Masarykova univerzita
Fakulta informatiky

Miniaplikace v desktopových prostředích KDE a GNOME

Bakalářská práce

Petr Šigut

Brno, podzim 2009

Prohlášení

Prohlašuji, že tato bakalářská práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Petr Šigut

Vedoucí práce: Mgr. Marek Grác

Poděkování

Chtěl bych poděkovat vedoucímu práce Marku Grácovi za přátelský přístup a věnovaný čas, Jaroslavu Řezníkovi za obětavou pomoc a Davidu Hrachovému za rady.

Shrnutí

Cílem bakalářské práce bylo prozkoumat stav miniaplikací v Linuxu, konkrétně v prostředích KDE a GNOME. Dále se seznámit s prostředky pro získávání informací z webových stránek a webových služeb. Po prozkoumání dostupných poznatků k dané problematice jsme navrhli a implementovali klient–server systém pro zjednodušení vývoje nových miniaplikací.

Klíčová slova

KDE, GNOME, plasma, plasmoid, screenlet, miniaplikace, applet, gadget, Ruby on Rails

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

V dnešní době jsme zahlceni informacemi. Bylo by příliš zjednodušující označit původce většiny z nich slovem Internet – i průměrně zdatný uživatel dnes umí rozlišit e-mail, RSS[1], chat, sociální sítě, webové stránky a mnoho dalších. Tyto technologie můžeme nazvat informačními kanály, denně se vyvíjejí, vznikají nové či se kombinují. Jako s příchodem kina nezaniklo divadlo, nezmizel e-mail s příchodem chatu. Každý z informačních kanálů má svá specifika a využití.

Jedním z těchto informačních kanálů jsou miniaplikace (gadgety, widgety, applety), jedná se o malé jednoúčelové programy umístitelné (nejtradičněji) na plochu desktopového prostředí. Zobrazují především jednoduché informace jako aktuální počasí, program kin či kurzy měn. Miniaplikace jsou poměrně novinkou, pro svůj vznik potřebovaly dostatečný výkon počítačů (uživatel dnes již nepostřehne několik desítek neustále běžících programů) a stálé připojení k Internetu (zobrazované informace musí být aktuální). I miniaplikace mají jako každý informační kanál svá specifika – uživatelé vyžadují moderní vzhled (průhlednost, hezké ikonky) a minimum nutné konfigurace.

Cílem bakalářské práce bylo zjednodušit vývoj a používání miniaplikací především v desktopových prostředích KDE[2] a GNOME[3]. Mnoho miniaplikací je poměrně jednoduchých a zobrazují pouze část nějaké existující webové stránky. Přesto musí programátor řešit problémy jako periodická aktualizace dat, opakování kódu při zpracovávání obsahu stránek atd. Uživatel zase musí instalovat pro každý typ informací novou miniaplikaci, přestože se velice podobají. Navíc je nutné hlídat její aktuálnost – když se změní struktura stránky poskytující informace, je pravděpodobné, že přestane fungovat i miniaplikace.

V kapitole Současné miniaplikace se snažíme popsat současný stav, jeho problémové části a identifikujeme jejich potencionální řešení. V další kapitole Unico Server se čtenář seznámí se serverovou částí poskytující miniaplikacím data – je nastíněna její struktura, použité technologie a její

1. Really Simple Syndication, formát pro sledování novinek na webových stránkách
2. komplexní grafické prostředí, http://www.kde.org
3. intuitivní grafické prostředí, http://www.gnome.org

možnosti. Krok za krokem je popsán návod jak si vytvořit model pro získávání obsahu. Následující část Implementace klientů nás seznamuje se situací miniaplikací v prostředích KDE a GNOME a představuje prostředky pro jejich vývoj.

2 Současné miniaplikace

Následující kapitola se snaží identifikovat některé problémy a jejich řešení, které by měl poskytnout náš systém. Na závěr jsou krátce představeny výsledky jiných autorů jako alternativní východiska.

2.1 Instalace miniaplikací

Prvním problémem je (obrazně řečeno) samotná existence miniaplikací. Uživatel si přeje mít neustále k dispozici například aktuální kurzy měn, či program místního kina. Instalace miniaplikace je až prostředek, jak toho dosáhnout. Moderní grafická prostředí, či systémy, které umožňují běh miniaplikací, se snaží instalaci samozřejmě co nejvíce usnadnit. Uživateli často stačí přetáhnout balík s miniaplikací do příslušného okna, či – například v KDE ve verzi 4.3 – lze instalovat miniaplikace přímo online po vybrání z automaticky generovaného aktuálního seznamu nových miniaplikací.

Obrázek 2.1: instalace QuickUrl, chybí balíček plasma-scriptengine-python

Přesto instalace není – především pro běžného uživatele – vždy snadná (obrázek 2.1), velká obliba webových aplikací, které instalaci nevyžadují, je (nejen) toho důkazem. Navíc software instalovaný z podepsaných repozitářů distribuce, poskytuje jistou úroveň záruky bezpečnosti. Desítky miniaplikací instalovaných z různých zdrojů již nejsou tak důkladně prověřeny[1]. Další problém nastává při aktualizacích miniaplikací, změní-li se

1. http://www.root.cz/zpravicky/malware-skryty-v-setrici/

například struktura stránky, ze které miniaplikace získává fotografii dne[2], uživatel musí aktualizovat miniaplikaci (či v tomto případě systém poskytující jí data – DataEngine [17]).

Řešením by byla jedna univerzální miniaplikace umožňující zobrazení libovolného *obsahu* (slovo obsah bude nadále používáno jako data, které zobrazuje naše univerzální miniaplikace). Použití jen jedné miniaplikace minimalizuje instalace. Logiku získávání a zpracování obsahu přeneseme na server poskytující obsah klientům (miniaplikacím), uživatel tak nemusí při změně ve zdroji dat aktualizovat miniaplikaci, ale jen čeká, až tak učiní správci serveru.

Dostáváme tak architekturu klient–server. Zatím jsme si popisovali výhody z hlediska uživatele, podstatný je i pohled vývojáře – čím jednodušší vývoj, tím více obsahu pro miniaplikaci a tím lépe pro uživatele. Oddělením serverové části a tím, že neběží u uživatele, si může programátor nainstalovat libovolné knihovny a podpůrné prostředky pro vývoj, či provádět složitá zpracování dat. Uživatel je od všeho odstíněn a dostane pouze hotová data.

Negativní na tomto modelu je nutnost provozování serveru, při řešení naší bakalářské práce jsme využili neplacených služeb blíže popsaných v podkapitole 3.9.

2.2 Roztříštěnost miniaplikací

Dalším nepříjemným jevem na současném stavu je roztříštěnost systémů pro provoz miniaplikací. Plasmoidy, Conky, Screenlety, (Super)Karamba, Google Gadgets, gDesklets – jsou pouze některé používané[3] v Linuxu.

Vývojář, chtějící svůj obsah poskytovat co největšímu množství uživatelů, musí vytvořit miniaplikaci zvlášť pro každý z těchto systémů.

Řešením je rozdíly mezi těmito systémy odstínit. Velké množství miniaplikací zobrazuje v podstatě jen výsek existující webové stránky nebo

2. http://kde-look.org/content/show.php/Photo+of+the+Day?content=104631
3. seřazeno podle počtu uživatelů dle ankety na http://www.abclinuxu.cz/ankety/miniaplikace-na-plochu

data, která je možno na HTML[4] převést. Možnosti formátování HTML jsou poměrně bohaté (tabulky, CSS[5]) a dostačující pro většinu informací. Navíc komponenta webového prohlížeče se dnes stává standardní součástí systémů pro provoz miniaplikací či příslušných knihoven.

Programování univerzální aplikace pro každý systém zvlášť se nevyhneme, díky serverové části však již nemusíme opakovaně implementovat získávání obsahu.

Navržená serverová část umí poskytovat obsah v různých formátech, automaticky umí vytvářet například HTML z XML[6] dokumentu, či čistý text z HTML dokumentu, další formáty je možné poměrně jednoduše přidávat. Programátorovi stačí uložit obsah ve formátu XML a server mu automaticky vytvoří další formáty. Miniaplikace (uživatel) si pak vybere formát, který je vhodný pro jeho miniaplikaci (například pro jednoduché informace prostý text, pro miniaplikaci chtějící zobrazovat informace v nativních widgetech[7] pak XML). Více informací nalezneme v podkapitole 3.6.

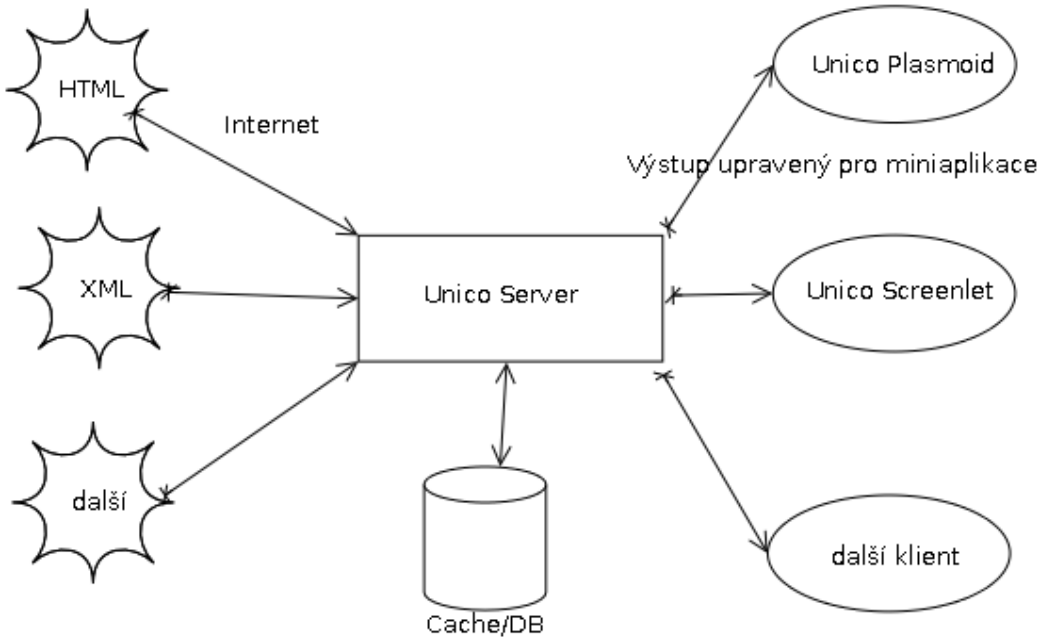
Název celého klient–server projektu byl zvolen Unico (ze španělského slova *único* znamenající *jediný*, *jedinečný*, *mimořádný*, či si unico můžeme vykládat jako spojení anglických slov *uni*ted *co*ntent – sjednocený obsah). Server označujeme jako Unico Server, miniaplikaci Unico klient, či konkrétně pro prostředí KDE jako Unico Plasmoid a pro GNOME Unico Screenlet. Obrázek 2.2 ilustruje Unico jako celek.

Unico se nesnaží nahradit veškeré miniaplikace – pro komplexní, s uživatelem interagující rozhraní nebo pro zobrazení jakýchkoli lokálních dat bude vždy výhodnější použití speciálních miniaplikací.

---

4. jazyk pro popis webových stránek, http://www.w3.org/TR/html4/
5. Cascading Style Sheets – kaskádové styly, http://www.w3.org/Style/CSS/
6. rozšiřitelný značkovací jazyk, http://www.w3.org/XML/
7. ovládací prvek, část grafického prostředí, http://en.wikipedia.org/wiki/GUI_widget

Obrázek 2.2: diagram systému Unico



2.3 Alternativní řešení

2.3.1 WebPage subset or HTML Widget

Plasmoid s názvem *WebPage subset or HTML Widget*[8] umožňuje zobrazovat zadanou webovou stránku, upravovanou na straně klienta JavaScriptem[9], konkrétně je využita knihova jQuery[10]. Není tak potřeba serveru.

První slabinu spatřuji v komplikovanosti pro uživatele, běžný uživatel není programátor a není schopen napsat potřebný kód. Ladění JavaScriptu nemusí být vždy snadné a programátorům je doporučeno použít například Firebug[11]. Tato překážka k zpřístupnění programu běžným uživatelům

8. http://kde-look.org/content/show.php/WebPage+subset+or+HTML+Widget?content=115536
9. skriptovací jazyk používám především ve webových prohlížečích, http://en.wikipedia.org/wiki/Javascript
10. http://www.jquery.com/
11. rozšíření do prohlížeče Firefox umožňující mimo jiné pohodlnější ladit JavaScript http://getfirebug.com/

by mohla být odstraněna vybudováním centrálního úložiště kusů kódů v jQuery určených pro úpravu jednotlivých stránek, ze kterých by si poté uživatel jednoduše volil.

Na uživatelův počítač jsou kladeny vyšší nároky na výpočetní výkon, jelikož zpracování stránky probíhá u něj. Dnešní počítače jsou však výkonné natolik, že to nepředstavuje problém.

Poslední nevýhodu spatřuji ve složitějším sdílení obecných funkcí mezi programátory. V Unico Serveru má programátor přístup např. k pomocným metodám pro vytvoření slideshow z obrázků pouhým zděděním třídy.

Plasmoid je poměrně mladý (15. listopadu 2009 zveřejněna první verze 0.1) a je tedy možné, že se funkcionalita bude rozšiřovat [10].

2.3.2 KDE DataEngine

Jako řešení dílčího problému opakovaného psaní kódu pro získávání dat se jeví DataEnginy z KDE, více informací v části 4.1.3. Bohužel jejich použití je v současné době omezeno pouze na KDE.

3 Unico Server

Unico Server je naprogramován ve webovém open-source (licence MIT[1]) frameworku[2] Ruby on Rails [8] (dále RoR). RoR používá architekturu Model–View–Controller (MVC)[3] pro organizaci struktury aplikace, pro oddělení logiky a prezentace [26]. Jako programovací jazyk frameworku je použit již z názvu patrný jazyk Ruby[4].

Jelikož je framework RoR určen pro vývoj webových aplikací, obsahuje (či lze jednoduše doinstalovat) množství knihoven hodících se pro účely Unico Serveru. Například knihovna Hpricot (více v podkapitole 3.4) pro zpracování HTML či knihovna REXML pro manipulaci s dokumenty XML. Každá aplikace vytvořená pomocí RoR ve výchozím stavu obsahuje webový server, potencionální vývojář či tester ho tak nemusí instalovat a konfigurovat (samozřejmě lze použít například i populární Apache[5] ve spojení s Phusion Passenger[6]).

Pro porozumění (alespoň rámcové) následujícího textu nejsou vyžadovány znalosti frameworku Ruby on Rails, pro zájemce pak doporučuji knihu *Agile Web Development with Rails* [21].

3.1 Adresářová struktura

Adresářová struktura Unico Serveru se ve své podstatě nijak neliší od jiné RoR aplikace. Všechny cesty jsou uváděny relativně od kořene serveru – od adresáře `unico`. Pro případné vývojáře jsou zajímavé především adresáře `app`, `config`, `lib/tasks` a `public`. Je-li naším úmyslem pouze přidání nového obsahu pro miniaplikace, můžeme přeskočit na podkapitolu 3.2.

adresář `app`

V adresáři `app` je drtivá většina zdrojových kódů Unico Serveru. Obsahuje podadresáře `controllers`, `helpers`, `models` a `views`.

---

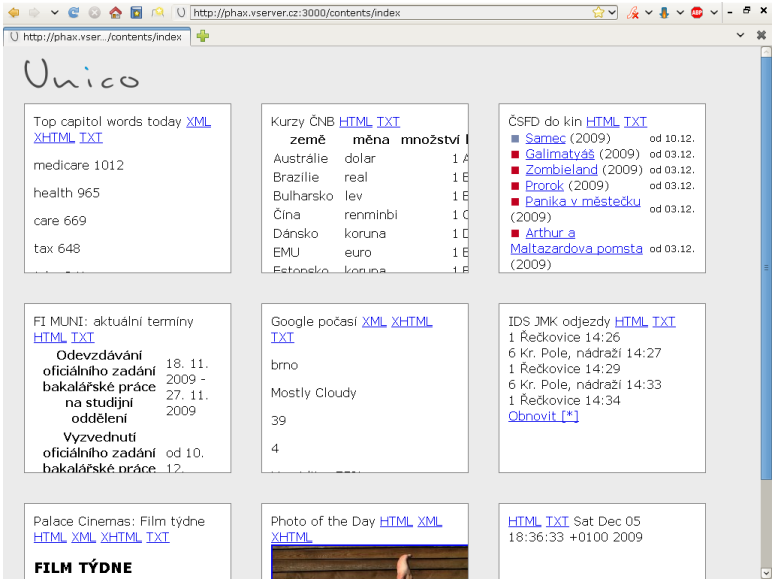
1. http://www.opensource.org/licenses/mit-license.php
2. *„je softwarová struktura, která slouží jako podpora při programování a vývoji a organizaci jiných softwarových projektů“* [32]
3. http://en.wikipedia.org/wiki/Model-View-Controller
4. http://www.ruby-lang.org/en/
5. http://www.apache.org/
6. http://www.modrails.com/

`controllers`

Řadiče[7] (controllers) v architektuře MVC přijímají vstup od uživatele (např. zadání URL) a poté provedou příslušné akce nad modelem.

V adresáři `controllers` se nachází soubor `contents_controller.rb`, který je hlavním řadičem celého Unico Serveru. Nejpodstatnějšími metodami jsou `index` a `show`, mající za úkol předat příslušná data šablonám na zobrazení stránek; `index` na zobrazení přehledů všech obsahů pro miniaplikace (obrázek 3.1); `show` pak konkrétního obsahu ve vyžádaném formátu. Pomocná metoda `layout` zajistí použití vyžádaného layoutu (rozvržení) pro daný obsah.

Obrázek 3.1: přehled všech obsahů – stránka index



`views`

Pohledy slouží k formátování a prezentaci dat uživatelům. Adresář `views` obsahuje dva podadresáře `layout` a `views`.

7. Překlad české Wikipedie (http://cs.wikipedia.org/wiki/Model-view-controller), lze se setkat i s pojmem „kontrolér“

V adresáři `layout` jsou uloženy layouty (rozvržení) stránek, v souboru `application.html.erb` je výchozí layout pro HTML soubory. Jedná se o jednoduchý soubor, obsahující jen nezbytnou HTML hlavičku a příkaz `<%= yield %>`, jež je nahrazen příslušným vyžádaným obsahem. Layout `index.html.erb` slouží k zobrazení přehledu všech obsahů. Soubor `slideshow.html.erb`, jehož autorem je Kevin Liew[8] (vydáno pod licencí Creative Commons Attribution 3.0[9]), obsahuje nezbytné věci (JavaScript, CSS) k vytvoření obrázkové slideshow.

Soubory v adresáři `contents` se starají o zobrazování samotného obsahu (jež je potom ještě obalen příslušným layoutem). Začínají názvem `show` (to odpovídá metodě `show` viz. podkapitola 3.1) a příponou podle vyžádaného obsahu (xml, html, txt, xhtml). Dále se zde nachází soubory začínající jménem `_juggernaut`, jež se vloží do stránky při povolení push aktualizací (podkapitola 3.7).

V souborech `_juggernaut` jsme využili „firebug console faker“ (pro simulaci ladícího rozšíření Firebug) z blogu Dora Kaleva[10], bez toho prohlížeč na každou proběhnutou push aktualizace upozorňoval vyskakovacím oknem.

`models`

Modely jsou abstrakce dat naší aplikace.

V naší aplikaci jsou data obsahy pro miniaplikace, implementace získávání obsahů se tedy nachází zde, v adresáři `app/models`. Každý obsah je v samostatném souboru (modelu), například soubor `cnbkurzy.rb` reprezentuje obsah „Kurzy České národní banky“. V modelu se typicky načte nějaký zdroj dat (např. webová stránka), upraví se (vyparsují se z ní jen podstatné informace) a uloží se do databáze. Podrobné informace jak model vypadá a jak si napsat vlastní se nacházejí v podkapitole 3.2.

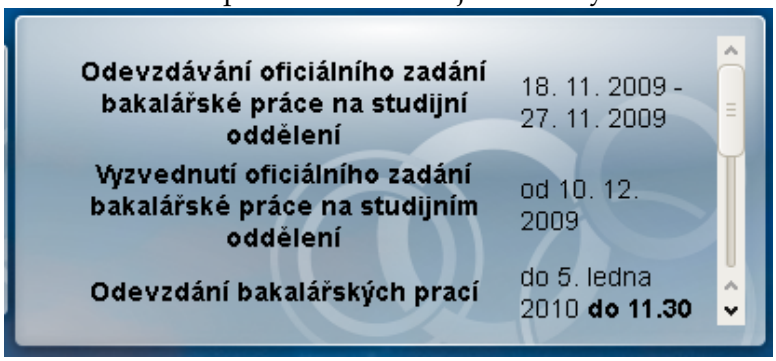
---

8. http://www.queness.com/post/152/simple-jquery-image-slide-show-with-semi-transparent-caption
9. http://www.creativecommons.cz/
10. http://www.dorkalev.com/2008/04/juggernaut-up-to-date-tutorial-1042008.html

3.2 Vytvoření vlastního modelu

V této části si ukážeme, jak vytvořit model pro získáváni obsahu. Ukázkový příklad (výsledek pro uživatele na obrázku 3.2) bude zobrazovat tabulku aktuálních termínů na Fakultě informatiky Masarykovy univerzity[11], půjde si zvolit, zda-li chceme zobrazit termíny pro bakalářské či magisterské studium.

Obrázek 3.2: plasmoid zobrazující termíny FI MUNI



Kostru pro vytvoření nového obsahu můžeme nalézt v souboru `app/model/sablona.rb`, jež slouží jako počáteční bod pro každý nový model

Obsah souboru `fiterminy.rb` vidíme na výpisu na obrázku 3.3, jelikož je kód podobný pro všechny modely a ukazuje poměrně dobře část struktury Unico Serveru, vysvětlíme kód řádek po řádku.

3.2.1 řádek 1

Každý obsah dědí třídu `Content` – tím získáme metody (funkce) pro usnadnění vývoje. Jelikož je třída `Content` potomek třídy `ActiveRecord::Base` při ukládání se data mapují do databáze (viz. podkapitola 3.3). Název souboru `fiterminy.rb` a název modelu `class`

11. http://www.fi.muni.cz/studies/dates.xhtml

Obrázek 3.3: model pro termíny na FI MUNI

```
1  class Fiterminy < Content
2
3    def self.parse_content(query = {})
4      content = Fiterminy.new
5      content.name_human = "FI␣MUNI:␣aktualni␣terminy"
6
7      rawhtml = Hpricot(open("http://www.fi.muni.cz/
           studies/dates.xhtml"))
8      rawhtml = rawhtml.search("//table")
9      #logger.fatal rawhtml.inspect
10
11     if (query["mgr"] == "yes")
12       content.rawhtml = rawhtml.second.to_s
13     else
14       content.rawhtml = rawhtml.first.to_s
15     end
16
17     content.save_me(query)
18   end
19 end
```

`Fiterminy < Content` si musejí odpovídat (název modelu ale začíná velkým písmenem).

3.2.2 řádek 3

Jediná povinná metoda na implementaci je `parse_content`. Z její celé hlavičky (`self.parse_content(query = {})`) lze vyvodit, že je jí předáván parametr `query` a nebude-li definován, uloží se do něj prázdné asociativní pole. Asociativní pole `query` nám je automaticky předáváno řadičem a obsahuje všechny proměnné (název proměnné však nesmí být `id`, `format`, `action` či `controller`) z URL zadané uživatelem, například: `http://example.cz/promena1=hodnota1&promena2=hodnota2`. Proměnné z této URL budou dostupné takto: `{"promena1"=>"hodnota1", "promena2"=>"hodnota2"}`. Povolené hodnoty obsahu proměnných jsou pouze znaky, čísla a podtržítko (`[A-Za-z0-9_]`). Proměnné, jejichž obsah nevyhovuje, řadič metodě

nepředá. Jedná se o bezpečnostní opatření, aby každý programátor modulu nemusel ošetřovat potencionálně nebezpečné znaky. V dalších verzích Unico Serveru by tato filtrace mohla být volitelná.

Takto flexibilní formát parametrů od uživatele byl zvolen, aby mohlo být využito existujících URL pro různé služby beze změny. Hledá-li uživatel např. odjezdy ze zastávek pomocí URL s proměnnými `stopId=1166&poleId=1` stačí, aby programátor Unico modulu využil stávajícího pojmenování a uživatel pouze zkopíruje původní parametry.

3.2.3 řádek 4 – 5

Následuje vytvoření instance třídy `Fiterminy` pomocí metody `new` (`content = Fiterminy.new`). Do atributu `human_name` uložíme výstižný popisek obsahu, například „FI: aktuální termíny“.

3.2.4 řádek 7 – 8

Na řádku sedmém načteme URL s pomocí knihovny Hpricot, více informací o jejím používání najdeme v podkapitole 3.4. Knihovna Hpricot nám umožňuje pohodlnou manipulaci s logickými celky souboru HTML. Na řádku osmém využijeme metodu `search` pro nalezení všech tabulek (tag `table`) v HTML.

3.2.5 řádek 9

Tento řádek řádek ukazuje logování pomocí standardní Ruby knihovny[12].

3.2.6 řádek 11 – 15

Zadá-li uživatel URL s parametrem `mgr=yes` zvolíme druhou tabulku (termíny pro magisterské studium), v jiném případě jsou zvoleny termíny bakalářského studia. Metodou `to_s` převedeme objekt typu Hpricot na textový řetězec (string).

3.2.7 řádek 17

Zavoláme metodu `save_me` (implementována v modelu `Content`), ta zkontroluje, liší-li se získaná data od dříve uložených a v případě, že ano, jsou uloženy do databáze.

12. http://www.ruby-doc.org/stdlib/libdoc/logger/rdoc/

3.2.8 inicializace dat nového modelu

Tímto je náš model hotov, jelikož řadič `Contents` kontroluje, jestli obsah požadovaný uživatelem je v databázi (není-li vrátí chybovou zprávu o nenalezení stránky), musíme poprvé zavolat metodu `parse_content` ručně z konzole `script/console`: `Fiterminy.parse_content`

3.3 Kešování dat, struktura databáze

Jelikož stejný obsah může v krátkém časovém rozmezí požadovat několik uživatelů, je většinou zbytečné, abychom vždy stahovali zdrojovou stránku a zpracovávali ji. Může se jednat o časově i zdrojově náročnou operaci. Místo toho využíváme tzv. kešování (caching), kdy si výsledek požadovaný prvním uživatelem uložíme a na další požadavky odpovídáme již výsledky z keše, které jsou velice rychle dostupné. Po nějaké době výsledky v keši zastarají a přemažou se opět plnohodnotně získanými daty.

Jako keš využíváme open-source databázi MySQL[13], potřebujeme pouze velice omezenou množinu standardních dotazů SQL, nebyl by tedy problém použít databázi jinou[14].

V tabulce 3.1 vidíme strukturu tabulky `contents`, která slouží k ukládání obsahů.

Tabulka 3.1: struktura tabulky contents

jméno sloupce	popis	obsah
id	pomocné id	1
name	jméno modelu	fiterminy
rawhtml	HTML získané parsováním stránky	<table><tr>...
xml	XML dokument (podkapitola 3.5)	NULL
created_at	kdy byl záznam vytvořen	2009-11-12 10:57:59
updated_at	kdy byl záznam aktualizován	2009-11-12 18:17:10
name_human	popisek obsahu	FI MUNI: aktuální termíny
xhtml	XHTML vzniklé transformací XML	NULL
plaintext	čistý text vzniklý z HTML	Odevzdávání oficiálního...

13. http://www.mysql.com/
14. http://guides.rubyonrails.org/getting_started.html#configuring-a-database

Dotazy, ve kterých není prázdná proměnná `query`, se nekešují. Proměnnou `query` nastavuje uživatel a každé její hodnotě může odpovídat jiný obsah, databáze by tak mohla nekontrolovaně narůstat záznamy užitečnými pouze pro jednoho uživatele.

3.4 Hpricot – HTML parser

Zpracovávání webových stránek je při získávání obsahů často řešenou úlohou. Hpricot je knihovna pro parsování (syntaktickou analýzu) HTML souborů, s její pomocí můžeme jednoduše získávat informace, jako například nalezení všech odstavců v HTML dokumentu (`search("//p")` nebo získání obsahu nadpisu úrovně tři (`at("//h3").inner_text`). Další příklady a informace nalezneme na stránce projektu Hpricot na githubu[15]. [5] Dalším zdrojem příkladů jsou také modely Unico Serveru.

3.5 XML

XML je *rozšiřitelný značkovací jazyk*, jedná se o vhodný nástroj pro vytváření *„konkrétních značkovacích jazyků (tzv. aplikací) pro různé účely a různé typy dat“*[31]. Mezi jeho výhody patří jednoduchá transformace na jiné typy dokumentů.

XML je na rozdíl od HTML obecnější, HTML v miniaplikacích přímo zobrazíme skrze komponentu webového prohlížeče, žádné další úpravy neprovádíme. XML je vhodnější pro popis struktury, obsahu.

Jelikož by měl Unico odstiňovat rozdíly mezi miniaplikacemi, formát XML se jeví jako vhodný kandidát. Navržená struktura a použití formátu XML především demonstruje možnosti využití a přípravu na budoucí rozšíření.

V povinné kořenové značce `document` používáme tagy `label` pro textové pole a `image` pro obrázky (respektive URL) – příklad vidíme na obrázku 3.4. Přidat další tagy je poměrně snadné (podkapitola 3.6). Unico Plasmoid, který v současné době jako jediný využívá XML, zobrazuje obsah tagů v nativních QT/Plasma widgetech (podrobněji v sekci 4.1.2).

15. http://github.com/whymirror/hpricot

Obrázek 3.4: ukázka xml souboru

```
1 <document>
2   <label>brno<label/>
3   <label>Overcast</label>
4   <label>39</label>
5   <image>http://www.google.com/images/weat/cloudy.gif<
        /image>
6   <label>Wind: SE at 7 mph</label>
7 </document>
```

Jedním z možných využití je tedy zobrazení informací z XML v nativních widgetech miniaplikace. V současné době jsou u obou nejvíce používaných toolkitů v Linuxu (GTK+[16] a QT[17]) preferovány XML soubory pro návrh GUI. Byly prozkoumány výstupní XML soubory z programu Glade[18] (návrhář GTK+ GUI) i QT Designer[19] (návrhář QT GUI), ale jejich syntax je příliš komplexní a pro naše potřeby příliš komplikovaná.

Pro vytváření XML nám Unico Server poskytuje metodu `create_xml_head` pro vytvoření kořenové značky a pomocí metody `create_xml_body(entity_name, entity_text)` tvoříme tělo dokumentu – libovolně opakující se značky `label` a `image`. Parametr `entity_name` pak udává jméno entity (`label` nebo `image`) a `entity_text` obsah elementu.

3.6 Automatické vytváření formátů

Unico Server se snaží o poskytování obsahu co nejvíce systémům pro miniaplikace. Pro různá použití se hodí různé formáty – v grafické miniaplikaci na výkonném stroji oceníme HTML formátované pomocí CSS s JavaScriptem pro efektní přechody, na pomalém stroji, kde se nám pouze vypisují textové informace na plochu, uvítáme čistý text.

Tato kapitola nás seznámí s tím, jaké formáty musí vytvořit programátor a jaké budou automaticky vytvořeny. V podkapitole 3.2 jsme viděli,

16. http://www.gtk.org/
17. http://qt.nokia.com/products
18. http://glade.gnome.org/
19. http://qt.nokia.com/products/appdev/developer-tools/developer-tools

že HTML získané zpracováním webové stránky se ukládá do atributu `rawhtml` („raw“ z anglického syrový, značící že takto získané HTML bude pouze nevalidní výsek). Druhým[20] atributem, který plníme daty, je `xml`, ten plníme nepřímo pomocí metod popsaných v podkapitole 3.5. Pro větší názornost je vždy automaticky vytvářený formát a jeho zdroj zachycen v tabulce 3.2.

Tabulka 3.2: automaticky vytvářené formáty

programátor	automaticky	
	xhtml	plaintext
rawhtml	NE	ANO
xml	ANO	ANO

Čistý text (plaintext) vzniká z HTML či XHTML, o tuto funkci se stará metoda `html2txt` v modelu `Content`. Z dat odstraní všechny HTML značky, dekóduje HTML entity[21] – například „`>`“ je převedeno na „>“ (využíváme gem `htmlentities`[22]). Dále jsou případné konce řádků používané v DOS/Windows (`\n\r`) převedeny na unixové(`\n`). Následně je odstraněno přebytečné místo tvořené netisknutelnými znaky[23].

XHTML se vytváří XSL (eXtensible Stylesheet Language) [33] transformací z XML, postup znázorňuje obrázek 3.6. Při XSL transformaci je potřeba zdrojový XML dokument (formát popsán v podkapitole 3.5) a *„vzorec pro transformaci“*[33], jež je zapsán v jazyce XSL. Pro transformaci je volána metoda `transform_xml2html` nacházející se v modelu `Content`.

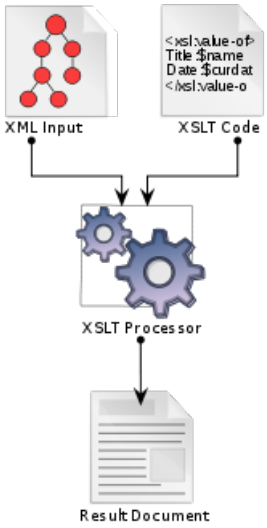
Obsah XSL souboru (nacházející se v adresáři `public/`) vidíme na výpisu na obrázku 3.5. Na řádcích 9 – 11 převádíme tag `label` na tag `p`, což je v (X)HTML značka pro odstavec. Na řádcích 12 – 18 použijeme obsah elementu `image` jako parametr atributu `src` v (X)HTML tagu `image`. Narazí-li se při transformaci na neznámou značku, je ignorována.

20. třetím a posledním atributem, kde programátor ukládá nějaká data, je `name_human` pro popis obsahu
21. Přehled HTML entit, http://www.jakpsatweb.cz/html/entity-vsechny.html
22. http://htmlentities.rubyforge.org/
23. http://en.wikipedia.org/wiki/Whitespace_%28computer_science%29

Obrázek 3.5: vzorec pro XSLT

```
1  <?xml version="1.0"?>
2  <html xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xsl:version="1.0">
3
4    <head></head>
5    <body>
6
7    <xsl:for-each select="document/*">
8        <xsl:choose>
9          <xsl:when test="name()='label'">
10           <p><xsl:value-of select="."/></p>
11         </xsl:when>
12         <xsl:when test="name()='image'">
13           <xsl:element name="img">
14             <xsl:attribute name="src">
15               <xsl:value-of select="."/>
16             </xsl:attribute>
17           </xsl:element>
18         </xsl:when>
19       </xsl:choose>
20     </xsl:for-each>
21
22 </body>
23 </html>
```

Obrázek 3.6: XSL transformace [28]



3.7 Push aktualizace

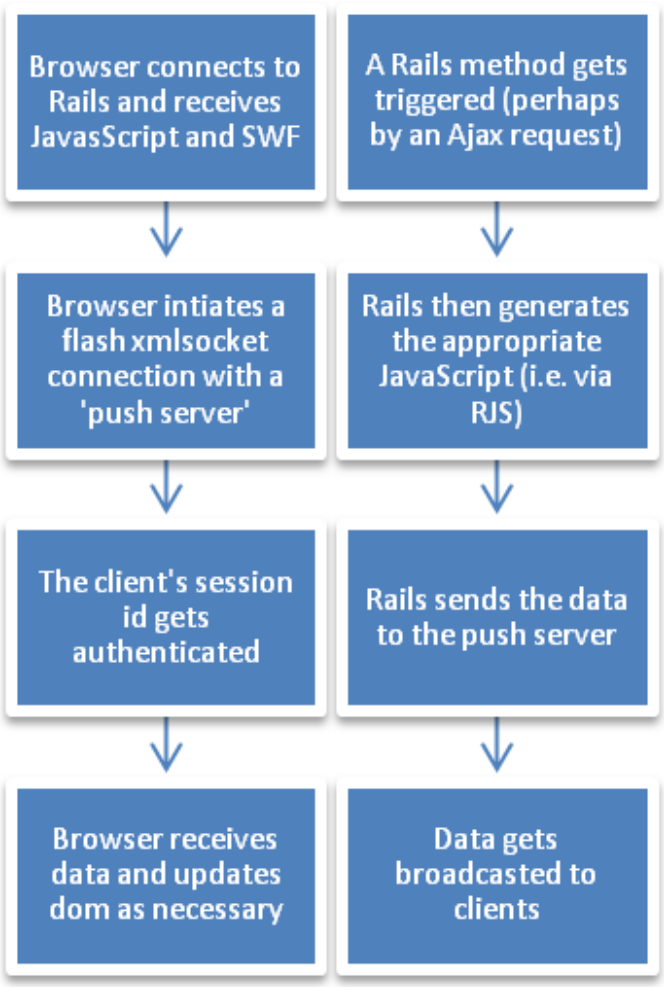
Pro získání nového obsahu se miniaplikace (klient) periodicky dotazuje serveru – toto se nazývá *polling* (aktualizace). Nevýhodou je prodleva mezi dostupností nového obsahu na serveru a jeho doručením klientovi – což je v nejhorším případě interval nastavený pro opakované stahování nového obsahu (typicky minuty).

Při *push* aktualizaci je to server, kdo inicializuje spojení. Jakmile má k dispozici nová data, pošle zprávu všem klientům, kteří o ně mají zájem. Odpadá čekání či zatěžování linky častými dotazy. Lze se setkat i s terminologií, že se klient *přihlásí k odběru* (subscribe) kanálu (channel). [25]

Pro implementaci push technologie do Unico Serveru byl využit projekt Juggernaut [20]. V našem případě probíhá komunikace v principu takto: miniaplikace si stáhne stránku, která obsahuje JavaScript a SWF (formát Adobe Flash[24]). Prohlížeč inicializuje spojení skrze xmlsocket [27] a spojí se s push serverem (Juggernaut server). Skrze tento socket posíláme klientovi data (v našem případě JavaScript s příkazem pro obnovení stránky). Detailněji na obrázku 3.7.

24. http://en.wikipedia.org/wiki/Adobe_Flash

Obrázek 3.7: průběh push komunikace [20]



Juggernaut je dostupný ve formě pluginu pro Ruby on Rails (ke své činnosti ještě potřebuje gemy[25] Json a EventMachine). Ve výchozím nastavení jsou push aktualizace zakázány, jelikož jsme neměli prostředí k jejich otestování v reálném provozu.

25. RubyGems – standardní formát balíčků pro distribuci programů a knihoven v Ruby [3]

3.7.1 Push aktualizace v Unico Serveru

Prvním krokem je povolení push aktualizací změnou v souboru `config/environment.rb`, je třeba změnit hodnotu konstanty `PUSH_ENABLE` na `true` (a restartovat RoR server). Tato volba se v kódu Unico Serveru kontroluje na dvou místech. Za prvé její povolení způsobí vkládání příslušného pomocného JavaScriptu do HTML obsahů (viz. podkapitola 3.1), tímto se miniaplikace přihlásí k určitému kanálu. Na druhém místě se její hodnota kontroluje, kdykoliv se stáhne nový obsah a liší se od staršího (což v podstatě znamená, že je novější a chceme tuto skutečnost sdělit miniaplikacím).

Liší-li se obsah od staršího a máme-li povoleny push aktualizace, provede se příkaz `Juggernaut.send_to_channel ("location.reload();", [self.class.name])`. Metodě `send_to_channel` se předávají jako první parametr data, které chceme zaslat klientům – zde řádek kódu v JavaScriptu `location.reload();` na opětovné načtení stránky. Druhý parametr udává, jakému kanálu chceme data zaslat, zde je to určeno jménem modelu obsahu. Zobrazuje-li miniaplikace obsah *Počasí Google* a jsou povolené push aktualizace, bude přihlášena ke kanálu `googlepocasi`.

Jak jsme zmínili výše, Juggernaut využívá svůj vlastní server, ten spustíme příkazem `juggernaut -c juggernaut.yml`[26] v kořenovém adresáři Unico Serveru. V souboru `juggernaut.yml` se nachází jeho konfigurace, můžeme nastavit na jakém portu má push server komunikovat (výchozí port je 5001). Výchozí a doporučená je autentizace pomocí IP adres, IP adresa klienta musí být uvedena v direktivě `:allowed_ips`, aby mohla miniaplikace komunikovat s push serverem.

Po těchto krocích jsou push aktualizace funkční. Problém nastává s četností aktualizace obsahu – server sám nic nespouští, stažení nového obsahu nastává pouze tehdy, vyžádá-li si ho některá miniaplikace (pull). Při velkém provozu je možné se spolehnout, že klientů žádajících periodicky obsah bude dostatek (tj. nebudou všichni využívat push aktualizace) a zajistí nám to rozumné aktualizace i pro klienty čekající na obsah metodou push. Druhá možnost je periodicky spouštět připravený Rake [2] task, více o Rake tasks v podkapitole 3.8.

---

26. nemáme-li adresář s gemy v proměnné `PATH`, uvedeme plnou cestu, např. `/var/lib/gems/1.8/bin`

3.8 Rake tasks (úlohy)

Rake je systém podobný známějšímu make[27], sloužící pro automatizaci úloh. Rakefile, soubor pro sestavovací (buildovací) systém Rake, (obdoba Makefile) je členěn na úlohy (tasks). Úlohy jsou části kódu v Ruby, mezi kterými můžeme definovat závislosti.

3.8.1 rake push

V adresáři `lib/tasks` se nachází soubor `push.rake`, jež obsahuje úlohu pojmenovanou `push`. Při spuštění `rake push` (v kořeni Unico Serveru) se potupně projdou všechny obsahy v databázi a pokusí se stáhnout nový obsah. V unixových systémech můžeme naplánovat spouštění například pomocí Cronu [30] a docílit tak pravidelného provádění push aktualizací popsaných v podkapitole 3.7. Jelikož *push rake* spouští aktualizaci pro obsahy uložené v databázi, push aktualizace se spouštějí pouze pro obsahy s uživatelem nedefinovaným `query` (viz. podkapitola 3.3).

3.8.2 rake downloadall

Tato úloha slouží k prvotnímu naplnění databáze (tabulky `contents`) při instalaci Unico Serveru, odpadá tak ruční volání metody `parse_content` jako v sekci 3.2.8. `downloadall` zavolá metodu `parse_content` pro každý model.

3.9 Testovací provoz Unico Serveru

Pro provoz Unico Serveru jsme využili RoR hostingu firmy KRAXNET s.r.o.[28] a virtuální server v programu START od firmy PIPNI s.r.o.[29]. Provoz služeb je zdarma a není garantovaná dostupnost či funkčnost.

Unico Server by měl být k dispozici na adresách
`http://phax.vserver.cz:3000/contents/index`
a `http://unico.free.railshosting.cz/contents/index`.

27. http://www.gnu.org/software/make/
28. http://www.railshosting.cz/railshosting/
29. http://new.pipni.cz/vindex.phtml

4 Implementace klientů

V této kapitole si postupně krátce představíme dvě nejpoužívanější desktopová prostředí v Linuxu a seznámíme se s vývojem miniaplikací pro ně. Naším cílem bylo vytvořit miniaplikace pro zobrazování obsahu poskytovaného Unico Serverem.

4.1 KDE

KDE je grafické prostředí běžící nejen v Linuxu. KDE je open-source a distribuováno pod svobodnými licencemi (přehled licencí [6]). Pojem KDE je zastřešující projekt pro dokumentaci, programy, uživatele i vývojáře [11]. KDE je napsáno s využitím knihovny QT [9].

4.1.1 Plasma

11. ledna 2009 byla vydána verze KDE 4 [22]. Jednalo se o vydání přinášející spoustu technologických změn a vylepšení. Vedle nových frameworků jako Phonon (multimédia), Solid (abstrakce hardware) či Strigi (vyhledávání) byla nejviditelnější Plasma. Jedná se o kompletní přepracování desktopového shellu, jeho sjednocení. Klasické položky desktopu (Wikipedií nazýváno „*Desktop metaphor*“ [23]) jako panel či ikony na ploše jsou nyní sjednoceny v tzv. Plasma widgety (Plasmoidy) [7].

4.1.2 Vývoj plasmoidu

Následující část práce čtenáře stručně seznámí s problematikou vývoje plasmoidu a s její pomocí bude čtenář schopen napsat jednoduchý plasmoid (testováno na KDE 4.3). Text se nesnaží být vyčerpávající, je spíše návodem a odkazem na další zdroje. Návod vychází především z tutoriálů na portálu KDE: TechBase.

Dokumentace

Rozcestník pro vývoj v Plasmě se nachází na `http://techbase.kde.org/Projects/Plasma`. V současné době lze kód plasmoidu psát v jazycích C++, JavaScript, Python a Ruby, přičemž C++ je nepsaným stan-

dardem a při použití jiných jazyků budete odkazováni například na API[1] psané v C++. Autor práce si zvolil jazyk Ruby.

Základním zdrojem pro úvod do programování plasmoidů v Ruby jsou tutoriály na portálu KDE TechBase[2]: Getting Started [15] (začínáme), Using widgets [14] (používání widgetů) a Simple Paste Applet [13] (jednoduchý applet pro vkládání). Nepostradatelná pro získání celkového přehledu nad Plasmou, seznamem tříd a metod, je stránka s Plasma API [1]. API je určeno pro jazyk C++ a abychom mohli používat dané programovací konstrukce v Ruby, je třeba provést určité změny, jež jsou zachyceny v obsáhlém přehledu na stránce portálu KDE TechBase – Vývoj v jazyku Ruby [12].

Užitečným zdrojem dalších informací mohou být e-mailové konference[3] (mailing list) KDE, především *plasma-devel* a *kde-bindings* (pro jiné programovací jazyky než C++). Přístupné jsou rovněž jejich archívy.

Vynikajícím zdrojem příkladů je stránka `www.kde-look.org`, kategorie Plasmoids, kde je zhruba přes 200 plasmoidů, jejichž zdrojové kódy mohou sloužit jako inspirace.

Jako zdroj v češtině je vhodný článek ze série o KDE 4 na portálu `www.root.cz`, KDE 4.2: vývoj plasmoidu pro Twitter [34] (v jazyce Python).

Potřebné knihovny

Potřebné knihovny pro vývoj a běh plasmoidu byly testovány v distribuci Linuxu Ubuntu[4] 9.10. Prostředí KDE jsme nainstalovali balíčkem `kde-minimal`, balíček `plasma-scriptengine-ruby` s následujícími závislostmi doinstaluje další potřebné balíčky pro vývoj a běh plasmoidu v Ruby:
`libkorundum4-ruby1.8 libqscintilla2-5 libqt4-ruby1.8 libruby1.8 libsmokekde4-2 libsmokeqt4-2 ruby1.8`.

1. http://cs.wikipedia.org/wiki/API
2. http://techbase.kde.org/Welcome_to_KDE_TechBase
3. http://www.kde.org/mailinglists/
4. http://www.ubuntu.com/

Struktura plasmoidu „Hello World“

Plasmoidy dodržují následující adresářovou strukturu. Adresáře končí lomítkem „/“.

```
hello-plasmoid/
├── contents/
│   ├── code/
│   │   └── main.rb
│   ├── config/
│   │   └── main.xml
│   └── ui/
│       └── config.ui
└── metadata.desktop
```

Kde `contents/code` obsahuje zdrojové kódy plasmoidu (zde `main.rb`, `.rb` jako přípona pro programy v jazyce Ruby). Soubory `config/main.xml` a `ui/config.ui` slouží pro vytvoření konfigurace plasmoidu, informace o tvorbě konfigurace lze najít v tutoriálu [16].

Kód plasmoidu – soubor main.rb

Výpis na obrázku 4.1 zobrazuje všechen potřebný zdrojový kód v jazyce Ruby pro plasmoid, na kterém bude jedno tlačítko s nápisem „Hello World“. Kód plasmoidu by měl začínat licencí. Řádek `require 'plasma_applet'` se postará o načtení QT a KDE knihoven pro použití s Ruby [15]. Třída `Main` je potomkem `PlasmaScripting::Applet`, budeme tak moci využít balíčkovací systém plasmy a distribuce plasmoidu pomocí tzv. *Get Hot New Stuff* (získat novinky), což uživatelům umožní pohodlně do KDE nainstalovat nové plasmiody (či nové tapety plochy a podobně).

Jméno modulu, zde `module HelloPlasmoid`, se musí shodovat s názvem adresáře `hello-plasmoid`, kde jméno modulu píšeme v tzv. Camel case[5] a název adresáře malými písmeny, slova oddělíme pomlčkou.

V metodě `init` vytvoříme vertikální (`QT::Vertical`) layout (rozložení). Prvky vkládané do tohoto layoutu budou uspořádány pod sebou.

5. mezery mezi slovy jsou odstraněny, pro rozlišení slov je každé psáno počátečním velkým písmenem, http://en.wikipedia.org/wiki/Camel_case

Obrázek 4.1: kód plasmoidu Hello Plasmoid

```
1  # License
2  #
3
4  require 'plasma_applet'
5
6  module HelloPlasmoid
7    class Main < PlasmaScripting::Applet
8      def initialize parent
9        super parent
10     end
11
12     def init
13       layout = Qt::GraphicsLinearLayout.new Qt::
             Vertical, self
14
15       button = Plasma::PushButton.new self
16       button.text = "Hello World"
17
18       layout.add_item button
19       self.layout = layout
20     end
21   end
22 end
```

Na řádku 15 vytvoříme nové tlačítko (button), na dalším řádku mu přiřadíme nápis „Hello World“. Na řádku 18 využijeme metodu `add_item` a přiřadíme do layoutu námi vytvořené tlačítko, řádek 19 pak přiřadí layout samotnému plasmoidu.

metadata plasmoidu

Druhým souborem, který je nutno pro ukázkový plasmoid vytvořit, je `metadata.desktop` (viz. výpis na obrázku 4.2), volby jsou poměrně odvoditelné z názvů, důležitá je hodnota volby `X-KDE-PluginInfo-Name`, která se využije v balíčkovacím systému plasmy. Není nezbytně nutné, aby se shodovala s názvem adresáře, pro přehlednost je to však vhodné.

Obrázek 4.2: metadata pro hello world plasmoid

```
1  [Desktop Entry]
2  Encoding=UTF-8
3  Name=Hello Plasmoid
4  Type=Service
5  ServiceTypes=Plasma/Applet
6  Icon=chronometer
7
8  X-Plasma-DefautSize=125,125
9  X-Plasma-API=ruby-script
10 X-Plasma-MainScript=code/main.rb
11 X-KDE-PluginInfo-Author=Petr Sigut
12 X-KDE-PluginInfo-Email=contact@sigut.net
13 X-KDE-PluginInfo-Name=hello-plasmoid
14 X-KDE-PluginInfo-Version=0.1
15 X-KDE-PluginInfo-Website=
16 X-KDE-PluginInfo-Category=Examples
17 X-KDE-PluginInfo-Depends=
18 X-KDE-PluginInfo-License=GPL
19 X-KDE-PluginInfo-EnabledByDefault=true
```

Balíčkovací systém Plasmy

Balíčkovací systém plasmy pracuje s námi vytvořenou adresářovou strukturou zabalenou pomocí zip komprese[6]. Pro ulehčení procesu komprese a instalace lze použít skript (zdrojový kód na obrázku 4.3). Skript `plasma_make` zkomprimuje aktuální adresář, přičemž vynechá dočasné soubory vytvářené editorem vim a adresář verzovacího systému git. Nakonec je provedena instalace (či aktualizace) daného plasmoidu.
Pro manipulaci s balíčky sytému Plasma slouží příkaz `plasmapkg`, kompletní seznam voleb příkazu zjistíme pomocí volby `-help`, ty nejpodstatnější jsou `-i` pro instalaci, `-u` pro aktualizaci, `-r` pro odinstalaci plasmoidu a `-l` pro seznam všech nainstalovaných plasmoidů. Balíčkovací systém plasmy je používán pro plasmoidy ve skriptovacích jazycích, pro plasmoidy psané v C++ se využívá balíčkovacího systému linuxové distribuce.

6. http://cs.wikipedia.org/wiki/ZIP_%28souborov%C3%BD_form%C3%A1t%29

Obrázek 4.3: plasma_make pro ulehčení komprese a instalace

```
1 #!/bin/bash
2 dir=$(basename `pwd`);
3 zip -r ../"$dir".zip␣.␣-x␣\*.swp␣-x␣"*.git*"␣;
    plasmapkg␣-u␣\
4 ../"$dir".zip
```

Pro otestování našeho plasmoidu slouží příkaz `plasmoidviewer`, jako parametr mu předáme jméno (to uvedeno v balíčkovacím systému) plasmoidu, v našem případě tedy `hello-plasmoid`.

4.1.3 Unico Plasmoid

Následující část zmiňuje některé problémy při vývoji Unico Plasmoidu.

odstraňování widgetů

Unico Plasmoid zobrazuje HTML obsah v jednom widgetu (`Plasma::WebView`), ale XML zobrazujeme v několika widgetech. Je tedy nutné vždy při změně formátu obsahu veškeré widgety vymazat a zobrazit jiné.

Bohužel při použití metody `dispose` na odstranění objektu typu `Plasma` celý plasmoid skončil s chybou, při odstraňování objektů typu `QT` je vše v pořádku. Dotaz, proč tomu tak je, zůstal na e-mailové konferenci *kde-bindings* nezodpovězen. Zdrojový kód na obrázku 4.4 volá `dispose` na objektu získaném pomocí `layout.itemAt(0)`, který je typu `QT` a k chybě nedochází.

Obrázek 4.4: odstraňování widgetů z plasmoidu

```
1 while ((@child = @layout.itemAt(0)) != nil)
2   @layout.removeItem(@child)
3   @child.dispose
4 end
```

DataEngine

S oddělením samotného obsahu a vzhledu se můžeme setkat u mnoha technologií (např. HTML a CSS), nejinak je tomu v případě Plasmy. O přípravu a poskytování dat samotných se starají DataEnginy. Stačí například jednou napsat DataEngine pro získávání aktuálního počasí a poté může vzniknout několik plasmoidů, které budou tyto údaje – každý jinak – zobrazovat. [17]

DataEnginy nebyly pro Unico Plasmoid využity, důvodem byla slabá dokumentace pro programování DataEnginů v Ruby – na TechBase není žádná dokumentace, nalezl jsem pouze blogové zápisky na portálu kdedevelopers.org: *Writing Plasma Data Engines in Ruby*[7] a *Writing Plasma Data Engines in C# and Ruby*[8]. Ukáže-li se DataEngine v budoucnu jako užitečný, rád bych jej v Unico Plasmoidu využil.

Nestabilita plasmoidů v Ruby

Při rychlém přidávání plasmoidů psaných v Ruby jsem pozoroval pády Plasmy, toto chování bylo popsáno a nahlášeno do systému pro hlášení chyb v KDE jako chyba (bug) 216833[9].

4.2 GNOME

GNOME[10] je grafické uživatelské prostředí určené pro unixové platformy[11]. GNOME se soustředí na použitelnost a přístupnost, je součástí projektu GNU. Pro vývoj GNOME (a aplikací pro něj určených) se používá toolkit/knihovna GTK+[12]. [24]

GNOME se momentálně (4. prosince 2009) nachází ve verzi 2.28 a součástí prostředí jsou Gnome Applets, tedy miniaplikace/applety, které je možno umístit pouze do panelu.

7. http://www.kdedevelopers.org/node/3393
8. http://www.kdedevelopers.org/node/3779
9. https://bugs.kde.org/show_bug.cgi?id=216833
10. GNOME je zkratka GNU Network Object Model Environment, původní význam již není aktuální a začíná se postupně přiklánět k označení Gnome
11. GNU, Linux, BSD, Solaris, http://en.wikipedia.org/wiki/GNOME
12. http://www.gtk.org/

Několik projektů se snažilo přinést do GNOME miniaplikace umístitelné na plochu. Dva pravděpodobně nejúspěšnější projekty jsou gDesklets[13] a Screenlets[14]. Screenlets mají dle ankety na portálu abclinuxu.cz[15] více uživatelů (Screenlets 5 % uživatelů, gDesklets 1 %). Bohužel oba projekty mají na svých stránkách poslední zmínku z roku 2008.

Někteří vývojáři projektu Screenlets začali vyvíjet nový systém Universal Applets[16], kterému se dostávalo velké pozornosti. Avšak nikdy se nedospělo do stabilního vydání a hlavní autor Natan Yellin nyní nemá na projekt čas:

„Unfortunately, several things came up in real life and I never had the time to release a stable version. As of now the project is on standby, awaiting a new maintainer with more time. –Natan Yellin 1. 12. 2009, e-mail“

Dále jsem kontaktoval Bjoerna Kocha, jednoho z aktivních členů vývojového týmu gDesklets, který mi poskytl detailní náhled na situaci kolem projektu. Veškeří původní vývojáři odešli a nový tým nemá hlubší znalosti systému:

„. . . right now we are a small team of 4-5 active developers. BUT: no of us knows the gDesklets core very well. So all we can do and all we are doing right now is keeping the project alive and fixing bits an pieces. Sometimes there are some new features etc. but no major changes are done. . . . –Bjoern Koch 8. 12. 2009, e-mail“

Autory systému Screenlets se mi bohužel nepodařilo kontaktovat, ale tento systém má v současné době pravděpodobně největší přízeň uživatelů (viz. anketa výše) a proto byl zvolen pro implementaci Unico klienta.

Na září 2010 je naplánováno vydání GNOME 3, které má přinést – jako KDE 4 – přepracovaný Desktop Shell. Změny by neměly být tak revoluční[17] jako v případě KDE 4, přesto prostředí čekají změny (například přechod na GTK+ verze 3) a vzhledem k nejasné budoucnosti nynějších

13. webová stránka trpěla v době psaní bakalářské práce častými výpadky, http://www.gdesklets.de/
14. http://www.screenlets.org/index.php/Home
15. http://www.abclinuxu.cz/ankety/miniaplikace-na-plochu
16. https://launchpad.net/universal-applets
17. http://live.gnome.org/ThreePointZero/Plan

systémů pro miniaplikace v GNOME nebylo vývoji Screenletu věnováno tolik času (z mého pohledu jsou plasmoidy nyní technologicky vyspělejší a uživatelsky přívětivější).

4.2.1 Screenlets

Na následujících řádcích si stručně představíme strukturu Screenletu.

Oficiálním zdrojem informací pro vývoj je dokumentace[18] na stránkách projektu. Pro informace v češtině lze doporučit seriál o vývoji Screenletů[19] na portálu root.cz. Jelikož jsou všechny Screenlety programovány v programovacím jazyku Python[20], jsou zdrojové kódy každého Screenletu čitelné a mohou posloužit jako výukový zdroj. Velké množství Screenletů na stažení nabízí portál gnome-look.org[21].

4.2.2 Balíky potřebné pro vývoj a používání Screenletů

V Ubuntu 9.10 nainstalujeme Screenlety pomocí balíku `screenlets`, který závisí na těchto balících: `python python-dbus python-gnome2 python-gtk2 python-rsvg python-support python-wnck python-xdg`. Z doporučených závislostí je vhodné nainstalovat `python-gtkmozembed`, jež umožňuje použít renderovací jádro prohlížeče Mozilla[22] jako GTK+ widget [4].

4.2.3 Struktura Screenletu

V dokumentaci Screenletů se nachází odkaz[23] na šablonu pro vytvoření nového Screenletu, využijeme této šablony pro ukázku struktury Screenletu. Po rozbalení archivu nás bude zajímat soubor `TemplateScreenlet.py`, jež obsahuje kód samotného Screenletu. Kromě

18. http://www.screenlets.org/index.php/Documentation
19. http://www.root.cz/clanky/jak-psat-screenlety/
20. http://www.python.org/
21. http://gnome-look.org/index.php?xcontentmode=6700
22. http://www.mozilla.org/
23. http://screenlets.org/images/c/c6/Template.tar.gz

něho se zde nachází soubor `icon.png` s ikonkou a prázdná adresářová struktura `themes/default` pro případná témata vzhledu.

Zdrojový soubor se Screenletem si můžeme rozdělit do několika částí, vybrané ukázky jsou z ukázkové šablony.

jméno souboru a třídy

Třída a název souboru si musejí odpovídat a končit slovem „Screenlet“ [19], v našem případě tedy `TemplateScreenlet.py` a `class TemplateScreenlet`. Adresář pojmenujeme bez přídavku Screenlet.

cesta k interpretu Pythonu

Na prvním řádku souboru `TemplateScreenlet.py` určíme, že chceme skript spouštět v Pythonu:
`#!/usr/bin/env python`, použití příkazu shellu `env` nám zajistí větší přenositelnost[24].

Licenční ujednání

Následují informace o licenci, jméno autora a kontakt na něj.

Import potřebných knihoven

V ukázkové šabloně vidíme importování nezbytných knihoven, lze přidat další dle potřeby, například `import gtkmozembed`, pro zmiňovanou komponentu webového prohlížeče.

třída `TemplateScreenlet`

Kód třídy `TemplateScreenlet` obsahuje samotnou implementaci funkcionality našeho Screenletu. Tato část by se dala opět rozdělit na tři části, v první definujeme meta informace o našem Screenletu, každá z těchto proměnných začíná a končí dvěma podtržítky, např `__name__`, či `__version__` kde uvedeme verzi našeho Screenletu.

---

24. na druhou stranu je možné, že se vybere nevhodný interpret, http://en.wikipedia.org/wiki/Env

Dále následuje výčet voleb, které chceme uživatele nechat editovat (například interval obnovy miniaplikace) a následně v metodě `__init__` pomocí `add_option` vytvoříme widgety pro jejich zobrazení v GUI (můžeme je například rozčlenit do skupin).

Následuje asi 30 prázdných předefinovaných metod, jejichž tělo je vykonáno při rozličných událostech dle názvu metody – například při skrytí (`on_hide`) Screenletu, při změně jeho velikosti apod. Samozřejmě můžeme definovat metody i zcela své. Každá metoda je v ukázkové šabloně okomentována, či český popis některých z nich lze najít ve zmiňovaném seriálu na portálu root.cz[25].

screenlets-packager

Screenlet připravíme pro instalaci pomocí příkazu `screenlets-packager`, jehož jediným parametrem je adresář se Screenletem – výsledkem bude `tar.gz` archív.
Instalace pak už probíhá skrze grafický `screenlets-manager`.

4.2.4 Unico Screenlet

Unico Screenlet je úpravou existujícího Screenletu Webframe[26] od autora Akira Ohgaki. Webframe je Screenlet sloužící k zobrazení webové stránky, nejlépe stránek přizpůsobených pro malou obrazovku, jako například verze stránek pro mobilní telefony.

Do Screenletu Webframe byla přidána podpora pro automatické obnovování obsahu v zadaném intervalu, některé části kódu byly převzaty z Screenletu MailCheck[27], který provádí periodickou kontrolu e-mailové schránky. Pro periodické kontroly jsme implementovali metody `set_check_interval` a `page_reload`.

Autorovi Webframe byla zaslána upravená verze a automatická obnova byla zařazena do nového vydání Webframe 0.2[28]. Implementace funkce autorem je díky jeho hlubším znalostem Screenletu nepochybně kvalitnější.

25. http://www.root.cz/clanky/jak-psat-screenlety/
26. http://akiraohgaki.com/goodies/widgets/webframe
27. http://www.screenlets.org/index.php/Information
28. https://launchpad.net/webframe

Místo samostatného programování forku[29] Webframu se vývoj nyní může soustředit do jednoho programu.

4.3 Další prostředí

Systém Unico není nijak svázán s konkrétním desktopovým prostředím, stačí vytvořit nového klienta schopného zobrazovat nějaký z formátů Unico Serveru. Jedním z těchto formátů je i prostý text (plain text), který je vhodným vstupem například pro program Conky.

Conky je systémový monitor a umožňuje vykreslovat na plochu (root desktop) jakékoliv informace (má zabudované proměnné pro informace o systému jako velikost zabrané paměti RAM) či výstupy skriptů [18]. Conky je populárním v rozličných lehkých a nenáročných prostředích (window managerech[30]) jako FluxBox[31] či Awesome[32].

Pro zobrazování obsahu pomocí programu Conky postačí do konfiguračního souboru `.conkyrc` přidat (do části `TEXT`) následující kód: `${execi 3000 wget -O \ - http://phax.vserver.cz:3000/contents/show/timetest.txt -q}`

Pomocí programu wget[33] se bude periodicky stahovat obsah (z uvedené URL adresy), jež se následně zobrazí pomocí Conky.

29. *„Forkem se v oblasti software označuje alternativní větev programu, která je vyvíjená nezávisle a zpravidla i jinými lidmi.* [29]“
30. http://en.wikipedia.org/wiki/Window_manager
31. http://www.fluxbox.org/
32. http://awesome.naquadah.org/
33. http://www.gnu.org/software/wget/

5 Závěr

Zadání bakalářské práce bylo splněno a výsledkem je systém Unico. Vznikla nová platforma pro programátory, kterým místo opakovaného tvoření miniaplikace stačí naprogramovat několik řádků kódu. Díky objektovému návrhu lze vývoj ještě více usnadnit implementací dalších metod pro často opakované úlohy. Uživatel získá instalací Unico Klienta univerzální miniaplikaci zobrazující obsah dodávaný Unico Serverem. Uživatel tak nemusí instalovat desítky velmi podobných miniaplikací. Navíc oproti zadání byly vyzkoušeny také serverem inicializované aktualizace.

Praktická část práce prohloubila moje znalosti programování ve frameworku Ruby on Rails a vyzkoušel jsem si práci s toolkitem QT. Zajímavou zkušeností byla také zpětná vazba od linuxové komunity.

Dalším bezprostředním krokem bude rozšíření mezi co nejvíce uživatelů, což také žádá co nejvíce nabízeného obsahu. Jedním z problémů, který bude třeba vyřešit, je správa modulů na získávání obsahu od různých vývojářů. Přínosem by byla i možnost programovat moduly i v jiných programovacích jazycích či vytvoření klientské miniaplikace pro další operační systémy. Rád bych v dalším vývoji pokračoval a pokusil se Unico zařadit do některých linuxových distribucí.

A Instalace Unico Serveru

Tato příloha popisuje potřebné kroky – především instalaci balíčků – pro provoz Unico Serveru. Postup byl testován na výchozí instalaci distribuce Ubuntu 9.10. Jelikož bylo testování prováděno ve virtuálním stroji VirtualBox[1], vyexportovaný obraz systému je přiložen na DVD (po startu systému je automaticky přihlášení uživatel `unico` s heslem `unico`).

Unico Server rozbalíme z přiloženého archivu `unico-server.tar.bz2`, nebo nainstalujeme (pravděpodobně aktuálnější) verzi z git repozitáře pomocí příkazu[2]
`git clone git://github.com/petrsigut/unico.git`

A.1 Balíčky

Tučným písmem je uveden vždy název balíčku a pod ním seznam instalovaných závislostí (někdy jde pouze o balík samotný či další metabalíky).

`mysql-server`
`libdbd-mysql-perl libdbi-perl libhtml-template-perl libnet-daemon-perl libplrpc-perl mysql-client-5.1 mysql-server mysql-server-5.1 mysql-server-core-5.1`

`ruby`
`libruby1.8 ruby ruby1.8`

`rubygems`
`irb1.8 libreadline-ruby1.8 libreadline5 rdoc1.8 rubygems rubygems1.8`

`libopenssl-ruby`
`libopenssl-ruby libopenssl-ruby1.8`

`libmysqlclient15-dev`
`libmysqlclient15-dev libmysqlclient15off zlib1g-dev`

`ruby-dev`
`ruby-dev ruby1.8-dev`

1. http://www.virtualbox.org/
2. program `git` se nachází v balíčku `git-core`

libxml-ruby libxslt-ruby
`libxml-ruby libxslt-ruby`

rake
`rake`

A.2 Gemy

```
gem install mysql rails hpricot htmlentities
```

A.3 Závislosti pro push aktualizace

A.3.1 balíčky

g++
`g++ g++-4.4 libstdc++6-4.4-dev`

A.3.2 gemy

```
gem install juggernatu eventmachine json
```

A.4 Poinstalační nastavení

Všechny uvedené cesty jsou uváděny relativně ke kořeni Unico Serveru.

V souboru `config/database.yml` je nutné nastavit heslo pro přístup k databázi (volby `username:` a `password:` v sekci příslušného módu).

Jakmile máme nastaveno heslo do databáze příkazem `rake db:create` v ní vytvoříme databázi pro Unico Server (v development módu se bude jmenovat `unico_development`) a příkazem `rake db:migrate` do ní zmigrujeme tabulky (více v kapitole 3.3).

Pro prvotní naplnění daty využijeme Rake task `downloadall` popsaný v sekci 3.8.2. Nyní by měl být Unico Server připraven ke spuštění skrze

příkaz `script/server`. Dostupné obsahy si můžeme prohlédnout pomocí webového prohlížeče na adrese
`http://localhost:3000/contents/index`

Literatura

[1] Dokumentace plasma api. http://api.kde.org/4.x-api/kdelibs-apidocs/plasma/html/hierarchy.html. [Online; cit. 20. 11. 2009].

[2] Dokumentace projektu rake. http://rake.rubyforge.org/. [Online; cit. 20. 11. 2009].

[3] Dokumentace projektu rubygems. http://docs.rubygems.org/read/book/1. [Online; cit. 19. 11. 2009].

[4] Gtkmozembed: Gtk mozilla embedding widget. http://www.mozilla.org/unix/gtk-embedding.html. [Online; cit. 4. 12. 2009].

[5] Hpricot – html parser. http://github.com/whymirror/hpricot. [Online; cit. 12. 12. 2009].

[6] Kde licenční politika. http://techbase.kde.org/index.php?title=Policies/Licensing_Policy. [Online; cit. 19. 11. 2009].

[7] Slovníček projektu plasma. http://techbase.kde.org/Projects/Plasma/Vocabulary. [Online; cit. 20. 11. 2009].

[8] Stránka frameworku ruby on rails. http://rubyonrails.org/. [Online; cit. 21. 11. 2009].

[9] Stránka knihovny qt. http://qt.nokia.com/products. [Online; cit. 19. 11. 2009].

[10] Stránka plasmoidu webpage subset or html widget na www.kde-look.org. http://kde-look.org/content/show.php/WebPage+subset+or+HTML+Widget?content=115536. [Online; cit. 1. 12. 2009].

[11] Stránka projektu kde. http://www.kde.org/. [Online; cit. 19. 11. 2009].

[12] Techbase – vývoj v jazyce ruby. http://techbase.kde.org/Development/Languages/Ruby. [Online; cit. 20. 11. 2009].

[13] Tutoriál plasma/ruby – jednoduchý applet pro vkládání (simple paste applet). http://techbase.kde.org/Development/Tutorials/Plasma/Ruby/SimplePasteApplet. [Online; cit. 20. 11. 2009].

[14] Tutoriál plasma/ruby – používání widgetů (using widgets). http://techbase.kde.org/Development/Tutorials/Plasma/Ruby/Using_widgets. [Online; cit. 20. 11. 2009].

[15] Tutoriál plasma/ruby začínáme – (getting started). http://techbase.kde.org/Development/Tutorials/Plasma/Ruby/GettingStarted. [Online; cit. 20. 11. 2009].

[16] Tutoriál používání kconfig. http://techbase.kde.org/Development/Tutorials/Using_KConfig_XT. [Online; cit. 6. 12. 2009].

[17] Tutoriál tvorby plasma dataenginů. http://techbase.kde.org/Development/Tutorials/Plasma/DataEngines. [Online; cit. 5. 12. 2009].

[18] Webová stránka projektu conky. http://conky.sourceforge.net/. [Online; cit. 5. 12. 2009].

[19] Michal Hořejšek. Jak psát screenlety. http://www.root.cz/clanky/jak-psat-screenlety/. [Online; cit. 4. 12. 2009].

[20] Alex MacCaw. Juggernaut. http://juggernaut.rubyforge.org/. [Online; cit. 19. 11. 2009].

[21] S. Ruby, D. Thomas, and D. Hansson. Agile Web Development with Rails. 2009.

[22] Wikipedia. Kde — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=KDE&oldid=325989195. [Online; cit. 20. 11. 2009].

[23] Wikipedia. Desktop metaphor — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Desktop_metaphor&oldid=325477079, 2009. [Online; cit. 20. 11. 2009].

[24] Wikipedia. Gnome — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=GNOME&oldid=328641174, 2009. [Online; cit 6. 12. 2009].

[25] Wikipedia. Push technology — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Push_technology&oldid=326287887, 2009. [Online; cit. 19. 11. 2009].

[26] Wikipedia. Ruby on rails — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Ruby_on_Rails&oldid=326622754, 2009. [Online; cit. 19. 11. 2009].

[27] Wikipedia. Xmlsocket — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=XMLSocket&oldid=313909088, 2009. [Online; cit. 20. 11. 2009].

[28] Wikipedia. Xslt — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=XSLT&oldid=330978771, 2009. [Online; cit. 22. 12. 2009].

[29] Wikipedie. Fork — wikipedie: Otevřená encyklopedie. http://cs.wikipedia.org/w/index.php?title=Fork&oldid=2812024, 2008. [Online; cit. 12. 12. 2009].

[30] Wikipedie. Cron — wikipedie: Otevřená encyklopedie. http://cs.wikipedia.org/w/index.php?title=Cron&oldid=4433333, 2009. [Online; cit. 20. 11. 2009].

[31] Wikipedie. Extensible markup language — wikipedie: Otevřená encyklopedie. http://cs.wikipedia.org/w/index.php?title=Extensible_Markup_Language&oldid=4558808, 2009. [Online; cit. 21. 11. 2009].

[32] Wikipedie. Framework — wikipedie: Otevřená encyklopedie. http://cs.wikipedia.org/w/index.php?title=Framework&oldid=4611706, 2009. [Online; cit. 5. 12. 2009].

[33] Wikipedie. Xslt — wikipedie: Otevřená encyklopedie. http://cs.wikipedia.org/w/index.php?title=XSLT&oldid=4137953, 2009. [Online; cit. 22. 11. 2009].

[34] Adam Štrauch. Kde 4.2: vývoj plasmoidu pro twitter. http://www.root.cz/clanky/kde-4-2-vyvoj-plasmoidu-pro-twitter/. [Online; cit. 6. 12. 2009].