Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Přehled webových mashupů

Plzeň, 2012 Bc. Josef Vrba

!!ZADÁNÍ!!

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 14. května 2012

Bc. Josef Vrba

Abstract

This thesis contains information about the web mashups. The thesis focuses on the tools that facilitate creation of mashups in enterprise environments.

The thesis describes a sample mashup which combines several data sources. This mashup is implemented in three different ways using different tools. In the last part of the thesis, the usage and feasibility of these tools in enterprise environments is evaluated.

Obsah

1 Úvod

V několika posledních letech se začal objevovat v prostředí internetu pojem *mashup*. Z počátku byly mashupy pouze webové stránky pro běžné uživatele, ale postupně se začalo využívat jejich výhod i ve firemním a korporátním prostředí.

V této práci bude čtenář seznámen s pojmem mashup, s jeho výhodami a nevýhodami. V práci bude dále uvedeno několik příkladů úspěšných webových mashupů. V závěru práce pak budou rozebrány možnosti, jak lze vytvořit jednoduchý mashup a jaké nástroje lze použít pro usnadnění jeho tvorby.

1.1 Co je mashup

Pro pojem *mashup* existuje v češtině několik ekvivalentů (např. míchanice, mixáž, kaše), ale žádný se ve velkém neujal. Proto i nadále v textu bude použit výraz *mashup*.

Definice pojmu *mashup* zatím není ustálená. Většina definicí je však velice podobná. Jedna z definic publikovaná v [4] říká:

„A mashup is a technique for building applications that combine data from multiple sources to create an integrated experience.“

Volně přeloženo lze říct, že mashup je technika vývoje aplikací, které kombinují více jak 2 zdroje dat a prezentují je novým způsobem.

Aplikace typu mashup má své výhody a nevýhody [4]. Mezi výhody lze řadit:

- Vývoj mashupu je několikanásobně rychlejší, než vývoj běžné aplikace.
- Vizualizovaná data jsou pro uživatele příjemnější.

Mezi nevýhody patří:

- Mashup je velmi často závislý na datech třetí strany.
- Vizualizovaná data mohou mít tendenci ke ztrátám přesnosti.

1.2 Typy mashupů

Mashupy lze dělit dvěma způsoby. První způsob vychází ze způsobu prezentace dat. Mashupy se pak dělí například do těchto skupin:

- mapové,
- sociální,
- multimediální,
- cestovatelské.

Kromě tohoto, existuje ještě další dělení [25].

- datové
- uživatelské (webové)
- enterprise

Datové mashupy kombinují více zdrojů dat a vytvářejí nový zdroj dat. Neřeší se v nich prezenční vrstva. Příkladem může být sloučení několika RSS kanálů a odfiltrování určitých položek.

Uživatelské (webové) mashupy jsou přístupné široké veřejnosti na internetu a prezentují veřejně přístupná data. Jako příklad lze uvést mapu, kde budou vyznačeny jednotlivé hotely s možností objednání ubytování.

Enterprise mashupy jsou určeny pro uzavřené firemní prostředí a zobrazují zpravidla důvěrná data smíchaná s veřejně dostupnými daty. Jako příklad lze uvést mapu, kde budou vyznačeny zákazníci firmy s výší jejich útraty za poslední měsíc. Enterprise mashupy navíc mohou pomoci řešit problém tzv. longtail aplikací (viz. 1.5).

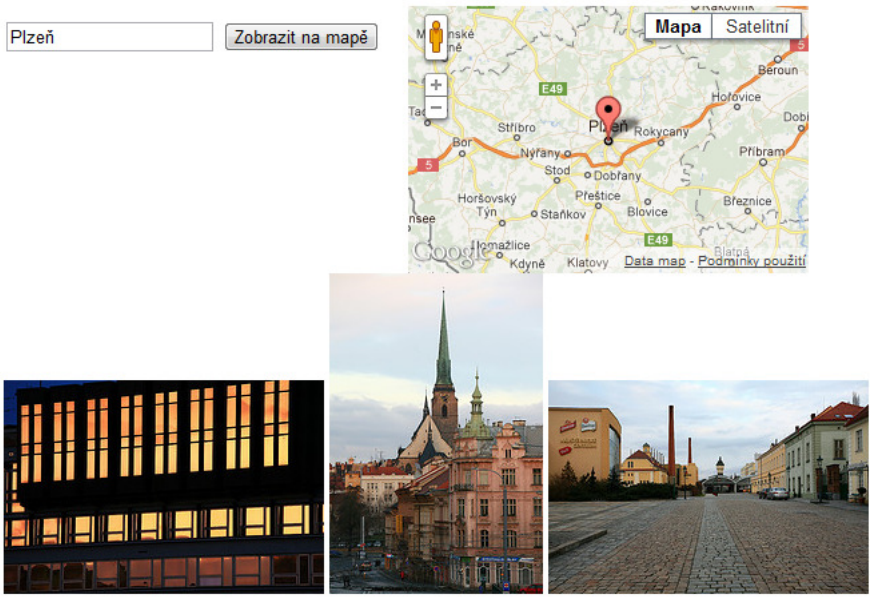
Zajímavostí je, že webové mashupy využívají většinou data z API postavených na technologii REST, kdežto ve firemním prostředí jsou častější zdroje dat typu SOAP [4]. Je to dáno z velké části lepší programovou zpracovatelností zdrojů dat s WSDL souborem.

1.3 Hello World Mashup

Pro lepší pochopení pojmu *mashup* jsem vytvořil velice jednoduchý webový mashup, na kterém lze snadno pochopit základní principy mashupů. Mashup jsem vytvořil pouze za použití HTML, CSS a JavaScriptu s knihovny jQuery.

1.3.1 Funkce mashupu

Funkce mashupu (náhled na obrázku 1.1) spočívá v tom, že návštěvník zadá název města do vstupního pole. Po kliknutí na tlačítko se zobrazí na mapě zadané město a zobrazí se obrázky města. Událost je rozdělena na čtyři akce:

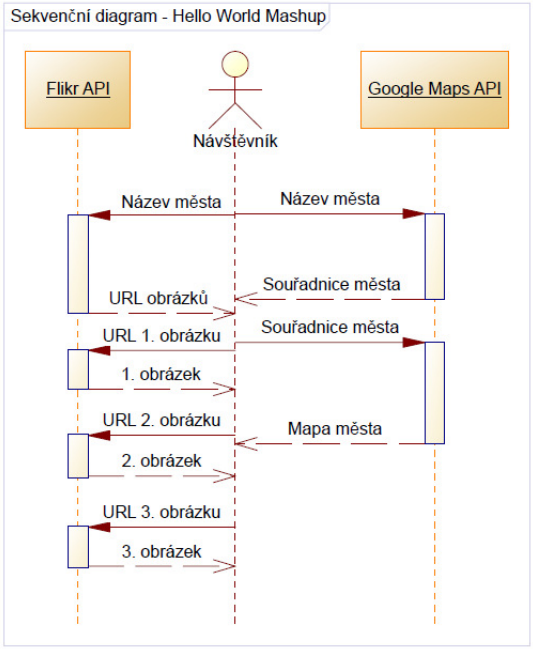


Obrázek 1.1: Hello World Mashup

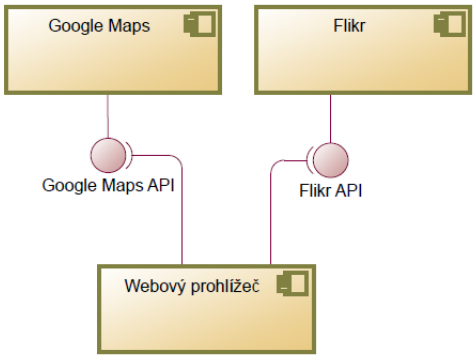
- Odešle se požadavek na servery Google Maps, kde se zjistí souřadnice daného města.
- Získané souřadnice se zobrazí na mapě Google Maps.
- Název města se odešle na servery Flickr.com. Jako odpověď přijde seznam fotek, které mají v popisu název hledaného města.
- Získané fotky se zobrazí na webové stránce.

V tomto jednoduchém Hello World Mashupu jsem spojil dva zdroje dat (Google Maps API a Flickr API). Volání těchto API je nezávislé, není to však podmínkou. Například získané souřadnice města z Google Maps API lze posílat jinému API, které zjistí počasí, nebo jiné informace o dané oblasti.

Průběh dotazování jednotlivých API je graficky znázorněn na sekvenčním diagramu 1.2. Jelikož se jedná o velice jednoduchý mashup je i komponentový diagram 1.3 velice jednoduchý.



Obrázek 1.2: Sekvečí diagram - Hello World Mashup



Obrázek 1.3: Komponentový diagram - Hello World Mashup

1.3.2 Návod na vytvoření

Pro vytvoření tohoto Hello World mashupu stačí provést několik jednoduchých kroků.

První krok je získání API klíčů k zdrojům dat a stažení knihovny jQuery. Vygenerování klíčů je k dispozici po přihlášení na stránce konkrétního API.

Druhý krok spočívá ve vytvoření HTML stránky s následujícím obsahem.

```
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <title>Hello World Mashup</title>
    <script type="text/javascript"
      src="http://maps.googleapis.com/maps/api/js?sensor=false"></script>
    <script type="text/javascript" src="jquery-1.7.1.min.js"></script>

    <script type="text/javascript"></script>
    <style type="text/css"></style>
  </head>
  <body>
    <div id="cities">
      <input type="text" id="city" />
      <input type="button" id="submit" value="Zobrazit na mapě" />
    </div>
    <div id="map_canvas"></div>
    <div id="photos"></div>
  </body>
</html>
```

Obsah stránky zatím není příliš zajímavý. V části uvozené tagem *head* je pouze vložená knihovna *jQuery* a Google Maps API. Tělo stránky uvozené tagem *body* obsahuje tři části oddělené tagy *div*. První část obsahuje dva

ovládací prvky (vstupní pole a tlačítko) pro zadání hledaného města. Druhá a třetí část neobsahuje nic. Jejich obsah bude generován JavaScriptem.

Největší úlohu v tomto mashupu hraje JavaScript. Nejprve je nutné inicializovat mapu z Google Maps API pomocí následujícího kódu.

```
var geocoder = new google.maps.Geocoder();
var marker = new google.maps.Marker();
var latlng = new google.maps.LatLng(50.0878114, 14.420459800000003);
var myOptions = {
   zoom: 8,
   center: latlng,
   mapTypeId: google.maps.MapTypeId.ROADMAP
};
var element = document.getElementById("map_canvas");
var map = new google.maps.Map(element, myOptions);
```

Jde pouze o vytvoření instancí geocoderu a markeru pro použití v následujících funkcích a inicializace zobrazení mapy. Uvedené GPS souřadnice jsou souřadnicemi Prahy.

Další krok spočívá v odchycení události kliknutí na tlačítko. K tomu postačí jednoduchý kód:

```
$("#submit").click(function(){
  var city = $("#city").val();
  downloadFlickr(city);
  geocodeAddress(city);
});
```

Tento kód po kliknutí na element s id *submit* zjistí hodnotu města uloženou ve vstupním poli s id *city*. Následně zavolá funkci *downloadFlickr* pro stažení obrázků a *geocodeAddress* pro aktualizaci mapy.

Následně je potřeba vytvořit dvě dříve zmíněné funkce. První z nich *downloadFlickr* vypadá takto:

```
function downloadFlickr(city) {
  var url = "http://api.flickr.com/services/rest/";
  var method = "flickr.photos.search";
  var api_key = "VAS_API_KLIC";
  var format = "json";
   $.get(url, {
     method: method,
     api_key: api_key,
     format: format,
     text: city,
```

```
     per_page: "5",
     jsoncallback: "parseFlickr"
   });
}
```

Obsah funkce může vypadat složitě, ale jde pouze o zavolání požadavku GET požadavku na adresu API Flickr. Veškeré parametry jsou samovysvětlující, pozornost je třeba dát pouze parametru *jsoncallback*, jehož hodnota je název funkce, která bude zavolána po úspěšném provedení požadavku. Obsah této funkce je následující:

```
function parseFlickr(response) {
$("#photos").html("");
$.each(response.photos.photo, function(i,photo){
  var imgUrl = "http://farm"+photo.farm+".staticflickr.com/";
  imgUrl = imgUrl + photo.server+"/"+photo.id+"_"+photo.secret+"_m.jpg"
  var img = "<img src='"+imgUrl+"' /> ";
  $("#photos").append(img);
});
}
```

Funkce z odpovědi přijaté od API převezme seznam fotek a pro každou vygeneruje element *img*, který přidá do elementu *div* s id *photos*.

Druhá funkce volaná při kliknutí na tlačítko byla funkce *geocodeAddress*. Obsah této funkce je následující:

```
function geocodeAddress(city) {
  var params = {"address": city};
  geocoder.geocode(params, function(results, status) {
    if (status == "OK") {
       var location = results[0].geometry.location;
       map.setCenter(location);
       marker.setPosition(location);
       marker.setMap(map);
    } else {
       alert("Zadané město nelze najít");
    }
  });
}
```

Tato funkce posílá požadavek na servery GoogleMaps API na zjištění pozice zadaného města. Pokud je město nalezeno, je vrácen *status ok*. Následně je vycentrována mapa na město a vložen ukazatel na střed města.

Poslední krok, který je pro úspěšně vytvoření mashupu udělat je vložení stylů. Pro základní funkci postačí pouze určit velikost elementu s mapou pomocí CSS:

```
#map_canvas {
  height: 200px;
  width: 300px;
}
```

Nyní již stačí otevřít HTML soubor ve webovém prohlížeči a mashup je funkční.

1.4 Příklady mashupů

Pro lepší představu, jak mohou vypadat i jiné mashupy, zde uvedu několik webových mashupů z ČR i ze světa. Mnoho dalších známých i neznámých mashupů lze nalézt na adrese *http://www.programmableweb.com/mashups*.

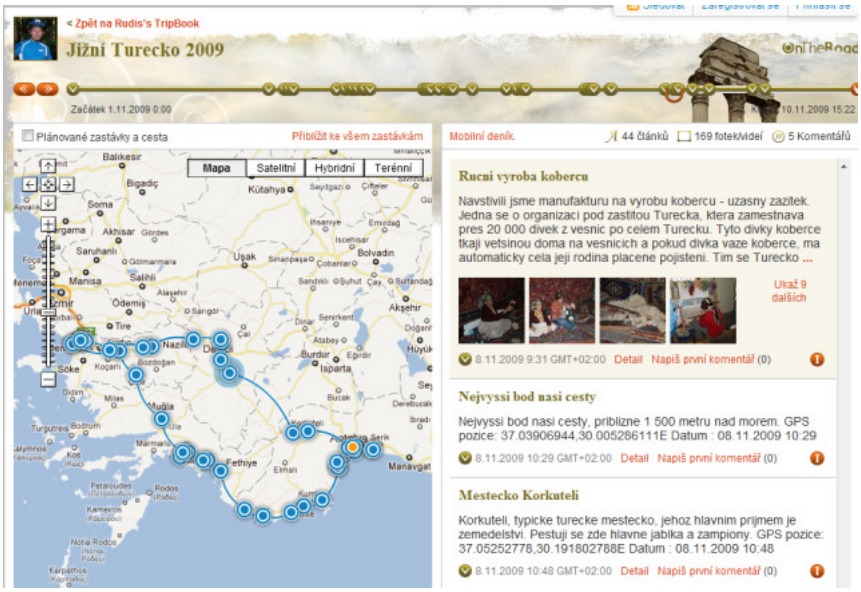
1.4.1 Ontheroad.to

Častý druh mashupů jsou tzv. *Mapové mashupy*. Tyto webové aplikace využívají některé z volně dostupných mapových API (např.: Google Maps API, Bing Maps API, Seznam Mapy API) a doplňují je o vlastní funkce.

Ontheroad.to umožňuje uživateli sdílet zážitky ze svých výletů s přáteli a dalšími uživateli webu. Celý proces vytvoření výletu je velice jednoduchý. Uživatel si vytvoří svůj účet a získá automaticky subdoménu, kde budou zobrazeny všechny jeho výlety (např. jmenoprijmeni.ontheroad.to). Následně již lze vytvořit první výlet. Postupně lze do mapy zadávat všechny zastávky na trase výletu a ke každé zastávce napsat příspěvek ve formě blogového příspěvku. Každá zastávka je následně automaticky doplněna odkazem na Wikipedii a odkazem na informace ze serveru Panoramio, týkajících se konkrétního místa. Po uložení celého výletu lze zážitek sdílet s přáteli na facebooku, twitteru a dalších webech pomocí přímé adresy.

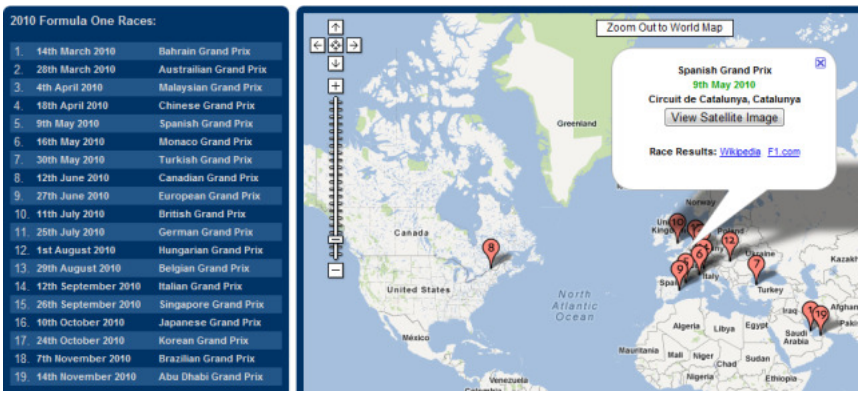
1.4.2 Formula One Map

Další mapový mashup, tentokrát ale mnohem jednodušší je *Formula One Map* (http://www.mapmash.in/f12010.html). Jak název napovídá, tento mashup zobrazuje na mapě všechny okruhy Velké ceny Formule 1 ve vybraném



Obrázek 1.4: Ukázka webu OnTheRoad.to.

roce. Mapové podklady poskytuje Google Maps API a seznam okruhů je získáván z oficiálních stránek Formule 1. Webová stránka je doplněna o vlajku pořádajícího státu.



Obrázek 1.5: Ukázka webu Formula One Map.

1.4.3 Collabim.cz

Poměrně netradiční mashup je také *Collabim.cz*. Collabim je nástroj pro webmastery a seo specialisty. Mezi dostupné funkce patří například kontrola

ranků, pozic ve vyhledávačích a zpětných odkazů. K tomu využívá různá API. Konkrétně Seznam Search API, Srank API, Google Search API, Google Adwords API, Google Analytics API, Yahoo Backlinks API. Collabim ukazuje, že šikovným využitím bezplatných API lze vytvořit placený hojně používaný nástroj.

1.4.4 1-zpravy.cz

Webové stránky 1-zpravy.cz lze také považovat za mashup. Hlavní funkce poskytuje návštěvníkům aktuální zprávy z mnoha zpravodajských webů (agregační funkce RSS a ATOM feedů). Dále je na webu zobrazeno aktuální počasí (widget z myweather.com) a aktuální kurzy měn (widget z i-kurzy.eu).

1.4.5 Shrnutí

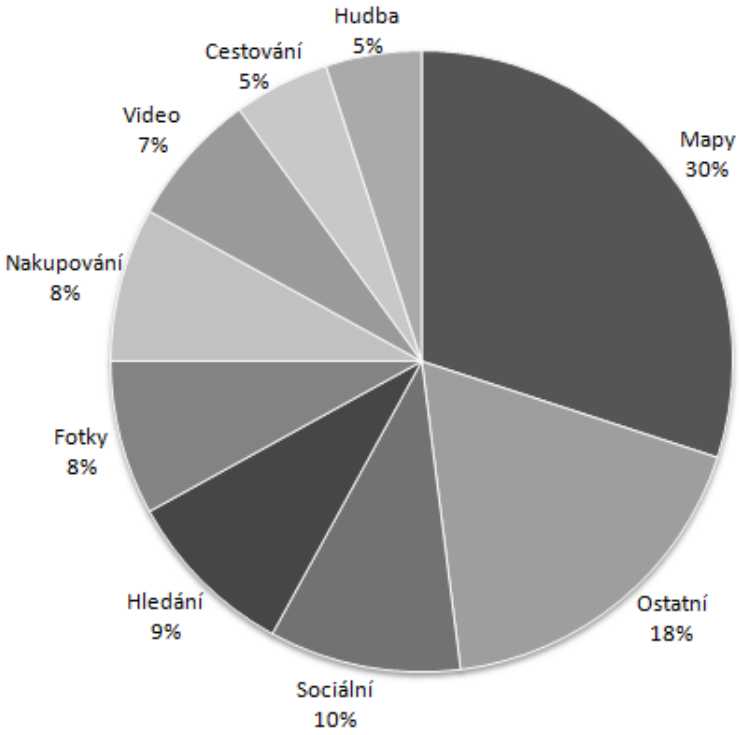
Všechny výše uvedené mashupy využívají dva a více zdrojů dat, přesto průnik společných zdrojů dat je pouze jediný a to Google Maps, které využívají mashupy *Formula One Map* a *Ontheroad.to*.

Daleko zajímavější je globální srovnání nejpoužívanějších zdrojů dat [22]. Toto srovnání ukazuje, že z dlouhodobého hlediska jsou nejpoužívanějším zdrojem dat mapové podklady. V novějších mashupech naopak převládají sociální zdroje dat (např.: Facebook a Twitter).

V grafu 1.6 je zobrazeno aktuální zastoupení zdrojů dat v mashupech na webu *Programmableweb.com*. Zdroje jsou rozděleny do skupiny

- Mapy - Mapové podklady (Google Maps, Bing Maps).
- Sociální - Například tlačítka sdílet na Facebook nebo Twitter.
- Hledání - Prohledávání internetu (Google Search, Bing Search).
- Fotky - Vyhledávání fotek (Picasa, Flickr, Panoramio).
- Nakupování - Vlastní internetový eshop postavený na existujícím (Amazon).
- Video - Vyhledávání videí (Youtube).

- Cestování - Například recenze a rezervace hotelů (Holidaycheck).
- Hudba - Vyhledávání hudby a informací o interpretech (LastFM).
- Ostatní - Všechny ostatní nezařaditelné zdroje.



Obrázek 1.6: Zastoupení zdrojů dat v mashupech.

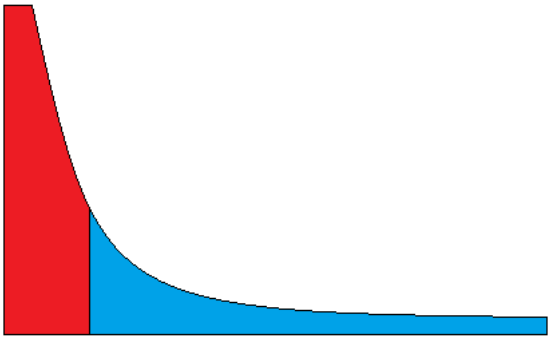
1.5 Long Tail

Koncept Long Tail (česky Dlouhý ocas nebo Dlouhý chvost) poprvé popsal Chris Anderson [3]. Konkrétně se zaměřil na vliv internetu na obchodní strategii internetových obchodů jako Amazon a Netfix. Ukázal, že i málo prodávané zboží, může tvořit velkou část příjmů, pokud je tohoto zboží velké množství.

Obdobné vlastnosti se ukázali i v případech optimalizace pro vyhledávače. Hojně hledaná obecná slova (např.: židle, notebooky), sice tvoří velký objem

vyhledávání, ale minimálně srovnatelný objem tvoří i skupina více konkrétních frází (např.: židle pro děti, notebooky pro studenty). Je proto často výhodnější zaměřit se právě na skupinu konkrétnějších slov, kde je mnohem menší konkurence, než na obecná slova, kde je velká konkurence.

Grafické znázornění Long Tail konceptu je vidět na obrázku 1.7. Část blíže k ose Y s nejvyššími hodnotami se nazývá hlava grafu, zbylá část grafu s nízkými hodnotami se nazývá ocasem (chvostem) grafu.



Obrázek 1.7: Grafické znázornění Long Tail konceptu.

V případě Long Tail konceptu na příkladu internetového obchodu, je na obrázku 1.7 osa Y popularita produktu a na ose X jsou jednotlivé produkty. Je vidět, že existuje několik málo produktů, které jsou značně populární. Zaměřit se na tyto produkty je sice snadné, ale je zde velká konkurence. Kdežto u produktů dále od osy Y je sice popularita mnohem menší, těchto produktů je ale výrazně více a navíc konkurence je mnohem menší.

V oblasti vývoje softwaru se koncept Long Tail také objevuje a to převážně u větších firem. Hlava grafu představuje stěžejní aplikace pro firmu, které jsou dlouhodobě využívané. Naopak ocas grafu představuje malé aplikace pro malé skupiny uživatelů, které mohou mít velmi krátkou dobu života.

Jelikož malé aplikace tvoří velkou část firemních aplikací, je snaha co nejvíce odstínit IT oddělení a vývojáře od vytváření, správy a údržby těchto aplikací. Tyto aplikace mají většinou velice jednoduchou logiku a pracují s již existujícími daty. Proto je cílem nechat vytváření těchto aplikací na samotných uživatelích. Uživatelé vlastně tvoří jednoduché mashupy a je potřeba jim dát nástroj, který tvorbu mashupů ulehčí a jako přídavek navíc umožní jednoduše sdílet vytvořený mashup s ostatními kolegy v celé firmě.

Rozdíl mezi aplikací typu mashup a běžnou aplikace není pouze v tom

kdo jí vytvořil, ale také ve velikostech nákladů na vývoj. Náklady na vývoj klasické aplikace, kterou využívá velká část zaměstnanců firmy jsou shodné s vývojem klasické aplikace pro několik málo zaměstnanců. Přitom právě tyto okrajové aplikace v drtivé většině využívají již existující data. Mashupy přicházejí do podniků jako nová technologie, která se snaží snížit náklady na okrajové aplikace tím, že IT oddělení dodá podklady (webové služby, zdroje, data, widgety) a běžní uživatelé si tyto data pouze propojí podle vlastních představ.

1.6 Web 2.0

S pojmem *mashup* je často spojený i pojem *Web 2.0*. Hlavním důvodem tohoto spojení je fakt, že mashupy jsou jedním z důsledků Webu 2.0.

Web 2.0 má řadu charakteristických vlastností [23]:

- Obsah webu tvoří z velké části uživatelé.
- Grafické zpracování má větší váhu.
- Podpora uživatelské komunity.
- Decentralizovaná správa obsahu.
- Volně dostupné API (uvolnění práv k datům).
- Sdílení informací.
- Sociální funkce.

Nyní již je zřejmé, že web 2.0 byl hlavním impulsem k tvorbě mashupů. Majitelé jedinečných dat (mapové podklady, předpovědi počasí, zprávy, různé seznamy, informace atd.) poskytli svá data volně ostatním tvůrcům webů. Ti měli možnost využít dříve zpoplatněná data pro svůj web a začali tvořit první mashupy.

2 Technologie a formáty pro získání dat

V této kapitole budou popsány aktuálně nejpoužívanější technologie a formáty dat používané při tvorbě mashupů. Ve všech zmíněných případech se pro přenos dat používá protokol HTTP, případně jeho zabezpečená varianta HTTPS. Důvody pro využití protokolu HTTP jsou zřejmé. HTTP je na internetu nejpoužívanější protokol. Díky jeho obecnému návrhu lze přes požadavek POST odeslat prakticky libovolná data.

2.1 XML-RPC

Protokol XML-RPC [31] přináší do prostředí webu známou technologie RPC (Remote procedure call). Přenášená data od klienta jsou zabalena do zprávy ve formátu XML. Server zprávu rozbalí, data zpracuje a odpověď před odesláním opět zabalí do zprávy ve formátu XML. Balení zpráv do formátu pro přenos se nazývá *marshalling* a rozbalení se nazývá *unmarshalling*.

2.1.1 Formát zpráv

Zprávy v XML-RPC jsou odesílány jako HTTP požadavky metodou POST. Na následujících příkladech 2.1.1 a 2.1.2 lze vidět ukázkovou XML-RPC komunikaci, kterou klient žádá server o výsledek součtu dvou hodnot.

Každý HTTP požadavek metodou POST obsahuje hlavičku a tělo. V hlavičce XML-RPC požadavku nejsou oproti běžným HTTP POST požadavkům žádné nové položky. Jedinou změnou je nastavení hodnoty vlastnosti *Content-Type*, která musí být nastavena na hodnotu *text/xml* u všech XML-RPC požadavků a odpovědí. Požadavek musí mít také správně nastavenou hodnotu *Content-length*, která udává délku zprávy v těle požadavku.

Tělo požadavku je XML dokument. Každý požadavek má kořenový element s názevem *methodCall*. Jeden požadavek může obsahovat pouze jeden element *methodCall*. Potomkem tohoto elementu je element *methodName*, který udává název volané procedury. Pokud volaná procedura má parame-

try, následuje element *params*. V tomto elementu je pro každou proměnnou separátní element *param*. Uvnitř elementu je element *value*, který určuje hodnotu parametru. V XML-RPC nemají parametry své jméno. Rozhodující je pořadí. Ukázkový XML-RPC požadavek je uveden v příkladu 2.1.1.

```
POST /xml-rpc.php HTTP/1.0
User-Agent: Mozilla/5.0 (Windows NT 6.1)
Host: rpcserver.com
Content-Type: text/xml
Content-length: 197

<?xml version="1.0"? encoding="utf-8">
<methodCall>
 <methodName>soucet</methodName>
   <params>
     <param>
       <value><int>34</int></value>
     </param>
     <param>
       <value><int>25</int></value>
     </param>
   </params>
</methodCall>
```

Příklad 2.1.1: Ukázka kompletního požadavku klienta.

Odpověď na požadavek je také XML dokument. Každá odpověď má kořenový element s názvem *methodResponse*. Jedena odpověď může obsahovat pouze jeden element *methodResponse*.

V případě bezchybného vyřízení požadavku následuje element *params*, které obsahuje návratové hodnoty v elementech *param*. Kladně vyřízená XML-RPC odpověď je uvedena v příkladu 2.1.2.

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 131
Content-Type: text/xml
Date: Fri, 20 May 2011 16:45:19 GMT
Server: lighttpd/1.4.28

<?xml version="1.0"?>
<methodResponse>
   <params>
      <param>
         <value><int>59</int></value>
      </param>
   </params>
</methodResponse>
```

Příklad 2.1.2: Ukázka kompletního odpovědi serveru.

V případě jakékoliv chyby na úrovni XML-RPC se návratový kód HTTP

nemění a je stále 200 OK. Chyba je uvedena až v těle odpovědi. Element *methodResponse* neobsahuje element *params*, ale element *fault*. Uvnitř tohoto elementu je v elementu *value* struktura popisující nastalou chybu. Ukázka chybové XML-RPC odpovědi je uvedena v příkladu 2.1.3.

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 426
Content-Type: text/xml
Date: Fri, 20 May 2011 16:45:19 GMT
Server: lighttpd/1.4.28

<?xml version="1.0"?>
<methodResponse>
   <fault>
      <value>
         <struct>
            <member>
               <name>faultCode</name>
               <value><int>12</int></value>
               </member>
            <member>
               <name>faultString</name>
               <value><string>Příliš mnoho parametrů.</string></value>
            </member>
         </struct>
      </value>
   </fault>
</methodResponse>
```

Příklad 2.1.3: Ukázka kompletního chybové odpovědi serveru.

2.1.2 Datové typy

Veškeré parametry přenášené pomocí XML-RPC jsou v element *value*. Hodnota parametru je před vložením obalena elementem určujícím datový typ. XML-RPC podporuje skalární proměnné, strukturované proměnné a pole.

Skalární datové typy

V XML-RPC je podpora pro skalární datové typy uvedené v tabulce 2.1.

Určení datového typu přenášeného parametru není povinné. Pokud není datový typ uveden, je hodnota považována za řetězec.

Element	Datový typ	Příklad
i4 nebo int	celé číslo se znaménkem	*-23*
boolean	pravdivostní hodnota	*true* nebo *1*
string	řetězec	*Hello World*
double	reálné číslo	*-12.9581*
dateTime.iso8601	datum a čas	*20110519T11:02:54*
base64	data zakódovaná funkcí base64	*eW91lYWQ...*

Tabulka 2.1: Tabulka skalárních datových typů [31].

Strukturované datové typy

Jediným strukturovaným datovým typem v XML-RPC je datový typ *struct* tvořený stejnojmenným elementem. Jeden element *struct* obsahuje minimálně 1 element *member*. Každý element *member* musí obsahovat element *name* a *value*. Ukázka strukturovaného datového typu je uvedena v příkladu 2.1.4.

```
<struct>
   <member>
      <name>dolniMez</name>
      <value><i4>16</i4></value>
      </member>
   <member>
      <name>horniMez</name>
      <value><i4>18</i4></value>
   </member>
</struct>
```

Příklad 2.1.4: Ukázka strukturovaného datového typu [31].

Datový typ *struct* lze používat rekurzivně. To znamená, že některé z položek mohou obsahovat další pole nebo strukturu.

Pole

Datový typ pole je vymezený elementem *array*. Tento element musí obsahovat element *data*. Uvnitř tohoto elementu se již nacházejí jednotlivé položky pole v elementech *value*. V poli lze libovolně míchat datové typy. Ukázka použití datového typu pole je uvedena v příkladu 2.1.5.

Datový typ *array* lze používat rekurzivně. To znamená, že některé z položek mohou obsahovat další pole nebo strukturu.

```
<array>
   <data>
      <value><i4>18</i4></value>
      <value><i4>16</i4></value>
      <value><string>Praha</string></value>
      <value><boolean>1</boolean></value>
      <value><dateTime.iso8601>20110519T11:02:54</dateTime.iso8601></value>
   </data>
</array>
```

Příklad 2.1.5: Ukázka datového typu pole [31].

2.2 SOAP

Další velmi rozšířený protokol pro získávání dat je *SOAP* [27]. Oproti *XML-RPC* je mnohem flexibilnějším a rozsáhlejším protokolem. Formát zpráv je opět na bázi XML a pro komunikaci se využívá protokol HTTP a požadavky HTTP POST. Zajímavostí je, že lze namísto HTTP použít i jiný protokol (např. SMTP). Reálné použití na jiném protokolu než HTTP je však raritou. V protokolu SOAP se používají i další standardy a to konkrétně WSDL a XSD.

Tento protokol se využívá velmi často například v oblasti B2B při exportu aktuálních informací o produktech pro internetové obchody. Mnohdy, pokud se mluví o webové službě, je myšlen právě protokol SOAP.

2.2.1 Formát zpráv

Jedna z výhod zpráv protokolu SOAP je stejná struktura požadavku i odpovědi [27]. Prvním elementem zprávy je *Envelope*. Jedná se o povinný element, který říká, že se jedná o SOAP zprávu. Uvnitř tohoto elementu se mohou nacházet až 3 elementy: *Header*, *Body* a *Fault*.

Element *header* je nepovinný a do zprávy se vkládá pouze v případě, kdy je využit. Jeho použití je poměrně ojedinělé. Používá se většinou pro přenos metadat o přenášené zprávě, pro účely logování a nebo pro přenos přihlašovacích údajů (jméno a heslo, následně ticket).

Element *body* je jako jediný povinný. Obsahuje samotné tělo zprávy. Struktura těla zprávy může být prakticky libovolná, obvykle se však používá následující model. Tělo zprávy obsahuje element s názvem volané funkce.

Parametry funkce jsou uvnitř elementu funkce a každý parametr uvnitř elementu se svým jménem. Kompletní tělo zprávy je uvedeno v příkladu 2.2.1

Volitelný element fault se používá v případě, kdy nastala v průběhu zpracování požadavku chyba. V případě bezchybného zpracování požadavku se element zpravidla neuvádí. Element fault má následující potomky [27]:

- faultcode - Kód identifikující chybu.
- faultstring - Text popisující chybu.
- faultactor - Informace o viníkovi chyby.
- detail - Místo pro další rozsáhlejší informace o chybě.

Element faultcode může mít jednu z následujících hodnot [27]:

- VersionMismatch - Nastavený špatný namespace pro požadavek.
- MustUnderstand - Obsah elementu v hlavičce zprávy s nastaveným atributem MustUndestand nebyl pochopen.
- Client - Požadavek byl ve špatném formátu, nebo obsahoval nesprávné informace. Za chybu může klient.
- Server - Nastal problém na serveru, který zprávu nedokáže zpracovat. Za chybu může server.

2.2.2 WSDL

WSDL [26] je jazyk založený na XML. Popisuje konkrétní webovou službu, její umístění, metody, parametry a další informace nutné pro přístup a komunikaci. WSDL má přesně danou strukturu, a je dobře programově zpracovatelný. Pokud webová služba obsahuje korektní WSDL soubor je přístup ke službě velice jednoduchý. WSDL lze totiž považovat za jakousi dokumentaci pro SOAP klienta. Většina dnešních SOAP klientů je schopna při inicializaci zpracovat WSDL soubor a následně nabízet přímé volání funkcí vzdálené webové služby. Programátor proto nemusí znát detaily komunikace.

```
POST /webservice.php HTTP/1.1
Host: www.webservice.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 309

<?xml version="1.0"?>
<soap:Envelope
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
   <soap:Body>
      <GetProduct>
         <ProductId>123456</ProductId>
      </GetProduct>
   </soap:Body>
</soap:Envelope>
```

Příklad 2.2.1: Ukázka obvyklé struktury SOAP zprávy odeslané přes HTTP.

Soubor s WSDL popisem webové služby obsahuje jeden kořenový element *definitions*. Potomci tohoto elementu jsou elementy *types*, *message*, *portType*, *binding* a *service*. Elementy musí být v uvedeném pořadí a každý z nich musí být uveden minimálně jednou.

Element types

V elementu *types* se definují uživatelské datové typy používané ve zprávách. WSDL používá pro definici datových typů XSD schéma. Popsání jazyka XSD by zabralo minimálně jednu kapitolu, proto zde shrnu jen základní informace, nutné pro pochopení jazyka WSDL.

Samo o sobě obsahuje XSD řadu běžných datových typů např.: Boolean, Byte, Date, Double, Float, Integer, String a Time. Tyto typy se nemusí v elementu *types* deklarovat a lze je použít přímo v elementu *message*.

V některých případech je potřeba blíže upřesnit datový typ. K tomuto účelu se používají jednoduché datové typy (*simpleType*). Pomocí *simpleType* lze vytvořit nový datový typ a nastavit mu pomocí *restriction* různé omezení. Například lze nastavit minimální a maximální délku, povolení určitých hodnot apod. Vytvoření jednoduchého datového typu s restrikcemi je uvedeno na příkladu 2.2.2.

Další možností, kterou XSD nabízí, jsou tzv. komplexní datové typy (*complexType*). Komplexní datový typ je takový typ, který může mít atributy a potomky. Jednoduchý datový typ je mít nemůže. Na příkladu 2.2.3 je ukázka

```
<xs:simpleType name="loginString">
   <xs:restriction base="xs:string">
      <xs:minLength value="3" />
      <xs:maxLength value="10" />
   </xs:restriction>
</xs:simpleType>
```

Příklad 2.2.2: Ukázka vytvoření vlastního jednoduchého datového typu v XSD.

definice komplexního datového typu *mujProdukt*. Tento datový typ lze využít například pro zprávu vracející produkt z internetového obchodu. Část XML dokumentu odpovídajícího tomuto schématu je uvedena na příkladu 2.2.4.

```
<xsd:complexType name="Product">
   <xsd:sequence>
      <xsd:element minOccurs="1" maxOccurs="1" name="ProductId" type="xsd:int"/>
      <xsd:element minOccurs="0" maxOccurs="1" name="Name" type="xsd:string"/>
      <xsd:element minOccurs="1" maxOccurs="1" name="Price" type="xsd:decimal"/>
      <xsd:element minOccurs="0" maxOccurs="1" name="Warranty" type="xsd:string"/>
      <xsd:element minOccurs="1" maxOccurs="1" name="Vat" type="xsd:decimal"/>
      <xsd:element minOccurs="1" maxOccurs="1" name="OnStock" type="xsd:boolean"/>
      <xsd:element minOccurs="0" maxOccurs="1" name="ImageUrl" type="xsd:uri"/>
      <xsd:element minOccurs="0" maxOccurs="1" name="Description" type="xsd:string"/>
   </xsd:sequence>
</xsd:complexType>
```

Příklad 2.2.3: Ukázka vytvoření vlastního komplexního datového typu v XSD.

```
<Product>
   <ProductId>123</ProductId>
   <Name>Acer Aspire 5110 Intel i3</Name>
   <Price>12300.00</Price>
   <Warranty>24 měsíců</Warranty>
   <Vat>20.00</Vat>
   <OnStock>false</OnStock>
</Product>
```

Příklad 2.2.4: Ukázka vlastního komplexního datového typu v XML.

Element message

V elementu *message* se přiřazují datové typy definované v části *types* jednotlivým parametrům funkcí webové služby. Tyto parametry budou následně použity v části *portType*.

Obsah tohoto elementu se liší podle toho, jaký je vybraný způsob vazeb. Existují dva typy vazeb: *RPC* a *Document*. Vazby se nastavují v poslední

části WSDL dokumentu, v elementu *binding*. Pro jednoduchost zde popíši situaci při použití vazby *RPC*.

V dokumentu WSDL musí být definována minimálně jedna zpráva. Většinou však bude zpráv více a navíc počet zpráv bude s velkou pravděpodobností dělitelný dvěma (první zpráva pro požadavek, druhá zpráva pro odpověď). Ukázka definované zprávy je uvedena v příkladu 2.2.5.

```
<message name="getProductRequest">
   <part name="id" type="xs:integer" />
   <part name="color" type="xs:string" />
</message>

<message name="getProductResponse">
   <part name="id" type="xs:integer" />
   <part name="name" type="xs:string" />
   <part name="price" type="xs:double" />
   <part name="vat" type="xs:double" />
</message>
```

Příklad 2.2.5: Ukázka definice zpráv v dokumentu WSDL (vazba RPC).

Element portType

V této části WSDL dokumentu se přiřazují jednotlivé zprávy funkcím (operacím) webové služby. Webová služba může pracovat ve 4 režimech [26]:

- **One-way** - Jednosměrná komunikace. Klient odešle serveru požadavek. Server neodpovídá. Tento režim se prakticky nepoužívá. Obvykle je nahrazen režimem *Request-response*, kdy server odpoví minimálně tím, že požadavek zpracoval nebo že požadavek skončil chybou.
- **Request-response** - Klasická obousměrná komunikace. Klient odešle serveru požadavek a server na něj odpoví. Tento režim je nejčastěji používaný.
- **Solicit-response** - Obrácená obousměrná komunikace. Server odešle klientovy výzvu a klient na ni odpoví.
- **Notification** - Jednosměrná komunikace. Server odešle klientovi výzvu a klient na ni neodpovídá.

Ze všech režimů se prakticky používá pouze *Request-response*. Na příkladu 2.2.6 je uvedena jednoduchá ukázka přiřažení funkcí ke zprávám z elementu *messages*.

```
<portType name="getProductPort">
   <operation name="getProduct">
      <input message="getProductRequest">
      <output message="getProductResponse">
   </operation>
   <operation name="getProductImage">
      <input message="getProductImageRequest">
      <output message="getProductImageResponse">
   </operation>
</message>
```

Příklad 2.2.6: Ukázka definice operací v části portType.

Element binding

Část WSDL dokumentu uvozená elementem *binding* obsahuje rozšířené informace o jednotlivých operacích definovaných v části *portType*. Tyto rozšiřující informace se používají pro konfiguraci režimu SOAP klienta. Struktura této části bývá obvykle velice podobné příkladu 2.2.7.

```
<binding name="ProductBind" type="getProductPort"
   <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
   <operation name="getProduct">
      <soap:operation />
      <input>
         <soap:body use="literal" />
      </input>
      <output>
         <soap:body use="literal" />
      </output>
    </operation>
</binding>
```

Příklad 2.2.7: Ukázka definice operací v části portType.

Element service

V poslední části WSDL souboru uvozené elementem *service* je definováno přesné umístění přípojného bodu. Zjednodušeně řečeno se v tomto elementu nachází URL adresa na které naslouchá konkrétní webová služba. Ukázka této části dokumentu je uvedena na příkladu 2.2.8.

```
<service name="Service">
   <port name="ServiceSoap" binding="tns:ServiceSoap">
      <soap:address location="http://www.example.com/webservice.asmx" />
   </wsdl:port>
</service>
```

Příklad 2.2.8: Ukázka části service v dokumentu WSDL.

2.2.3 UDDI

Poslední součástí, která patří k webovým službám založeným na protokolu SOAP je UDDI. Zjednodušeně řečeno je UDDI katalog webových služeb. K celému katalogu je umožněn přístup pomocí protokolu SOAP. Celý systém webových služeb založených na protokolu SOAP může pracovat takto:

1. Vyhledání webové služby v registru UDDI (SOAP zprávy)
2. Získání adresy WSDL dokumentu z UDDI
3. Stažení WSDL dokumentu
4. Zjištění adresy webové služby, parametrů a dalších informací z WSDL
5. Komunikace s webovou službou (SOAP zprávy)

Velké korporace využívající SOAP mohou mít vlastní UDDI registry [18]. Veřejné UDDI registry jsou například:

- http://webservices.seekda.com
- http://www.xmethods.com/ve2/index.po

Celkově je UDDI nejméně využívaná část patřící k protokolu SOAP.

2.3 REST

Do skupiny k protokolům SOAP a XML-RPC se přiřazuje také REST, ač se nejdená o protokol, ani formát. REST (Representational State Transfer) je architektura popsána v [8]. Hlavní výhodou RESTu je jeho jednoduchost. Požadavky jsou klasické HTTP požadavky (obvykle GET). Odpověď na požadavek může být v libovolném formátu (např.: XML, JSON, RSS, plain-text).

REST je bezestavový, proto je nutné u každého požadavku zadávat všechny parametry, které jsou potřeba.

Velmi často se REST používá na webových stránkách využívajících technologii AJAX. Důvod je jednoduchý, požadavek je běžný HTTP požadavek, který lze odeslat pomocí objektu XMLHttpRequest a odpověď může být například ve formátu JSON, který lze v JavaScriptu velice snadno zpracovat. Odpadá tak vytváření XML s požadavkem a parsování XML s odpovědí, které by bylo nutné při použití XML-RPC nebo SOAP.

2.4 RSS a ATOM

RSS [32] a ATOM [19] jsou o dva velice podobné a velmi populární formáty. Podobnost je dána vývojem, kdy novější ATOM vychází z původního dříve navrženého RSS.

Oba formáty jsou primárně určeny pro získávání často aktualizovaných informací jako například zprávy, komentáře a výsledky. Zpravodajské servery (například idnes.cz, živě.cz nebo NewYorkTimes.com) velice často poskytují svým čtenářům volně k dispozici aktuální zprávy ve formě feedu ve formátu RSS resp. ATOM. Proto jsou tyto formáty velice rozšířené a známé i u běžných uživatelů.

2.4.1 RSS

Pod pojmem RSS je skryto hned několik XML formátů. V průběhu vývoje tohoto formátu vznikly dvě navzájem nekompatibilní větve.

- RSS 0.90, 1.0 a 1.1
- RSS 0.91, 0.92 a 2.0

Zde bude popsána druhá větev a to konkrétně RSS 2.0, která je častěji používaná a má jednodušší strukturu.

Formát RSS verze 2.0 (Really Simple Syndication původně Rich Site Summary) [32] je kompatibilní se staršími verzemi 0.91 a 0.92. Největším rozdílem je přidání nepovinných nových elementů například pro podporu podcastů.

Začátek RSS 2.0 dokumentu je uvozen elementem *rss* s atributem *version*, který určuje použitou verzi RSS. Následuje vnořený element *chanel*, který obsahuje metadata kanálu a samotný obsah kanálu. Kanál je pouze jiné pojmenování pro zdroj obsahu.

V metadatech se vyskytují elementy pro název, popis a jazyk kanálu. Dále také odkaz na zdrojový web, datum poslední aktualizace a mnoho dalších. Následně jsou v dokumentu umístěny jednotlivé položky (elementy *item*). Každá položka má několik elementů, které určují její název, popis, datum zveřejnění a odkaz na zdroj. Kompletní obsah RSS 2.0 feedu je uveden na příkladu 2.4.1.

```
<?xml version="1.0" encoding="utf-8" ?>
<rss version="2.0">
  <channel>
    <title>Můj blog</title>
    <link>http://www.mujblog.tld/</link>
    <description>Můj osobní blog.</description>
    <language>cs</language>
    <pubDate>Sun, 29 Oct 2011 18:18:00 GMT</pubDate>
    <lastBuildDate>un, 29 Oct 2011 20:18:00 GMT</lastBuildDate>
    <docs>http://www.rssboard.org/rss-specification</docs>
    <generator>RSS System 1.0</generator>
    <managingEditor>info@mujblog.tld</managingEditor>
    <webMaster>webmaster@mujblog.tld</webMaster>
    <item>
      <title>Můj druhý příspěvek</title>
      <link>http://www.mujblog.tld/muj-druhy-prispevek</link>
      <description>Na světě je můj druhý příspěvek na blogu.</description>
      <pubDate>Sun, 29 Oct 2011 10:18:00 GMT</pubDate>
    </item>
    <item>
      <title>První příspěvek</title>
      <link>http://www.mujblog.tld/prvni-prispevek</link>
      <description>Popisek mého prvního příspěvku.</description>
      <pubDate>Sun, 29 Oct 2011 06:18:00 GMT</pubDate>
    </item>
  </channel>
</rss>
```

Příklad 2.4.1: Ukázka RSS 2.0 dokumentu.

2.4.2 ATOM

Formát ATOM vychází z RSS a snaží se spojit výhody obou rozštěpených verzí RSS. V roce 2005 vznikla první verze ATOM 1.0. [19].

Oproti RSS jsou některé elementy pojmenovány jinak, ale základní struktura zůstala zachována. Samovysvětlující ukázka formátu atom je uvedena

na příkladu 2.4.2.

```
<?xml version="1.0" encoding="utf-8" ?>
<feed xmlns="http://www.w3.org/2005/Atom" >
 <title>Můj blog</title>
 <link href="http://www.mujblog.tld/feed/" rel="self" />
 <link href="http://www.mujblog.tld/" />
 <updated>2011-10-30T18:30:02Z</updated>
 <author>
   <name>Josef Vrba</name>
   <email>vrba@students.zcu.cz</email>
 </author>
 <entry>
   <title>Můj první příspěvek</title>
   <link href="http://www.mujblog.tld/muj-prvni-prispevek" />
   <id>/muj-prvni-prispevek</id>
   <updated>2011-10-30T18:20:02Z</updated>
   <summary>V mém prvním příspěvku vítám všechny návštěvníky mého blogu.</summary>
 </entry>
</feed>
```

Příklad 2.4.2: Ukázka ATOM 1.0 dokumentu.

2.5 JSON

JSON (JavaScript Object Notation) [5] je velmi oblíbený formát dat používaný na webu. Zásadním rozdílem tohoto formátu oproti předchozím je skutečnost, že není založen na formátu XML. JSON je textový formát, který využívá syntax programovacího jazyka JavaScript. Přenáší se vlastně programový kód JavaScriptu. Pokud tento kód JavaScriptový klient vyhodnotí pomocí funkce *Eval*, získá veškerá data v podobě lokálních objektů.

Ve formátu JSON jsou k dispozici následující datové typy:

- Číslo
- Textový řetězec
- Boolovská proměnná
- Pole
- Objekt
- null

JSON zapisuje obsah proměnných ve formátu název/hodnota, kdy název je oddělen od hodnoty dvojtečkou. Obsah polí je uvozen znakem hranaté závorky, nový objekt je uvozen znakem složené závorky. Jednotlivé proměnné a objekty lze do sebe neomezeně vnořovat. Ukázka formátu JSON je uvedena na příkladu 2.5.1.

```
{ produkt: {
      nazev: "Acer Aspire 5110 Intel Core i3"
      partNo: "LX.12211.A21"
      cena: 12500
      skladem: true
      barva: ["černá", "stříbrná"]
      kategorie: {
         hlavni: "/notebooky/15 palcu/acer"
         vedlejsi: "/notebooky/Intel/Acer/15 palcu"
      }
   }
}
```

Příklad 2.5.1: Ukázka formátu JSON.

Formát JSON se používá i v jiných programovacích jazycích. Existuje mnoho knihoven, které umožňují práci s formátem JSON. Nejčastější použití JSON je však ve webových aplikacích využívajících AJAX, kde se využívá prakticky jenom JSON. Důvodů je hned několik. JSON je znatelně menší co se týče objemu přenášených dat oproti XML. Lze jej také jednoduše vyhodnotit na straně klienta pomocí JavaScriptu. Data ve formátu JSON lze získat i z jiné domény a to bez použití jakékoliv proxy brány.

Způsob získávání JSON dat z jiných domén se nazývá JSONP. Jde vlastně o techniku jak obejít ochranu webových prohlížečů proti útokům typu Cross-Site script. Tato technika je velice jednoduchá. Požadavek se vytváří klasickým HTML tagem *script*, kde se jako zdroj uvede vzdálená doména a parametry dotazu se uvedou do URL. Tento tag se vloží pomocí JavaScriptu do webové stránky. Prohlížeč tento skript musí v každém případě stáhnout a vyhodnotit. Této vlastnosti se využije. Stažený skript obsahuje volání funkce a jako parametry jsou uvedena návratová data. Ukázka obsahu tohoto skriptu s voláním funkce *callback* je uvedeno na příkladu 2.5.2.

2.6 Lokální soubory

Pro úplnost zde uvádím i možnost získávat data z lokálních souborů. Lokální soubor může být v libovolném formátu. V úvahu připadá například XML,

```
callback({ produkt: {
      nazev: "Acer Aspire 5110 Intel Core i3"
      partNo: "LX.12211.A21"
      cena: 12500
      skladem: true
      barva: ["černá", "stříbrná"]
      kategorie: {
         hlavni: "/notebooky/15 palcu/acer"
         vedlejsi: "/notebooky/Intel/Acer/15 palcu"
      }
   }
});
```

Příklad 2.5.2: Ukázka skriptu při použití techniky JSONP.

CSV ale i TXT. Získání přístupu k těmto datům je pak ze všech vyjmenovaných možností nejjednodušší.

2.7 Screening obrazovky - vytěžování stránek

Získat konkrétní data z webových stránek lze i jinak než pomocí API nebo veřejných webových služeb. Jednou z možností je tzv. *Screening obrazovky* [2] jinak též *vytěžování stránek* [15]. Tímto způsobem lze získat prakticky jakákoliv data, která jsou volně k dispozici na internetu. Před dalším popisem je nutné upozornit, že tento způsob získání dat nemusí být za všech okolností legální. Podrobněji jsou podmínky legálního získávání dat tímto způsobem uvedeny v části 3.2.

Z cizích webových stránek lze získat prakticky jakákoliv součást. Může to být obrázek, video, text, data nebo složitější věci vytvořené v JavaScriptu. V této části se však zaměřím na získávání surových dat - různé formy seznamů či tabulek. Metody pro získávání těchto dat jsou různé, já se zde omezím pouze na dvě nejzajímavější metody. Obě metody budou popsány na příkladu. Cílem bude získat seznamu kempů z webu DoKempu.cz.

2.7.1 Parsování zdrojového kódu

Nejzákladnější metodu získávání dat z webu je tzv. *Parsování zdrojového kódu*. Tato metoda lze použít pro libovolnou webovou stránku, která nepoužívá pro načítání dat AJAX. Jinak řečeno, tuto metodu lze použít na všechny webové stránky, které vrací na požadavek GET/POST hledaný obsah.

Nyní k příkladu. Po prohledání webu se dostaneme k detailu konkrétního kempu na URL adresu například:

```
http://www.dokempu.cz/kempy/straznice/autokempink-217/
```

Jak je vidět, adresa obsahuje na svém konci ID kempu. Pokud ID ručně změním a nechám předchozí hodnoty bez povšimnutí, systém mě automaticky přesměrovává na detaily kempů se zadaným ID. Této funkce lze zneužít a jednoduše tak získat URL adresy detailů všech kempů.

Když už máme URL adresy všech kempů, přijde na řadu parsování. Parsuje se zdrojový kód stránky, takže je nejprve nutné stáhnout zdrojový kód všech stránek. Následně ve zdrojových kódech vyhledat element *h1* a získat jeho obsah. Uvnitř tohoto elementu je název kempu.

Upozorňuji, že pro získání názvů kempů lze použít i jednodušší parsování využívající seznam kempů na adrese:

```
http://www.dokempu.cz/kempy/cr/
```

Ve většině případů se však hodí získat více informací i za cenu větší složitosti.

2.7.2 Využití AJAXu

Další možnost získání dat lze využít na stránkách, které využívají AJAX a dynamické načítání obsahu.

Nyní k příkladu. Na úvodní stránce webu DoKempu.cz je k dispozici interaktivní mapa, na které jsou zobrazeny kempy. Pokud je s mapou provedena akce posun nebo změna měřítka, automaticky se načtou kempy na viditelné části mapy. Toto automatické načítání kempů je věc kterou hledáme. Po prozkoumání kódu lze zjistit, že na adresu:

```
http://dokempu.cz/xml-markers
```

jsou odesílány asynchronní požadavky s žádostmi o seznam kempů v rozsahu GPS souřadnic předaných pomocí parametrů GET.

Nyní již je vyhráno. Byla objevena jednoduchá REST služba. Požadavky se posílají ve formátu HTTP GET a odpovědi jsou XML s kempy a jejich GPS souřadnice.

2.8 Srovnání technologií

Na závěr této kapitoly se pokusím srovnat všechny výše uvedené technologie a formáty z pohledu programátora. Přestože se jedná o velice odlišné technologie a v několika případech se jedná pouze o formát dat, pokusil jsem se o jejich srovnání, které je v tabulce 2.2.

	XML-RPC	SOAP	REST	RSS a ATOM	JSON	Lokální soubory	Screaming obrazovky
Def. způsob přístupu	ANO	ANO	ANO	NE	NE	ANO	ANO
Def. formát dat	ANO	ANO	NE	ANO	ANO	NE	ANO
Složitost přístupu	střední	střední	malá	-	-	malá	velká
Složitost formátu	střední	velká	-	malá	malá	-	velká

Tabulka 2.2: Srovnání formátů a technologií.

Ze srovnání se může zdát, že některé technologie jsou složité a proto budou málo využívané. To ovšem nemusí být vždy pravda. Například SOAP je nejsložitější z popsaných technologií. Pro jednoduchou komunikace na webu se téměř nepoužívá, ale pro složité a obsáhlé datové výměny je to velice často používaná technologie.

3 Zdroje dat

V této kapitole budou stručně popsána jednotlivá API a zdroje dat použité v ukázkové aplikaci.

3.1 API

API (Application Programming Interface) je programové rozhraní aplikace, které umožňuje programově přistupovat k metodám konkrétní aplikace. Pro představu například mapový portál má webové rozhraní pro uživatele. Toto rozhraní však nelze jednoduše programově ovládat. K tomu slouží API, které poskytuje veškerou funkčnost jako webové rozhraní, ale je jednoduše ovladatelné z programu pomocí metod a funkcí.

3.2 Legálnost získávání dat

Před použitím získaných dat z webových stránek je nutné ujistit se, že data byla získána legálně a smí být použita v mashupu.

V případě veřejně otevřených API je situace poměrně jednoduchá. Data jsou sice autorsky chráněna, ale autor svolil k jejich využití. Toto svolení je obvykle vyjádřeno v podmínkách použití, kde je specifikováno za jakých okolností mohou být data použita. Nejčastěji se lze setkat s omezením, že API nesmí být použito pro komerční účely nebo je omezen počet dotazů za časový úsek. Při tvorbě mashupu v rámci této práce nehrozí porušení ani jedné z těchto častých podmínek.

Jiná situace je však v případě, kdy webové stránky neposkytují žádné API a data by byla získávána bez vědomí autora. I v tomto případě jsou data autorsky chráněná, ale autor nesouhlasil s poskytnutím dat a proto je nutné získat souhlas autora k použití dat.

3.3 Použitá API

V této části budou představeny zdroje dat, použité ve vzorové aplikace. Řada z nich patří mezi nejoblíbenější API současného internetu. U každého zdroje dat bude uveden příklad na vytvoření jednoduché "Hello World"aplikace. Složitější příklady jsou uvedeny v dokumentaci příslušného API.

3.3.1 Google Maps API

Google Maps API [11] patří k nejoblíbenějším a nejpoužívanějším [22] zdrojům dat současného internetu. Aktuálně již je k dispozici API verze 3. Jedná se o API postavené na JavaScriptu, které poskytuje funkce zobrazení mapových podkladů světa, vyhledávání států, měst, ulic a objektů na celém světě.

Před začátkem programování je potřeba získat vlastní jedinečný klíč k přístupu do API. Tento klíč lze získat na stránce s dokumentací, která je uvedena na konci této části.

Prvním krokem, k vytvoření mapy je vložení zdrojového JavaScriptu. To se provede vložením následujícího kódu mezi tagy *head* v HTML dokumentu.

```
<script type="text/javascript"
  src="http://maps.googleapis.com/maps/api/js?sensor=false>
</script>
```

Druhým krokem je vytvoření elementu *DIV* a určení jeho rozměrů. Uvnitř tohoto elementu bude JavaScriptem automaticky vložena mapa. To lze provést například tímto kódem.

```
<div id="mapa" style="width:200px; height: 200px;"></div>
```

Třetí krok již obsahuje kód v JavaScriptu, který zařídí zobrazení mapy. Kód najde element s id *mapa*, a do něj vloží mapu podle vlastností z proměnné *nastaveni*. V této proměnné je nastaven střed mapy na souřadnice Prahy, je nastaveno přiblížení na hodnotu 10 a klasický typ mapy.

```
<script type="text/javascript">
  function inicializace() {
    var nastaveni = {
      center: new google.maps.LatLng(50.08, 14.42),
      zoom: 10,
      mapTypeId: google.maps.MapTypeId.ROADMAP
    };
    var mapa = new google.maps.Map(
                    document.getElementById("mapa"),
                    nastaveni);
  }
</script>
```

Poslední krok obsahuje kód pro spuštění JavaScriptové funkce *inicializace* po dokončení načtení dokumentu.

```
<body onload="initialize()">
```

Po spuštění bude zobrazena mapa o velikosti 200 x 200 pixelů se středem v Praze. Google Maps API obsahuje mnoho dalších funkcí. Na mapu lze umístit značky (*Marker*), lze zobrazovat různé informační okna (*InfoWindow*). Lze také naslouchat různým událostem (např. změna přiblížení, změna pozice mapy). Všechny tyto vlastnosti jsou přehledně popsány v dokumentaci.

Ukázkový výsledek použití tohoto API je na obrázku 3.1.

- Dokumentace: https://developers.google.com/maps/documentation/
- Alternativní API: Bing Maps API, Seznam Mapy API, Yahoo Maps API

3.3.2 Weather Underground API

Dalším zdrojem dat, tentokrát zdroj předpovědi počasí, je Weather Underground API [30]. Api obsahuje mimo jiné i funkce jednoduchého geocodéru, kterému stačí poslat název města a jako výsledek je vrácen seznam meterologických stanic v okolí. Následně lze vybrat nejbližší stanici a zjistit předpověď.

Nejzajímavější funkcí tohoto API je funkce forecast. Jako parametry lze poslat GPS souřadnice, ID meterologické stanice nebo název města. Tato funkce se volá jako GET požadavek. Odpověď může být ve formátu XML nebo JSON. Získání aktuální předpovědi v Plzni ve formátu JSON je možné posláním požadavku na adresu:

```
http://api.wunderground.com/api/API_KLIC/forecast/q/plzen.json
```

Jedná se tedy o API využívající technologii REST. Při registraci je přidělen každé aplikaci jednoznačný API klíč, kterým se aplikace musí při každém dotazu prokázat. Pokud je na jeden klíč zaznamenáno více než 100 požadavků, jsou další požadavky zablokovány. Neomezený počet požadavků je zpoplatněn.

Ukázkový výsledek použití tohoto API je na obrázku 3.2.

- Dokumentace: http://www.wunderground.com/weather/api/
- Alternativní API: Yahoo Weather API, Google Weather API, InPocasi API

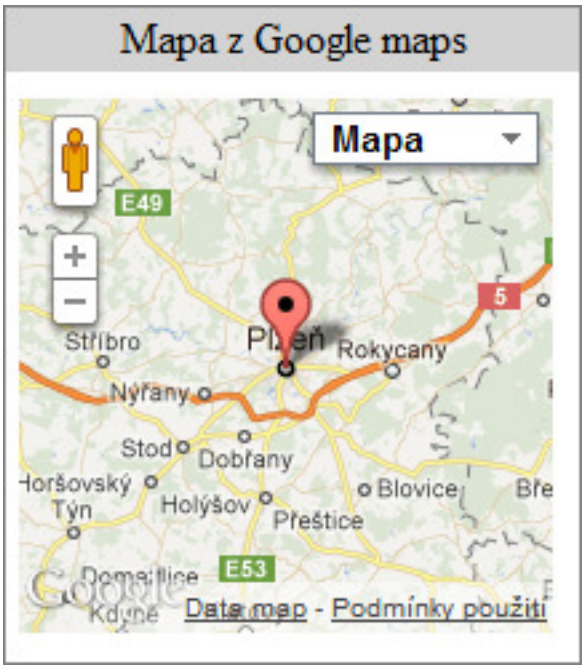
3.3.3 Panoramio API

Na webu Panoramio.com se uživatelé mohou podělit o své fotografie. Každá fotografie musí být přiřazena místu, kde byla pořízena. K dispozici je veřejné API, které poskytuje funkce na získání fotek podle GPS souřadnic.

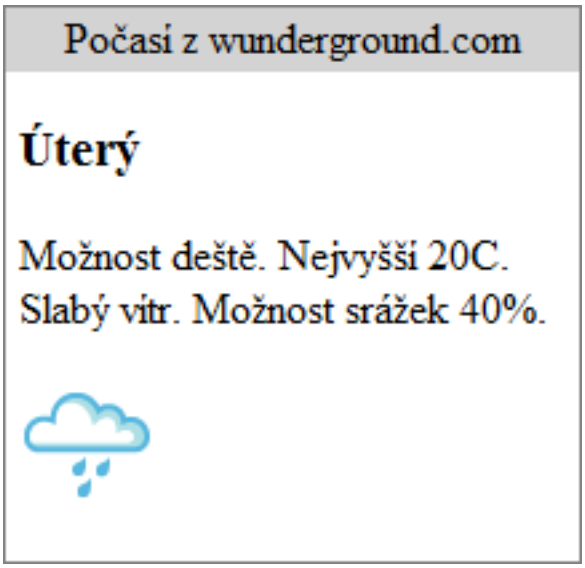
Toto API [12] nabízí několik možností jak získat fotografie. Na výběr je několik hotových komponent, které se vkládají pomocí JavaScriptu obdobným způsobem jako GoogleMaps. Výhodnější však může být v některých případech použit funkce, které vrací adresy fotografií. Mezi tyto funkce patří funkce *get panoramas* naslouchající na adrese:

```
http://www.panoramio.com/map/get_panoramas.php
```

Funkce má následující parametry:



Obrázek 3.1: Ukázka použití Google Maps API



Obrázek 3.2: Ukázka použití Weather Underground API

- **set** - typ vrácených fotografií. Možnosti jsou *full* nebo *public*.
- **from** - číslo první vrácené fotografie.
- **to** - číslo poslední vrácené fotografie.
- **minx** - určení polohy pořízení fotografie
- **miny** - určení polohy pořízení fotografie
- **maxx** - určení polohy pořízení fotografie
- **maxy** - určení polohy pořízení fotografie
- **size** - velikost vrácených fotografií. Možnosti jsou *medium*, *small*, *original* a další.

Vrácen je seznam fotografií, které byly pořízeny ve výseku GPS souřadnic. Oproti podobné funkci, kterou nabízí Flickr API vrací Panoramio API několikanásobně větší množství fotografií.

Ukázkový výsledek použití tohoto API je na obrázku 3.3.

Obrázek 3.3: Ukázka použití Panoramio API

- Dokumentace: http://www.panoramio.com/api/widget/api.html
- Alternativní API: Flickr API, Picasa API

3.3.4 Wikipedia API

Poměrně málo známé API poskytuje i server Wikipedia [17]. I přes velké množství nejrůznějších funkcí, které umožňují export prakticky úplně všeho z wikipedie zde chybí jenda zásadní funkce. Pokud chceme získat čistý text o nějakém pojmu, nezbývá nic jiného, než získat HTML a odstranit z něj HTML tagy. Programové rozhraní API naslouchá na adrese:

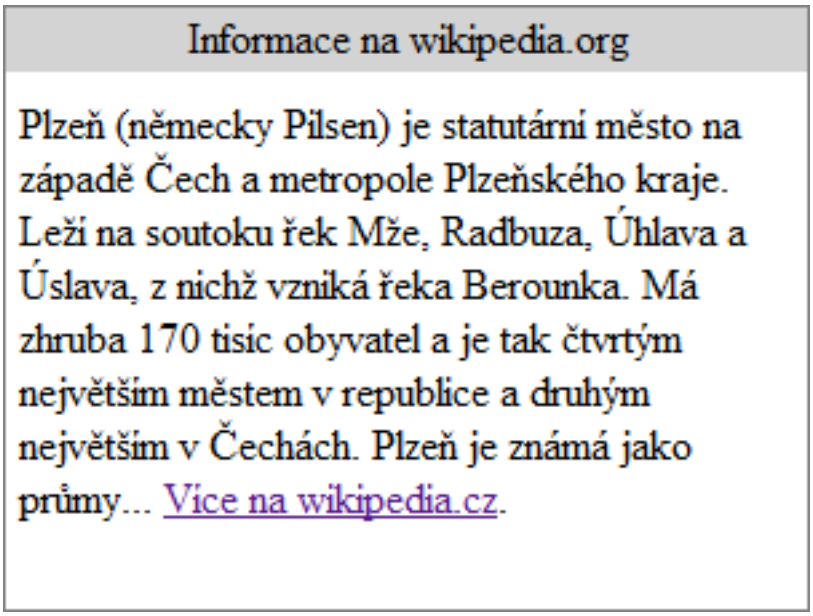
```
http://cs.wikipedia.org/w/api.php
```

Nejvíce vyhovující nastavení pro získání čistého textu se zdá být kombinace následujících parametrů:

- **action** s hodnotou *query* oznamuje požadavek na získání dat.
- **prop** s hodnotou *revisions* oznamuje požadavek na získání stránky.
- **titles** s názvem stránky určuje, jakou stránku chceme získat.
- **rvprop** s hodnotou *content* oznamuje požadavek na získání obsahu stránky.
- **rvsection** s hodnotou *0* omezuje vrácená data pouze na první sekci.
- **rvparse** s hodnotou *1* určuje, že požadavek nemá být ve formátu wikitext, ale v HTML.

Výsledkem výše uvedeného dotazu je JSON (případně XML), které obsahuje HTML obsah konkrétní wikistránky. V případě nutnosti získat čistý text, je nutné z obsahu odstranit HTML tagy. Aby dával čistý text smysl je nutností odstranit veškeré tabulky z textu.

Ukázkový výsledek použití tohoto API je na obrázku 3.4.



Obrázek 3.4: Ukázka použití Wikipedia API

3.3.5 DoKempu.cz API

DoKempu.cz nabízí ojedinělé API [9], díky kterému lze získat seznam kempů v okolí určité souřadnice. Podobná API bývají často zpoplatněná. Toto API je kompletně zdarma, ale nabízí pouze velice omezené možnosti. Zavoláním níže uvedené adresy lze získat seznam kempů v okolí 20 kilometrů od Prahy. Souřadnice je nutné zjistit předem například z Google Maps API.

```
http://www.dokempu.cz/export/?x=50.08&y=14.42&r=20
```

Toto API využívá technologii REST. Další informace a příklady jsou uvedeny v jednoduché dokumentaci. Ukázkový výsledek použití tohoto API je na obrázku 3.5.

- Dokumentace: http://www.dokempu.cz/spoluprace/
- Alternativní API: Active.com API

3.3.6 HejbejteSe.cz API

Api z webu HejbejteSe.cz je neveřejné, ale po dohodě s provozovatelem ho mohu využívat ve všech aplikacích spojených s touto prací. Jedná se o velice jednoduché REST API, které obsahuje jedinou funkci *get companies*. Jako parametry jsou GPS souřadnice čtverce *gps-lng-1*, *gps-lng-2*, *gps-lat-1* a *gps-lat-2*. Tyto parametry je nutné poslat jako HTTP POST požadavek na adresu:

```
http://www.hejbejtese.cz/get/companies
```

Bohužel k tomuto API není k dispozici žádný manuál ani dokumentace. Ukázkový výsledek použití tohoto API je na obrázku 3.6.

- Dokumentace: není k dispozici
- Alternativní API: Active.com API

3.3.7 KudyZnudy.cz RSS

Web KudyZnudy.cz poskytuje několik druhů RSS [6]. Patří mezi ně výlety, aktivity a akce. Všechny RSS dostupné buď pro celou republiku, nebo pro jednotlivé kraje. RSS jsou dostupné na adrese:

```
http://www.kudyznudy.cz/Rss.aspx
```

Ukázkový výsledek použití tohoto RSS je na obrázku 3.7.

Kempy z dokempu.cz		
Autocamp Ostende - Plzeň	Ukázat na mapě	Více info
Autocamp INA - Plzeň - Litice	Ukázat na mapě	Více info
Bílá Hora - Plzeň	Ukázat na mapě	Více info
U Dolanského mostu - Třemošná	Ukázat na mapě	Více info
Autokemp U Jezera - Ejpovice	Ukázat na mapě	Více info
U přívozu - Třemošná	Ukázat na mapě	Více info

Obrázek 3.5: Ukázka použití DoKempu API

Sportoviště z hejbetese.cz			
M Factory	Rokycanská 33, Plzeň	Ukázat na mapě	Více info
Sportcentrum Koloseum	Sokolovská 74, Plzeň	Ukázat na mapě	Více info
TJ Lokomotiva Plzeň	Úslavská 75, Plzeň	Ukázat na mapě	Více info
Fitness Bolevec	Vondruškova 1, Plzeň	Ukázat na mapě	Více info
Squashcentrum Plzeň	Kollárova 19, Plzeň	Ukázat na mapě	Více info

Obrázek 3.6: Ukázka použití HejbejteSe API

Výlety na kudyznudy.cz

Pro vás, co máte rádi honosné zámky

Hledáte hrad či zámek, kde se určitě nebudete nudit? Pak se vydejte do Litomyšle! Zdejší zámecký areál není zajímavý jen tím, že byl zapsán mezi památky UNESCO, ale nabízí mnohem víc než jen pouhou prohlídku interiérů, za vidění stojí i leccos ve městě a v jeho okolí.

Obrázek 3.7: Ukázka použití Kudy z nudy RSS

4 Widgety

Widget nebo též Gadget je jednoúčelová miniaplikace. Widgety mohou obsahovat informace například o předpovědi počasí, kurzech měn, ale mohou také poskytovat funkce například kalendáře či mapy. Webové widgety jsou nejčastěji vytvořeny v JavaScriptu, ale lze najít i widgety tvořené obrázkem, flashem či statickým html.

4.1 Formáty Widgetů

Zásadním problémem prvních widgetů na internetu byla integrace do webových stránek. Nejčastější metody vkládání byly přes HTML prvky *iframe*, *embed* případně *img*. Dodnes se tyto metody hojně používají na mnoha webových stránkách. Nevýhodou těchto řešení je jejich nekonzistentnost. Každý widget se vkládá jiným způsobem a vyžaduje jiné programově nezpracovatelné nastavení. Jelikož měl každý widget jiný formát, nebyla možná ani jejich jednoduchá kategorizace.

Z výše uvedených důvodů vzniklo několik formátů pro přenos widgetů. V podstatě jsou zde dva typy formátů. První je tvořený pouze jedním XML souborem. Výhodou je velice jednoduché zpracování. Veškerý obsah je v jednom souboru spolu s metadaty. Hlavní nevýhodou je nemožnost přenášet některé součásti widgetu jako například obrázky. Druhý typ je tvořen archivem v kterém jsou oddělené soubory s daty a metadaty. Widgety tohoto typu nejsou většinou řazeny mezi *webové widgety*, přesto je některé webové aplikace využívají jako běžné webové widgety. Výhodou tohoto typu je možnost přenášet spolu s widgetem i obrázky, nevýhodou je pak složitější zpracování obsahu a vytvoření widgetu.

V následující části budou popsány formáty, které lze použít v některé z aplikací uvedených v kapitole 6.

4.1.1 Google Gadgets

Google Gadget je formát od společnosti Google. Widgety v tomto formátu lze používat v mnoha službách společnosti Google (GMail, iGoogle, Google

Calendar), ale také na mnoha další webových stránkách. Widgety v tomto formátu lze nalézt ve veřejném katalogu Google Directory.

Tento formát je postavený na XML. Celý widget je uzavřen do elementu *module*. V tomto elementu se nachází veškeré informace o widgetu. Tyto informace jsou uvedeny v následujících elementech:

- **ModulePrefs** - tento element se může nacházet pouze jednou v XML. Obsahuje základní metadata o widgetu. Například název, rozměry a jméno autora.
- **UserPref** - je nepovinný element. Může se v těle vyskytnou vícekrát. Každý výskyt popisuje jednu uživatelsky nastavitelnou proměnnou.
- **Content** - nejdůležitější prvek celého widgetu. Může obsahovat buď url adresu na webovou stránku, která se má zobrazit, nebo může obsahovat HTML a JavaScript kód widgetu.

Ukázka jednoduchého widgetu v tomto formátu je uvedena na příkladu 4.1.1.

```
<?xml version="1.0" encoding="utf-8" ?>
<Module>
  <ModulePrefs title="Ukázka formátu Google Gadget" />
  <UserPref name="uzivatel" display_name="Uživatelské jméno" required="true" />
  <UserPref name="mapa" display_name="Zobrazit mapu" datatype="bool" default_value="true"/>
  <Content type="html">
     <![CDATA[
       <p>Na tomto místě lze používat HTML kód, ale také JavaScript.</p>
       <p>Přístup k uživatelským proměnným je možný
  přes JavaScriptový objekt new gadgets.Prefs();</p>
     ]]>
  </Content>
</Module>
```

Příklad 4.1.1: Ukázka widget formátu Google Gadget.

Přes element *ModulePrefs* lze do widgetu přidat podporu pro další funkce. Podporovány jsou následující moduly [10]:

- MiniMessage
- Tab
- TabSet

- Flash
- PubSub
- Rpc
- Skins
- Views
- Window

Pro mashupy je nejdůležitější modul *PubSub - Publish Subscribe*, díky kterému lze komunikovat mezi widgety. Podobnou funkci by měl podle dokumentace umožňovat i modul *Rpc*, bohužel však tuto funkčnost neimplementuje žádný dostupný kontejner.

4.1.2 OpenSocial Gadgets

Formát OpenSocial Gadgets vychází z formátu *Google Gadgets* uvedeného v části 4.1.1. Formát je naprosto shodný a jediným rozdílem je nutnost přidat v elementu *ModulPrefs* požadavek na modul *OpenSocial* pomocí kódu:

```
<Require feature="opensocial-0.8"/>
```

Tento kód zpřístupní další JavaScriptové funkce, které poskytují podporu pro různé sociální funkce, které lze volat z JavaScriptu. Tento formát je proto zpětně kompatibilní formátem *Google Gadgets*.

Za tímto formátem stojí OpenSocial Foundation, v kterém je řada internetových gigantů. Inicializátorem vzniku tohoto združení byla společnost Google. Nyní jsou součástí sdružení také firmy jako Yahoo, MySpace a další.

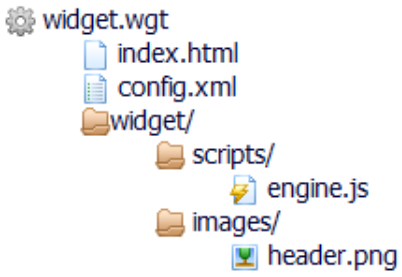
Kromě formátu *OpenSocial Gadgets* vytvořilo sdružení OpenSocial Foundation také systém autorizace uživatelů *OAuth*, který je mimo jiné integrován do jejich vlastního formátu widgetů.

Vzhledem k otevřenosti a jednoduchosti tohoto formátu, se jedná o velice oblíbený a používaný formát.

4.1.3 W3C Widget

Nejnovější z uvedených formátů je *W3C Widget*. Jeho poslední verze byla vydána až v prosinci roku 2011 [28]. Z tohoto důvodu tento formát implementuje velice málo aplikací.

Zásadní rozdíl oproti předchozím formátům, je způsob distribuce. W3C Widget je distribuován jako ZIP archiv s příponou WGT. Název souboru může být například *widget.wgt*. Struktura tohoto archivu je vyobrazena na obrázku 4.1.



Obrázek 4.1: Struktura W3C Widgetu [29].

Obdobně jako v předchozích formátech, i tento formát ukládá metadata do XML souboru. Všechna metadata se ukládají do souboru s názvem *config.xml*. Tento soubor musí být v rootu celého archivu a může vypadat například takto [28]:

```
<widget xmlns = "http://www.w3.org/ns/widgets"
  id = "http://example.org/exampleWidget"
  version = "1.1"
  width = "190"
  height = "150"
  viewmodes = "floating">
    <name short="Ukázka">Ukázkový widget v1.1</name>
    <description>Jednoduchý ukázkový widget.</description>
    <author>Josef Vrba</author>
</widget>
```

Součástí tohoto souboru mohou být další elementy. Například element *preference*, ve kterém jsou definovány globální proměnné widgetu.

Dalším důležitým souborem v archivu je startovací soubor, který má obvykle název *index.html*. Jedná se o klasický HTML (XHTML) soubor, který je zobrazen po načtení widgetu. Ostatní soubory jsou již nepovinné a lze je téměř libovolně zanořovat do různých složek.

Hlavní výhodou tohoto formátu je možnost přenášet v rámci widgetu i další soubory. Není tedy nutné odkazovat se absolutní cestou na veřejně dostupný webserver, ale postačí nahrát soubory do archivu a odkazovat se na ně relativně.

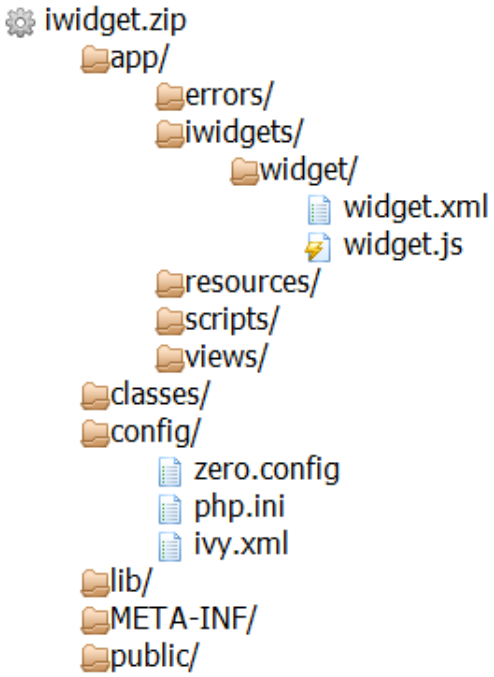
4.1.4 iWidget

Formát iWidget byl vyvinut firmou IBM a je hlavním formátem widgetů ve všech aplikacích této společnosti. Svojí strukturou je blízký W3C Widgetu, který je uveden v části 4.1.3. Opět se jedná o ZIP archiv, který v sobě obsahuje veškerá data widgetu. Struktura tohoto archivu je však poněkud obsáhlejší. Archiv může kromě samotného widgetu obsahovat také kód zpracovatelný na serverové straně. Struktura archivu iWigetu ve verzi 2.1. [14] je vyobrazena na obrázku 4.2.

V celé struktuře je nejdůležitější složka *apps/iwidgets*. V této složce jsou umístěny všechny widgety. Je tedy možnost mít v jednom archivu více widgetů. Každý widget má vlastní složku, která obsahuje všechny podpůrné soubory (obrázky, javascript) a XML soubor v následujícím formátu [14]:

```
<iw:iwidget
   name="TestovaciWidget"
   xmlns:iw="http://www.ibm.com/xmlns/prod/iWidget"
   supportedModes="view"
   mode="view"
   iScope="TestovaciWidgetScope">
     <iw:event id="VlozenText"
               eventDescName="TextEvent"
               published="true"/>
     <iw:eventDescription id="TextEvent"
                          payloadType="text"
                          lang="en" />
     <iw:resource uri="TestovaciWidget.js"/>
     <iw:content mode="view">
        <![CDATA[
            <div>Tělo testovacího widgetu</div>
        ]]>
     </iw:content>
</iw:iwidget>
```

Nástroje na tvorbu iWidgetů vývojáře kompletně odstíní od tvorby výše



Obrázek 4.2: Struktura formátu iWidget.

uvedeného XML, takže není potřeba podrobně rozebírat jednotlivé elementy. Je zde však několik důležitých věcí.

Obsah elementu *iw:content* obsahuje klasický HTML kód. Všechny editory iWidgetů tento obsah nepovolují editovat, ale celý obsah prezentují jako fiktivní soubor *TestovaciWidget-view.html*. Tento fiktivní soubor lze již normálně editovat.

Za povšimnutí stojí také elementy *iw:event* a *iw:eventDescription*. Díky těmto elementům lze widgety propojovat. Pokud widget přijímá nebo odesílá nějaké zprávy jiným widgetům, je to vždy vyznačeno pomocí těchto elementů.

Všechny ostatní složky jsou většinou prázdné. Případně jejich obsah generují nástroje na tvorbu iWidgetů bez zásahu uživatele.

4.2 Mediace zpráv

V mashupech je nejdůležitější vlastností widgetu schopnost komunikace s ostatními widgety. Ač se může zdát, že se jedná o triviální vlastnost widgetů, často se lze setkat s různými problémy.

Většina problémů vychází z dřívějších nekompatibilních implementací zasílání zpráv a stále neimplementovaným nebo nekorektně implementovaným standardem *OpenAjax Hub*.

OpenAjax Hub [20] aktuálně ve verzi 2.0. poskytuje všem widgetům hub. V tomto hubu se mohou widgety přihlásit k odběru zpráv. Každá zpráva má svůj zástupný název, který určuje o jakou zprávu se jedná. Widgety, které jsou registrovány pro příjem zpráv s tímto názvem, zprávu dostanou. Odesílání zpráv do hubu není nikterak podmíněno, takže libovolný widget může odeslat zprávu s libovolným názvem. Jedná se vlastně o implementaci návrhového vzoru *Publisher-Subscriber*.

Veškerá komunikace widgetů je vždy založena na návrhovém vzoru *Publisher-Subscriber*. U jednotlivých implementací se často liší jen názvy metod a pořadí atributů.

Kontejnery pro běh widgetů, které podporují zasílání zpráv a podporují widgety ve formátu *OpenSocial Gadgets*, neumožňují uživateli vybírat, které widgety mezi sebou budou komunikovat. Je však otázkou času, kdy toto bude umožněno, jelikož standard *OpenAjax Hub 2.0* podporuje dynamické propojování widgetů uživatelem (jinak též *wiring*). Pro formát *iWidget* je již dynamcické propojování widgetů dostupné v aplikaci IBM Mashup Center.

4.3 Další využití widgetů

V této práci jsou widgety využívány výhradně pro tvorbu mashupů. Jejich použití je však širší. Některé widgety lze používat také samostatně. Jako příklad lze uvést widget zobrazující aktuální jídelní lístek, u kterého nelze očekávat interakce s ostatními widgety.

Některé klasické aplikace (např. IBM Lotus Notes) umožňují vkládat a zobrazovat webové widgety v běžných formátech. Lze tak jednoduše rozšířit grafické rozhraní aplikace o další funkce. Často lze jednotlivé widgety vkládat

také do webových aplikací. Jako příklad lze uvést webový vyhledávač *Google* a jeho vyhledávací rozhraní *iGoogle*, které umožňuje vkládat widgety (náhled na obrázku 4.3).

Obrázek 4.3: Ukázka webové stránky s widgety.

Od webových widgetů již není daleko například k widgetům, které lze vkládat přímo do operačního systému Windows na plochu. Je zde nutný drobný zásah do zdrojového kódu widgetu, ale jedná se v principu pouze o převod stávajícího widgetu do jiného formátu.

5 Frameworky a nástroje pro tvorbu mashupů

V této kapitole budou popsány vybrané frameworky a nástroje, které usnadňují tvorbu mashupů. Největší pozornost je věnována nástrojům na podporu tvorby enterprise mashupů.

5.1 IBM řešení

Řešení od společnosti IBM je komerční placené řešení na tvorbu enterprise mashupů, které se skládá z několik aplikací.

- IBM Websphere sMash nebo Rational Application Developer
- IBM Mashup Center
  - Lotus Mashups
  - InfoSphere MashupHub

Stěžejní částí tohoto řešení jsou widgety, konkrétně toto řešení spoléhá na widgety ve formátu *iWidget*. Částečně jsou podporovány i widgety ve formátu *OpenSocial Gadgets*. Ty bohužel nepodporují uživatelské propojování.

Pro každý zdroj dat je vhodné mít jeden widget. Následně je možné povolit vybranou komunikaci mezi widgety. Uživatelé mohou na stránce mashupu přidávat a odebírat jednotlivé widgety a dále si stránku přizpůsobovat.

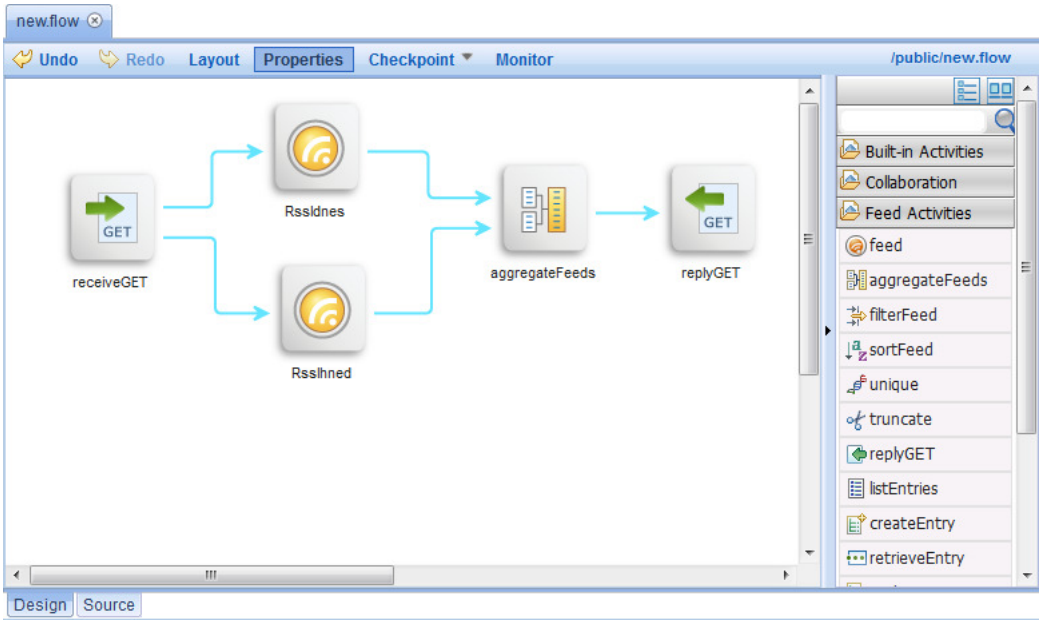
V následujícím textu budou popsány jednotlivé části tohoto řešení.

5.1.1 IBM Websphere sMash

Celý proces vytváření mashupu začíná vytvoření jednotlivých widgetů. K vytváření nových widgetů je určen právě IBM Websphere sMash. Spojení *nový widget* je důležité. Protože některé úpravy již existujících widgetů lze provést přímo v aplikaci IBM Mashup Center.

V rámci této práce jsem měl k dispozici pouze funkčně omezenou verzi tohoto produktu, která je poskytována v rámci *Project Zero*. Z tohoto důvodu se skutečnosti uvedené v následujícím textu vztahují výhradně na tuto omezenou verzi.

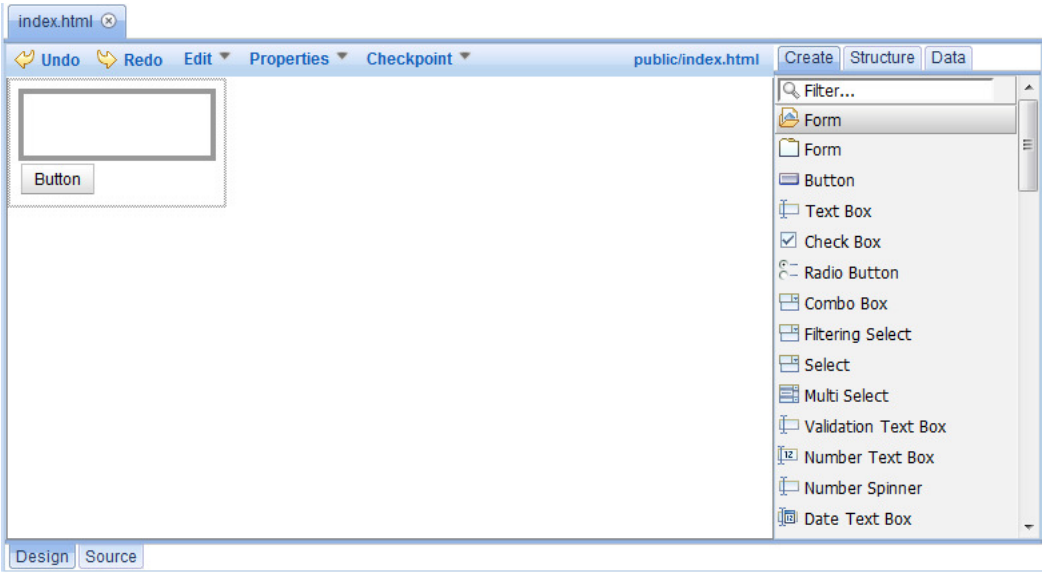
Vývoj probíhá ve webové aplikaci s názvem *App Builder*. Tato aplikace obsahuje dvě zajímavé částí týkající se mashupů. První část je tvorba *flow* skriptů zobrazená na obrázku 5.1. Druhá část je tvorba widgetů pomocí WYSIWYG HTML editoru zobrazeného na obrázku 5.2.



Obrázek 5.1: Ukázka flow editoru.

Tvorba *flow* skriptů je ve skutečnosti vytvoření datového mashupu. Vstupem je několik zdrojů dat a výstupem jsou data. Součástí flow skriptů není prezenční vrstva. Celá filozofie je shodná s nástrojem Yahoo! Pipes uvedeným v části 5.3.1.

Tvorbu widgetů usnadňuje WYSIWYG HTML editor, který pomáhá s tvorbou HTML a umožňuje i nastavení vlastností CSS přes průvodce. Částečně pomáhá i při tvorbě JavaScriptu, kdy například generuje hlavičku funkce *onClick*. Samotné tělo funkce již musí logicky napsat vývojář. Samozřejmostí je, že vytvořený widget lze exportovat ve formátu iWidget. Takto vytvořený widget lze bez problémů využít v aplikaci IBM Mashup Center.



Obrázek 5.2: Ukázka HTML WYSIWYG editoru.

Celkově bych hodnotil tento editor jako nepovedený. V některých průvodcích chybí zvýrazňování syntaxe a umísťování prvků HTML je velice zmatené. Pro vývojáře je jednodušší napsat HTML kód přímo a běžný uživatel skončí u toho, že nedokáže napsat kód v JavaScriptu.

5.1.2 Rational Application Developer

Druhou možností na vývoj iWidgetů je *Ration Application Developer*. Jedná se o standardní vývojové prostředí postavené na Eclipse IDE, které odstraňuje nevýhody webového *AppBuilderu* s nedostatečným zvýrazňováním syntaxe. Aplikace je určen přímo pro vývojáře a není zde předpoklad, že by ji ovládal běžný uživatel.

V rámci této práce jsem tuto aplikaci neměl k dispozici. Vzhledem k dostupným informacím, neočekávám, že by tato aplikace nějakým zásadním způsobem měnila způsob tvorby iWidgetů.

5.1.3 IBM Mashup Center

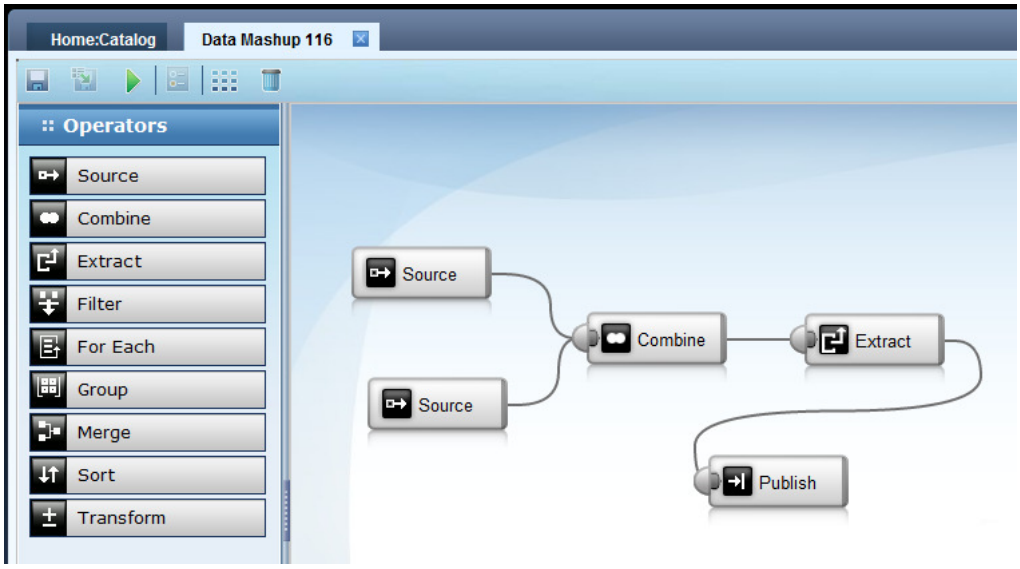
Předchozí aplikace byly určeny převážně pro vývojáře. IBM Mashup Center zapojuje do tvorby mashupů i běžné uživatele. Produkt *IBM Mashup Center* je složen ze dvou aplikací *Lotus Mashups* a *InfoSphere MashupHub*. Pro běh je nutný aplikační server *WebSphere Application Server*.

InfoSphere MashupHub

Tato aplikace má za úkol správu widgetů a datových zdrojů. Widgety zde lze přidávat, odebírat, hodnotit a komentovat. Je zde též možnost editace názvu a popisu. Nové widgety lze přidávat několika způsoby.

Prvním způsobem je možnost uploadu widgetu. Widget vytvořený například v programu *Websphere sMash* stačí nahrát pomocí webového prohlížeče, vyplnit popisek a widget se zobrazí v katalogu. Podporovány jsou formáty *iWdiget* a *OpenSocial Gadgets*.

Druhý způsob je vytvoření widgetu z již existujícího widgetu. Lze to pochopit jako způsob vytvoření stejného widgetu, ale například s jiným rozměrem, popisem, ikonou a nastavením.



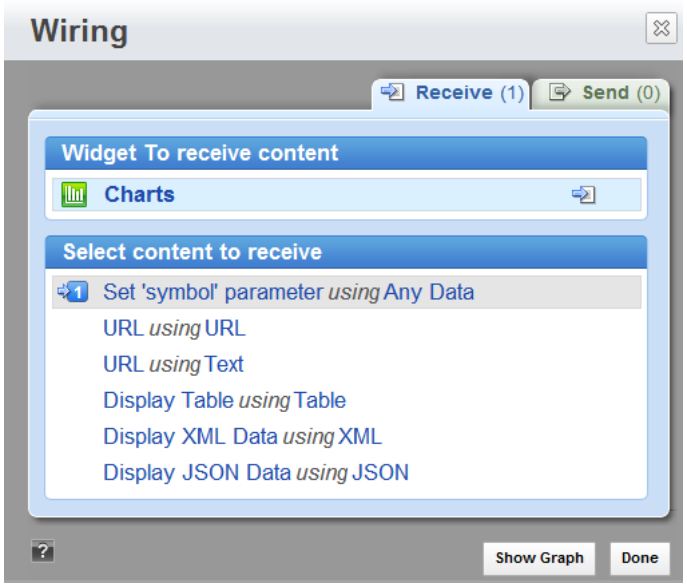
Obrázek 5.3: Ukázka editoru na tvorbu datových mashupů.

Kromě vytváření widgetů umožňuje *InfoSphere MashupHub* také přidá-

vat nové zdroje dat. Do katalogu lze vložit již existující zdroj dat bez jakékoliv modifikace nebo lze vytvořit datový mashup. Datový mashup vytvoření sloučením a filtrací několika zdrojů dat pak lze umístit do katalogu. Tvorba datového mashupu je obdobná jako v aplikaci *Websphere sMash* v části 5.1.1. Náhled webové stránky s nástrojem na tvorbu datového mashupu je na obrázku 5.3

Lotus Mashups

Smyslem této aplikace je zobrazení dat uživateli s možností personifikace. Uživatel si může vytvořit stránky a na každou stránku si libovolně umístit widgety podle svého zájmu. Widgety lze vybírat z katalogu, který poskytuje *InfoSphere MashupHub*. Widgety, které byly vloženy do katalogu ve formátu *iWidget* může propojovat uživatel (ukázka na obrázku 5.4), ostatní widgety musí být propojeny již dříve od vývojáře. Vytvořené stránky lze sdílet s ostatními uživateli.



Obrázek 5.4: Dialogové okno propojování widgetů.

5.2 OpenSource řešení

Open source řešení je složené s několika aplikací.

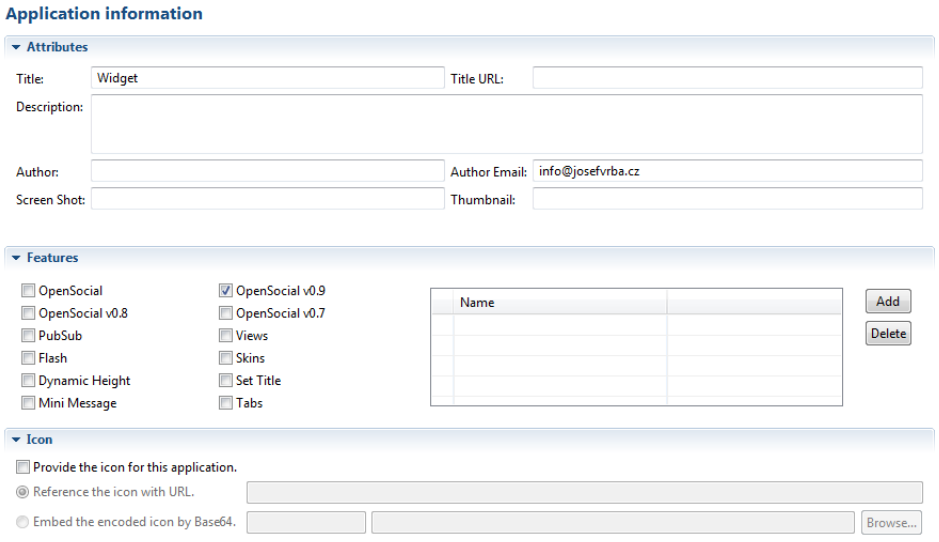
- OpenSocial Developer Environment
- Apache Shindig nebo Apache Wookie
- Apache Rave nebo WSO2 Gadget Server

Opět jsou stěžejním prvkem tohoto řešení widgety. Vývoj widgetů pro toto řešení je možný například v *OpenSocial Developer Environment*. Všechny výše uvedené aplikace podporují widgety ve formátu *OpenSocial Gadgets*.

5.2.1 OpenSocial Developer Enviroment

OpenSocial Developer Enviroment je vývojové prostředí pro tvorbu widgetů ve formátu *OpenSocial Gadgets* vytvořené jako Plug-In do *Eclipse IDE*. Podobnou roli jako OpenSocial Developer Enviroment v OpenSource řešení má Rational Application Developer v IBM řešní.

Tvorba widgetů touto aplikací bude vždy v rukou vývojářů. Není zde předpoklad, že by se běžní uživatelé pokoušeli vyvíjet widgety v této aplikaci. Tomuto faktu je upraveno i prostředí. K dispozici je jednoduchý grafický průvodce (viz. obrázek 5.5) pro nastavení metadat.



Obrázek 5.5: Ukázka průvodce v aplikaci OpenSocial Developer Enviroment.

Pro zápis kódu je k dispozici pouze editor se zvýrazňováním syntaxe. Tvorba kódu je tedy plně v rukou vývojáře. Z toho vyplývá, že pokud vývojář

pochopí způsob zápisu metadat do XML, stane se pro něj tento nástroj téměř zbytečným.

5.2.2 Apache Shindig a Apache Wookie

Vytvořené widgety ve formátu *OpenSocial Gadgets* je nutné spouštět na aplikačním serveru. Podporu pro tento formát poskytují dva servery:

- Apache Shindig
- Apache Wookie

Apache Shindig je referenční implementace pro tento formát. Přesto však existuje množství funkcionality, které prozatím není implementováno. Proto někteří tvůrci kontejnerů tyto funkce implementují sami na úrovni kontejneru. To bohužel může vést k nekompatibilitě widgetů mezi jednotlivými kontejnery.

Apache Wookie je aplikační server primárně určený pro widgety ve formátu *W3C Widget*. Podporuje však i widgety ve formátu *OpenSocial Gadgets*. Vzhledem k tomu, že se jedná o poměrně nový projekt, jsou použitelné kontejnery postavené na tomto serveru spíše výjimkou.

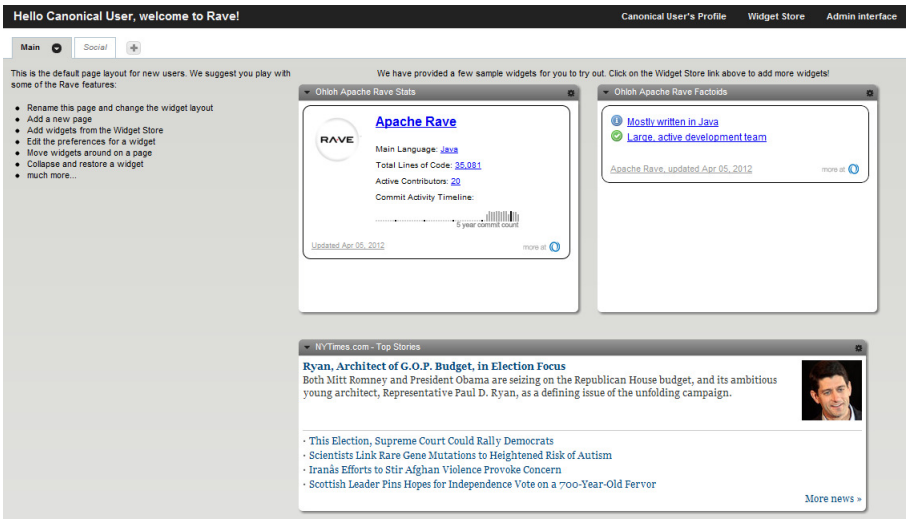
Pro formát *OpenSocial Gadgets* existují již hotové aplikace (kontejnery), které využívají výše uvedené servery a poskytují grafické rozhraní pro správu widgetů. V následujícím textu budou uvedeny dva OpenSource kontejnery.

5.2.3 Apache Rave

Tento kontejner využívá pro svůj chod *Apache Shindig*. Celé prostředí kontejneru je velice intuitivní. Jsou zde dvě skupiny uživatelů.

- Uživatelé
- Administrátoři

Každý uživatel má vlastní sadu pracovních stránek (viz. obrázek 5.6) a může si přidávat libovolné schválené widgety z *widget store* (viz. obrázek 5.7).



Obrázek 5.6: Apache Rave - Pracovní stránky uživatele.

Widget store obsahuje kategorizovaný seznam schválených widgetů. Každý widget mohou uživatelé hodnotit a je u něj také informace o používanosti. Uživatelé mají také možnost přidávat nové vlastní widgety. Tyto widgety musí před použitím nejprve schválit uživatel ze skupiny *administrátoři*.

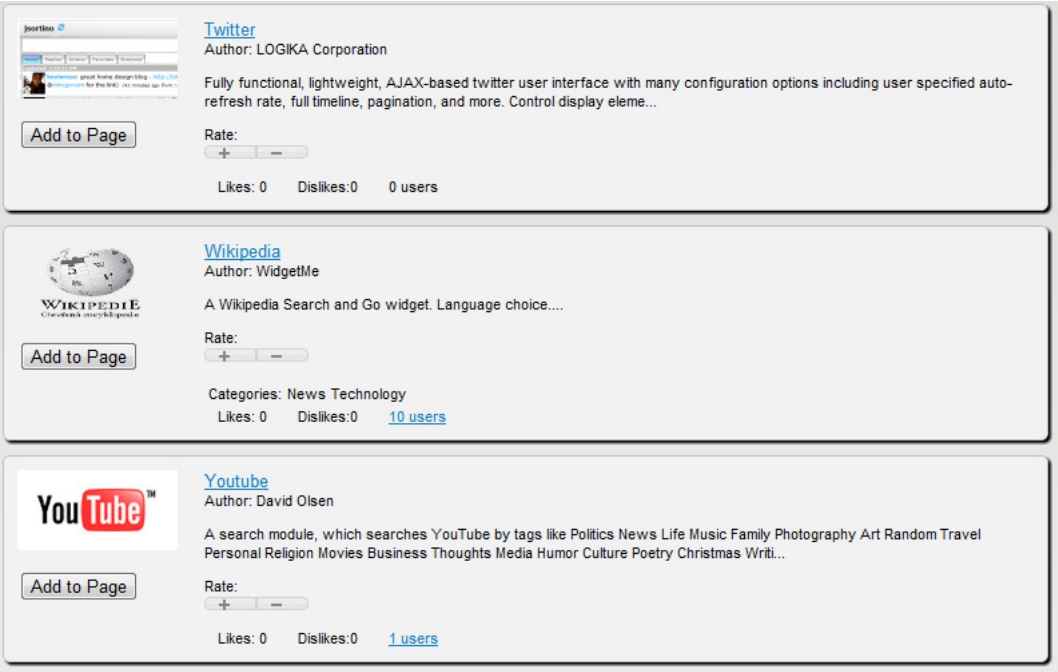
Součástí kontejneru Apache Rave je také administrátorské rozhraní, kde mohou uživatelé ze skupiny *Administrátoři* schvalovat widgety, přidávat nové uživatele, spravovat kategorie a další.

5.2.4 WSO2 Gadget Server

Druhý představený kontejner pro svůj chod taktéž využívá *Apache Shindig* a částečně i *Apache Wookie*. Funkčností je tento kontejner téměř nerozeznatelný od kontejneru *Apache Rave*.

Každý uživatel má opět vlastní sadu pracovních stránek. Podle aktuálního nastavení může buď přímo přidávat nové widgety, nebo musí každý widget projít schválením od administrátora.

Oproti kontejneru *Apache Rave* jsou zde větší možnosti nastavení kontejneru. Například právě možnost vypnout nutnost schvalování nových widgetů, nebo možnost zakázat registraci novým uživatelům. Zajímavostí je možnost přihlašování přes OpenID.



Obrázek 5.7: Apache Rave - Widget store.

Za zmínku stojí také produkt *WSO2 Mashup Server*, který poskytuje prostředí pro zobrazování widgetů, ale chybí zde katelog widgetů. Určen je především pro převod webových služeb postavených na WSDL na jednodušší JavaScriptové služby.

5.3 Jednoúčelové nástroje

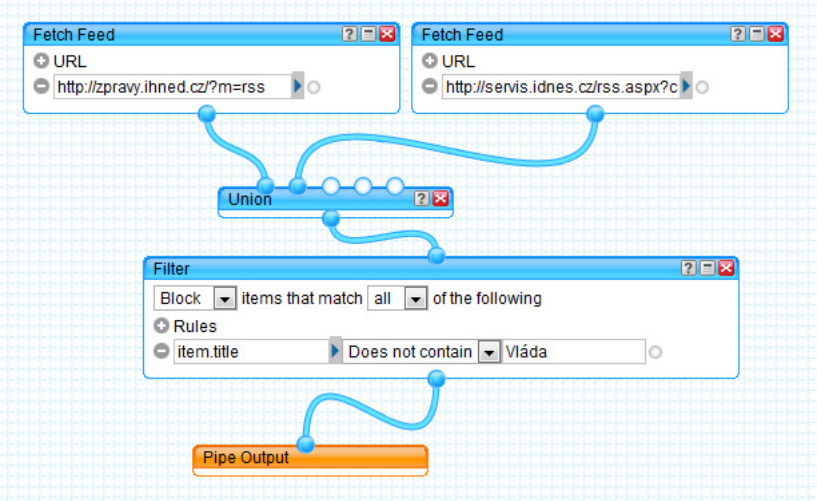
V této části budou uvedeny vybrané nástroje, které souvisí s tvorbou mashupů. Tyto nástroje jsou nazvány jako jednoúčelové, jelikož s jejich pomocí nelze běžným způsobem vytvořit mashup podle návrhu v kapitole 6.

5.3.1 Yahoo! Pipes

Yahoo! Pipes je velice jednoduchá a zajímavá služba od společnosti Yahoo!. Celá webová aplikace funguje na jednoduchém principu. Jako vstup je vložen

jeden nebo více zdrojů dat (obvykle RSS nebo ATOM). Tyto zdroje dat jsou propojeny s nějakým filtrem nebo sloučením a jsou poslány na výstup. Výsledkem je pak upravený dokument podle nastavených filtrů, který je dostupný přes vygenerovanou URL adresu.

Jako příklad běžného využití Yahoo! Pipes lze uvést filtrování RSS kanálů. Odebírám zprávy z RSS iDnes.cz a iHned.cz. Zajímají mě pouze informace týkající se vlády. Mohu tedy jako vstup použít oba RSS kanály a sloučit je do jednoho. Ve sloučeném kanálu následně vyfiltrovat pouze položky obsahující slovo vláda. Do vlastní RSS čtečky následně vložím adresu, kterou dostanu po uložení tohoto příkladu. Na obrázku 5.8 je vidět propojení nutné k vytvoření tohoto příkladu.



Obrázek 5.8: Příklad použití Yahoo! Pipes.

5.3.2 Google Mashup Editor

Tento již neexistující editor [24] byl otevřen pro veřejnost v roce 2007. Určen byl hlavně pro uživatele, kteří umí HTML a částečně ovládají JavaScript. Jednalo se spíše o editor, kde je nutné funkčnost vytvářet programováním v kódu. Po vytvoření aplikace v internetovém editoru byl mashup zdarma hostován na serverech společnosti Google. I přes značnou oblibu u uživatelů byl Google Mashup Editor uzavřen v srpnu roku 2009. Všechny vytvořené mashupy byly přesunuty na Google Apps Engine. Funkčnost již vytvořených mashupů zůstala zachována, ale samotné Google Apps Engine pracuje s odlišnou filozofií a není primárně určené pro vývoj mashupů.

5.3.3 Microsoft Popfly

Další již neexistující editor [21] byl jako betaverze otevřen pro veřejnout taktéž v roce 2007. Umožňoval tvorbu webových stránek a aplikací na platformě Silvelight. Bohužel byl provoz v roce 2009 ukončen.

5.3.4 Yahoo! Dapper

Tento zajímavý nástroj umožňoval z libovolné webové stránky vytvořit zdroj dat ve formátu XML a JSON [33]. Bohužel bez jakýchkoliv informací přestal Yahoo! Dapper koncem roku 2011 fungovat. S velkou pravděpodobností se Yahoo rozhodlo ukončit provoz tohoto projektu. Zajímavostí je, že tento projekt Yahoo! koupilo teprve v roce 2010.

5.3.5 Další nástroje

Existuje několik dalších nástrojů, které s tvorbou mashupů souvisí. V tomto odvětví mají však podobné nástroje velmi krátkou dobu života (viz. například Google Mashup Editor a Microsoft Popfly). V [25] bylo uvedeno několik dalších nástrojů, které zde nejsou zmíněny. Některé uvedené nástroje se specializují na jiné odvětví než je tvorba mashupů, jiné již přestaly existovat a poslední část nástrojů je placená bez možnosti získat zkušební verzi. V poslední skupině jsou například programy od společnosti Kapow Software, která zkušební verze poskytuje pouze potencionálním zákazníkům.

6 Vzorová aplikace

V této kapitole bude popsána vzorová aplikace implementovaná třemi různými způsoby. První implementace je bez použití nástrojů pro tvorbu mashupů. Další dvě využijí nástroje na tvorbu mashupů.

6.1 Návrh vzorové aplikace

Vzorová aplikace bude získávat data z těchto zdrojů:

- Dokempu.cz
- Maps.google.com
- Wunderground.com
- Hejbejtese.cz
- Panoramio.com
- KudyZnudy.cz
- Wikipedia.org

K předchozím zdrojům dat přibude ještě vstup od uživatele, který zadá název města, které ho zajímá. Celá aplikace bude fungovat následovně. Uživatel zadá název města a získá následující informace:

- Seznam kempů v okolí s možností zobrazit kemp na mapě.
- Mapu s vyznačeným městem.
- Aktuální počasí a předpověď na následující dny v okolí.
- Seznam sportovišť v okolí
- Fotografie ze zadaného města.
- Tipy na výlet v okolí.

- Informace o městě.

Funkčnost aplikace je shrnuta v jednoduchém modelu, který je na obrázku 6.1.

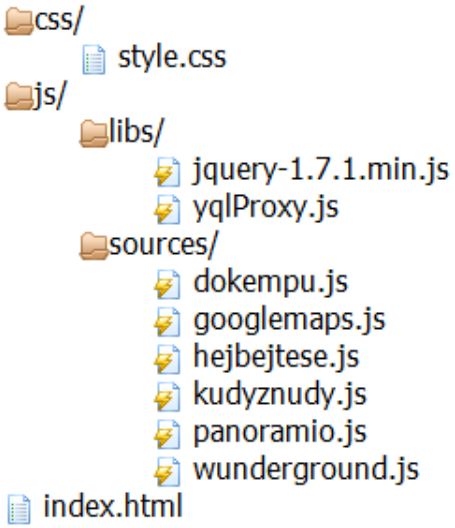
Obrázek 6.1: Model vzorové aplikace.

6.2 Ručně vytvořená stránka

První implementace vzorové aplikace byla vytvořena pouze za použití HTML, CSS a JavaScriptu. Nebylo nutné využívat žádné skripty na serverové straně. Veškeré získání a zpracování dat řeší JavaScriptové moduly. Každý modul je ve vlastním souboru. Stromová struktura souborů vzorové aplikace je na obrázku 6.2.

6.2.1 HTML a CSS část

Samotná webová stránka s HTML obsahem je velice jednoduchá. Základem je prázdná kostra HTML dokumentu s tagy *html*, *head* a *body*. Do kostry je



Obrázek 6.2: Strom souborů vlastní implementace.

nutné vložit odkazy na veškeré JavaScriptové soubory pomocí tagu *script*. Pro modul panoramio vypadá vložení následovně:

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

Dále je nutné do těla *body* pro každý modul vložit jeden element *div*. Obsah tohoto prvku je vždy velice podobný. Například element pro předpověď počasí vypadá takto:

```
<div class="widget" id="wunderground">
   <div class="header">Počasí z wunderground.com</div>
   <div id="forecast" class="content">
      K dispozici nejsou žádná data.
   </div>
</div>
```

U všech modulů jsou shodně pojmenované třídy jednotlivých prvků a tak lze vzhled jednotlivých modulů ostylovat z jednoho místa přes třídy *widget*, *header* a *content*. Atributy ID se používají pouze pro určení rozměrů a obsluhu událostí v JavaScriptu.

6.2.2 JavaScriptová část

Do webové stránky je vloženo také několik JavaScriptových souborů. V této části budou popsány některé z nich.

jQuery - jquery-1.7.1.min.js

Pro zpřehlednění a zjednodušení JavaScriptového kódu jsem využil nejpoužívanější [1] JavaScript Framework *jQuery* a to ve verzi 1.7.1. Díky tomuto frameworku lze využívat pokročilejší funkce bez znalosti implementačních rozdílů jednotlivých prohlížečů. Například použití *XmlHTTPRequest* je odlišně implementované ve všech verzích prohlížeče Internet Explorer oproti ostatním prohlížečům. Díky jQuery lze využít například funkci *get*, která vyřeší odlišnosti jednotlivých prohlížečů. O kvalitách tohoto frameworku svědčí i fakt, že jej společnost Microsoft integrovala do svého ASP.NET Frameworku.

YQL Proxy - yqlproxy.js

Jak již bylo řečeno v části 2.5, při stahování dat z jiných domén použitím JavaScriptu nastávají problémy. Jde o ochranu prohlížeče proti Cross-Site Scriptingu. Pokud jsou požadovaná data ve formátu JSON, je poměrně jednoduchým řešením využít JSONP (jQuery nabízí podporu pro tento styl stahování dat). Pokud jsou požadovaná data v jiném formátu, nelze je na daném serveru konvertovat na JSON a není možnost využít skript na straně serveru, je nejčastěji doporučováno využití YQL Proxy.

YQL neboli Yahoo! Query Language je dotazovací jazyk na bázi SQL. Byl vytvořen pro snadné získávání dat z webových stránek. Například dotaz

```
SELECT * FROM html WHERE url="http://www.zcu.cz"
```

vrací veškerý obsah webové stránky *http://www.zcu.cz*. Do klauzule *where* lze však doplnit i jiná kritéria. Například lze využít jazyka *xpath*. YQL podporuje i další typy dat. Mezi tyto typy patří XML, RSS, ATOM a další. Stačí pouze do klauzule *from* vložit příslušný typ.

Zprovoznění YQL Proxy je velice jednoduché. Yahoo poskytuje REST API, na které lze dotazy ve formátu YQL posílat. Toto API běží na adrese:

```
http://query.yahooapis.com/v1/public/yql
```

Pokud na tuto adresu pošleme výše uvedený dotaz na získání obsahu HTML stránky s parametrem *format=json*, API vrátí obsah webové stránky zkonvertovaný do formátu JSON. Následně již lze stáhnout tyto data pomocí JSONP. YQL Proxy je vlastně serverový skript - konvertor vybraného datového typu na formát JSON.

Pro toto řešení existuje i zásuvný modul do jQuery. Přesto jsem tuto proxy implementoval samostatně jako jednoduchou třídu *yqlproxy* s funkcemi *html(url, callback)* a *rss(url, callback)*.

Modul Google Maps - googlemaps.js

Tento modul zobrazuje mapu využitím Google Maps API. Zobrazení mapy je řešeno stejně jako v části 3.3.1. Jsou zde navíc doplněny dvě funkce. Zobrazování značky (*Marker*) a převod názvu města na GPS souřadnice (*Geocoder*).

Zobrazování značky zajišťuje jednoduchý kód:

```
var marker = new google.maps.Marker();
    marker.setPosition(pozice);
    marker.setMap(instanceMapy);
```

Výše uvedený kód vytvoří novou značku a nastaví pozici, na které se má značka zobrazovat. Pozice je instance objektu *google.maps.Latlng*. Jako poslední je nutné nastavit mapu, na které se má značka zobrazovat. V případě změny pozice značky stačí volat pouze funkci *setPosition()*.

Převod názvu města na GPS souřadnice není taktéž nikterak obtížný. Stačí správně zavolat příslušnou funkci. Vše řeší následující kód:

```
   var geocoder = new google.maps.Geocoder();
   var parametry = {
      "address": "Plzeň"
   };
   geocoder.geocode(parametry, function(results, status) {
      if (status == "OK") {
```

```
        var pozice = results[0].geometry.location;
    } else {
        alert("Zadané město nelze najít");
    }
});
```

Výše uvedený kód se pokusí získat souřadnice města Plzeň. V případě, že dotaz skončí úspěchem, bude v proměnné *pozice* instance objetku *google.maps.Latlng*, který obsahuje GPS souřadnice. V případě, že město nebude nalezeno, bude zobrazena chybová hláška.

Tento modul používají i některé ostatní moduly a to voláním funkcí *updateMarker* případně *updateMarkerByAddress*. Obě funkce mění pozici značky na mapě.

Moduly Panoramio, Hejbejtese a Wunderground - panoramio.js, hejbejtese.js a wunderground.js

Tyto moduly si jsou velice podobné. Mezi hlavní znaky těchto modulů patří komunikace s API využitím technologií REST a JSONP. Struktura všech modulů je velice podobná a proto zde bude uvedena pouze na příkladu modulu *Panoramio*.

Modul Panoramio přijímá od uživatele název města. Jelikož Panoramio.com neumožňuje hledání podle názvu města, je nutné získat GPS souřadnice. Realizace získání GPS souřadnic je realizována skrze Google Maps API obdobně jako v modulu *Google Maps*.

Ve chvíli, kdy jsou k dispozici GPS souřadnice, stačí odeslat požadavek na server služby Panoramio. Protože se jedná o server na jiné doméně, je nutné využít technologii JSONP. Díky frameworku jQuery, lze odeslání požadavku zařídit velice jednoduchým kódem:

```
$.ajax({
        url: urlAddress,
        data: params,
        dataType: "jsonp",
        jsonpCallback: "callback"
    });
```

Výše uvedený kód odešle technologií JSONP požadavek na adresu v proměnné *urlAddress* s parametry v proměnné *params*. Obsahem této proměnné jsou dříve získané GPS souřadnice a informace o požadované velikosti fotografií. Po přijetí odpovědi od serveru, bude zavolána funkce *callback*.

Obsah funkce callback je opět velice jednoduchý. Díky využití *jQuery* již není potřeba starat se o vyhodnocování odpovědi ve formátu JSON. Veškeré konverze a ochranné mechanizmy proti různým útokům řeší *jQuery*. Funkce *callback* může vypadat například takto:

```
callback(response) {
      $("#panoramio #photos").html("");
      $.each(response.photos, function(i, photo) {
         var img = "<img src='"+photo.photo_file_url+"' />";
         $("#panoramio #photos").append(img);
      });
 }
```

Tato funkce nejprve vyprázdní předchozí viditelný obsah. Následně v cyklu *each* pro každou fotografii v odpovědi vytvoří element *img* a vloží ho do obsahu.

Moduly KudyZnudy a DoKempu - kudyznudy.js a dokempu.js

Další skupinu podobných modulů tvoří moduly *KudyZnudy* a *DoKempu*. Hlavním znakem těchto modulů je odpověď vzdáleného serveru ve formátu XML. Z tohoto důvodu nelze využít technologie JSONP, ale je nutné využít dříve popsaný modul YqlProxy. Jelikož je opět struktura obou modulů velice podobná, bude zde detailněji popsán pouze modul *KudyZnudy*.

Vstupem od uživatele je hledané město. Server KudyZnudy poskytuje seznamy výletů pouze pro regiony. Je proto nutné převést název města na ID regionu, ve kterém se zadané město nachází. K získání názvu kraje je využito opět Google Maps API. Jediný rozdíl oproti řešení v modulu *Google Maps* je získávání názvu kraje místo GPS souřadnic. V kódu to znamená jediný rozdíl. V odpovědi stačí místo objektu *location* použít objekt *address* a jeho atribut *administrative area level 1*. Následně je název kraje převeden pomocí tabulky na ID odpovídající ID regionu na serveru KudyZnudy.cz.

Samotné získání dat je díky modulu *YqlProxy* jednoduché a stačí k tomu pouze následující řádky:

```
var url ="http://www.kudyznudy.cz/Trips.aspx?region="+regionId;
yqlproxy.rss(url,"callback");
```

Odpověď je obdobně jako u modulu *Panoramio* zpracována *callback* funkcí.

Modul Wikipedia - wikipedia.js

Základní struktura tohoto modulu je obdobná s modulem *Panoramio*. Zásadní rozdíl je zde ve zpracování odpovědi. Data získaná z Wikipedia API jsou bohužel ve formátu HTML s relativními odkazy a proto nejsou vhodná pro přímé zobrazení ve widgetu.

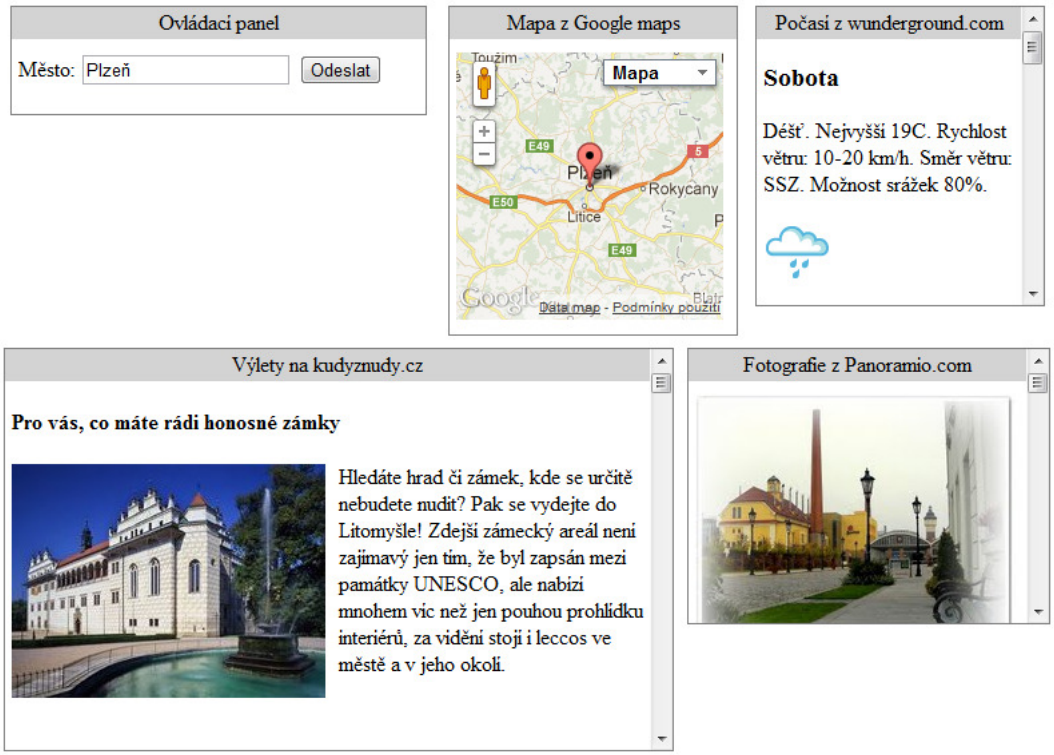
První úprava dat spočívá v odstranění tabulek. Toto odstranění spočívá v odstranění veškerého obsahu uvnitř tagu *table*. Důvodem je snaha získat krátký výstižný text o hledaném městě. Vyplněnou tabulku některé města nemají a často má tabulka různé položky. Zobrazovaná data by pak byla značně nekonzistentní.

Druhá úprava spočívá v odstranění veškerých html tagů. Odstranění je realizováno regulárním výrazem, který odstraňuje veškerý obsah uzavřený ve znacích < a >.

Poslední úpravou je zkrácení textu na 300 znaků a doplnění pokračovacího odkazu na zdroj dat.

6.2.3 Shrnutí

Toto řešení bylo tvořeno bez využití jakýchkoliv nástrojů na usnadnění tvorby mashupů. Není zde možná žádná personifikace. Oproti následujícím řešením je nutné podotknout fakt, že jednotlivé moduly mohou mezi sebou komunikovat přímo. Z jedné strany je to znatelná výhoda a zjednodušení pro vývojáře. Z druhé strany nejsou data jednotlivých modulů nijak zabezpečená a modul vytvořený třetí stranou může velice jednoduše odcizit důvěrná data. Výsledná aplikace vytvořená tímto řešením je na obrázku 6.3.



Obrázek 6.3: Výsledná aplikace vytvořená bez Mashup Frameworků.

6.3 OpenSource řešení

Druhá implementace je již závislá na několika produktech třetích stran. Ve všech případech se jedná o OpenSource produkty.

6.3.1 Widgety

Toto řešení je složeno z widgetů ve formátu *OpenSocial Gadgets*. Každý widget odpovídá jednomu zdroji dat.

Obsah jednotlivých widgetů je velice podobný jednotlivým modulům z části 6.2.2. Jedná se zde pouze o obalení veškerého HTML, CSS a JavaScript kódu do jednoho XML souboru a provedení několika drobných úprav.

Metadata

Součástí XML s widgetem jsou i metadata. Tyto metadata vypadají například u widgetu *Panoramio* takto:

```
<ModulePrefs title="Panoramio Widget" height="205">
        <Require feature="pubsub" />
</ModulePrefs>
```

Hodnota atributu *title* obsahuje název widgetu, atribut *height* obsahuje výšku celého těla widgetu v pixelech. Atribut *width* je možný, ale bohužel některé kontejnery pro zobrazování widgetů neberou hodnotu v úvahu. Mnohem důležitější než popsané atributy je element *require* s atributem *feature* a hodnotou *pubsub*. Tímto elementem je vyžadována podpora modulu *pubsub* od hostujícího kontejneru.

Mediace zpráv

PubSub je zkrácený název pro funkce *Publish* a *Subscribe*, které slouží pro vzájemnou komunikaci widgetů. Tyto funkce jsou klíčové pro tvorbu mashupů. Aplikační kontejner hostující widgety obsahuje *hub*, z kterého mohou všechny widgety přijímat nebo odesílat data. Přímá komunikace widgetů je téměř nemožná[1], jelikož každý widget je umístěn do vlastního tagu *iframe* a má proto jiný jmenný prostor.

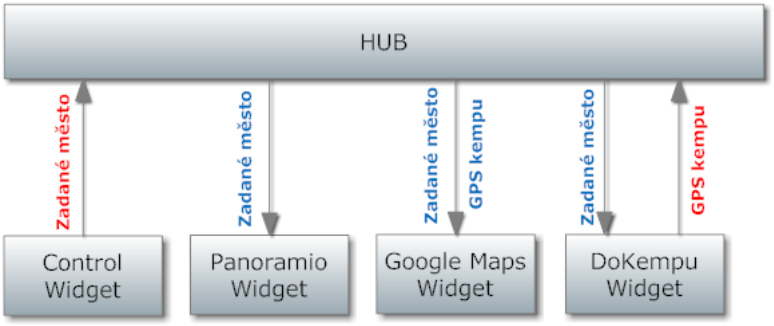
Komunikace probíhá tak, že příjemce se zaregistruje v *hubu* jako příjemce konkrétního typu zpráv pomocí metody *subscribe*. Pokud jakýkoliv widget odešle zprávu tohoto typu pomocí metody *publish*, všechny registrované widgety ji získají.

V implementované aplikaci je modul *pubsub* využíván všemy widgety. Na obrázku 6.4 je vyobrazena část widgetů a typ zpráv které přijímají a odesílají.

V podobě kódu je registrace pro příjem zpráv realizována kódem:

```
gadgets.pubsub.subscribe("nazev-zpravy", callback);
```

[1]Existuje poměrně složitá technika, díky které lze zajistit komunikaci dvou widgetů. Tato technika spočívá v získání rodičovského DOM stromu a vyhledávání dalších tagů *iframe* s tím, že jeden z nalezených musí být příjemce zprávy.



Obrázek 6.4: Způsob komunikace pomocí PubSub.

Funkce *callback* je zavolána po přijetí zprávy. Tato funkce může vypadat například takto:

```
function callback(topic, data, subscriberData) {
   var message = gadgets.util.escapeString(data);
}
```

V proměnné *message* bude po zavolání funkce *callback* obsah přijaté zprávy. Odeslání zprávy je realizováno pomocí kódu:

```
gadgets.pubsub.publish("nazev-zpravy", parametry);
```

Gadgets IO

V případě ručně psané aplikace, bylo nutné za určitých okolností využívat pro dotazování se serverů na jiné doméně YQL proxy. OpenSocial widgety umožňují využívat funkce z balíku *gadgets.io*, které zajišťují přesměrovávání požadavků na jiné domény. Lze tedy opustit řešení YQL proxy a využít funkce z *gadgets.io*.

Odeslání požadavku je realizováno funkcí *makeRequest*. Kód může vypadat například takto:

```
gadgets.io.makeRequest(url, callback);
```

Funkce *makeRequest* má naprosto shodný způsob volání jako YQL proxy. První parametr je url adresa včetně parametrů. Druhý parametr je funkce, která bude zavolána po přijetí požadavku.

6.3.2 WSO2 Gadget Server

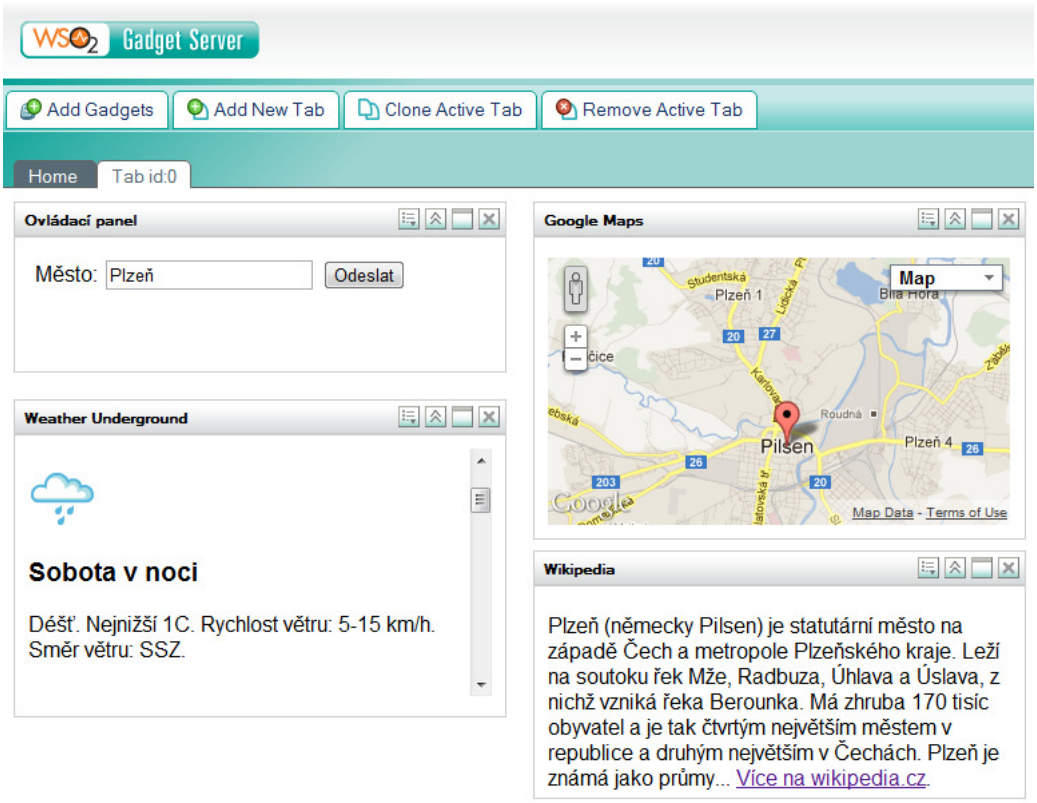
Zprovoznění mashupů v této aplikaci je velice snadné. Stačí provést několik kroků:

1. Stáhnout archiv s aplikací *WSO2 Gadget Server* ze stránek výrobce *www.wso2.com*.
2. Rozbalit archiv do složky s právem zápisu.
3. Spustit ve složce *bin* soubor s názvem *wso2server.bat* resp. *wso2server.sh*. Pro spuštění jsou potřeba administrátorská práva.
4. V internetovém prohlížeči přistoupit na adresu *localhost:8443*.
5. Přihlásit pomocí výchozích přihlašovacích údajům: jméno *admin* a heslo *admin*.
6. Nyní již lze přidávat widgety v menu *Add Gadgets*. Výsledek může vypadat například jako na obrázku 6.5.
7. Administrátorské menu je dostupné na adrese *localhost:8443/admin*

Pro vytvoření vzorové aplikace stačí pouze vložit všechny widgety. Výsledek může vypadat pokaždé mírně odlišně. Záleží na použitém layoutu stránky a poskládání widgetů do jednotlivých pozic. Funkčnost však bude vždy stejná.

6.3.3 Apache Rave

V době psaní této práce, bohužel nebylo možné běžným způsobem zprovoznit žádný mashup. Důvodem byla nefunkční podpora modulu *pubsub*. Po dotazu v oficiálním mailing listu mi bylo odpovězeno, že ve verzi 1.0 již bude modul *pubsub* funkční.



Obrázek 6.5: WSO2 Gadget Server - Pracovní stránky uživatele.

Widgety vzorové aplikace jsou vytvořeny podle specifikace, proto lze očekávat, že při použití v budoucí verzi 1.0 bude vzorový mashup plně funkční.

Postup pro zprovoznění bude obdobný jako u aplikace *WSO2 Gadget Server*:

1. Stáhnout archiv s aplikací *Apache Rave* ze stránek výrobce *rave.apache.org*.
2. Rozbalit archiv do složky s právem zápisu.
3. Spustit ve složce *bin* soubor s názvem *startup.bat* resp. *startup.sh*. Pro spuštění jsou potřeba administrátorská práva.
4. V internetovém prohlížeči přistoupit na adresu *localhost:8080*.
5. Přihlásit pomocí výchozích přihlašovacích údajům: jméno *canonical* a heslo *canonical*.

6. Nyní již lze přidávat widgety v menu *Add Gadgets*.
7. Widgety musí schválit administrátor v menu *Admin interface - Widgets*.

6.3.4 Shrnutí

Při tvorbě této implementace byly využity OpenSource nástroje pro usnadnění tvorby mashupů. S využitím vhodného OpenSocial Gadget kontejneru pro běh widgetů, může běžný uživatel personifikovat své stránky a vytvářet jednoduché mashupy. Bohužel zatím chybí podpora pro dynamické propojování widgetů. Propojení jednotlivých widgetů musí[2] být vždy vytvořeno předem vývojářem. Oproti předchozímu řešení jsou data jednotlivých widgetů lépe chráněna (každý widget běží ve vlastním iframu).

6.4 IBM řešení

Další implementované řešní je vytvořeno s využitím komerčních produktů od společnosti IBM.

6.4.1 Widgety

Toto řešení je opět postaveno na widgetech, tentokrát ve formátu *iWidget*. U předchozího řešení vždy jeden widget odpovídal jednomu zdroji dat. V tomto řešení tomu tak vždycky být nemusí. Na stránce může být více widgetů, než je zdrojů. Lze totiž vkládat skryté widgety, které mohou řešit na pozadí například konverzi předávaných zpráv mezi widgety. Podrobněji bude popsáno dále v části 6.4.3.

[2]Částečně lze řešit propojování widgetů přes uživatelské nastavení widgetu (metadata element *UserPrefs*). Nelze však dosáhnout stejného výsledku jako v IBM řešení. Do budoucna již existuje návrh, který toto propojování řeší.

Metadata

Obecně je formát *iWidget* složitější a komplexnější než formát *OpenSocial Gadgets*. Proto jsou i metadata tohoto formátu obsáhlejší.

Metadata u všech widgetů jsou kromě malých odlišností stejná. Jejich obsah bude popsán na příkladu widgetu *Hejbejtese*.

```
<iw:event id="hledaneMesto" eventDescName="pozice"
handled="true" onEvent="onReceiveEvent" />

<iw:event id="poziceSportoviste" eventDescName="pozice"
published="true" />

<iw:eventDescription id="pozice" payloadType="text" />
<iw:resource uri="Hejbejtese.js" />
<iw:resource uri="Hejbejtese.css" />
```

Jako první jsou definované události pomocí elementů *event*. První událost *hledaneMesto* je příchozí. Nastává v případě, když uživatel zadá název města. Toto město je pak parametrem této události. Druhá událost *poziceSportoviste* je odchozí. Widget v případě, že uživatel klikne na název sportoviště, vygeneruje událost s pozicí sportoviště. Z příkladu je patrné, že příchozí události mají nastaven atribut *handled* na *true* a odchozí události mají nastaven atribut *published* na *true*. K události je nutné definovat také popis pomocí elementu*eventDescription*. Popis událost určuje jaký typ dat se bude přenášet (v tomto případě pouze text) a dále umožňuje vložit pomocný popisek, který se zobrazí uživateli při projování widgetů.

Následně jsou definovány zdrojové soubory jako CSS a JavaScript. Tyto soubory budou při spuštění widgetu přidány do části *head* dokumentu, takže k nim bude možné přistupovat.

U ostatních widgetů v této implementaci se metadata liší pouze v názvech událostí a vkládaných souborech. Navíc některé widgetů mají pouze příchozí událost *hledaneMesto*.

Dojo

Zásadně rozdílná, oproti widgetům ve formátu *OpenSocial Gadgets*, je tvorba JavaScriptového kódu. V předchozím formátu bylo možné zvolit libovolnou JavaScriptovovou knihovnu či framework. Widgety ve formátu *iWidget* striktně vyžadují a využívají *Dojo*.

Dojo Toolkit [7] je open source JavaScriptový framework, který umožňuje rychlejší vývoj webových aplikací v JavaScriptu. Za vývojem stojí nezisková organizace *Dojo Foundation*, ve které má svůj podíl právě IBM, ale také například Oracle. Dojo je v podstatě jedna z mnoha alternativ ke knihovně jQuery zmíněné dříve.

Dojo klade velký důraz na objektově orientovaný přístup. Celý framework byl navržen tak, aby napomáhal programátorům tvořit objektově orientovaný JavaScriptový kód. Tato vlastnost však značně komplikuje tvorbu jednoduchých a jednoúčelových JavaScriptových funkcí. To je jeden z důvodů, proč se Dojo nestalo příliš oblíbené a má zanedbatelný podíl používání oproti jQuery, které vyniká svojí jednoduchostí.

Rozdíly v JavaScritovém kódu

Rozdíl v JavaScriptovém kódu mezi vlastní implementací a widgety ve formátu *OpenSocial Gadgets* byl téměř zanedbatelný. Oproti widgetům ve formátu *iWidget* jsou rozdíly mnohem častější.

Asi nejzásadnější rozdíl je způsob zpracování JavaScriptu. JavaScript se píše do třídy deklarované příkazem *dojo.declare*. Po spuštění je tato třída interně zpracována interpretem. To má za následek velice obtížné odhalování chyb a ladění. Chyby totiž neodhaluje prohlížeč, ale interpret a pouze ve velmi omezené míře. Často se pak může stát, že chyba způsobí zastavení zpracování JavaScriptového kódu bez jakékoliv chybové hlášky. Deklarace třídy s kostrou metod vypadá například takto:

```
dojo.declare("PanoramioScope",null,{
onLoad: function () { },
    sendEvent:function(data){ }
}
```

Druhý rozdíl je ve zpracování HTML. V předchozích implementacích se obsah HTML objevil v prohlížeči přesně tak, jak byl napsán, včetně vloženého JavaScriptu. V případě iWidgetů se některé HTML prvky kompilují (v implementaci není použito), ale co je podstatnější, nestahuje se žádný vložený JavaScript pomocí tagu *script*. Toto je poměrně podstatná věc, která značně komplikuje používání vložených JavaScriptových funkcí jako GoogleMaps. Vkládání většiny JavaScriptu lze řešit pomocí funkce *dojo.io.script*, ale například *GoogleMaps* musí být vloženy jinak.

Jediný funkční způsob, který byl popsán v [16] je ruční vkládání tagu *script* s odkazem na GoogleMaps a s callback funkcí do DOM stromu. Po načtení widgetu proběhne zprovoznění GoogleMaps následovně:

- Vložení funkce *callback* do těla stránky. Funkce pouze přesměrovává veškeré požadavky na funkci *instanceWidgetu.callback*.
- Vložení tagu *script* požadujícího stažení GoogleMaps s parametrem odkazujícím na funkci *callback*.
- Stažený skript GoogleMaps zavolá funkci *callback* v těle stránky. Ta přesměruje požadavek do těla widgetu.
- Widget po přijetí callback odpovědi může začít používat veškeré funkce GoogleMaps.

V JavaScriptovém kódu vypadá vkládání GoogleMaps následovně:

```
script = document.createElement("script");
script.type = "text/javascript";
script.text = "function gCallback(data){" +
   this.getCompleteiContextIdentifier() +
   ".iScope().googleCallback(data)}";
dojo.doc.getElementsByTagName("head")[0].appendChild(script);

script = document.createElement("script");
script.type = "text/javascript";
script.id="api";
script.src = "http://maps.google.com/api/js?callback=callback";
dojo.doc.getElementsByTagName("head")[0].appendChild(script);
```

Tento způsob funguje bezproblémově v případě, kdy je na stránce pouze jeden widget využívající GoogleMaps. V případě, kdy je widgetů více, končí chybou. Proto je nutné před zahájením inicializace zjistit, zda jsou již dostupné GoogleMaps a pokud ano, nevkládat je znovu.

Ve výše uvedeném kódu stojí ještě za zmínku funkce *getCompleteiContextIdentifier* publikovaná v [13]. Zjednodušeně lze říct, že tato funkce vrací název proměnné, pod kterou je uložena aktuální instance JavaScriptové třídy widgetu. Lze tak přistupovat k funkcím uvnitř widgetu.

Další rozdíly v JavaScriptu jsou spíše kosmetické. Například místo jQuery funkce *$.each* se používá funkce *dojo.forEach*.

Mediace zpráv

Zasílání zpráv mezi widgety je ve formátu *iWidget* řešeno opět pomocí návrhového vzoru *Publisher-Subscriber*. K odběru zpráv se widget přihlašuje pouze pomocí metadat. V metadatech je uveden název funkce, která bude po přijetí zprávy vyvolána. Odeslání zprávy lze realizovat pomocí funkce *fireEvent* jako na následujícím příkladu:

```
this.iContext.iEvents.fireEvent("TextEntered",null,data);
```

Podmínkou pro zasílání zpráv je uvedení odpovídajících informací v metadatech widgetu. Navíc při použití widgetů v aplikaci *IBM Mashup Center* nutné propojit komunikující widgety (více viz. 6.4.3).

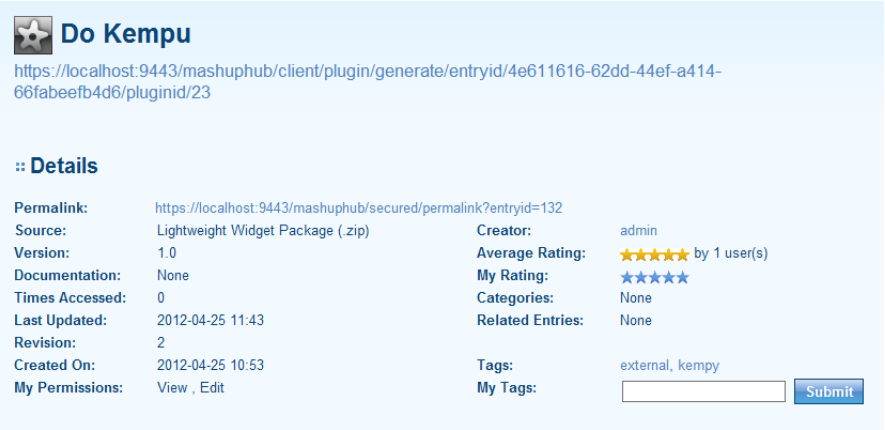
6.4.2 InfoSphere MashupHub

Funkce této aplikace je ve vzorové implementaci velice jednoduchá. Vytvořené widgety lze díky této aplikaci nahrát a vložit je do katalogu.

Celá operace vložení do katalogu spočívá ve dvou krocích. Nejprve je nutné widget nahrát na server. To lze udělat přes menu pomocí příkazů *Upload - Upload Widget or Gadget*.

Po nahrání je nutné widget zpřístupnit v aplikaci *Lotus Mashups*. To lze udělat pomocí příkazů v menu *Add to - Mashup Builder*.

Tato aplikace navíc shromažďuje veškeré dostupné informace o widgetu. Jako například informace o používání, komentáře, hodnocení, tagy a další. Detail widgetu je uveden na obrázku 6.6.



Obrázek 6.6: InfoSphere MashupHub - Detail widgetu.

6.4.3 Lotus Mashups

Nejdůležitější částí celého IBM řešení je právě aplikace *Lotus Mashups*. V aplikaci lze vytvářet různě strukturované stránky, tyto stránky navíc lze i sdílet s ostatními uživateli.

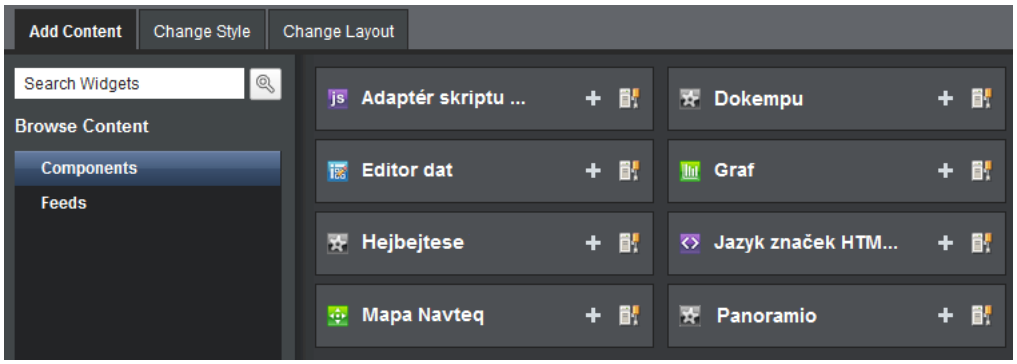
Správa obsahu stránky

Na vytvořenou stránku lze nyní přidat všechny widgety, které byly k dispozici v InfoSphere MashupHubu. Kromě již předpřipravených widgetů (viz. 6.4.3) jsou k dispozici i námi vložené widgety. Přídávání widgetů lze realizovat přes menu *Actions - Edit page - Customize*. Zobrazené menu (viz. obrázek 6.7) umožňuje kromě přidávání widgetů též změnu vzhledu (stylu) a rozložení stránky. Přidávání widgetů ze zobrazeného menu lze již velice jednoduše přetažením na požadované místo na stránce.

Předpřipravené widgety

Součástí aplikace je již několik předpřipravených widgetů. Řada widgetů slouží pouze jako ukázky funkčnosti aplikace, ale lze zde nalézt i několik užitečných widgetů.

Jako velice přínosný považuji widget s názvem *Uživatelský vstup*. Podstata činnosti tohoto widgetu je velice jednoduchá. V nastavení widgetu lze přidat



Obrázek 6.7: Lotus Mashups - Vkládání widgetů na stránku.

libovolné množství uživatelských vstupů. Tyto vstupy budou po uložení zobrazeny v těle widgetu spolu s tlačítkem *Odeslat*. Jednotlivé uživatelské vstupy lze následně propojit s jinými widgety. Tento widget může bez jakýchkoliv problémů nahradit mnou vyvinutý widget *Ovládací panel*.

Za další přínosný widget považuji widget *Mapa Navteq*. Jedná se o widget s mapu, jehož vstupem může být například název města. Zásadním problémem tohoto widgetu je nutnost zakoupit licenci k mapovým podkladům u společnosti Navteq. V případě platné licence může tento widget nahradit mnou vyvinutý widget *Google Maps*.

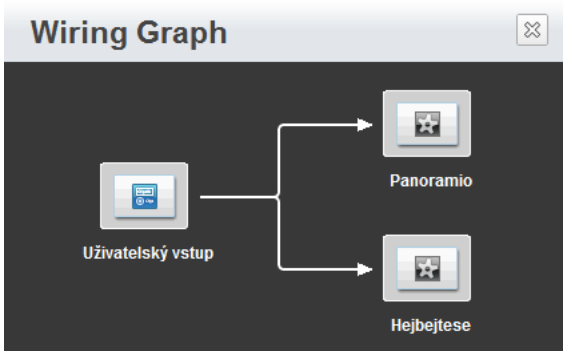
Za další přínosné widgety považuji též widgety *Graf*, *Čtečka kanálu* a další. Ve vzorové aplikaci je lze použít pouze velice obtížně.

Mediace zpráv

V této aplikaci je k dispozici velice ojedinělá, ale o to více mocná a užitečná funkce pro mediaci zpráv. Tato funkce pojmenovaná jako *wiring* umožňuje velice jednoduše uživatelům vytvářet datové proudy mezi widgety. V tomto případě si lze propojování představit jako dráty mezi widgety.

Na obrázku 6.8 je zobrazená podstata této funkce propojování. K tomuto oknu se lze dostat u kteréhokoliv widgetu ve formátu *iWidget* přes menu *Edit Wiring*. V horní části uživatel vybere widget, který má data odesílat. Po vybrání se rozbalí seznam dostupných událostí, které může widget propagovat. Po vybrání jedné z událostí se objeví seznam ostatních widgetů na stránce, které jsou schopné přijímat takovýto typ události. Po vybrání přijímajícího

Obrázek 6.8: Lotus Mashups - Propojování widgetů



Obrázek 6.9: Lotus Mashups - Propojování widgetů

widgetu stačí okno zavřít pomocí tlačítka *save* a propojení je vytvořené.

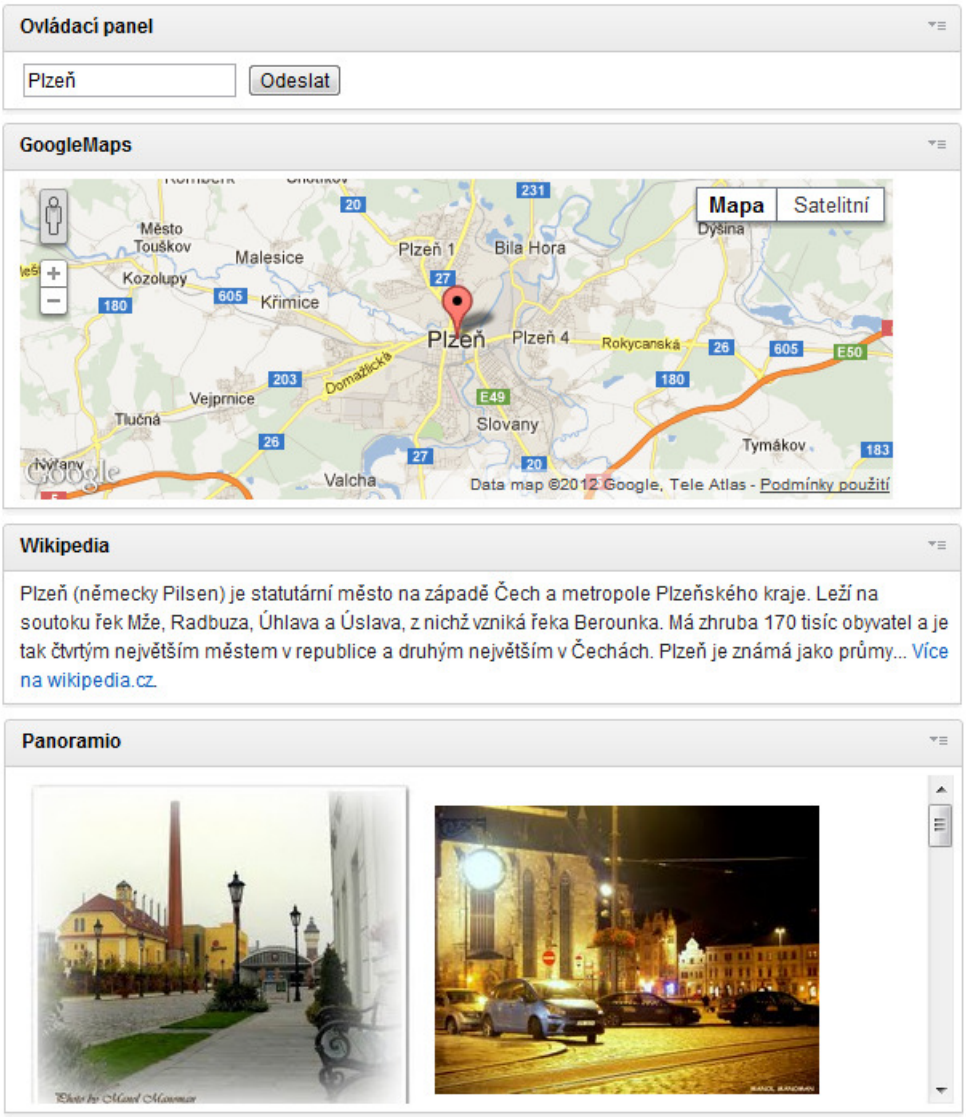
Po vytvoření všech datových toků, lze toky z/do widgetu zobrazit pomocí menu *View Wiring Graph*. Lze tak jednoduše zjistit, které widgety spolu komunikují. Ukázka propojovacího grafu je uvedena na obrázku 6.9

Bez ručního propojení nemohou widgety ve formátu *iWidget* mezi sebou komunikovat. Nutno je však také podotknout, že widgety musí být připraveny na možnost propojování a musí korektně prezentovat příchozí a odchozí data ve svých metadatech.

6.4.4 Shrnutí

Toto řešení využívá komerční nástroje pro usnadnění tvorby mashupů. Výhody tohoto řešení jsou ve velice rozsáhlé možnosti personifikace stránek a mashupů. Oproti předchozímu řešení zde může uživatel libovolně propojovat widgety. Propojení nemusí být kompletně vytvořeno programátorem. Data v jednotlivých widgetech jsou navíc chráněna právě nutností ručního propojení.

Část výsledného mashupu vytvořeného tímto řešením je uveden na obrázku 6.10.



Obrázek 6.10: Část výsledného mashupu vytvořeného v Lotus Mashups

7 Zhodnocení

V této práci byl vytvořen vzorový mashup, který byl implementován třemi způsoby:

- Ručně vytvořená stránka
- OpenSource řešení
- IBM řešení

V této kapitole budou zhodnoceny klady a zápory jednotlivých řešení.

7.1 Náročnost na tvorbu

Z pohledu náročnosti tvorby je pro jednoúčelové a nepersonifikovatelné mashupy nejméně náročné na vytvoření řešení *Ručně vytvořená stránka*. V případě požadavků na víceúčelovost a personifikaci (běžné u mashupů) se toto řešení stává nejnáročnějším z hodnocených řešení. Důvodem je nutnost navrhnout a implementovat velice rozsáhlou a netriviální funkcionalitu. Zbylé dvě řešení jsou na tom v tomto případě o poznání lépe a to díky widgetům.

U *OpenSource řešení* a *IBM řešení* se náročnost na tvorbu vztahuje prakticky pouze k vývoji widgetů. Běhové kontejnery jsou již k dispozici a není potřeba je vyvíjet.

Vývoj widgetů je znatelně jednodušší v případě *OpenSocial řešení*. Programátor může využít libovolný JavaScriptový framework a knihovny. V tomto řešení nejsou žádná omezení a vývoj probíhá jako u běžné webové aplikace. Pro vývoj widgetů nejsou nutné žádné specifické zkušenosti a znalosti.

Oproti tomu vývoj widgetu pro *IBM řešení* je znatelně obtížnější. Z velké části za to může vynucení JavaScriptového frameworku Dojo. Nemalý vliv na náročnost mají též bezpečnostní opatření, kvůli kterým nelze například běžným způsobem vkládat JavaScript z cizích serverů.

Pořadí v disciplíně *Náročnost na tvorbu*:

1. OpenSource řešení
2. IBM řešení
3. Ručně vytvořená stránka

7.2 Personifikace

Řešení *ručně vytvořená stránka* neumožňuje žádný způsob personifikace. *OpenSource řešení* umožňuje uživatelům tvořit různě strukturované stránky s různými widgety, ale neumožňuje upravovat propojení jednotlivých widgetů. Proto je z hlediska personifikace jednoznačně nejlepší *IBM řešení*, které umožňuje jak tvořit libovolně strukturované stránky s widgety, tak umožňuje uživatelům propojovat libovolně jednotlivé widgety.

Pořadí v disciplíně *Personifikace*:

1. IBM řešení
2. OpenSource řešení
3. Ručně vytvořená stránka

7.3 Použitelnost v podnicích

Z hlediska použitelnosti v podnicích vychází nejlépe *IBM řešení* a to hlavně z důvodů jednodušší integrace v rámci podniku. Pokud již podnik využívá některé produkty IBM například pro vývoj portletů pro portály, bude pro vývojáře velice snadné vyvíjet i widgety ve formátu *iWidget*. Dále díky možnosti libovolně propojovat widgety umožňuje vytvářet uživatelům doopravdy libovolné mashupy.

Jako použitelné řešení lze též hodnotit i *OpenSource řešení*. Vývoj lze provádět prakticky v libovolném vývojovém prostředí (lze tedy snadno vyvíjet ve stávajících aplikacích). Problém nastává s propojováním widgetů, které musí být řešeno již při tvorbě widgetu. Toto řešení se tím pádem hodí spíše jako prezenční nástroj, nikoliv nástroj na tvorbu mashupů.

Ručně vytvořená stránka je pro využití v podnicích nevhodná. Zásadní nevýhodou je vytvoření nestandardního nástroje, který po delší době půjde velice špatně upravovat a v případě odchodu vývojářů, kteří toto řešení vytvářeli, se stane celé toto řešení prakticky neupravitelné.

Pořadí v disciplíně *Použitelnost v podnicích*:

1. IBM řešení
2. OpenSource řešení
3. Ručně vytvořená stránka

7.4 Omezení

Z pohledu omezení, čímž je myšleno nemožností vytvořit nějaký druh mashupu, je na tom nejlépe *Ručně vytovřená stránka*. Je to vcelku logické, vývojář není ničím omezen a může vytvořit jakýkoliv mashup.

IBM řešení díky možnostem uživatelského propojování widgetů pokrývá širší množství mashupů a je tak na tom lépe než *OpenSource řešení*.

Pořadí v disciplíně *Omezení*:

1. Ručně vytvořená stránka
2. IBM řešení
3. OpenSource řešení

7.5 Shrnutí

Pokud shrneme předchozí hodnocení, lze říct, že nejvhodnější řešení pro tvorbu mashupů v podnicích je *IBM řešení*. Toto hodnocení se zakládá na funkčnosti a celkové podpoře tvorby mashupů uživateli. V hodnocení není zahrnuta finanční stránka věci, kdy pořizovací náklady jednotlivých řešení jsou značně rozdílné.

8 Závěr

Tato práce poskytuje čtenáři přehled o aktuálních možnostech tvorby mashupů. Úvod práce je věnován základním informacím o mashupech a příkladům úspěšných internetových mashupů. V práci jsou též uvedeny používané technologie získávání dat a formáty widgetů, které tvoří základ většiny enterprise mashupů.

Součástí práce je vzorový mashup, který byl implementován třemi způsoby pomocí různých nástrojů na tvorbu mashupů.

První řešení bylo tvořeno ručně bez využití podpůrných nástrojů na tvorbu mashupů. Toto řešení se ukázalo jako velice jednoduché, ale nevhodné pro použití v korporátním prostředí pro jeho špatnou personifikovatelnost. Hlavním prvkem dalších dvou řešení byly již widgety.

Druhé řešení využívalo volně dostupné nástroje. Výhodou tohoto řešení byla velmi jednoduchá tvorba widgetů, která spočívala pouze ve vložení HTML a JavaScriptového kódu do XML. Jako zásadní nevýhoda tohoto řešení se ukázala nedokončená implementace klíčových funkcí jako je komunikace mezi widgety.

Poslední řešení využívalo produkty od společnosti IBM. Toto řešení se ukázalo jako velmi vhodné pro použití v korporátním prostředí. Mezi jeho hlavní výhody patří velice rozsáhlé možnosti personifikace a tvorby mashupů.

Při pohledu na způsob vývoje mashupů, je tvorba mashupů značně jednodušší než vývoj běžné aplikace. Jedním z hlavních důvodů je velké zjednodušení správy dat. Mashup je primárně závislý na již existujících datech, která pouze čte. Proto většina mashupů nepotřebuje žádnou databázi ani pokročilé nástroje na editaci dat.

Podle mého názoru mohou být webové mashupy tvořené profesionálními nástroji velkým přínosem pro velké korporace. Použití v menších a středních firmách za současného stavu postrádá dle mého názoru finanční návratnost. Hlavním důvodem je nutnost vývoje a podpory uživatelů při tvorbě mashupů, která se vyplatí až od určitého množství uživatelů.

Literatura

[1] Builtwith.com: *JavaScript Usage Statistics* [online]. [cit. 2012-05-01]. Dostupné z: http://trends.builtwith.com/javascript

[2] Chow, S.-W.: *Programujeme Mashup aplikace pro Web 2.0 v PHP*. Brno: Computer Press, a.s., 2008, ISBN 978-80-251-2057-6.

[3] Chris, A.: *The Long Tail: Why the Future of Business Is Selling Less of More*. New York: Hyperion, 2006, ISBN 978-1-4013-0237-5.

[4] Clarkin, L.; Holmes, J.: *Enterprise Mashups*. [online]. [cit. 2012-03-02]. Dostupné z: http://msdn.microsoft.com/en-us/library/bb906060.aspx

[5] Crockford, D.: *RFC 4627 - The application/json Media Type for JavaScript Object Notation (JSON)*. IETF. [online]. [cit. 2011-10-25]. Dostupné z: http://tools.ietf.org/html/rfc4627

[6] CzechTourism: *RSS Kudy z nudy* [online]. [cit. 2012-01-14]. Dostupné z: http://www.kudyznudy.cz/rss.aspx

[7] The Dojo Foundation: *Dojo Toolkit* [online]. [cit. 2012-04-28]. Dostupné z: http://www.dojotoolkit.org/

[8] Fielding, R. T.: *Architectural styles and the design of network-based software architectures*. University of California, Irvine, 2000, ISBN 0-599-87118-0.

[9] FTOnline: *DoKempu.cz API* [online]. [cit. 2012-04-28]. Dostupné z: http://www.dokempu.cz/spoluprace

[10] Google: *Gadgets API Reference* [online]. [cit. 2012-04-28]. Dostupné z: http://code.google.com/apis/gadgets/docs

[11] Google: *Google Maps API* [online]. [cit. 2012-04-28].
Dostupné z: https://developers.google.com/maps

[12] Google: *Panoramio Widget API* [online]. [cit. 2011-12-05].
Dostupné z: http://www.panoramio.com/api

[13] Grammel, L.: *iWidgets: Referring to iContext from innerHTML* [online]. [cit. 2012-04-01].
Dostupné z: http://lgrammel.blogspot.com/2009/02/iwidgets-referring-to-icontext-from.html

[14] IBM: *iWidget Specification v2.1* [online]. [cit. 2011-10-05].
Dostupné z: http://public.dhe.ibm.com/software/dw/lotus/mashups/developer/iwidget-spec-v2.1.pdf

[15] Janovský, D.: *Vytěžování stránek* [online]. [cit. 2011-09-05].
Dostupné z: http://www.jakpsatweb.cz/vytezovani.html

[16] Koyuk, H.: *Loading Google Maps asynchronously via a Dojo Widget* [online]. [cit. 2012-04-28].
Dostupné z: http://heather.koyuk.net/refractions/?p=257

[17] MediaWiki: *MediaWiki API* [online]. [cit. 2011-11-03].
Dostupné z: http://cs.wikipedia.org/w/api.php

[18] Monson-Haefel, R.: *J2EE Web services*. London: Pearson Education, Inc., 2004, ISBN 0-321-14618-2.

[19] Nottingham, M.; Sayre, R.: *RFC 4287 The Atom Syndication Format* [online]. [cit. 2011-08-29].
Dostupné z: http://tools.ietf.org/html/rfc4287

[20] OpenAjax Alliance: *OpenAjax Hub 2.0 Specification* [online]. [cit. 2012-04-29].
Dostupné z: http://www.openajax.org/member/wiki/OpenAjax_Hub_2.0_Specification

[21] Polzer, J.: *Microsoft Popfly končí* [online]. [cit. 2011-12-20].
Dostupné z: http://www.maxiorel.cz/microsoft-popfly-konci

[22] ProgrammableWeb.com: *Mashup Dashboard* [online]. [cit. 2011-12-19].
Dostupné z: http://www.programmableweb.com/mashups/

[23] Susnova, E.: *Web 2.0* [online]. [cit. 2011-11-02].
Dostupné z: http://suewebik.net/webove/web20-def.html

[24] Tholomé, E.: *Farewell to Mashup Editor* [online]. [cit. 2011-10-11].
Dostupné z: http://googlemashupeditor.blogspot.com/2009/07/farewell-to-mashup-editor.html

[25] Volker, H.; Marco, F.: Market Overview of Enterprise Mashup Tools. In *Proceedings of the 6th International Conference on Service-Oriented Computing*, ICSOC '08, Berlin, Heidelberg: Springer-Verlag, 2008, ISBN 978-3-540-89647-0, s. 708–721, doi:10.1007/978-3-540-89652-4˙62 [online]. [cit. 2011-12-01].
Dostupné z: http://dx.doi.org/10.1007/978-3-540-89652-4_62

[26] W3C: *Web Services Description Language (WSDL) 1.1* [online]. [cit. 2012-01-05].
Dostupné z: http://www.w3.org/TR/wsdl

[27] W3C: *SOAP Version 1.2* [online]. [cit. 2012-02-24].
Dostupné z: http://www.w3.org/TR/soap12-part1/

[28] W3C: *Widget Interface* [online]. [cit. 2012-02-25].
Dostupné z: http://www.w3.org/TR/widgets-apis/

[29] W3C: *Widget Packaging and XML Configuration* [online]. [cit. 2012-02-25].
Dostupné z: http://www.w3.org/TR/widgets/

[30] Weather Underground: *Weather API* [online]. [cit. 2012-03-05].
Dostupné z: http://www.wunderground.com/weather/api/

[31] Wine, D.: *XML-RPC Specification* [online]. [cit. 2011-11-25].
Dostupné z: http://www.xmlrpc.com/spec/

[32] Winner, D.: *RSS 2.0 Specification* [online]. [cit. 2011-11-20].
Dostupné z: http://cyber.law.harvard.edu/rss/rss.html

[33] Yahoo: *Yahoo! Dapper* [online]. [cit. 2012-05-01].
Dostupné z: http://open.dapper.net/