat.

Odkazy:

- Eclipse – http://www.eclipse.org.
- XulBooster – http://cms.xulbooster.org.

- Aptana – http://www.aptana.com.

6.1.3 XUL Explorer

Jednoduchý editor z dílen Mozilla.org, který slouží k rychlému navržení uživatelského rozhraní. Pro základní design jsou dostupné jednoduché šablony. Vytvořené rozhraní se zobrazuje v náhledu. Bohužel editor obsahuje pouze základní editační funkce. Neobsahuje ani zvýrazňování syntaxe. Proto ho nelze doporučit pro návrh složitějších formulářů. Nejlépe se hodí k naučení jazyka XUL nebo k rychlému vyzkoušení myšlenky.

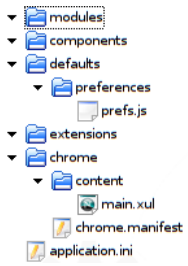
Odkazy:

- Xul Explorer – http://developer.mozilla.org/en/docs/XUL_Explorer.

6.2 Založení projektu

Každý projekt je tvořen speciální strukturou adresářů a souborů, která je znázorněna na obrázku 6.1. Význam jednotlivých adresářů je následující:

- modules – pro JS knihovny (novinka XULRunneru 1.9),
- components – adresář pro komponenty XPCOM,
- defaults – v podadresáři preferences jsou v souborech s příponou „js“ definovaná globální nastavení,
- extensions – adresář pro nainstalované rozšíření,
- chrome – obsahuje veškerá data adresovaná pomocí protokolu chrome (viz kapitola Chrome).



Obrázek 6.1: Základní adresářová struktura aplikace.

Každá aplikace musí mít v kořenu struktury soubor *application.ini*, který popisuje aplikaci. Jsou zde obsaženy údaje jako např. jméno, verze, jednoznačný identifikátor, autor a nutná min/max verze běhového prostředí potřebného ke spuštění.

V globálním nastavení je nutné definovat umístění okna, které se má spustit při startu aplikace. Dále je zde možno změnit implicitní nastavení platformy nebo vytvořit svoje vlastní.

Posledním zajímavým adresářem je chrome, ten ve svém kořenu obsahuje soubory s příponou „manifest“. Vlastní jména souborů nejsou významná – všechny budou zpracovány. Manifesty v sobě obsahují mapování adres protokolu chrome na adresáře (vysvětleno dále).

Ukázka této jednoduché struktury se nachází na doprovodném CD. Ukázka obsahuje jeden okenní dialog, které po spuštění (viz kapitola Spuštění aplikace) obsahuje známý text „Hello word!“ [42].

6.3 Chrome

Je část aplikace, kde se nachází obsah spojený s uživatelským rozhraním. Jde např. o dialogy, okna, CSS styly nebo lokalizační řetězce [47]. Chrome je rozděleno do tří částí, přičemž každá má svého správce (anglicky provider):

- **skin** – spravuje soubory určující vzhled chrome – typicky CSS, obrázky,
- **content** – spravuje struktury GUI komponent chrome (typicky XUL, JS, XBL),
- **localization** – spravuje všechny textové řetězce, které se v chrome vyskytují (např. texty menu, popisky ikon, ...).

6.4 Chrome registry

Je služba, která slouží k namapování jména balíčku na fyzické umístění na disku. Pod pojmem balíček se myslí kolekce zájmově stejného obsahu pod jedním správcem. Registry se konfigurují pomocí textových souborů s příponou „manifest“, které jsou umístěny ve složce *chrome*. Manifesty mají textový formát založený na řádcích. Pokud je obsah na řádku ve správném tvaru, tak se provede příslušná akce, v opačném případě se ignoruje.

content

```
content jméno_balíčku uri/k/souborům/
```

Cesta k souborům musí končit znakem „/“.

locale

```
locale jméno_balíčku jmeno_lokalizace uri/k/souborům/
```

Za jméno_lokalizace se dosazuje jazyk lokalizace, jako je například „cs“ pro češtinu nebo uven-US pro americkou angličtinu. Pro stejný balíček lze vytvořit více pravidel, pokaždé s jiným jménem lokalizace. Dosáhne se tak efektu, že aplikace obsahuje více jazyků.

skin

```
skin jméno_balíčku jméno_vzhledu uri/k/souborům/
```

Jméno_vzhledu má obdobné použití jako příkaz locale. Aplikace díky tomu může obsahovat více vzhledů.

6.5 Protokol chrome

Zatím bylo vysvětleno co *chrome* je a jak se definuje, ale chybělo vysvětlení, k čemu slouží. Celý mechanizmus dostává význam až s protokolem chrome. Jedná se klasický URI protokol, který má následující předpis: `chrome://jmeno_balicku/spravce/cesta/k/souboru`. Vytváří se tak virtuální adresový prostor nad celou aplikací, ve kterém balíčky hrají úlohu jmenných prostorů a správci určují kontext.

Při bližším prozkoumání protokolu může zarazit nemožnost specifikovat *jméno_lokalizace* pro správce *locale*. Je to dáno tím, že při vyhodnocování adresy se lokalizace automaticky určí s ohledem na nastavení aplikace. Obdobně to funguje i pro vzhled.

6.5.1 Příklad

Mějme adresářovou strukturu jako na obrázku 6.1. V souboru *chrome.manifest* je následující řádek:

```
content app content/
```

Pokud bychom chtěli otevřít okno *main.xul* v adresáři *content* pomocí JS, tak se k tomu použije následující příkaz:

```
window.open("chrome://app/content/main.xul", "_blank")
```

6.6 XPCOM

Platforma v sobě obsahuje velké množství komponent pro použití. Lze je rozdělit do dvou skupin:

- instance,
- služby.

Navzájem se liší tak, že instancí lze vytvořit neomezené množství. Služba je oproti tomu implementována podle návrhového vzoru jedináček a je tak v celém systému unikátní (existuje pouze jedna instance na kterou se dostává reference).

Získaní objektu/služby se provádí ve dvou krocích. V prvním se získá třída, která se specifikuje řetězcovým identifikátorem. V druhém kroku se pomocí explicitní definice rozhraní dostane vlastní objekt. Na příkladu lze vidět, jak vytvořit objekt reprezentující lokální soubor na disku.

```
var class = Components.classes["@mozilla.org/file/local;1"];
var file = class.createInstance(Components.interfaces.nsILocalFile);
```

Vytvoření objektu reprezentujícího službu se liší pouze v záměně metody *createInstance* za *getService*.

Vytváření vlastních komponent není zcela triviální a je nad *rámec* této práce.

6.7 Moduly

Velmi často je potřeba při programování vytvářet knihovnu funkcí, které se často opakují. Bohužel standardní jazyk JS tuto neumožňuje. Danou situaci by sice bylo možné řešit pomocí XPCOM implementovaných v JS, ale jak již bylo naznačeno, není to příliš jednoduché.

Proto tvůrci Mozilly přidali moduly. Ty nejsou ničím jiným než běžným JS skriptem, obsahující navíc pole *EXPORTED_SYMBOLS*, ve kterém se uvádí jaké funkce, objekty nebo proměnné modul poskytuje navenek.

Nejlepší vysvětlení je na příkladě. Ten obsahuje tři funkce: foo, bar a none. V poli pro export jsou definovány pouze první dvě. Mimo modul tak půjde použít pouze je.

```
function foo(){ ... }
function bar(){ ... }
function none(){ ... }
var EXPORTED_SYMBOLS = ["foo", "bar"]
```

Pro použití modulu je prvně nutné jeho načtení. K tomu slouží funkce *Components.utils.import(adresa)*, která je obdobou příkazu „include“, známého z jinýchvarianty jazyků. Pro určení adresy se používá protokol resource, který má následující tvar: `resource://alias/relativni-cesta/soubor.js|jsm`, kde alias může nabývat hodnot:

- **app** – bude se hledat ve složce aplikace,
- **gre** – bude se hledat ve složce běhového prostředí.

Relativní cesta může nabývat libovolného jména, ale je vhodné použít „modules“ [48].

6.8 Prostředí pro běh aplikací

V kapitole 4.1 bylo řečeno, že pro spuštění aplikací slouží XULRunner. Příklady a aplikace v této práci jsou postavené na nejnovější verzi 1.9. Kromě XULRunneru, lze ke spuštění také použít prohlížeč Firefox 3.0, ale pro vývoj je lepší použít první variantu [49].

6.8.1 Instalace XULRunneru

Postup při instalaci je následující:

1. Pořídit si nejnovější verzi XULRunneru. Možnosti získání jsou následující:
   (a) Stažení binární verze ze stránky: `http://developer.mozilla.org/en/docs/XULRunner`.
   (b) Kompilace ze zdrojových kódu – poměrně náročné, ale při kompilaci je možné si zapnout podporu vlastností, které nejsou ve výchozím stavu povoleny (většinou jde o experimentální kód).
2. Vlastní instalace. Postup se různí podle cílového OS:
   - Windows – rozbalení archivu do určeného adresáře (doporučeno do C:\Program Files\XULRunner). Poté je nutné pomocí příkazového řádku spustit příkaz:
     xulrunner.exe --register-global – pro všechny uživatele systému,
     xulrunner.exe --register-user – pouze pro jednoho uživatele.
     Tyto příkazy zaregistrují běhové prostředí do systému.
3. Mac OS X – otevření .pkg souboru pomocí instalátoru.
4. Linux – obdobné jako u OS Windows, s ohledem na přístupová práva (instalace pro všechny uživatele vyžaduje práva účtu root).

6.8.2 Odinstalování prostředí

Pokud už běhové prostředí není potřeba, tak se dá lehce odstranit. Postup se opět liší podle OS:

- Windows/Linux – spustit příkaz **xulrunner --unregister-global** nebo **xulrunner --unregister-user** podle typu instalace. Po provedení se pak musí ručně smazat vlastní adresář.
- Mac OS X – Smazání adresáře */Library/Frameworks/XUL.framework*

6.9 Spuštění aplikace

Startovací bod aplikace tvoří vždy soubor *application.ini*. Vlastní spuštění lze provést následujícími způsoby, které se liší v podobě umístění programu.

Při prvním způsobu, nejběžnějším při vývoji, je aplikace umístěna na disku v adresářové struktuře. Program se pak spouští příkazem:

xulrunner(.exe) /cesta/k/application.ini

nebo v případě použití Firefoxu verze 3:

firefox(.exe) -app cesta/k/application.ini

U druhého způsobu je aplikace uložena v archivu typu zip, nejčastěji s příponou „xulapp“. Pro spuštění se prvně musí provést instalace příkazem:

xulrunner(.exe) --install-app /cesta/k/archívu [/cesta/kam/instalovat]

pokud se vynechá cesta kam instalovat, tak se použije standardní systémová cesta pro instalaci programu, která je závislá na hostitelském OS. Při instalaci se v cílovém adresáři vytvoří spustitelný soubor se shodným jménem, jaké má aplikace. Vlastní spuštění se provede podle běžné konvence daného OS.

6.9.1 Distribuce aplikace

Možná forma distribuce aplikace byla naznačena v minulé podkapitole – a to jako archiv zip. Tento způsob má výhodu, že se využije sdílení běhového prostředí. Nevýhoda spočívá v poměrně uživatelsky nepříjemné instalaci, zejména pak pro uživatele OS windows, kteří nejsou zvyklí používat příkazový řádek.

Nabízí se možnost vytvořit instalační skript, který by zařídil vše potřebné v uživatelsky příjemné formě, například formou průvodců. Princip by byl jednoduchý: skript by nejprve detekoval přítomnost běhového prostředí, pokud by ho nenalezl, tak by se provedla jeho instalace a registrace. Kromě přítomnosti prostředí se musí brát ohled i na jeho verzi, některé aplikace nemusí na starší nebo naopak novější verzi fungovat. Dále by se pokračovalo vlastní instalací aplikace pomocí XULRunneru do uživatelem vybraného umístění. Tento způsob je velmi elegantní, ale bohužel zatím takovýto instalační program neexistuje.

Poslední možnost je poněkud těžkopádná, ale časem prověřená. Používá ji i organizace Mozilla.org na své projekty. Princip spočívá v distribuci aplikace včetně běhového prostředí, ztrácí se tak výhoda sdílení a multiplatformnosti instalace, ale předchází se problémy s verzemi. Příprava distribuce je následující:

- Do adresáře s aplikací se zkopíruje do adresáře XULRunner běhové prostředí pro cílový OS (stačí do aplikace jednoduše rozbalit archiv s XULRunnerem).
- Z adresáře XULRunner zkopírovat spustitelný soubor xulrunner-stub(.exe) do kořenového adresáře aplikace.
- Přejmenovat tento soubor na jméno aplikace.
- Na takto předpřipravenou aplikaci, lze použít už běžné instalační nastroje, jako je například NSIS od firmy Nullsoft[1].

6.10 Nápověda v aplikaci

Každá seriozní aplikace nutně musí obsahovat nápovědu k ovládaní. I na tento aspekt je v platformě pamatováno a pro jeho tvorbu je nabízeno jednoduché API. Nápověda je založena na formátu (x)html. Vazby mezi dokumenty jsou popsány formátem RDF.

Vstupní bod do nápovědy tvoří soubor, většinou pojmenovaný „help.rdf“, ve kterém se specifikují adresy čtyř základních okruhů nápovědy:

- Významový slovník (anglicky glossary).
- Rejstřík (anglicky index).
- Vyhledávaní (anglicky search).
- Tabulka kontextu (TOC).

Jednotlivé okruhy není povinné definovat. Z pohledu nápovědy je nejdůležitější tabulka kontextu, ve které je definována struktura (kapitoly a podkapitoly) s odkazy na (x)html dokumenty obsahující vlastní text. Ukázka jednoduché nápovědy je v příloze C.

V aplikaci je použití nápovědy velmi jednoduché. Slouží k tomu funkce *openHelp*, která je deklarovaná v souboru `chrome://help/content/contextHelp.js`. Funkce má dva parametry. První říká, na jakém tématu se má nápověda otevřít a druhý specifikuje umístění vstupního souboru „help.rdf“) [50].

6.11 Tipy & triky pro vývoj

Tato podkapitola se věnuje nástrojům a pomůckám, které jsou užitečné při vlastním vývoji. První tři nástroje jsou z dílny Mozilly, další jsou jednoduché funkce, které vznikly při vytváření projektu Tester. Jejich implementace je provedena v modulu base.jsm, který lze nalézt na doprovodném CD-ROM.

6.11.1 Chybová konzola

Slouží k informování o chybách v CSS, JS a vlastních poznámek. Jednotlivé zprávy jsou rozdělené do tří kategorii, dle vážnosti:

- chyby (např. chybná syntaxe v JS),
- varování (např. neznámá vlastnost v CSS),

[1] Volně šiřitelný instalační skript *nsis.sourceforge.net/Main_Page*

- zprávy (informační poznámky).

Zprávy lze podle kategorii filtrovat.

Konzole se musí explicitně zapnout, buď při startu aplikace parametrem „-jsconsole“ tj.

xulrunner(.exe) cesta/k/application.ini -jsconsole

nebo programové, pomocí následujícího kódu:

```
window.openDialog("chrome://inspector/content/", "Inspekt",
                  "chrome,dialog=no,all", window.location.href);
```

kde poslední parametr znamená adresu okna, které se má zkoumat. V tomto případě se předá adresa okna, ze kterého je funkce volaná. Po spuštění lze v menu Inspektoru zaměnit okno za libovolné otevřené.

6.11.2 Venkman (Javascript debugger)

Jak již nadpis napovídá, tak tento nástroj slouží k ladění JS skriptů. Obsluhou a možnostmi se nikterak nevymyká zavedenému standartu a zvládá všechny běžné úkony jako:

- prohlížeč kódu ze zvýrazněnou syntaxí,
- definování bodu zastavení (anglicky breakpoint),
- krokování programu (zanoření, vynoření a přes funkci),
- prohlížení stavu v proměnných,
- stav zásobníku volání,
- profilování (časová analýza) programu (anglicky profiler).

Obdobně jako DOM inspektor není debugger součástí běhového prostředí a musí se doinstalovat jako rozšíření, které lze nalézt na adrese: http://developer.mozilla.org/en/docs/Venkman. Pro otevření debuggeru v aplikaci je potřeba nejprve vytvořit funkci *toOpenWindowByType*. Vhodná implementace je následující:

```
const Cc = Components.classes;
const Ci = Components.interfaces;
function toOpenWindowByType(inType, uri, features) {
  //spravce oken
  var wM = Cc['@mozilla.org/appshell/window-mediator;'].getService();
  //rozhrani spravce oken
  var wMI = wM.QueryInterface(Ci.nsIWindowMediator);
  var topWindow = wMI.getMostRecentWindow(inType);
  if (topWindow) //pokud jiz okno existuje
    topWindow.focus();
  else if (features) //nove okno s prednastavenyma vlastnostma
    window.open(uri, "_blank", features);
  else //nové okno s vlastnim nastavenim
    window.open(uri, "_blank!, "chrome,extrachrome,menubar,
        resizable,scrollbars,status,toolbar");
}
```

Vlastní otevření se provede zavoláním funkce „*start_venkman()*“.

6.11.3 Modul Base

Obsahuje několik užitečných funkcí pro vypsání ladících informací a práci s XPCOM komponentami. Obsažené funkce jsou:

- **jsdump**(object) – obdoba vestavěné funkce dump, která vypíše textovou reprezentaci objektu na std. výstup. Jsdump tuto funkcionalitu rozšiřuje o výpis na JS konzoli.
- **jsPropertyDump**(objekt) – na std. výstup a konzoli vypíše o zadaném objektu následující údaje: typ, jméno, jména všech atributů společně s jejich hodnotou (včetně typů) a jména všech metod.
- **printDOM**(node) – vypíše stromovou strukturu potomků zadaného XML nebo HTML uzlu. U každého uzlu se vypíše: jméno, typ, atributy (včetně hodnot), pozice vůči rodičovskému uzlu a pokud uzel obsahuje textový uzel, tak jeho obsah.
- **createInstance**(identifikator, rozhraní) – vytvoří instanci XPCOM objektu. Funkce je synonymem pro zápis:

  Components.classes[*identifikátor*].createInstance(Components.interfaces.*rozhraní*),

- **getService**(identifikator, rozhraní) – obdoba funkce *createInstance*, určená pro získání služby.
- **getPlatform**() – zjistí textovou reprezentaci hostitelského OS. Ta je stejná jako u unixového příkazu „uname - s“. Pro OS Windows se vrací řetězec „WINNT“.

6.12 Ukázková demo aplikace

Říká se, že jeden obrázek zastoupí tisíce slov. U programování lze toto rčení s nadsázkou parafrázovat tak, že praktický příklad zastoupí mnoho textu. Proto jsem vytvořil velmi jednoduchou aplikaci, která shrnuje poznatky získané v této kapitole.

Jako příklad jsem si zvolil hru Šibenice. Jedná se o klasiku, známou ze školních lavic na ukrácení dlouhé chvíle. Její princip je velmi jednoduchý – hráč se snaží uhodnout slovo, které je reprezentované políčkem za každé písmeno. Hádání probíhá postupným výběrem písmen z anglické abecedy. V případě, že je písmeno ve slově obsaženo, napíše se na všechny jeho výskyty do příslušných políček. Pokud tomu tak není, započítá se chyba. Hra končí v případě, že hráč uhodne celé slovo nebo dosáhne předem určeného počtu chyb. Obtížnost hry je daná počtem dovolených chyb. Platí zde nepřímá úměra, čím méně chyb, tím obtížnější je slovo uhodnout a naopak. Vhodný rozsah by měl ležet v intervalu 6–9 neúspěchů.

Hra dostala jméno podle grafické reprezentace chyb, která znázorňuje stavbu šibenice. Při každé chybě se postaví část. Začíná se základnou a postupuje se jednotlivými trámy ve tvaru L. Poslední krok je uvázání oprátky nebo v drastičtější variantě oběšení figurky.

6.12.1 Technický popis

Aplikace je především zaměřená na ukázku uživatelského rozhraní, vlastní logika hry je velmi jednoduchá a je tvořena dvěma objekty:

Game (soubor Game.js) – jádro hry, které má v sobě uloženo stav hry. Přejímá písmena a na jejich základě modifikuje svůj stav. S okolím komunikuje pomocí událostí, které jsou:

- **onWin** – vyvolá se v případě výhry,
- **onLose** – vyvolá se v případě prohry,
- **onAddLetter** – vyvolá se v případě, kdy bylo vloženo správné písmeno,
- **onAddFault** – vyvolá se v případě, kdy bylo vloženo chybné písmeno.

Kromě událostí objekt obsahuje následující metody:

- **play**(slovo, obtížnost) – metoda vynuluje stav hry, nastaví slovo, které je předmětem hádání a obtížnost (vyjádřenou počtem možných chyb). Nakonec spustí hru,
- **isPlay**() – příznak, jestli hra probíhá,
- **nextLetter**(písmeno) – otestuje zadané písmeno.

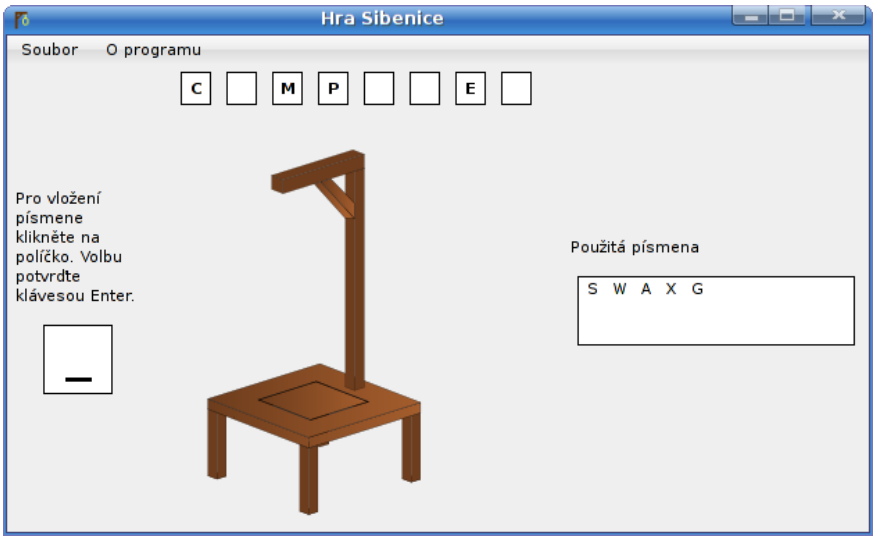
Words (soubor Words.js) – slouží k získávání slov. Ta jsou uložena jako jeden řetězec v globálním uložišti nastavení. Mezi sebou jsou vzájemně oddělena separačním znakem „;“. Objekt obsahuje následující metodu:

- **getWord**() – vrátí náhodné slovo.

Vlastní uživatelské rozhraní je deklarováno v souboru *sibenice.xul* a obsluha v *sibenice.js*. Hlavní okno obsahuje v horní části programové menu. Zbývající část prostoru, která obsahuje prvky pro hru, je rozdělena pomyslně do tří vertikálních kvadrantů. V prvním se nalézá box pro zadávání písmen. Střední část obsahuje pole políček reprezentujících písmena hledaného slova a prostor pro obrázek znázorňující počet chyb. Poslední část je tvořena boxem, který obsahuje použitá chybná písmena. Tyto čtyři prvky jsou implementovány jako XBL komponenty.

Za povšimnutí stojí komponenta pro zobrazení stavu hry, která zobrazuje graficky stavbu šibenice (viz obrázek 6.2). Pro kreslení obrázku je využit vektorový formát SVG. Jednotlivé fáze animace (stavby) jsou nakresleny ve vrstvách. Na začátku hry mají všechny nastavenou neviditelnost. Po každé chybě se pomocí JS zviditelní příslušná vrstva.

Poslední částí aplikace je dialog nastavení, ve kterém si uživatel může zvolit obtížnost hry nebo spravovat seznam slovíček. V dialogu se využívají možnosti platformy, které jeho vytvoření velmi usnadňují. Dovoluje například provázání kontextového rolovacího menu s položkou v globálním úložišti nastavení.



Obrázek 6.2: Hlavní okno programu Šibenice, ve kterém právě probíhá hra.

Kapitola 7

Demonstrační aplikace – Tester

Pro demonstraci možností platformy jsem si vymyslel aplikaci z e-learningového prostředí. Přesněji se jedná o program pro podporu učení na základě kvízových (testových) otázek se zaměřením na výuku slovní zásoby.

Pro tento projekt jsem se rozhodl, protože již delší dobu hledám aplikaci, ve které bych si mohl vytvořit sadu otázek s odpověďmi, na jichž základě by mně procvičovala. Takovouto nekomerční, multiplatformní aplikaci jsem zatím nenalezl.

Pro výběr tohoto projektu mluví také fakt, že není výpočetně náročný a je převážně orientován na uživatelské rozhraní, jež platforma zvládá velmi dobře.

Pro aplikaci jsem si vytyčil následující požadavky:

- multiplatformnost – automaticky splněno díky platformě,
- možnost vytvářet/upravovat balíčky otázek,
- otázky můžou být textové, obrazové nebo popřípadě zvukové,
- inteligentní systém zkoušení,
- speciální podporu pro výuku cizích slovíček,
- jednoduchou hru pro procvičování slovní zásoby,
- jednoduchý slovník.

7.1 Analýza

Při analýze problému jsem se nechal, kromě vlastních představ, inspirovat existujícími komerčními produkty, které jsou zaměřeny na výuku jazyka. Šlo mi zejména o to zjistit, jaká je nejlepší forma učení a zkoušení. K tomu jsem použil následující produkty:

- Domácí učitel angličtiny od firmy Eddica – *www.eddica.cz*.
- Langugage Lab od firmy Vitvare – *www.vitware.cz*.

Než začnu s rozborem, tak definuji několik pojmů, které v textu dále používám:

Balíček – jedná se o kolekci otázek, které k sobě patří.

Otázka – může být slovní, grafická, zvuková nebo kombinace.

Odpovědi – kolekce správných a špatných odpovědí, která je součástí otázky.

Na základě prozkoumání produktu jsem vytvořil schéma výuky, které je rozděleno do tří režimů.

První slouží k naučení otázek. Uživatel postupně prochází všemi otázkami z vybraných balíčků, u kterých vidí správné odpovědi. V případě slovní zásoby se jedná o slovíčko v cizím jazyce a jeho správný překlady.

Druhý režim (dále v textu se používá pojem zkoušení) má podobu asistovaného testu. Na začátku si uživatel vybere jednu z následujících forem testu:

- **Podle otázky** – se určí, zda lze vybrat pouze jednu nebo množinu odpovědí. Určení je provedeno na základě počtu správných odpovědí. Tato forma je pro uživatele nejjednodušší.
- **Více možností** – u všech otázek se vybírá množina odpovědí. Uživatel má tak možnost vybrat více odpovědí i v případě, kdy správně je pouze jedna.
- **Textová odpověď** – místo pasivního výběru musí uživatel odpověď napsat. V případě, že otázka obsahuje víc správných odpovědí, stačí odpovědět na libovolnou z nich. Tato forma je nejtěžší, ale nejúčelnější pro procvičování slovní zásoby.

Pro ztížení testu lze vybrat maximální dobu na odpověď, po jejímž vypršení se přejde na další otázku. Po výběru nastavení začne vlastní test. Asistence spočívá ve vyhodnocení odpovědi při přechodu na další otázku. Pokud nebyla korektně zodpovězena, tak je na to uživatel upozorněn včetně správné odpovědi. V tomto režimu tedy stále probíhá učení, které je o něco zábavnější. Po ukončení testu se zobrazí vyhodnocení, které obsahuje seznam otázek, statistické vyhodnocení a výsledné oznámkování. Uživatel si následně může znovu zkusit špatně zodpovězené otázky.

Pod posledním režimem se skrývá klasický test, na kterém si uživatel prověří naučenou látku. Jeho možnosti a průběh je stejný jako v režimu zkoušení, jenom uživatel není upozorňován na chyby. Po vyhodnocení testu je možné si znovu projít všechny otázky. U každé jsou zobrazeny všechny odpovědi včetně jejich správnosti. Z důvodu poučení jsou zvýrazněny uživatelovy chybně vybrané odpovědi.

Aby nedošlo ke stereotypu, tak se pořadí otázek ve všech fázích náhodně generuje. U testu se kromě toho mění pořadí odpovědí.

7.1.1 Hra pro procvičení slovní zásoby

Pro zábavnou formu učení slovíček jsem vybral hru Šibenice, která je jednoduchá a velmi zábavná. Její bližší popis je v kapitole 6.12.

Aby hra splnila svůj pedagogický účel, tak musí po případném zdaru či nezdaru hráče zobrazit i význam hledaného slova (tj. jeho překlad).

7.1.2 Jednoduchý slovník

Slovník je doplňkem k výuce slovní zásoby. K tomuto účelu postačuje, aby byl velmi jednoduchý. Měl by tedy umět:

- vyhledání slov podle jeho prvních písmen,
- obousměrný překlad/vyhledávání,

- možnost měnit jazyky.

Potencionálně nejobtížnější část je získat obsah slovníků. Naštěstí existují projekty, které se tímto problémy zabývají a jsou přístupné pod otevřenou svobodnou licencí. Pro anglický jazyk se nachází velmi kvalitní slovník na adrese: http://slovnik.zcu.cz/download.php. Podstatně méně obsáhlé, ale s větším výběre jazyků lze nalézt na adrese: http://www.dicts.info/uddl.php.

7.2 Návrh a implementace

Při návrhu a implementaci jsem kladl důraz na co nejvyšší využití platformy. Mou snahou také bylo využití nových vlastností připravovaného xulrunneru verze 1.9. Kompletní a příkladný návrh je nad možnostmi tohoto textu. Proto jsem vybral jen důležitá a zajímavá témata vzhledem k platformě.

V návrhu bylo nejdůležitější vytvořit formát pro reprezentaci balíčků. U něho jsem se nechal inspirovat formátem OpenDocument. Ten je založen na zip archivu, který obsahuje data uložená v textovém formátu a přílohy v adresářích. Zbývá tedy vyřešit strukturu textového formátu. Nabízejí se tři následující možnosti:

- jazyk XML – velká podpora v prog. jazycích a nástrojích,
- formát JSON – poměrně nový, velmi jednoduché použití,
- vlastní formát – např. založen na řádcích.

Poslední možnost je v dnešní době nevhodná pro svoji neuniverzálnost a náročnost. Zbývající dvě řešení jsou poměrně vyrovnané, ale nakonec jsem zvolil konzervativnější XML. Důvodem je jeho rozšířenost, jednoduchost a je poměrně přehledný pro ruční editaci.

Strukturu balíčku jsem definoval pomocí jazyka DTD. Při návrhu jsem bral ohled na budoucí rozšiřování formátu. Výsledné schéma s popisem se náchází v příloze D.

Pro jednoduší použití v XML souborech jsem schéma pojmenoval *lection* a umístil na internetovou adresu: http://dtd.pandasoft.org/lection.dtd. Ukázka XML dokumentu podle tohoto DTD je v příloze E. Konečná podoba balíčku je tedy následující:

- **package.tcm** – jméno balíčku s příponou – zip archív:
  - **images/** – adresář pro ukládání obrázku z otázek (adresa u obrázků se adresuje vůči tomuto adresáři),
  - **sounds/** – adresář pro ukládání zvuků z otázek (adresa u zvuků se adresuje vůči tomuto adresáři),
  - **data.xml** – soubor obsahující data, podle schématu *lection.dtd*.

Pro práci s balíčky slouží v aplikaci objekt *PackageManager*, který je implementován podle návrhového vzoru „singleton“ (česky jedináček). Pro vlastní reprezentaci baličku slouží objekt *PackageItem*. Oba objekty jsou vytvořeny jako moduly, aby šly jednoduše použít v JS skriptech.

Kvůli efektivnější správě paměti a rychlému vyhledávání si objekt *PackageManager* načítá signaturu balíčků, která obsahuje ID, jméno, skupinu a podobně – viz struktura. Kromě toho je doplněna statistickými údaji o počtu otázek a jejich rozložení (tj. kolik

z nich obsahuje text, obrázek nebo zvuk). Detailnější popis objektu je součástí příslušných souborů.

Z technologického hlediska je zajímavé ukládání a nahrávání dat z balíčku, které se provádějí pomocí XPCOM komponent pro manipulaci se zip archívy. Protože jejich použití není zcela přímočaré, vytvořil jsem v JS objektový obal, který práci velice usnadní. Implementovaný je jako modul pod jménem *zip.jsm*. Pro zpracování dat z XML souboru je použit dotazovací jazyk XPath.

Při vlastní implementaci nastaly problémy s multimediálním obsahem. Při návrhu bylo předpokládáno, že se stihne do platformy včlenit podpora audia. Kvůli komplikacím to ale autoři do odevzdání práce nestihli. U obrázku nastal problém u uživatelského prvku pro jeho zobrazování. Jediná možnost jak propojit externí obrázek s touto komponentou je přes adresu URL. Ale obrázky jsou uložené v zip souboru a tak se nedají adresovat přes standardní protokoly. Řešením by bylo extrahovat soubor do adresáře. Pak by šel adresovat pomocí protokolu *file*. Po ukončení práce by se z disku vymazal. Existuje ale elegantnější způsob, jak tento problém vyřešit. V platformě se vyskytuje protokol *jar*, který se používá u mapování balíčků v chrome kvůli ušetření místa (místo adresáře je jar archív). Protože je formát jaru velmi podobný zipu, lze ho použít pro získání obrázku přímo z archivu.

Zaměřením aplikace Tester je slovní zásoba. Aby se jí aplikace mohla přizpůsobit, je potřeba ji moci rozlišit od ostatních balíčků. K tomu lze použít jejich kolekci nastavení. Každá položka v ní je definovaná jako dvojice jméno – hodnota. Protože se předpokládá, že v budoucnosti může být aplikace zaměřená více směry, zvolil jsem jméno položky „type“ a vlastní rozlišení je na její hodnotě. V případě slovní zásoby má hodnotu „vocables“.

V požadavcích na Tester jsou dvě mini aplikace – hra Šibenice a slovník. Protože platforma standardně podporuje rozšiřování známým z prohlížeče Firefox, rozhodl jsem se toho využít. Vlastní úprava aplikace, aby umožňovala spravovat rozšíření, není vůbec složitá. Je to pouze otázka nastavení a minima programování.

7.2.1 Hra Šibenice

Protože je hra v aplikaci spíše doplňkem, je tak i implementována a není tak součástí Testeru po instalaci. Uživatel si v případě zájmu může toto rozšíření doinstalovat.

Návrh a implementace je obdobná jako u ukázkového dema v kapitole 6.12.1. Kromě změn spojených s portací na rozšíření se musel předělat objekt words, který se stará o získávání slov. Jejich nový zdroj představují uživatelem vybrané balíčky.

Je zřejmé, že ne všechny balíčky jsou ideálními kandidáty, protože některé obsahují dlouhé otázky. Vhodnými balíčky jsou pouze ty, co obsahuji jako otázku jedno slovo. Příkladem může být slovní zásoba. Řešení problému je poměrně jednoduché. Pro rozlišení, které balíčky se mají pro hru použít stačí použít příznak v jejich nastavení. Ten jsem zvolil „gallows=1“. Kromě této konfigurační dvojce se ještě akceptuje „type=vocables“, která reprezentuje slovní zásobu.

I přes tento výběr není stále zaručeno, že jsou všechny otázky (slova) vhodné. Proto je ještě zařazen filtr, který provede následující akce v uvedeném pořadí:

1. Pokud je slovo delší než 12 znaků, je ignorováno.
2. Pokud slovo obsahuje znaky s diakritikou, jsou převedeny na odpovídající písmena anglické abecedy.
3. Jsou odstraněny veškeré znaky, které nejsou písmeny v anglické abecedě.

4. Výsledné slovo je převedeno na velké znaky.

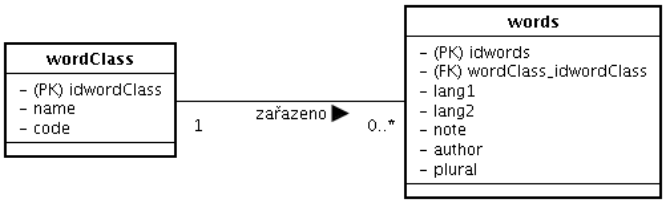
7.2.2 Slovník

Je dalším nadstandardním rozšířením aplikace. U slovníku jsou nejdůležitější slovíčka a formát jejich uložení. V projektu jsem chtěl využít možnost přímého provázání uživatelského rozhraní s databází. Princip je velmi jednoduchý, prvku stačí určit, jakou databázi má využít. A poté pomocí jazyka pro manipulaci dat (DML) říci, jaké položky se mají vybrat pro zobrazení. Nakonec se musí vytvořit šablona jednoho řádku výstupu. Podle té se naformátují výsledky dotazu. XULrunner v poslední verzi umožňuje tři zdroje dat:

- Formát RDF – nejstarší, dnes se od něho ustupuje.
- Čistý formát XML – hodí se na menší objem dat.
- Vestavěnou databázi Sqlite – hodí se na větší objem dat.

Pro datovou reprezentaci slovníku jsem si vybral poslední zmíněný zdroj, protože datové slovníky mají většinou velký objem. Nemalou výhodou Sqlite je o to, že se jedná o klasickou databázi, se kterou se pracuje pomocí dialektu jazyka SQL.

Na obrázku 7.1 je zobrazen ER diagram databáze pro uložení slovíček. Význam jednotlivých entit a jejich atributů je následující:



Obrázek 7.1: E/R diagram databáze pro slovník.

- **wordClass** – slovní druh slovíčka:

  name – jméno druhu,

  code – zkratka.

- **words** – slovíčka:

  lang1 – slovíčko v prvním jazyku,

  lang2 – slovíčko v druhém jazyku,

  note – poznámka,

  author – autor slovíčka (kvůli licenci),

  plural – nastaven pokud je slovo v množném čísle.

Každá dvojice jazyků má vlastní soubor s databází. Pro popis metadat slovníku jsem použil jednoduchý XML soubor, jehož schéma v DTD je následující:

```
<!ELEMENT config (lang1,lang2,name,version,description,path)>
<!ELEMENT lang1 (#PCDATA)>
<!ELEMENT lang2 (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT path (#PCDATA)>
<!ELEMENT version (#PCDATA)>
```

Význam položek je:

- **lang1** – jméno prvního jazyka (např. angličtina),
- **lang2** – jméno druhého jazyka (např. čeština),
- **description** – detailnější popis slovníku,
- **name** – jméno slovníku (např. EN-CZ slovník),
- **path** – relativní cesta k souboru s databází,
- **version** – verze slovníku.

Protože je struktura souboru velmi jednoduchá, je pro načítání obsahu použit jazyk E4X. Ten je rozšířením jazyka JS o přímou práci s XML. DOM model se tváří jako kolekce a dá se tak lehce přistupovat k datům. Implementace tohoto jazyka je poměrně mladá a obsahuje chyby. Proto není vhodná na rozsáhlejší struktury.

Pro větší flexibilitu jsou data slovníku distribuována jako samostatné doplňky, které jsou závislé na rozšíření slovníku.

7.3 Ovládání aplikace

Při návrhu uživatelského rozhraní byla snaha o co nejintuitivnější ovládání. Přesto je celkový popis všech možností rozsáhlý a do tohoto textu tak nevhodný. Proto je zde uvedeno základní ovládání aplikace z pohledu uživatele, který si chce vyplnit test. Zbývající témata, jakými například jsou:

- vytváření/editace balíčků,
- přidání nových balíčků,
- přidání rozšíření,

obsahuje vestavěná nápověda aplikace.

Po spuštění aplikace se zobrazí podobné okno, jako na obrázku 7.2. Jeho horní prostor zabírá hlavní menu. Zbývající plocha pod ním je obsazena tlačítky, které reprezentují jeho nejdůležitější volby. Nabídky v menu jsou:

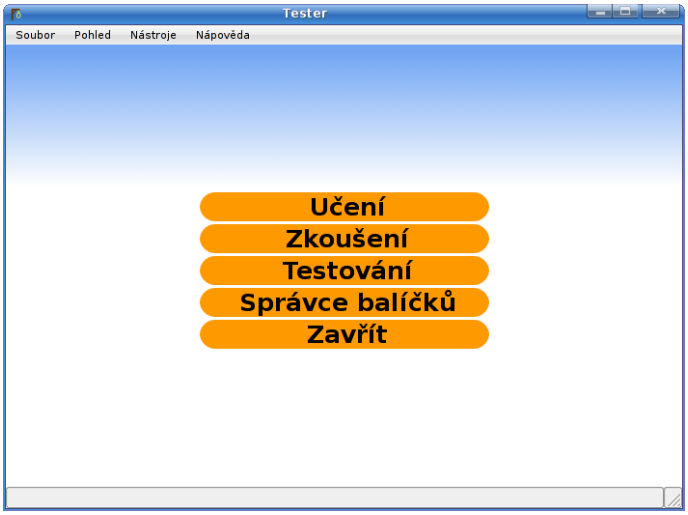
- **Soubor**:

  Zavřít – uzavře aplikaci.

- **Pohled**:

  Úvod – panel s tlačítky.

  Učení – volba režimu.



Obrázek 7.2: Hlavní okno aplikace Tester.

Zkoušení – volba režimu.

Testování – volba režimu.

Správce balíčků – zobrazí dialog pro správu balíčku.

- **Nástroje**:

  Rozšíření – zobrazí dialog pro správu rozšíření.

  Nastavení – zobrazí dialog pro nastavení.

  Restartovat – restartuje aplikaci (zavře a znovu ji spustí).

- **Nápověda**:

  Hlavní – zobrazí hlavní nápovědu.

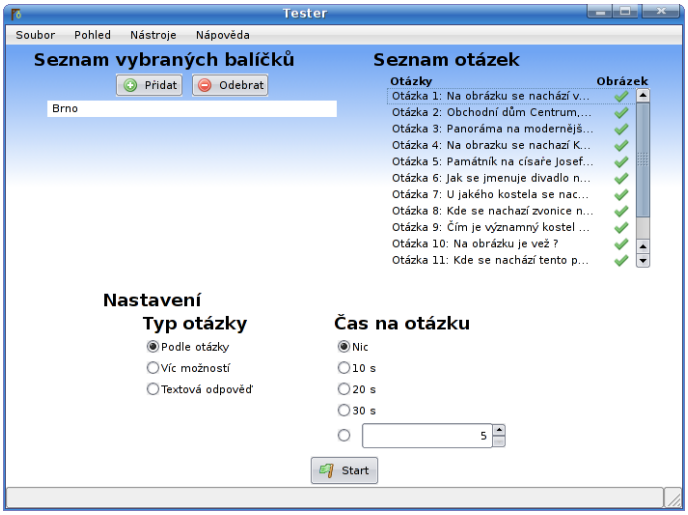
  O programu – zobrazí dialog informující o programu.

Po zvolení jednoho z režimů se přepne na panel, který je znázorněn na obrázku 7.3. V případě učení se nezobrazí spodní část s nastavením. Přidání/odebrání balíčku se provádí příslušnými tlačítky. Při výběru balíčku se v pravé části zobrazí jeho seznam otázek. Vlastní test se spustí tlačítkem start. Okno zobrazující otázky se zobrazí přes celou obrazovku, aby uživatel nebyl ničím rušen. Obsah okna se různí podle otázky a zvoleného režimu. Na další otázku se přejde pomocí tlačítka **další** nebo klávesou **Enter**. Pokud je otázka časově omezená, tak je vedle zmíněného tlačítka časomíra zobrazující zbývající čas v sekundách.

7.4 Možnosti rozšíření do budoucna

Aplikace je dobře připravena pro účely, na které byla vytvořena a splňuje podmínky, které jsem si na počátku stanovil. Ale samozřejmě jako každý projekt, i tuto aplikaci lze dále vylepšovat. Tato podkapitola obsahuje nápady, jakým směrem by měl jít další vývoj.

Jedním z nejdůležitějších bodů je rozšíření otázek o možnost zvuku, který by se hodil například pro výslovnost u slovíček. Návrh aplikace se zvukovou složkou počítá a jeho podpora vázne na vlastní platformě, která zatím neumožňuje rozumně přehrávat zvuk.



Obrázek 7.3: Volba balíčků a nastavení typu otázek.

Dalším bodem vylepšování jsou nastavení u balíčků. Zatím slouží pouze pro odlišení balíčků se slovní zásobou od ostatních. V budoucnosti by mohl autor balíčku nastavení využít například pro omezení výběru typů odpovědí, nastavení maximálního času na odpověď nebo definici vlastní stupnice hodnocení.

Návrhu uživatelského rozhraní byla věnována patřičná míra a snaha, aby bylo co nejvíce příjemné. Přesto se může v praktických podmínkách ukázat, že si uživatelé přejí jeho změnu. Podle jejich podnětů by se tedy mělo upravovat.

S uživateli je spojen i poslední bod. V aplikaci není ideálně řešeno přidávání nových balíčků otázek. Ty jsou alfou i omegou celé aplikace. I sebelepší technický produkt by se neměl šanci prosadit, pokud by pro něho neexistovala velká základna balíčků. Pro vybudování takovéto základny je nutno použít komunitní systém známý z web 2.0 aplikací. Celá myšlenka je založená na faktu, že balíčky tvoří lidé, kteří je potřebují pro svoje účely. Může se například jednat o studenta přírodovědy, který si vytvoří otázky s obrázky květin a odpověďmi budou jejich latinská jména. Takovýto balíček se určitě bude hodit více lidem. Ale je otázkou, jak ho k nim distribuovat, protože je potřeba, aby to bylo co nejjednodušší. Proto je zapotřebí vybudovat webový portál, který bude katalogizovat balíčky od uživatelů. Do aplikace pak zabudovat klienta (rozšíření správce balíčků) pro tento portál, který by umožňoval jednoduché vyhledávání, stahování, aktualizace a nahrávání balíčků.

Kapitola 8

Závěr

Za svou profesní kariéru jsem měl možnost vyzkoušet poměrně vysoký počet platforem nebo aplikačních rámců. Některé mně velmi zklamaly, jiné naopak nadchly. Platforma Mozilla se s jistotou řadí do druhé skupiny.

Nejsilnější divizí platformy jsou bezesporu technologie kolem uživatelského rozhraní. Trojice jazyků XUL, CSS a XBL poskytují pro vývojáře vysokou flexibilitu a rychlé prototypování. Je škoda, že si tato kombinace nenašla cestu do ostatních prohlížečů, protože poměrně elegantním způsobem řeší problémy dnešního webu. Nemusely by se tak vytvářet složité Javascriptové knihovny nebo vymýšlet nové standardy v podobě Xforms nebo Web Forms.

Další příjemnou vlastností je velké množství vestavěných komponent, které usnadňují vývoj. Z oblastí, které pokrývají, lze vystopovat původ platformy, protože nejvíc pokrytá funkcionalita je z oblasti internetu a multimédií.

Zajímavá je též možnost vybavit aplikaci mechanizmem pro její rozšiřování. Tato vlastnost velmi pomohla prohlížeči Firefox obstát v tvrdé konkurenci. Možnost nechat komunitu lidí vytvářet moduly a tak rozšiřovat aplikaci podle jejích představ je bezesporu užitečným benefitem.

Ne vše se ale dá hodnotit kladně. Příkladem může být dokumentace celé platformy, která je často neaktuální a pro některé komponenty není dostupná vůbec. Důsledkem toho je vývojář nucen studovat zdrojové kódy, což je velmi náročné na čas. Tímto nešvarem ale trpí mnoho opensource projektů, protože se klade větší důraz na přidávání nových funkcí a na nepopulární psaní dokumentace nezbývá čas. Mozilla si tuto situaci uvědomuje a v současné době buduje nový dokumentační portál, založený na systému Wiki. Díky tomu se otevře možnost podílet se na budování obsahu komukoliv.

Výkon platformy pro aplikaci Tester byl plně dostačující. Uživatelské rozhraní reaguje na podněty v reálném čase. Ale je zřejmé, že skriptovací jazyk má své výkonností limity. Je to daň za jednoduchou přenositelnost mezi softwarovými a hardwarovými platformami. Na zvyšování výkonu se neustále pracuje a každá nová verze běhového prostředí tento aspekt vylepšuje.

Budoucnost platformy lze těžko odhadnout. Ale je velmi pravděpodobné, že bude kopírovat vývoj událostí v organizaci Mozilla, které se na jejím vývoji ve velké míře podílí. Pro její budoucnost je potřeba větší zpopularizování, ke kterému snad přispěje i tento text. Kvalita celé platformy je na velmi dobré úrovni a potencionál k masivnějšímu používaní rozhodně má.

Literatura

[1] Softwarová platforma. http://www.agtivity.com/def/software_platform.htm [Verze L, 2.1.2006][Online; cit. 6.2.2008].

[2] Groovy dynamic language for the java platform. http://groovy.codehaus.org [Online; cit. 1.2.2008].

[3] The jython project. http://www.jython.org [Online; cit. 1.2.2008].

[4] Mono project. http://www.mono-project.com [Online; cit. 1.2.2008].

[5] Rich client platform. http://wiki.eclipse.org/index.php/Rich_Client_Platform [Online; cit. 10.2.2008].

[6] Eclipse rcp (rich client platform). http://nb.vse.cz/~zelenyj/it380/eseje/xmalj35/EclipseRCP.htm [Verze L, prosinec 2005][Online; cit. 10.2.2008].

[7] Netbeans platform. http://platform.netbeans.org [Online; cit. 10.2.2008].

[8] Spring richclient. http://spring-rich-c.sourceforge.net/1.0.0/index.html [Online; cit. 10.2.2008].

[9] Adobe air. http://labs.adobe.com/technologies/air [Online; cit. 11.2.2008].

[10] Wikipedia – microsoft silverlight. http://en.wikipedia.org/wiki/Silverlight [Online; cit. 11.2.2008].

[11] Javafx developer central. https://openjfx.dev.java.net [Online; cit. 11.2.2008].

[12] Wikipedia – framework. http://cs.wikipedia.org/wiki/Framework [Verze L, 1.5.2008][Online; cit. 11.2.2008].

[13] Historie webových prohlížečů, část 2. – worldwideweb a mosai. http://www.zive.cz/default.aspx?article=127213 [Verze L, 24. 10. 2005][Online; cit. 5.11.2007].

[14] Historie webových prohlížečů, část 3. – netscape. http://www.zive.cz/default.aspx?article=127343 [Verze L, 31. 10. 2005][Online; cit. 5.11.2007].

[15] Internet explorer history. http://www.microsoft.com/windows/WinHistoryIE.mspx [Verze L, 30. 6. 2003][Online; cit. 5.11.2007].

[16] Wikipedia – internet explorer verze 4. http://en.wikipedia.org/wiki/Internet_Explorer#Version_4 [Verze L, 5.11.2007][Online; cit. 5.11.2007].

[17] WWW stránky. Wikipedia – mozilla. http://en.wikipedia.org/wiki/Mozilla [Verze L, 5.11.2007][Online; cit. 5.11.2007].

[18] 10,000 firefox enthusiasts make history. http://www.spreadfirefox.com/node/8769 [Verze L, 16.12.2004][Online; cit. 6.11.2007].

[19] Spread firefox, akce z učebnice virálního marketingu. http://interval.cz/clanky/spread-firefox-akce-z-ucebnice-viralniho-marketingu [Verze L, 4. 1. 2005][Online; cit. 6.11.2007].

[20] Mozilla firefox's use share stabilises in the european countries. http://www.xitimonitor.com/en-us/browsers-barometer/firefox-september-2007/index-1-2-3-110.html [Verze L, 30.10.2007][Online; cit. 7.11.2007].

[21] Wikipedia – netscape. http://en.wikipedia.org/wiki/Netscape [Verze L, 31.10.2007][Online; cit. 5.11.2007].

[22] D. Majda. Vývojová platforma mozilla. In *LinuxExpo 2006*, Praha, 12.4.2006 2006. ftp://ftp.czilla.cz/other/propagation/vyvojova-platforma-mozilla-linuxexpo-2006.ppt [Online; cit. 3.3.2008].

[23] Xulrunner. http://developer.mozilla.org/cs/docs/XULRunner [Online; cit. 11.2.2008].

[24] Diskuse nad budoucností xulrunneru. http://jasnapaka.bloguje.cz/538290-diskuse-nad-budoucnosti-xulrunneru.php [Verze L, 19.5.2007][Online; cit. 11.2.2008].

[25] Wikipedia – xulrunner. http://cs.wikipedia.org/wiki/XULRunner [Verze L, 22.5.2007][Online; cit. 11.2.2008].

[26] Wikipedia – digital rights management. cs.wikipedia.org/wiki/Digital_rights_management [Verze L, 6.1.2008][Online; cit. 20.1.2008].

[27] Wikipedia – peer-to-peer. http://cs.wikipedia.org/wiki/Peer-to-peer [Verze L, 5.10.2007][Online; cit. 12.12.2007].

[28] Celtx - firefox pro psaní scénářů a podporu pre-produkce. http://interval.cz/clanky/celtx-firefox-pro-psani-scenaru-a-podporu-pre-produkce [Verze L, 9. 5. 2006][Online; cit. 18.12.2007].

[29] Flock – the social web browser. http://www.flock.com/ [Online; cit. 16.2.2008].

[30] Wikipedia – netscape portable runtime. http://en.wikipedia.org/wiki/Netscape_Portable_Runtime [Verze L, 15.12.2007][Online; cit. 16.12.2007].

[31] Jrex – the java browser component. http://jrex.mozdev.org [Online; cit. 16.12.2007].

[32] D. Majda. Mozilla – platforma pro vývoj aplikací. 2003. http://www.majda.cz/tvorba/skolni-veci [Verze L, 21.8.2003][Online; cit. 3.12.2007].

[33] D. Turner and I. Oeschger. Creating XPCOM Components. *Open Source*, 2003. http://www.mozilla.org/projects/xpcom/book/cxc [Verze L, 14.1.2005][Online; cit. 20.11.2007].

[34] Java xpcom. http://developer.mozilla.org/en/docs/JavaXPCOM [Verze L, 19.11.2006][Online; cit. 16.12.2007].

[35] Perl xpcom. http://plxpcom.mozdev.org [Verze L, 30.3.2005][Online; cit. 16.12.2007].

[36] Pyxpcom. http://developer.mozilla.org/en/docs/PyXPCOM [Verze L, 27.7.2007][Online; cit. 16.12.2007].

[37] Wikipedia – garbage collector. http://cs.wikipedia.org/wiki/Garbage_collector [Verze L, 4.10.2000][Online; cit. 17.12.2007].

[38] Xpconnect (scriptable components). http://www.mozilla.org/scriptable [Verze L, 1.2.2000][Online; cit. 17.12.2007].

[39] Rdf: Často kladené dotazy. http://dsic.zapisky.info/RDF/FAQ [Verze L, 30.11.2003][Online; cit. 19.12.2007].

[40] Rdf. http://developer.mozilla.org/en/docs/RDF [Online; cit. 19.12.2007].

[41] Sémantický web a jeho technologie. http://www.ics.muni.cz/zpravodaj/articles/296.html [Online; cit. 19.12.2007].

[42] D. Boswell, B. King, E. Murphy, I. Oescheger, and P. Collins. *Creating Applications with Mozilla*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.

[43] Luxor - xml ui language (xul) toolkit. http://luxor-xul.sourceforge.net [Online; cit. 18.12.2007].

[44] N. McFarlane. *Rapid Application Development with Mozilla*. Prentice Hall PTR, 2003. ISBN-13 978-0131423435.

[45] Support for html video element in firefox. http://www.bluishcoder.co.nz/2007/05/support-for-html-video-element-in.html [Verze L, 27.2.2007][Online; cit. 5.3.2008].

[46] Mathml in mozilla. http://www.mozilla.org/projects/mathml [Verze L, 25.9.2006][Online; cit. 5.3.2008].

[47] Chrome registration. http://developer.mozilla.org/en/docs/Chrome_Registration [Verze L, 22.2.2008][Online; cit. 5.3.2008].

[48] Using javascript code modules.
http://developer.mozilla.org/en/docs/Using_JavaScript_code_modules
[Verze L, 5.2.2008][Online; cit. 11.3.2008].

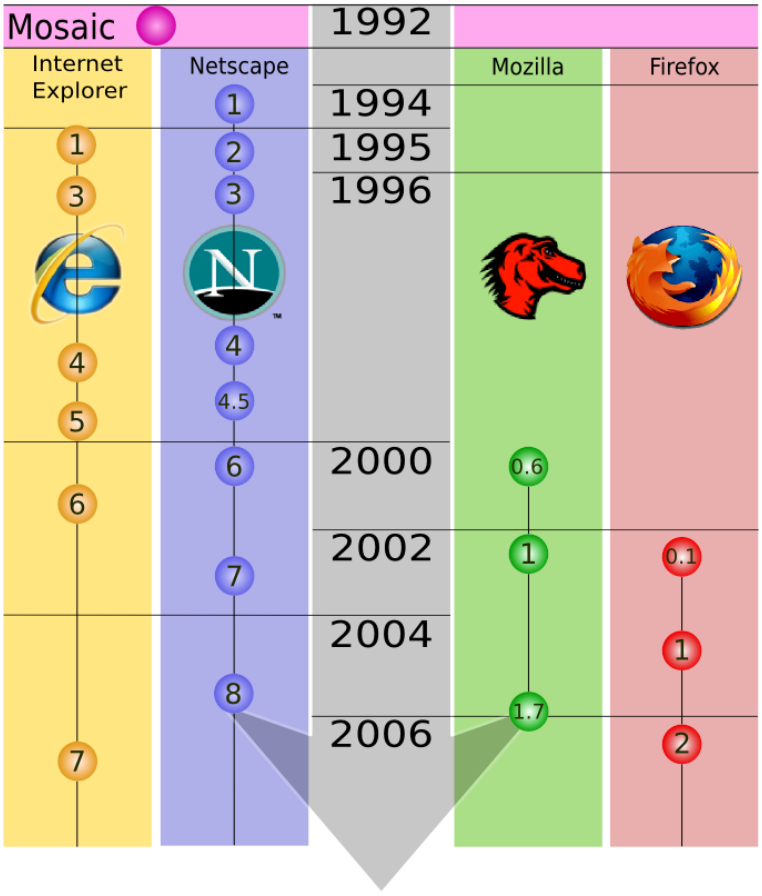
[49] Xulrunner 1.8.0.1 release notes.
http://developer.mozilla.org/en/docgs/XULRunner_1.8.0.1_Release_Notes
[Verze L, 22.8.2006][Online; cit. 13.3.2008].

[50] Help viewer: Creating a help content pack.
http://developer.mozilla.org/en/docs/Help_Viewer:
Creating_a_Help_Content_Pack [Verze L, 9.6.2007][Online; cit. 15.3.2008].

Seznam příloh

Dodatek A

Diagram vývoje jednotlivých verzí prohlížečů



Dodatek B

Přehled nových vlastností JavaScriptu podle verzí

JavaScript 1.6 – Firefox 1.5 (Gecko 1.8)

- **E4X** – rozšíření jazyka obohacující ho o přirozenou podporu XML.
- Rozšíření objektu pole o nové metody:
  - **every** – vrátí pravdu pokud zadaná funkce nad všemi položkami vrátí pravdu.
  - **filter** – vrátí nové pole, které obsahuje jen ty položky původního, u kterých zadaná funkce vrátila pravdu.
  - **forEach** – nad každou položkou je vykonaná zadaná funkce.
  - **map** – vrátí nové pole, s položkami původního, na které byla aplikovaná zadaná funkce.
  - **some** – vrátí pravdu, pokud alespoň jedna položka na zadanou funkci vrátí pravdu.

  Řetězce jsou také pole

JavaScript 1.7 – Firefox 2.0 (Gecko 1.8.1)

- **Generátory** – vlastnost známá z jazyka Python. Tato nová vlastnost se hodí ke generování posloupností s ušetřením paměti, protože každý nový člen posloupnosti se generuje až na požádání.
- **Interátory** – pro iterací nad daty.
- **Let-bloky** – velmi rozsáhlá funkcionalita týkající se rozsah platnosti viz zdroj.
- **Vícenásobné přiřazení** – na pravé i levé straně přiřazení – může být více hodnot.
- **Vícenásobná návratová hodnota** – výsledek funkce může vrátit více hodnot.

JavaScript 1.8 – Firefox 3.0 (Gecko 1.9)

- **Výrazové uzávěry** – lambda funkce.
- **Výrazové generátory** – zjednodušení vytváření generátorů.
- Rozšíření objektu pole o nové metody:

– **reduce** – pomocí zadané funkce se redukuje pole na jednu hodnotu. při redukci se postupuje od začátku pole do konce.
– **reduceRight** – stejný jako redukce jen se postupuje opačným směrem.

Dodatek C

Ukázka souborů potřebných pro vytvoření nápovědy

Soubor „help.rdf“ – obsahuje základní popis struktury nápovědy.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:nc="http://home.netscape.com/NC-rdf#">
    <rdf:Description about="urn:root"
      nc:title="Tester help"
      nc:defaulttopic="Select"
      nc:base="chrome://aplikace/locale/help/">
    <nc:panellist>
      <rdf:Seq>
        <rdf:li>
    <rdf:Description nc:panelid="glossary" nc:datasources=""/>
  </rdf:li>
        <rdf:li>
    <rdf:Description nc:panelid="index" nc:datasources=""/>
  </rdf:li>
        <rdf:li>
    <rdf:Description nc:panelid="search" nc:datasources="" nc:emptysearchtext=""
    nc:emptysearchlink="chrome://help/locale/help_help.html#search_tips"/>
  </rdf:li>
        <rdf:li>
      <rdf:Description nc:panelid="toc" nc:datasources="help-toc.rdf"/>
    </rdf:li>
      </rdf:Seq>
    </nc:panellist>
  </rdf:Description>
</rdf:RDF>
```

Soubor „help-toc.rdf“ – obsahuje kontextovou strukturu nápovědy.

```
<?xml version="1.0"?>
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:nc="http://home.netscape.com/NC-rdf#">
  <Description about="urn:root">
    <nc:subheadings>
      <Seq>
        <li>
      <Description ID="Main" nc:name="Tester" nc:link="main.html"/>
  </li>
      </Seq>
    </nc:subheadings>
  </Description>
  <Description about="#Main">
    <nc:subheadings>
    <Seq>
      <li>
        <Description ID="Main_start" nc:name="Úvod"
         nc:link="main.html#start"/>
      </li>
      <li>
        <Description ID="Main_body" nc:name="Tělo"
         nc:link="main.html#body"/>
      </li>
      <li>
        <Description ID="Main_end" nc:name="Konec"
         nc:link="main.html#end"/>
      </li>
    </Seq>
    </nc:subheadings>
  </Description>
</RDF>
```

Soubor „main.html“ – obsahuje vlastní text nápovědy

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
  <title>Nápověda k aplikaci</title>
</head>
<body>
  <div id="main">
    <h1 id="start">Úvod</h1>
    ...
    <h1 id="body">Tělo</h1>
    ...
    <h1 id="end">Konec</h1>
    ...
  </div>
</body>
</html>
```

Dodatek D

DTD schéma struktury balíčku

```
<!ELEMENT lection (name, group?, description, author, date,
                   setting?, firstname, secondname, questions)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT group (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT setting (value*)>
<!ELEMENT value (#PCDATA)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT secondname (#PCDATA)>
<!ELEMENT questions (question*)>
<!ELEMENT question (word?,sound?, picture?,answer+,note?)>
<!ELEMENT word (#PCDATA)>
<!ELEMENT sound EMPTY>
<!ELEMENT picture EMPTY>
<!ELEMENT note (#PCDATA)>
<!ELEMENT answer (#PCDATA)>

<!ATTLIST lection id ID #REQUIRED >

<!ATTLIST author homepage CDATA #IMPLIED >

<!ATTLIST value name CDATA #REQUIRED >

<!ATTLIST firstname lang CDATA #IMPLIED >

<!ATTLIST secondname lang CDATA #IMPLIED >

<!ATTLIST sound src CDATA #REQUIRED >

<!ATTLIST picture src CDATA #REQUIRED >
<!ATTLIST picture width NMTOKEN #IMPLIED >
<!ATTLIST picture height NMTOKEN #IMPLIED >

<!ATTLIST word data CDATA #IMPLIED >

<!ATTLIST answer correct (true|false) "false" >
```

Význam jednotlivých elementů je následující:

- **lection** – kořenový element:
  atribut id – celosvětově jednoznačné identifikační číslo.
- **name** – jméno balíčku.
- **group** – začlenění/specializace balíčku (např. angličtina, biologie).
- **description** – detailní popis balíčku.
- **author** – autor balíčku:
  atribut homepage – vyhrazeno pro stránky autora/balíčku.
- **date** – datum poslední aktualizace (slouží jako verze). Datum musí byt v zóně GMT a podle normy RFC #822.
- **setting** – kolekce nastavení balíčku.
- **value** – reprezentuje jednu položku v nastavení:
  atribut name – jméno položky.
  firstname – jméno pro otázky.
- **secondname** – jméno pro odpovědi.
- **questions** – kolekce otázek.
- **question** – jedna otázka.
- **word** – slovní otázka:
  atribut data – vhodný například pro fonetiku slovíčka.
- **sound** – pro zvukovou otázku,
  atribut src adresa zvukové stopy, vůči struktuře balíčku,
- **picture** – pro zvukovou otázku:
  atribut src adresa obrázku, vůči struktuře balíčku.
  width, height – velikost obrázku.
- **note** – doplňující text k otázce.
- **answer** – jedna odpověď na otázku:
  atribut correct – příznak, zda je otázka správná.

Dodatek E

Příklad XML struktury balíčku

```
<?xml version="1.0"?>
<!DOCTYPE lection SYSTEM "http://dtd.pandasoft.org/lection.dtd">
<lection id="1196713d-c380-36c5-3005-212a0ae3cd48">
  <name>Informatika</name>
  <group>Angličtina</group>
  <description>Slovíčka z oboru informatika</description>
  <author>Martin Vídeňský</author>
  <date>Sat, 19 Apr 2008 19:30:15 GMT</date>
  <setting>
    <value name="type">vocables</value>
  </setting>
  <firstname>Anglicky</firstname>
  <secondname>Česky</secondname>
  <questions>
  <question>
    <word data="brauz">browse</word>
    <answer correct="true">brouzdat</answer>
    <answer correct="false">jezdit</answer>
    <answer correct="true">procházet</answer>
    <note>to look for information on a computer, e
          sp. in a database or the internet
    </note>
  </question>
  <question>
    <word data="k.m"pjú:t.">computer</word>
    <answer correct="true">počítač</answer>
    <answer correct="false">kalkulačka</answer>
    <answer correct="false">počítadlo</answer>
    <note>A machine that can be used to do maths, type, draw, play games,
          watch videos, listen to music, surf the internet etc.
    </note>
  </question>
  </questions>
</lection>
```

Dodatek F

Obsah CD

Součástí práce je CD-ROM medium, na kterém se nachází následující obsah:

- Technická zpráva v elektronické podobě.
- Počeštěné běhové prostředí XULRunner.
- Aplikaci Testr pro ruzné platformy včetně zdrojových kódů.
- Rozšíření pro aplikaci Tester.
- Demoapliakce Šibenice včetně zdrojových kódů.
- Základní adresářová struktura pro zahájení projektu.

Přesnější pokyny se nachází na mediu v souboru README.