VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

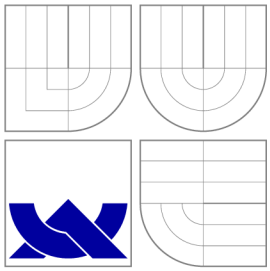
VÝVOJ WEBOVÝCH KOMPONENT AJAX V PROSTŘEDÍ EXTJS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

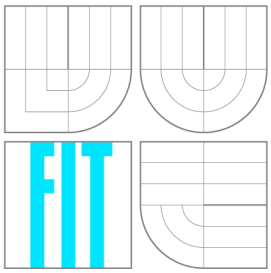
AUTOR PRÁCE
AUTHOR

Bc. JAROSLAV DYTRYCH

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VÝVOJ WEBOVÝCH KOMPONENT AJAX V PROSTŘEDÍ EXTJS

DEVELOPMENT OF AJAX WEB COMPONENTS IN EXTJS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. JAROSLAV DYTRYCH
AUTHOR

VEDOUCÍ PRÁCE — Doc. RNDr. PAVEL SMRŽ, Ph.D.
SUPERVISOR

BRNO 2009

Zadání diplomové práce/5743/2008/xdytry00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2008/2009

Zadání diplomové práce

Řešitel: **Dytrych Jaroslav, Bc.**
Obor: Informační systémy
Téma: **Vývoj webových komponent AJAX v prostředí ExtJS**
Kategorie: Web

Pokyny:

1. Seznamte se s možnostmi nástroje ExtJS
2. Prozkoumejte možnosti umisťování komponent se zaměřením na deklarativní zadání rozmístění.
3. Navrhněte a implementujte systém pro umisťování jednotlivých komponent a jejich synchronizaci.
4. Vyhodnoťte realizované řešení a porovnejte zvolený přístup s alternativními metodami
5. Vytvořte stručný plakát prezentující práci, její cíle a výsledky

Literatura:

- podle dohody

Při obhajobě semestrální části diplomového projektu je požadováno:

- funkční prototyp řešení

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Smrž Pavel, doc. RNDr., Ph.D.**, UPGM FIT VUT
Datum zadání: 22. září 2008
Datum odevzdání: 26. května 2009

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato práce se zabývá návrhem a implementaci správce rozložení stránky pro webové uživatelské rozhraní složené z komponent. Součástí řešení je i nový formát XML pro konfiguraci správce rozložení a sestavení stránky a nové aplikační rozhraní pro komponenty. Vytvořený systém splňuje požadavky evropského projektu KiWi a prošel akceptačními testy jeho projektového týmu.

Abstract

This thesis deals with the design and implementation of the page layout manager for a web user interface composed of components. A part of solution is also a new XML format for configuring of the manager of page layout and composition and a new application interface for components. The created system has met the requirements of the European project KiWi and has passed the acceptance tests of its project team.

Klíčová slova

uživatelské rozhraní, rozložení webové stránky, XML, JavaScript, AJAX, ExtJS, Google Gadgets, komponent, web

Keywords

user interface, web page layout, XML, JavaScript, AJAX, ExtJS, Google Gadgets, component, web

Citace

Jaroslav Dytrych: Vývoj webových komponent AJAX v prostředí ExtJS, diplomová práce, Brno, FIT VUT v Brně, 2009

Vývoj webových komponent AJAX v prostředí ExtJS

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Doc. RNDr. Pavla Smrže Ph.D. Další informace mi poskytli členové projektového týmu projektu KiWi. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jaroslav Dytrych
26. února 2009

Poděkování

Děkuji Doc. RNDr. Pavlu Smržovi, Ph.D., za odborné vedení práce, vstřícnost a ochotu při řešení této práce. Děkuji také všem členům projektového týmu české části projektu KiWi za spolupráci, poskytnuté informace a pomoc při testování.

Obsah

Kapitola 1

Úvod

Cílem práce bylo seznámit se s principy a vybranými existujícími implementacemi pokročilých webových uživatelských rozhraní složených z komponent využívajících AJAX a následně vytvořit pokročilý systém pro správu rozložení stránky a některé další součásti klientské části obdobného systému pro projekt KiWi.

V rámci své diplomové práce jsem se nejprve seznámil s obecnými principy tvorby webových uživatelských rozhraní složených z komponent a podrobně jsem prostudoval existující řešení od firem Google a Netvibes. Seznámil jsem se také z řešeními od dalších firem, jako např. Yahoo! a Pageflakes. Základní principy prostudovaných řešení jsou popsány v kapitole 2.

Následně jsem seznámil vývojářský tým české části projektu KiWi s funkcionalitou existujících systémů a analyzoval požadavky týmu na funkcionalitu nového správce rozložení stránky a rozhraní poskytované komponentům uživatelského rozhraní. Shrnutí těchto požadavků je uvedeno v kapitole 3.

Dle analýzy požadavků jsem zvolil technologie potřebné pro realizaci systému a tyto technologie prostudoval. Prostudoval jsem také knihovnu Ext JS, která byla zvolena jako nejvýhodnější řešení pro abstrakci mírně se lišících aplikačních programových rozhraní (dále API) různých webových prohlížečů. Informace o jednotlivých využitých technologiích a knihovně Ext JS jsou uvedeny v kapitole 4.

Na základě analýzy požadavků jsem navrhl pokročilý systém pro správu rozložení stránky, který oproti jiným existujícím systémům poskytuje větší možnosti přizpůsobení vzhledu stránky požadavkům jednotlivých uživatelů, při zachování principů ovládání uživatelského rozhraní těchto systémů, na které jsou uživatelé zvyklí. Rovněž možnosti tvorby komponentů jsou značně rozšířeny oproti jiným řešením, aby bylo možno systém využít pro tvorbu komplexního uživatelského rozhraní složeného ze vzájemně komunikujících interaktivních komponentů, které budou moci reagovat i na změny v rozložení stránky a další události vyvolané uživatelskou činností. Navrhl jsem také jednoduchý komunikační mechanismus pro vzájemnou komunikaci komponentů. Návrh API tohoto systému pro server je popsán v kapitole 5, stejně jako návrh konfiguračního XML s informacemi o rozložení stránky i jednotlivých komponentech. Navržené API pro komponenty je popsáno v kapitole 6.

Po dokončení všech detailů API a jejich schválení projektovým týmem KiWi jsem navrhl strukturu stránky (z hlediska implementace), strukturu správce rozložení stránky, včetně důležitých datových struktur, a API pro jeho inicializaci. Navrhl jsem také řešení některých problémů s webovými prohlížeči (spouštění dynamicky načtených skriptů, vyvolání událostí ve správný čas nezávisle na způsobu, jakým prohlížeč spouští skripty v komponentech, apod.). Výsledky jsou uvedeny v kapitole 7.

Popsané řešení jsem implementoval, přičemž jsem vyřešil i další problémy zjištěné v průběhu implementace. O vlastní implementaci a největších problémech řešených v této fázi vývoje systému pojednává kapitola 8.

Řešení jsem otestoval a dal k dispozici ostatním členům týmu, kteří při tvorbě komponentů ověřili, že výsledné řešení poskytuje všechny potřebné funkce a plně vyhovuje jejich potřebám. V průběhu testování jsem ostatním členům týmu poskytoval technickou podporu ke správci rozložení a prováděl také drobné úpravy a dolaďování systému dle nově vznikajících požadavků zjištěných v průběhu vývoje komponentů. Informace o testování systému jsou uvedeny v kapitole 9.

Spolupracoval jsem také na vývoji některých komponentů uživatelského rozhraní pro projekt KiWi, přičemž jsem vytvářel především jejich vizuální součásti využívající mnou vytvořené řešení a knihovnu Ext JS. Na závěr jsem zhodnotil dosažené výsledky a možnosti budoucích rozšíření systému. U rozšiřitelnosti jsem se zabýval především možnostmi doplnění API pro komponenty od jiných výrobců, mezi které patří např. Google Gadgets, UWA Widgets a Flakes. Závěr se nachází v kapitole 10.

Ve třetím semestru svého magisterského studia jsem v rámci semestrálního projektu provedl studijní část práce a navrhl API systému včetně konfiguračního XML. Navrhl jsem také správce rozložení a implementoval jeho funkční prototyp. Implementaci jsem dokončil v navazujícím semestru, ve kterém jsem v rámci své diplomové práce provedl také testování a ladění systému. Vytvořil jsem rovněž dokumentace pro vývojáře serveru i komponentů, které jsou obsaženy v přílohách A a B.

Kapitola 2

Webové uživatelské rozhraní složené z komponent

Mezi moderní trendy v oblasti uživatelských rozhraní patří přizpůsobitelnost uživateli. Webové prohlížeče, postupující standardizace API pro klientský JavaScript (standardizován jako ECMAScript) a asynchronní komunikace pomocí technologie AJAX umožňují uplatnění tohoto trendu v oblasti webových aplikací. Uživatelé začínají ve stále větší míře vyžadovat možnost přizpůsobení svých domovských stránek a webových rozhraní často využívaných služeb (e-mail, instant messaging apod.) svým potřebám a zvyklostem (viz [23] a [7]).

2.1 Existující řešení

Mezi první poskytovatele přizpůsobitelné domovské stránky patří firmy Google, Microsoft a Yahoo!, po nichž následovala řada menších firem poskytujících více či méně kvalitní alternativy. Tyto stránky jsou složeny z jednotlivých komponentů, přičemž uživatel si může zvolit požadované komponenty a v rámci možností daného systému si je rozmístit na stránce. Některé systémy umožňují i uživatelské nastavení vlastností a obsahu komponentů, či vytváření vlastních komponentů a jejich sdílení s dalšími uživateli [23]. Srovnání funkcionality různých systémů od větších i menších výrobců lze nalézt např. v [32] a [28].

2.1.1 Principy

Existují různé přístupy k implementaci systému (někdy též prostředí) tvořícího přizpůsobitelnou domovskou stránku a poskytujícího API komponentům. Jednotlivá řešení se liší především mírou závislosti komponentů na daném systému, způsobem umístění komponentů na stránce a strukturou API. Z existujících řešení jsem vybral tři, která jsou typickými zástupci odlišných konceptů, a popíši zde jejich základní vlastnosti, výhody a nevýhody.

iGoogle

Systém iGoogle [20] od firmy Google nabízí domovskou stránku s maximálně třemi sloupci, do nichž uživatel může umisťovat komponenty nazvané Google Gadgets (více viz [40] a [18]). Tento systém pro jednotlivé komponenty využívá vložené rámce (elementy `iframe`), které skriptům v komponentu umožňují jednoduché zjištění vlastní identity (viz dále), a současně poskytují maximální izolovanost komponentů. Každý komponent má přidělen svůj rámec,

ve kterém může provádět všechny operace potřebné pro manipulaci se svým obsahem. Jednotlivé vložené rámce jsou obaleny do elementů `div` tvořících ohraničení a hlavičku komponentu, která obsahuje titulek a ikony či odkazy s nástroji umožňujícími nastavení vlastností komponentu, jeho odstranění ze stránky a jiné funkce poskytované systémem.

Systém provádí správu rozložení, která uživateli umožňuje přemisťovat komponenty na stránce tažením myši, přepínat mezi záložkami apod. Jednotlivým komponentům poskytuje JavaScriptové API umožňující přístup k částem rozhraní spravovaným systémem (hlavička komponentu, nastavení, rozměry komponentu apod.), komunikaci se serverem a ostatními komponenty a další potřebné funkce. Část této funkcionality je naprogramována na klientské straně (běží ve webovém prohlížeči uživatele), ale velké množství funkcí je prováděno i na serveru. Server dle potřeby rozhoduje o tom, které funkce budou v API dostupné, a vkládá do stránky příslušné knihovny. Provádí také náhrady zástupných znaků ve zdrojových kódech komponentů za lokalizované texty či hodnoty voleb nastavení poskytnuté uživatelem. Zdrojový kód komponentů je tedy na serveru předzpracován, čímž je dosaženo zjednodušení klientské části systému. Uživatelská nastavení jsou ukládána do cookies v prohlížeči a u registrovaných a přihlášených uživatelů i na server firmy Google.

Mezi výhody tohoto systému patří jednoduchost implementace komponentů, vyšší odolnost vůči chybám v komponentu (chybný skript v komponentu nemusí způsobit chybné zobrazení celé stránky, či zablokování systému pro daného uživatele) a možnost využití mnoha služeb Google pro jejich vývoj a sdílení.

Hlavní nevýhodou je výrazná závislost komponentů na daném systému. Provoz komponentu mimo tento systém vyžaduje implementaci poměrně složitého API, které je obtížně integrovatelné do jiného systému. Struktura API také vyžaduje užití vložených rámců, které jako jediné každému skriptu umožňují zjištění id rodičovského komponentu, což může výrazně zkomplikovat výsledný systém (práce s vloženými rámci může v některých systémech výrazně navýšit jejich složitost).

Více informací o API pro Google Gadgets lze nalézt v příručce pro vývojáře [19].

Mezi alternativní implementace API pro Google Gadgets patří implementace od nadace Apache nazvaná Shindig [3]. Toto řešení ale neumožňuje jednoduchou úpravu a implementaci některých funkcí týkajících se především formuláře pro nastavení a hlavičky komponentu. V případě jeho využití v heterogenním systému s různými typy komponentů komplikuje vytvoření jednotného uživatelského rozhraní a přidává výrazné množství zbytečného kódu (další implementace tvorby formuláře apod.), který zpomaluje načítání stránky.

Netvibes

Netvibes [24] je řešení poskytované menší firmou, která si dala za cíl dosažení maximální kompatibility svých komponentů se systémy od jiných výrobců (více viz [41]). Komponenty jsou nazvané UWA (Universal Widget API) a jsou tvořeny kompletní webovou stránkou, do které jsou vloženy skripty poskytující API systému [25]. Zdrojový kód těchto skriptů je otevřený (open source), což umožňuje snadné prostudování využitých principů a jednodušší integraci do jiných systémů. Každý komponent je tedy možné provozovat či testovat samostatně, vložit do rámce v libovolné webové stránce, umístit do jiného systému (např. iGoogle) nebo přímo do systému Netvibes.

V případě, že je komponent umístěn přímo v systému Netvibes, serverová část systému provede analýzu komponentu a na stránce jej umístí do elementu `div`, kolem kterého obdobně jako iGoogle vytvoří obalující elementy se záhlavím komponentu. Klientská část potom poskytuje zobrazování formulářů s nastavením, možnosti přetahování komponentů

myší, změny šířek sloupců apod. Pokud je komponent umístěn v jiném systému, tento systém nemusí poskytovat stejnou funkcionalitu, může poskytnout pouze proxy server pro komunikaci a vložený rámec pro umístění komponentu. Implementace zbylé funkcionality je volitelná a její míra udává rozsah omezení funkcionality komponentu. Nastavení komponentů je dle cílového umístění ukládáno do cookies či databáze systému.

Výhodou tohoto řešení oproti iGoogle je menší závislost komponentů na cílovém systému a větší svoboda ve tvorbě alternativních implementací API. Při využití pouze základních součástí systému lze také jednoduše doimplementovat některé jeho vizuální součásti (např. formulář pro nastavení), což umožňuje dosažení jednotného vzhledu a ovládání komponentů v heterogenním systému při zachování efektivity. Pokud však cílový systém neimplementuje potřebné funkce pro nastavení komponentu a správu rozložení stránky, nebo pokud pro daný systém neexistuje skript pro komunikaci systému se skripty Netvibes poskytujícími API danému komponentu (skript prostředí nazvaný UWA Environment), funkcionalita komponentů je výrazně omezena (nemožnost nastavení komponentu apod.). Výhodou je dále možnost změny šířek sloupců s komponenty, a tím i lepší přizpůsobitelnost vzhledu stránky.

Nevýhodou je menší odolnost vůči chybným komponentům (chybný skript v komponentu může způsobit chybné zobrazení celé stránky) a skutečnost, že systém komponentům neposkytuje žádné prostředky pro vzájemnou komunikaci.

My Yahoo

My Yahoo! [48] od firmy Yahoo! [42] je řešení, které je určeno převážně pro registrované uživatele služeb Yahoo! Do cookies se zde ukládá pouze část informací, a bez registrace tedy není možné provádět některá nastavení. Komponenty obsahují především služby Yahoo! a RSS kanály a chybí zde možnost jednoduchého vytváření vlastních komponentů.

Pro uživatele Yahoo! má toto řešení výhodu v možnosti volby obsahu a přizpůsobení vzhledu uživatelského rozhraní. Výraznými nevýhodami jsou specifické zaměření a uzavřenost systému, které znemožňují integraci komponentů do jiných systémů i běh komponentů z jiných systémů v My Yahoo!.

Další řešení

Řešení od jiných firem jsou téměř vždy obdobou uvedených systémů a liší se především ve vzhledu, možnostech přizpůsobení a detailech API. Obecné principy jsou v převážné většině obdobou výše uvedených. Informace o některých těchto systémech lze nalézt v [28] a [32].

Kapitola 3

Analýza požadavků projektu KiWi

Do projektu KiWi [35] bylo třeba vytvořit správce rozložení (layout manager), který pomocí technologie AJAX ze serveru načte konfigurační XML a dynamicky sestaví stránku složenou z komponentů. Pro XML může být využit existující formát (existuje-li dostupný formát s dostatečnou flexibilitou, který umožní uložit všechny potřebné informace), nebo může být navržen formát zcela nový. Stránka musí umožňovat vytvoření obdobného vzhledu, jako má iGoogle, ale zároveň poskytovat mnohem větší variabilitu rozložení. Správce rozložení má také poskytovat další níže uvedené služby komponentům a do budoucna umožní dosažení maximální kompatibility s jinými systémy a využití jejich komponentů. Bylo také třeba vytvořit jednoduchý mechanizmus pro vzájemnou komunikaci komponentů.

Pro tvorbu systému bude využita knihovna Ext JS, která byla zvolena pro svou velkou flexibilitu, funkcionalitu a dostatečnou abstrakci rozdílného API různých webových prohlížečů. Všechny komponenty mohou být závislé na této knihovně a lze předpokládat, že tvůrci komponentů budou znát její API.

3.1 Rozložení stránky

Rozložení stránky bude navrženo tak, aby poskytovalo dostatečnou flexibilitu, ale nebylo příliš komplexní (jako např. plocha v operačním systému Microsoft Windows). Stránka bude rozdělena na oblasti, přičemž vhodné je např. rozdělení na jednu centrální a čtyři volitelné okolní oblasti (horní, dolní, levá a pravá). Každá oblast může mít titulek a bude ji možné sbalit (skrýt) či zavřít. V každé oblasti může být libovolný počet sloupců, ve kterých budou umístěny komponenty. Bude také možné vytvořit oblast s jedním komponentem přes celou plochu oblasti a taková oblast následně může být využita jako postranní skrývatelný panel, či stále zobrazený panel s důležitými informacemi. Komponenty bude možné přesouvat tažením myší, a to jak uvnitř oblasti, tak i mezi oblastmi. Tažením myší bude možná i změna rozměrů oblastí. Veškerá tato funkcionalita musí být volitelná (nastavitelná v konfiguračním XML), aby byla poskytnuta dostatečná flexibilita. Bude tedy možné vytvořit oblasti bez záhlaví, které nepůjde sbalit ani zavřít, komponenty které nebude možné přesouvat apod. Pomocí konfigurace v XML musí být rovněž možno vytvořit sloupce bez okrajů a měnit poměry velikosti jednotlivých sloupců v oblasti.

3.2 API

Stránka musí být načtena celá současně (nikoliv po jednotlivých komponentech), aby byl zjednodušen návrh spolupracujících komponentů a snížen počet požadavků na server. Jednotlivé komponenty budou uvedeny přímo v konfiguračním XML a správce rozložení musí provést jejich analýzu a nahradit v nich zástupné symboly za lokalizované texty a hodnoty proměnných. Formát XML a API pro server musejí být navrženy tak, aby se minimalizovaly požadavky na server a správce rozložení bylo možno využít s různými servery.

API pro komponenty by mělo být navrženo tak, aby bylo možno vytvářet komponenty, které budou maximálně nezávislé na daném systému a bude je možné využít i samostatně či v jiných systémech. Musí zde však být i možnost vytvářet komponenty, které sice budou závislé na vytvořeném systému, ale budou mít možnost spolupracovat, zjišťovat vzájemně svoje pozice a nastavení, reagovat na změny v rozložení stránky apod.

Veškerá nastavení budou ukládána pomocí speciálního komponentu určeného k tomuto účelu. Správce rozložení musí ve svém API zahrnovat metody, pomocí kterých bude možné získat všechny potřebné informace. Komponent pro ukládání bude vytvořen členem projektového týmu, který bude vytvářet server.

3.3 Další požadavky

Systém musí být lokalizovatelný do cizích jazyků. Názvy proměnných, vlastností a metod v API musejí být zapsány v angličtině, aby byly lehce srozumitelné pro zahraniční členy týmu.

Systém musí poskytovat částečnou kompatibilitu a podporu pro jednoduché komponenty Google Gadgets. Kompatibilita bude zahrnovat zpracování elementů XML s Gadgetem a všechny metody potřebné pro emulaci základních součástí API Google Gadgets. Mým úkolem bude vytvořit emulaci všech součásti tohoto API týkajících se rozložení stránky, zobrazení, nastavení a pomocných funkcí (třída `gadgets.util`) s výjimkou volitelných součástí (součástí, které nemusejí být podporovány ve všech implementacích), které kolidují s rozložením stránky, či jinými součástmi systému. Kolidující součásti musejí být korektně ošetřeny (např. u pohledů (views) musí být zobrazen správný pohled a Gadget musí korektně zjistit, že alternativní pohledy nejsou podporovány). Pro emulaci součástí souvisejících s rozložením stránky by měly být vytvořeny obalující třídy, které budou nadstavbou nad metodami správce rozložení, který ke stejnému účelu poskytne flexibilnější API. Součásti API určené pro komunikaci budou vytvořeny současně se serverem jiným členem projektového týmu, stejně jako třídy pro činnost složitějších Gadget (konkrétně Tab, TabSet, flash a skins). Rovněž není vyžadována podpora všech Gadgetů bez nutnosti jejich úprav (Gadget může být např. upraven skriptem, či drobnějším zásahem programátora). Specifický kód pro podporu Google Gadgets, který nelze využít k jinému účelu, by měl být minimální a využití obalujících tříd volitelné.

V komponentech musí být možné využívat komplexní textový editor TinyMCE, správce rozložení musí být kompatibilní s tímto editorem a poskytovat metody potřebné pro jeho provoz.

Do budoucna by mělo být možné systém rozšířit tak, aby v něm bylo možné provozovat komponenty z jiných systémů (Netvibes, Pageflakes, eskobo apod.).

Kapitola 4

Využité technologie

V této kapitole uvedu základní informace o využitých technologiích. Technologie jsem volil především na základě požadavků na výsledný systém a dle jejich dostupnosti v běžně využívaných prohlížečích.

4.1 Jazyky pro popis dokumentu a jeho vzhledu

Pro popis dokumentu a jeho vzhledu jsem využil XHTML a CSS, protože tyto jazyky mají podporu ve všech běžně využívaných prohlížečích a jsou využívány i knihovnou Ext JS.

4.1.1 XML

XML (Extensible Markup Language) je obecný flexibilní značkovací jazyk, který umožňuje vytváření značkovacích jazyků pro různé účely a typy dat. Umožňuje popsat strukturu dokumentů, přičemž se nezabývá jejich vzhledem (vizuální prezentací). Byl vyvinut konsorciem W3C jako nástupce dříve využívaného SGML (Standard Generalized Markup Language), přičemž odstranil všechny jeho hlavní nevýhody (např. nedeterminističnost způsobenou možností vynechávání značek). Více o XML se lze dozvědět v literatuře [46] či z webových stránek konsorcia W3C [4].

4.1.2 XHTML

XHTML (The Extensible HyperText Markup Language) je značkovací jazyk pro popis struktury dokumentu, který je aplikací univerzálního jazyka XML. Je využíván převážně k tvorbě hypertextových dokumentů v prostředí webu (webových stránek). Vytvořilo jej konsorcium W3C reformulací jazyka HTML 4 (HyperText Markup Language) v XML 1.0. Jazyk HTML v roce 1990 navrhl Tim Berners-Lee a v průběhu následujících let se vyvíjel a byl standardizován. Verze 4.01 pochází z roku 1999 a je poslední stabilní verzí tohoto jazyka, který se nyní vyvíjí paralelně s XHTML.

Více informací o XHTML se lze dozvědět z [30], [34] a [26], aktuální informace o vývoji a nových verzích se nacházejí na webových stránkách konsorcia W3C [4].

4.1.3 DOM

DOM (Document Object Model) je objektově orientovaná stromová reprezentace XML nebo XHTML dokumentu. DOM je také API pro přístup k obsahu, struktuře a stylům dokumentu

a jejich modifikaci. Toto API se v jednotlivých prohlížečích mírně liší od standardizované verze od W3C a aplikace, které jej využívají, musejí tyto odlišnosti zohlednit. Více o DOM lze nalézt v [43].

4.1.4 CSS

CSS (Cascading Style Sheets) je jazyk pro popis způsobu zobrazení dokumentů napsaných ve značkovacích jazycích HTML, XML nebo XHTML. Navrhl jej Håkon Wium Lie v roce 1994 a je udržován organizací W3C. Aktuální verze je CSS 2.1 z roku 2002, která byla naposledy aktualizována v prosinci 2006. Nyní již existuje i nová verze CSS 3, která je stále ve vývoji a její podpora v prohlížečích není dostatečná.

Pro svoji práci jsem zvolil podmnožinu jazyka CSS 2.1, která je podporována běžně využívanými prohlížeči. Většina stylových předpisů však bude využita přímo z knihovny Ext JS, kterou využiji pro zobrazování.

Více o CSS se lze dozvědět z [29] a [31], přesnou specifikaci a aktuální informace o vývoji a nových verzích lze nalézt na stránkách konsorcia W3C [4].

4.2 Skriptovací jazyky

Skript je zdrojový kód programu, který je čten a vykonáván za běhu (tzv. interpretován). Programovací jazyky pro psaní skriptů jsou nazývány skriptovací jazyky (více viz [45]).

Skripty ve webových aplikacích se dělí podle toho, jestli jsou prováděny na straně klienta (ve webovém prohlížeči), nebo serveru. Pro skriptování na straně klienta jsem zvolil JavaScript, protože je velmi dobře podporován ve většině běžně využívaných prohlížečů.

Vzhledem k tomu, že mým úkolem je tvorba součástí systému běžících na klientovi, skriptování na straně serveru využiji pouze pro tvorbu pomocných souborů pro testování a demonstraci činnosti vytvořeného řešení. Za tímto účelem jsem zvolil jazyk PHP pro jeho velkou rozšířenost a dostupnost.

4.2.1 JavaScript

JavaScript je multiplatformní objektově orientovaný skriptovací jazyk. Nejčastěji se využívá jako interpretovaný skriptovací jazyk pro webové stránky, který se ve zdrojovém či komprimovaném (minimalizovaném) tvaru vkládá do zdrojového kódu stránek a je prováděn na straně klienta. Může být také v externím souboru, který je ke stránce připojen příslušným elementem, nebo může být do stránky načten dynamicky až za běhu.

Tvůrcem JavaScriptu je Brendan Eich, který jej navrhl v roce 1995. Následně byl standardizační organizací ECMA standardizován jako ECMAScript a firma Microsoft vytvořila vlastní implementaci nazvanou JScript. Protože ke standardizaci klientského JavaScriptu v prohlížečích docházelo postupně, až po vzniku různých implementací, podpora JavaScriptu se v jednotlivých webových prohlížečích mírně liší, přičemž největší odlišnosti jsou v API pro modifikaci struktury a obsahu stránky (přístup k DOM, obsahu elementů apod.) a komunikaci (objekt XMLHttpRequest).

Aktuální verze je JavaScript 2.0, která odpovídá vývojové verzi standardu ECMAScript Edition 4 a bude standardizována ve standardu ECMA-262 Edition 4. Poslední stabilní verze standardu ECMA je ECMAScript Edition 3 (ECMA262 Edition 3) z roku 1999, která odpovídá JavaScriptu ve verzi 1.5. Pro práci s XML existuje také standard ECMA-357 (ECMAScript for XML (E4X)) z roku 2004.

Více o JavaScriptu se lze dozvědět z literatury [15] a [22], aktuální informace o vývoji a nových verzích lze nalézt na stránkách [6].

4.2.2 PHP

PHP (rekurzivní akronym pro „PHP: Hypertext Preprocessor“) je univerzální skriptovací jazyk určený především pro vývoj webových aplikací, ale lze jej využít i pro tvorbu desktopových aplikací. Navrhl jej Rasmus Lerdorf, který jej společně s Andi Gutmansem, Zeevem Suraskim a dalšími členy skupiny PHP Group dále vyvíjí.

Skripty jsou prováděny na straně serveru a do uživatelského prohlížeče se zasílá pouze výstup těchto skriptů.

Aktuální verze PHP je 5.2.6. Více o PHP se lze dozvědět z literatury [17] a [49] či webových stránek PHP Group, kde naleznete i informace o vývoji a nejnovějších verzích [27]. V literatuře naleznete rovněž český manuál [1].

4.3 Komunikační mechanismy

Webový klient pro komunikaci se serverem využívá protokol HTTP. Tradiční model komunikace webového klienta se serverem je synchronní, což znamená, že klient jako reakci na činnost uživatele (kliknutí na odkaz, odeslání formuláře apod.) zašle požadavek na server, který mu odpoví zasláním stránky, která bude zobrazena. Při každé akci tedy dojde k obnovení či načtení jiné stránky. Poslední trendy vývoje se posouvají k asynchronní komunikaci, kdy zobrazená stránka na pozadí komunikuje se serverem nezávisle na činnosti uživatele a nedochází zde k obnovování celé stránky, ale pouze k načítání a odesílání potřebných dat. Výhodou tohoto přístupu je, že uživatel není při své práci rušen čekáním na načtení stránky, nevýhodou je obtížné či nemožné uložení odkazu na konkrétní dynamicky vytvořenou stránku. Nejčastěji využívanou strategií pro tuto komunikaci je AJAX, který jsem vzhledem k požadavkům na výsledný systém využil.

4.3.1 HTTP

HTTP (Hypertext Transfer Protocol) je internetový protokol určený pro výměnu hypertextových dokumentů mezi klientem a serverem. Pracuje v modelu požadavek/odpověď, je bezstavový a má vstupně-výstupní charakter. S rozšířením MIME (Multipurpose Internet Mail Extension) definovaným dokumenty RFC-822, RFC-2045, RFC-2046, RFC-2047, RFC-2048 a RFC-2049 umožňuje přenos téměř libovolných souborů a ve značné míře tak nahrazuje i protokol FTP (File Transfer Protocol). Spojení iniciuje klient a ukončuje server společně se zaslanou odpovědí.

Aktuální verze je HTTP 1.1, která oproti předchozí verzi HTTP 1.0 optimalizuje využití spojení a šířky pásma, obsahuje lepší podporu zabezpečení a umožňuje vytváření virtuálních hostů (více URL na jedné IP adrese). Poslední verze specifikace HTTP 1.1 z roku 1999 je obsažena v dokumentu RFC-2616 [14]. Více informací o tomto protokolu lze nalézt v [44] a [39].

4.3.2 AJAX

Ajax je strategie pro tvorbu webových aplikací umožňující načítání dat ze serveru pomocí skriptů v zobrazené webové stránce a dynamickou aktualizaci stránky bez obnovení (nového

načtení). Jeho název a popis v roce 2005 zformuloval Jesse James Garrett ve svém článku Ajax: A New Approach to Web Applications [16].

Ajax se skládá především z prezentace tvořené standardním XHTML a CSS, dynamického zobrazování pomocí DOM, výměny dat se serverem pomocí XML a XSLT realizované asynchronně pomocí objektu XMLHttpRequest a JavaScriptu.

Více informací o této strategii, její historii a detailní popis lze nalézt v [5], [37], [36] a [2].

XSLT (eXtensible Stylesheet Language Transformations) je transformace, která slouží k převodu zdrojových dat ve formátu XML do libovolného jiného formátu, nejčastěji jiného XML či XHTML (více viz [47]). Objekt XMLHttpRequest je součástí JavaScriptového API webových prohlížečů a umožňuje asynchronní výměnu dat se serverem.

4.4 Knihovna Ext JS

Knihovna Ext JS je javascriptová knihovna pro tvorbu interaktivních webových aplikací s využitím technologie AJAX a manipulací s DOM. Může být provozována samostatně, nebo s různými uživatelskými rozšířeními poskytujícími přidanou funkcionalitu nebo snadnou integraci produktů třetích stran (např. textový editor TinyMCE).

Základní verze knihovny poskytuje funkcionalitu pro práci s rozložením stránky, okny, formuláři, dynamickými tabulkami, stromy, nabídkami a dalšími prvky uživatelského rozhraní. Poskytuje rozhraní pro práci s technologií AJAX nezávislé na prohlížeči a metody pro práci s různými formáty dat a jejich načítání ze serveru, dočasné uložení v prohlížeči a převod do vhodných datových struktur, které lze využívat přímo pro tvorbu prvků uživatelského rozhraní (např. pole pro výběr ze seznamu hodnot). Upravuje také některé základní konstruktory JavaScriptu, a rozšiřuje tak jeho funkcionalitu o možnosti zpožděného volání metod, některé operace s poli a řetězci a další potřebné metody. Má i pomocné vlastnosti pro identifikaci operačního systému, prohlížeče a jeho režimu.

Hlavní výhodou využití knihovny Ext JS je abstrakce práce s prohlížeči umožňující vytváření aplikací s minimálním množstvím kódu závislým na konkrétním prohlížeči. Je kompatibilní s většinou běžně využívaných prohlížečů v nejčastěji využívaných verzích. Při vhodném návrhu aplikace lze při vzniku nových prohlížečů (nebo nových verzí stávajících) dosáhnout kompatibility s těmito prohlížeči pouhou aktualizací knihovny bez nutnosti zásahu do kódu aplikace.

Aktuální verze je Ext 2.1 z roku 2008, která se od předchozí verze 2.0 liší především novou duální licencí (komerční a LGPL pro osobní, vzdělávací a nekomerční využití). Již od verze 2.0 z roku 2007 je výrazně vylepšen výkon, vzhled a funkcionalita. Jsou také zmenšeny velikosti souborů. Více informací o Ext JS a jeho nových verzích lze nalézt v [10], [13] a [38]. Dokumentaci API lze nalézt na stránkách [9], které byly rovněž vytvořeny pomocí této knihovny.

Kapitola 5

Návrh API pro server

V této kapitole se budu zabývat návrhem API správce rozložení pro server. Výsledný server bude muset provádět nejenom níže popsané činnosti, ale bude muset pracovat i s komponenty, které budou pro svoji komunikaci se serverem využívat vlastní API. Komunikace komponentů se serverem bude pro dosažení požadované nezávislosti na správci rozložení prováděna přímo pomocí příslušných metod Ext JS.

Vzhledem k požadavkům jsem API pro server navrhl tak, aby byly na server kladeny minimální nároky. Veškerá činnost serveru se skládá z generování indexové stránky webu a konfiguračního XML, jehož návrh je uveden níže.

5.1 Indexová stránka webu

Indexová stránka webu bude ve své hlavičce obsahovat elementy s vazbami na skripty a stylové předpisy Ext JS, správce rozložení, observeru a dalších potřebných knihoven a součástí systému. V těle stránky bude element s informacemi pro uživatele bez podpory JavaScriptu, který bude ihned po úspěšné inicializaci Ext JS automaticky skryt. Bude zde také element s textem zobrazeným v průběhu načítání konfiguračního XML ze serveru a prázdný element pro umístění indikátoru průběhu činnosti (progressbar).

Pro lokalizaci správce rozložení bude využit obdobný princip, jaký je využit v knihovně Ext JS, a pro lokalizaci tedy bude postačovat vložení vazeb na příslušné lokalizační skripty do hlavičky stránky.

Příklad vygenerované indexové stránky s detailním popisem je obsažen v příloze B.

5.2 Návrh konfiguračního XML

Konfigurační XML pro správce rozložení musí obsahovat informace o rozložení stránky i jednotlivé komponenty zobrazené na stránce. Vzhledem k tomu, že k danému účelu dosud neexistuje žádný standardizovaný formát, navrhl jsem vlastní strukturu tohoto XML. Při návrhu jsem vycházel z požadavků na výsledné řešení, možností a API Ext JS a částečně i z API Google Gadgets, s nímž má systém poskytovat částečnou kompatibilitu. Kladl jsem důraz především na přehlednost XML a intuitivní pojmenování elementů, přičemž jsem předpokládal, že uživatel systému (vývojář serveru či komponentů) má alespoň minimální znalosti API Ext JS a API Google Gadgets.

Nejprve jsem navrhl základní strukturu XML a jeho minimální povinný obsah. Minimální obsah neobsahuje žádné užitečné informace, ale pouze základ struktury, podle kterého

lze rozlišit, zda se jedná o mnou navržený formát XML. Tento obsah je následující:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<pageData>
<layoutSettings>
</layoutSettings>
</pageData>
```

Do tohoto XML jsem postupně přidával definice oblastí, sloupců a jednoduchých komponentů, přičemž jsem definoval výchozí chování správce rozložení při neuvedení jednotlivých informací. Pro názvy oblastí a některých elementů a atributů jsem pro přehlednost využil názvy z Ext JS. Takto jsem definoval všechny součásti komponentu, které jsou třeba pro základní komponent s minimální závislostí na správci rozložení (titulek, zavírání, sbalování, přesunutelnost, ...).

Příklad obsahu XML s definicí centrální oblasti s jedním sloupcem bez okrajů a jedním komponentem:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<pageData>
<layoutSettings>
  <area location="center">
    <title>Centrální oblast</title>
    <collapsing enabled="false" collapsed="false"/>
    <cols>
      <col width="100%" padding="0"/>
    </cols>
  </area>
</layoutSettings>
<component id="2" location="center" col="0" row="0">
  <preferences>
    <title>Titulek v záhlaví komponentu</title>
    <collapsing enabled="true" collapsed="false"/>
    <closing enabled="true"/>
    <fixedHeight value="100" scrolling="true"/>
    <dragging enabled="true"/>
  </preferences>
  <content type="text/html">
  <![CDATA[
     Text v těle komponentu.
  ]]>
  </content>
</component>
</pageData>
```

Následně jsem definoval rozšířené možnosti komponentu, přičemž jsem využil i některé ze součástí API Google Gadgets, které jsem k tomuto účelu zjednodušil. Následuje příklad komponentu s těmito možnostmi:

```
<component id="2" location="center" col="0" row="0">
    <Locale lang="cs" country="CZ"/>
    <UserPref name="myName" display_name="Name" required="true"/>
```

```
    <UserPref name="myEnum" display_name="__MSG_colors__"
              datatype="enum">
      <EnumValue value="Red" display_value="__MSG_red__ color"/>
      <EnumValue value="Green"/>
      <EnumValue value="Yellow" display_value="Yellow color"/>
    </UserPref>
    <UserPref name="myList" datatype="list"
              default_value="Red|Green|Blue"/>
    <messagebundle>
      <msg name="myLocalizedText">
        Lokalizovaná část textu.
      </msg>
      <msg name="colors">Colors</msg>
      <msg name="red">Red</msg>
    </messagebundle>
  <content type="text/html">
  <![CDATA[
    Text v těle komponentu.
    __MSG_myLocalizedText__
  ]]>
  </content>
</component>
```

Nakonec jsem definoval elementy a atributy nutné pro kompatibilitu s Google Gadgets, jejichž sémantika byla v již navrženém XML obsažena (titulek, výška komponentu apod.). Tyto definice sice zavádějí nutnost určit chování při uvedení různých informací v obou elementech se stejnou sémantikou, ale přímé využití elementů z Google Gadgets by bylo nevhodné. Např. titulek komponentu je v Gadgetu definován atributem elementu `ModulePrefs`, ale v mnou navrženém komponentu je umístěn v elementu `title`, který je k danému účelu využíván častěji a poskytuje tak větší přehlednost a srozumitelnost.

Navržený formát je flexibilní a většina elementů je volitelná. Např. element `preferences` nemusí být uveden a jeho obsah může být přímo v elementu `component`. Rovněž element `Locale` může být uveden přímo v elementu `component`, nebo na libovolné úrovni vnoření v tomto elementu. Bude-li některý element, který má v komponentu význam pouze jednou (`collapsing`, `closing`, ...), uveden v komponentu vícekrát, bude využita první nalezená instance. Neznámé elementy a atributy budou při analýze ignorovány. Tato volnost formátu umožňuje budoucí analýzu komponentů převzatých z jiných systémů s nutností minimálních úprav (není nutno měnit strukturu, ale pouze syntax).

Výsledná struktura XML, včetně detailního popisu významu jednotlivých elementů, je uvedena v příloze A.

Vzhledem k předpokladu, že do tohoto XML budou vkládány či transformovány XML s komponenty z jiných systémů, z nichž některé elementy lze bez ztráty důležitých informací ignorovat, není nutné, aby bylo vygenerované XML vždy validní (odpovídalo formální definici). Míra odlišnosti od „správného“ formátu XML zde udává pouze míru správnosti funkce a zobrazení obsažených komponentů.

Kapitola 6

Návrh API pro komponenty

Při návrhu API jsem vycházel z požadavků na funkcionalitu komunikačního mechanismu a správce rozložení. Protože API Google Gadgets ani jiného existujícího systému neposkytuje dostatečnou funkcionalitu (chybí zde např. metody pro zjištění pozice komponentu, možnost reakce na některé události, apod.), API pro komponenty jsem navrhl zcela nové.

Nejprve jsem navrhl komunikační mechanismus a jeho API a toho jsem následně využil při návrhu konzistentního API pro správce rozložení.

6.1 Komunikační mechanismus

Vzhledem k požadavku možnosti vytváření skupin spolupracujících komponentů, které nemusejí mít o sobě vzájemně informace, je nejvýhodnějším komunikační mechanismus vytvořený pomocí návrhového vzoru observer. Jedná se o stavový objekt, jehož vlastnosti může každý komponent aktualizovat, přičemž jsou na provedenou změnu upozorněny všechny komponenty, které jej sledují (voláním metod registrovaných v observeru). Základní observer jsem zde rozšířil o možnost definování více stavových objektů a uživatelských událostí. Každý komponent tak může využít standardní stavový objekt a metody pro jeho aktualizaci a získání jeho vlastností, nebo definovat vlastní stavový objekt a jednu nebo více událostí jeho změny. Každá událost může mít definovány i parametry, které lze využít pro předání potřebných informací i bez využití stavového objektu.

Protože Ext JS obsahuje abstraktní bázovou třídu pro vytváření observerů s většinou potřebných vlastností a metod (`Ext.util.Observable`), využil jsem jako základ pro vytvoření observeru tuto třídu. Z metod, které tato třída poskytuje, jsou důležité zejména následující:

- `addEvents()` – metoda pro definování nových událostí observeru
- `addListener()` – metoda pro registraci posluchače událostí (funkce, která bude volána v reakci na událost)
- `fireEvent()` – metoda pro vyvolání události
- `hasListener()` – metoda pro zjištění, zda má objekt zaregistrovanou nějakou funkci pro obsluhu události
- `removeListener()` – metoda pro odregistrování posluchače událostí (funkce volané v reakci na událost)

- `relayEvents()` – metoda pro nastavení přeposílání událostí mezi observery

Kromě výše uvedených jsou v této třídě i další metody, které však k danému účelu nejsou potřebné a využití některých z nich by bylo nevhodné (např. metoda pro zablokování všech událostí by způsobila úplné zablokování komunikace komponentů).

Protože ve třídě `Ext.util.Observable` není definován stavový objekt, rozšířil jsem ji o tento objekt a metody pro jeho aktualizaci a získání hodnot jeho vlastností. Pro optimalizaci práce se stavovým objektem při komunikaci více komponentů (ignorování událostí, které se daného komponentu netýkají) je výhodné, aby při události změny stavového objektu bylo možné zjistit, který objekt provedl poslední aktualizaci objektu. Nejjednodušším způsobem řešení tohoto problému (z hlediska využití výsledného API) je předávání identifikátoru objektu, který provádí aktualizaci, jako parametru metody pro aktualizaci stavového objektu.

Výsledné API observeru tedy bude mít metody zděděné ze třídy `Ext.util.Observable` a následující nově vytvořené metody:

- `update()` – metoda pro aktualizaci hodnoty vlastnosti stavového objektu
- `getStateVariable()` – metoda pro získání hodnoty vlastnosti stavového objektu
- `getState()` – metoda pro získání přístupu ke stavovému objektu

Detailní popis uvedených metod, včetně popisu parametrů a návratových hodnot, je obsažen v příloze A, kde jsou popsány i důležité metody zděděné z bázové třídy (detailní popis všech metod z bázové třídy naleznete v dokumentaci API Ext JS [9]).

Událost aktualizace stavového objektu má jako parametr objekt s následujícími vlastnostmi:

- `lastModifiedBy` – id objektu, který provedl poslední aktualizaci stavového objektu
- `changedVariable` – název změněné vlastnosti stavového objektu
- `oldValue` – původní hodnota měněné vlastnosti
- `newValue` – nová hodnota měněné vlastnosti

6.2 API správce rozložení

Pro dosažení deterministočnosti jsem správce rozložení navrhl tak, že jeho objekt musí mít vždy název „`layoutManager`“ a objekt observeru, se kterým bude pracovat, musí mít název „`observer`“.

API správce rozložení pro komponenty je tvořeno metodami, umožňujícími provádět všechny požadované operace, a událostmi, na které mohou komponenty reagovat. Nejprve jsem navrhl metody pro provádění základních operací a metody pro speciální komponenty, které budou sloužit pro správu rozložení stránky a ukládání informací na server. Potom jsem navrhl metody pro lokalizaci umožňující kompatibilitu s Google Gadgets a na závěr jsem definoval textově nahrazované zkratky pro zjednodušení zápisu některých metod a texty, které budou nahrazeny pro podporu Google Gadgets (viz níže).

6.2.1 Základní metody

Základní metody jsou:

- `getComponent()` – získání reference na komponent
- `getComponentBody()` – získání reference na tělo komponentu
- `getComponentLocation()` – zjištění umístění komponentu na stránce
- `getComponentPreferences()` – získání reference na uživatelská nastavení komponentu
- `getComponentPrefsItem()` – získání položky uživatelských nastavení komponentu
- `setComponentPrefsItem()` – změna hodnoty položky uživatelských nastavení komponentu
- `refresh()` – obnovení stránky (kompletní, včetně všech komponentů a jejich rozložení)
- `registerEventListener()` – registrace funkce pro reakci na událost do observeru
- `unregisterEventListener()` – odregistrace funkce pro reakci na událost z observeru

Každý komponent je tvořen objektem třídy `Ext.Panel` (nebo třídy, která z této třídy dědí), a bude tedy možno využít většinu metod z API Ext JS. Bude však strukturální rozdíl, mezi komponenty, které mají formulář s uživatelskými nastaveními, a komponenty, které jej nemají. Komponenty bez tohoto formuláře budou mít svoje tělo uloženo přímo v těle panelu tvořícího komponent. Komponenty s formulářem budou mít ve svém panelu vnořeny další 3 panely: formulář pro nastavení, oddělovací panel a panel, v jehož těle bude umístěno tělo komponentu. Pro konzistentní přístup k tělu komponentu je tedy třeba separátní metoda.

Každý komponent bude moci přistupovat ke svým nastavením, a to buď k jednotlivým položkám, nebo k poli se všemi položkami. Nastavení bude možné měnit pomocí k tomu určené metody, která aktualizuje také formulář s nastavením a vyvolá událost změny nastavení komponentu.

Při uzavření komponentu budou zrušeny všechny jeho součásti kromě funkcí, které budou registrovány v observeru jako reakce na události. Pokud komponent v reakci na událost mění svůj obsah, nebo provádí jiné operace se součástmi, které jsou při zavírání odstraněny, po zavření komponentu by při reakci na událost došlo k chybě. Správce rozložení bude mít k tomuto účelu metodu, která umožní registraci funkcí do observeru tak, aby byly při zavření komponentu automaticky odregistrovány. Pro konzistenci zde bude i metoda pro odregistraci. Každý komponent může využít tyto metody, nebo pracovat přímo s observerem a uzavření komponentu ošetřit jiným způsobem, což mu umožní např. reakci na vlastní zavření.

Detailní popisy těchto i níže uvedených metod, včetně popisů parametrů a návratových hodnot, jsou obsaženy v příloze A.

6.2.2 Metody pro speciální komponenty

Správce rozložení sestavuje stránku dle konfiguračního XML vytvořeného serverem. Pro přístup uživatele k možnostem nastavení stránky (volba zobrazených oblastí, počtů jejich

sloupců apod.) mohou být vytvořeny speciální komponenty. Takový komponent může rovněž ukládat informace o aktuálním rozložení na server (zobrazené oblasti, rozměry oblastí, pozice komponentů apod.). Aby tento komponent mohl snadno přistupovat ke všem potřebným informacím, správce rozložení bude mít následující metody pro získání reference na příslušné datové struktury:

- `getAreasInfo()` – získání reference na objekt s informacemi o oblastech, ze kterých se stránka skládá
- `getComponentPositions()` – získání reference na objekt s informacemi o umístění všech komponentů
- `getComponentPrefsPresence()` – získání reference na pole s informacemi o tom, které komponenty mají uživatelská nastavení
- `getComponentPrefs()` – získání reference na pole s uživatelskými nastaveními všech komponentů

6.2.3 Metody pro lokalizaci

Google Gadgets je možné lokalizovat pomocí lokalizovatelných textů umístěných v externích souborech. Pro dosažení kompatibility musí správce rozložení disponovat obdobným principem. Lokalizované texty a lokalizační informace tedy budou umístěny ve stejných elementech jako u Google Gadgets, ale tyto elementy budou všechny umístěny v konfiguračním XML (tedy nikoliv v externích souborech) a element `Locale` bude mít jiný význam – bude sloužit k určení lokalizace komponentu (u iGoogle je vše, včetně zpracování textů, prováděno na serveru). Z důvodů konzistence bude API pro získání těchto informací skripty v komponentech rovněž jiné a pro kompatibilitu s Google Gadgets bude nutno vytvořit obalující třídy. Tato součást API pro komponenty je tvořena následujícími metodami:

- `getComponentLocalizedText()` – získání lokalizovaného textu pro komponent
- `getComponentLocales()` – získání lokalizačních informací pro komponent

6.2.4 Textově nahrazované metody

Pro zjednodušení zápisu některých metod jsem navrhl následující zkratky, které budou při analýze konfiguračního XML textově nahrazeny:

- `layoutManager.getMyComponent()`
  - bude nahrazeno za `layoutManager.getComponent("__MY_ID__")`
- `layoutManager.getMyComponentBody()`
  - bude nahrazeno za `layoutManager.getComponentBody("__MY_ID__")`
- `layoutManager.getMyComponentLocation()`
  - bude nahrazeno za `layoutManager.getComponentLocation("__MY_ID__")`

Kromě uvedených metod budou nahrazeny také zástupné symboly. Zástupný symbol "`__MY_ID__`" bude v těle komponentu nahrazen za textové id komponentu bez uvozovek, zástupný symbol "`__COMPONENT_ID__`" za id komponentu s uvozovkami. Druhá z variant je kompatibilní s Google Gadgets.

Dalšími nahrazovanými symboly jsou zástupné symboly za proměnné uživatelského nastavení. Ty budou při analýze konfiguračního XML nahrazeny za jejich aktuální hodnoty v titulku i v těle komponentu. Zástupné symboly za lokalizované texty budou nahrazeny v titulku, těle, zobrazovaných názvech proměnných uživatelského nastavení a zobrazovaných hodnotách výčtů.

Protože Google Gadgets jsou určeny do systému využívajícího vložené rámce, jejich API předpokládá, že v těle metody je možno zjistit identifikátor komponentu (každý vložený rámec má v prohlížeči jiný globální objekt). V mnou vytvářeném systému však nejsou využity vložené rámce, a identifikátor tedy zjistit nelze. Zavedení vložených rámců by bylo velmi nevýhodné pro nově vytvářené komponenty a využití vložených rámců pouze pro komponenty s Gadgety by bylo značně komplikované a příliš by zpomalovalo sestavení stránky. Pro dosažení alespoň částečné kompatibility je možno textově nahradit volání některých metod tak, aby měly jako první parametr identifikátor komponentu. Pokud Gadget nevolá tyto metody v dynamicky generovaném kódu, nebo nevyužívá jiné techniky, při kterých není nahrazování možné, lze jej upravit tak, že všechny skripty jsou z externích souborů vloženy přímo do příslušných elementů v jeho obsahu. Textová náhrada potom umožní běh takového Gadgetu s obalujícími třídami, které zapouzdří API správce rozložení do upraveného API Google Gadgets.

Protože některé z nahrazovaných metod mají volitelné parametry, je nutno rozlišovat variantu volání metody bez parametrů, kde se nový parametr vloží do prázdných závorek za identifikátorem metody, a s parametry, kde se nový parametr vloží před seznam uvedených (za otevírací závorku) a oddělí čárkou. Pokud bude tato vlastnost správce rozložení zapnuta, budou při analýze konfiguračního XML textově nahrazeny následující volání metod:

- `new gadgets.Prefs()`
  - bude nahrazeno za `new gadgets.Prefs(__MODULE_ID__)`
- `gadgets.window.adjustHeight()`
  - bude nahrazeno za `gadgets.window.adjustHeight(__MODULE_ID__)`
- `gadgets.window.adjustHeight(`
  - bude nahrazeno za `gadgets.window.adjustHeight(__MODULE_ID__,`
- `gadgets.window.getViewportDimensions()`
  - bude nahrazeno `gadgets.window.getViewportDimensions(__MODULE_ID__)`
- `gadgets.window.setTitle(`
  - bude nahrazeno za `gadgets.window.setTitle(__MODULE_ID__,`

Obdobně budou nahrazeny i varianty ze staršího (legacy) API Google Gadgets.

6.2.5 Události

Správce rozložení bude generovat události observeru, na které mohou komponenty reagovat. Každá událost bude mít jako parametr objekt s detailními informacemi o události. Události budou generovány v následujících situacích:

- dokončení inicializace stránky (registrace funkce pro reakci na tuto událost do observeru nahradí registraci metodou `Ext.onReady()`)
- skrytí těla oblasti (událost generována před skrytím i po skrytí)
- zobrazení skrytého těla oblasti (událost generována před zobrazením i po zobrazení)
- zavření oblasti
- změna velikosti oblasti
- přesun komponentu
- zavření komponentu
- změna uživatelských nastavení komponentu

Všechny události změn v rozložení stránky a jejich parametry jsem navrhl tak, aby byly konzistentní a bylo možné je v komponentech jednoduše zpracovat. Skrytí a zobrazení těla oblasti budou generovat dvě události, aby komponenty mohly na dobu provádění těchto operací přerušit manipulaci se svým obsahem, případně deaktivovat některé svoje součásti. Názvy konkrétních událostí, jejich detailní popisy a parametry jsou obsaženy v příloze A.

Kapitola 7

Návrh správce rozložení

7.1 API pro inicializaci

Správce rozložení bude mít kromě API pro komponenty také API pro inicializaci samotného správce rozložení, načtení konfiguračního XML a zobrazení stránky. Toto API se bude skládat z veřejných vlastností objektu správce rozložení pro nastavení jeho chování (adresa, ze které se bude načítat konfigurační XML, podpora Google Gadgets apod.) a z metody pro načtení stránky, která bude volána po jejich nastavení a inicializaci Ext JS a dalších knihoven a produktů. Detailní popis tohoto API je obsažen v příloze A.

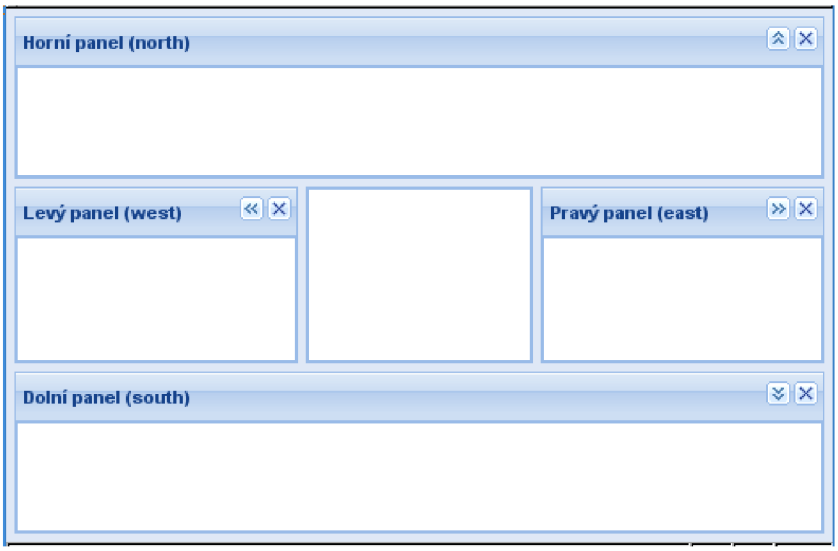
Inicializace správce rozložení bude provedena skriptem, který bude připojen k indexové stránce webu nebo v ní bude přímo obsažen. Generování tohoto skriptu serverem lze využít k realizaci některých pokročilejších funkcí, jako např. skrytí indikátoru průběhu načítání v těle stránky při jejím druhém a dalších načteních.

7.2 Struktura stránky

Stránka bude vytvořena pomocí objektu `Ext.Viewport` z knihovny Ext JS, který slouží k sestavení a zobrazení stránky. Její rozložení bude realizováno pomocí rozložení nazvaného „border layout“, protože toto přesně odpovídá zadaným požadavkům. Staticky naprogramovaná ukázka tohoto rozložení je vyobrazena na obrázku 7.1.

Pro přesouvání komponentů v rámci oblasti i mezi nimi využiji rozšiřujících tříd pro tvorbu rozložení stránky se sloupci z příkladu „Portal“, který je součástí Ext JS a který daný problém řeší. Tělo každé oblasti bude tvořeno objektem třídy `Ext.ux.Portal` (dále portál) z tohoto příkladu.

Každý komponent bude objektem třídy `Ext.ux.Portlet`, který dědí ze třídy `Ext.Panel` a mění výchozí vlastnosti objektů této třídy. Pokud komponent bude mít formulář s nastavením, budou v tomto panelu umístěny tři vnořené panely. Formulář s nastavením bude tvořen objektem třídy `Ext.form.FormPanel` (`Ext.Panel` rozšířený o metody pro práci s formuláři, který má v těle s formulářovým rozložením „FormLayout“ umístěn formulář) a pod ním bude umístěn oddělovací panel (`Ext.Panel` s oddělovačem v těle), který bude skrýván současně s formulářem. Pod oddělovacím panelem bude panel s tělem komponentu.



Obrázek 7.1: Ukázka rozložení „border layout“

7.3 Analýza konfiguračního XML a sestavení stránky

Konfigurační XML bude načteno pomocí metod třídy `Ext.Ajax`, která abstrahuje implementačně závislá rozhraní objektu `XMLHttpRequest` v jednotlivých prohlížečích. V průběhu načítání konfiguračního XML bude animován indikátor průběhu načítání stránky, který bude umístěn v elementu určeném k danému účelu (lze deaktivovat, více viz příloha A). Dialog zobrazený po načtení XML bude modální, což uživateli zabrání v manipulaci s právě inicializovanou stránkou.

XML bude analyzováno pomocí standardních metod pro přístup k DOM načteného XML dokumentu. Při analýze budou vytvářeny níže popsané datové struktury s informacemi o stránce a komponentech a objekt s konfigurací pro vytvoření objektu třídy `Ext.Viewport`. Nejprve budou analyzovány informace o jednotlivých oblastech na stránce. Pro neuvedené informace se využijí výchozí hodnoty (viz příloha A). Následně se provede analýza jednotlivých komponentů a jejich umístění do datových struktur.

Komponenty mohou být v XML uvedeny v libovolném pořadí a pro efektivní zpracování je nutné je umisťovat do takových struktur přímo na pozice definované jejich parametry. Objekt třídy `Ext.Viewport` vyžaduje konzistentní a kompletní konfigurační informace, je tedy nutné, aby v každé oblasti byl nejméně jeden sloupec a v každém sloupci nejméně jeden komponent. Komponenty musejí být v jednotlivých sloupcích umístěny od nultého řádku a mezi řádky nesmí být mezera. Aby server nemusel daný problém řešit a aby v případě prázdného řádku nebylo nutné posouvat komponenty ve sloupci a řešení bylo konzistentní s řešením pro prázdné sloupce, budou po analýze XML chybějící pozice doplněny prázdnými výplňovými komponenty. Po sestavení stránky budou výplňové komponenty odstraněny a dojde pouze ke změnám v datové struktuře s pozicemi komponentů.

Po sestavení stránky budou také spuštěny skripty v komponentech, příslušným objektům budou přiřazeny funkce pro obsluhy událostí a bude vyvolána událost dokončení inicializace stránky. Návrh řešení těchto problémů je popsán níže.

7.3.1 Skripty v komponentech

Aby byl dynamicky načtený skript prohlížečem proveden, musí být vložen do DOM stránky pomocí metody `appendChild` nebo `replaceChild`. Jednotlivé komponenty budou do DOM stránky vloženy při vytváření objektu `Ext.Viewport`, který skripty tímto způsobem korektně nevkládá a ty tak nejsou spuštěny. Skripty je tedy nutné spustit programově až po sestavení stránky.

Jako nejjednodušší a nejefektivnější řešení tohoto problému jsem zvolil záměnu elementů se skripty za jejich kopie. Po sestavení stránky tedy dojde k procházení DOM dokumentu, při kterém bude každý element se skriptem nahrazen kopií. Při kopírování budou využity standardní metody pro manipulaci s DOM dokumentem a v případě prohlížeče Microsoft Internet Explorer i některé nestandardní vlastnosti tohoto prohlížeče (přístup k textovému obsahu elementů `script` a `style`). Pro zjednodušení zpracování budou u elementů `script` analyzovány a kopírovány pouze atributy `type` a `src`. Zbylé atributy budou ignorovány a zahozeny.

7.4 Události změn v rozložení stránky

Pokud uživatel provede nějakou změnu v rozložení stránky (změnu šířky oblasti, přesun komponentu apod.), je nutné vyvolat příslušnou událost observeru. K tomuto účelu budou využity události oblastí, portálů a jednotlivých komponentů. Každý z těchto objektů dědí ze třídy `Ext.util.Observable`, a je tedy možné zaregistrovat do něj další funkce na obsluhu událostí, které budou aktualizovat datové struktury správce rozložení a generovat příslušné události observeru.

7.5 Událost dokončení inicializace stránky

Protože událost dokončení inicializace stránky musí být vyvolána až po vykonání všech skriptů v komponentech, aby skripty mohly registrovat funkce pro její obsluhu, a skripty umístěné v externích souborech jsou v některých prohlížečích (např. Mozilla Firefox) prováděny v oddělených vláknech paralelně s dokončováním inicializace stránky, není možné tuto událost generovat přímo správcem rozložení.

Tento problém jsem vyřešil tak, že správce rozložení po dokončení inicializace stránky připojí k elementu `body` celé stránky element `script`, který bude obsahovat vazbu na skript pro vyvolání příslušné události umístěný v externím souboru. Protože skripty v externích souborech jsou prováděny v pořadí, ve kterém byly přidány do stránky, tento skript bude vždy proveden jako poslední z nich. Jeho vložení do stránky až po dokončení inicializace stránky zajistí, že bude vždy vykonán ve správnou dobu.

Dialogové okno zobrazené po dobu sestavování stránky bude skryto ve funkci, kterou správce rozložení zaregistruje pro reakci na tuto událost.

7.6 Datové struktury

Správce rozložení bude mít v observeru uloženy stavové objekty s datovými strukturami s informacemi o oblastech stránky, umístění a nastavení komponentů apod. Tyto datové struktury jsem navrhl tak, aby byly přehledné a manipulace s nimi byla jednoduchá. Pomocí příslušných metod správce rozložení budou některé z nich zpřístupněny komponentům

určeným k ukládání dat na server a k uživatelskému nastavení rozložení stránky. Tyto struktury budou obsahovat následující informace:

- přítomnost jednotlivých oblastí na stránce
- výchozí stav oblastí (zda byly oblasti při načtení stránky sbalené, jejich rozměry apod.)
- aktuální stav oblastí
- pozice komponentů (oblast, sloupec, řádek) – budou uloženy ve struktuře navržené tak, aby bylo možno rychle a efektivně zjistit jak pozici komponentu, tak id komponentu na dané pozici (pouhou indexací, bez nutnosti sekvenčního procházení polem)
- funkce, které si komponenty registrovaly do observeru
- lokalizační informace pro komponenty (země, jazyk)
- lokalizované texty pro komponenty
- které komponenty mají uživatelská nastavení
- uživatelská nastavení komponentů

Konkrétní názvy struktur a jejich detailní popis jsou obsaženy v příloze A.

7.7 Lokalizace

Pro lokalizaci správce rozložení využiji obdobný princip, jaký je využit v knihovně Ext JS. Lokalizovatelné texty tedy budou uloženy ve veřejných vlastnostech třídy a v případě potřeby předefinovány lokalizačními skripty.

7.8 Zavírání oblastí

Při zavírání oblasti budou nejprve uzavřeny všechny komponenty v dané oblasti a následně bude uzavřena celá oblast.

V případě, že je v oblasti komponent bez zavíracího tlačítka (definováno v konfiguračním XML), správce rozložení může tento komponent zavřít, nebo zobrazit chybové hlášení a operaci přerušit. Toto chování lze definovat příslušným elementem konfiguračního XML. Ve výchozím nastavení bude komponent uzavřen.

Kapitola 8

Implementace

8.1 Observer

Observer dědí z abstraktní bázové třídy `Ext.util.Observable`. K realizaci dědičnosti jsem využil stejný princip, jaký je využíván v knihovně Ext JS. Nejprve je tedy vytvořen konstruktor objektu s přidanými vlastnostmi a ten je následně rozšířen o vlastnosti a metody rodičovského konstruktoru metodou `Ext.extend`.

Metoda pro aktualizaci stavového objektu mimo provedení samotné aktualizace vygeneruje také událost změny stavového objektu, která bude mít jako parametr objekt s informacemi o tom, který komponent změnu provedl, jakou vlastnost stavového objektu měnil a původní i novou hodnotu této vlastnosti.

8.2 Správce rozložení

Správce rozložení jsem implementoval dle návrhu, zmíním zde pouze strukturu metody pro analýzu konfiguračního XML a sestavení stránky, implementaci některých dalších složitějších metod a vybrané problémy řešené ve fázi implementace. Detailnější informace lze zjistit z bohatě komentovaných zdrojových kódů.

Součásti správce rozložení plně převzaté z příkladu Portal (rozšíření `Ext.ux.Portal`, `Ext.ux.PortalColumn` a `Ext.ux.Portlet`) jsou umístěny v separátních souborech. V případě potřeby je možné využít verze těchto souborů, které jsou umístěny přímo v knihovně. Avšak vzhledem k tomu, že jsou tyto soubory v knihovně umístěny v příkladech a v budoucích verzích mohou být změněny či odstraněny, doporučuji využívat verze souborů z Ext JS 2.1 přiložené k projektu.

8.2.1 Metoda pro analýzu konfiguračního XML a sestavení stránky

Tato metoda nejprve zjistí, zda je v XML obsažen povinný element `layoutSettings`, bez kterého stránku nelze sestavit. Následně pomocí Ext JS zobrazí modální dialogové okno s indikátorem průběhu, který bude až do zavření tohoto okna animován. Potom jsou vytvořeny a inicializovány datové struktury pro informace o stránce a komponentech a je analyzován element určující chování při zavírání oblastí (viz výše).

Poté následuje analýza konfiguračního XML. Nejprve jsou v cyklu procházeny elementy s definicemi oblastí a získané informace jsou ukládány do příslušných objektů. Pokud v XML nebyly uvedeny informace pro vytvoření centrální oblasti, je tato oblast vytvořena s jedním sloupcem a výchozími hodnotami.

Po analýze informací o oblastech je obdobným způsobem prováděna analýza komponentů. Pokud panel nemá žádný titulek, vůbec se nezobrazí záhlaví panelu s nástroji (není-li využita konfigurační volba pro jeho vynucení). Této vlastnosti jsem využil pro možnost vytvoření komponentu bez záhlaví a veškeré řízení zobrazování záhlaví jsem vyřešil pomocí obsahu titulku.

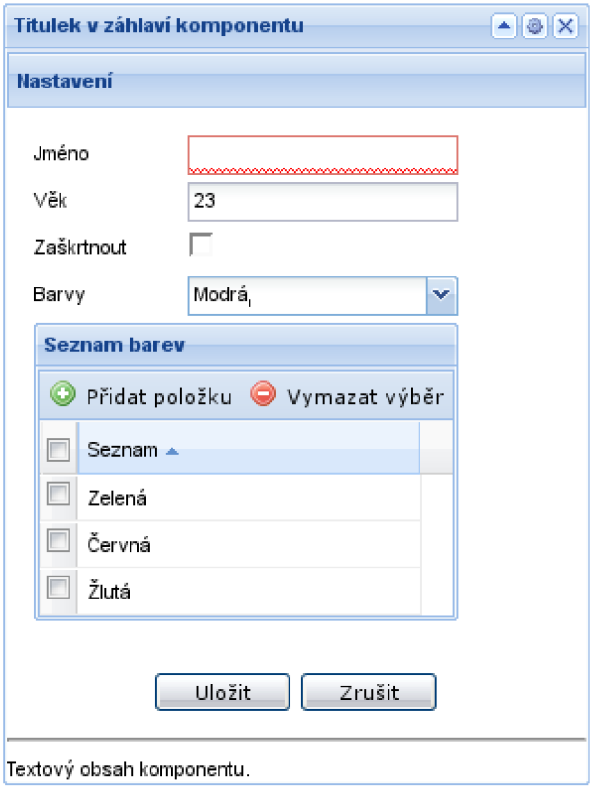
U titulku, obsahu, i dalších elementů s textovým obsahem bylo nutno vyřešit nestandardní API pro přístup k textovému obsahu v prohlížeči Microsoft Internet Explorer. Pokud se tedy obsah elementu nepodaří získat standardním způsobem a jedná se o některou z verzí Internet Exploreru, analyzátor se pokusí získat textový obsah pomocí tohoto nestandardního API.

Uživatelská nastavení jsou nejprve analyzována a následně je v případě potřeby sestaven formulář pro nastavení. Největším problémem při tvorbě tohoto formuláře bylo vytvoření formulářového pole pro datový typ list (seznam uživatelem poskytovaných hodnot). K tomuto účelu jsem využil editovatelnou dynamickou tabulku `Ext.grid.EditorGridPanel` s jedním sloupcem. Aby bylo možné hodnoty v tabulce přesouvat (vylepšení oproti iGoogle a podobným systémům), využil jsem implementaci vlastní dopadové oblasti dle [8] a vlastního výběrového modelu z [12]. Aby v seznamu nebyly prázdné položky, implementoval jsem rovněž automatické odstraňování prázdných položek před vložením nové položky, po editaci položky a před uložením nastavení. Odstraňování prázdných položek může být prováděno pouze v těchto situacích a musí být prováděno tak, aby nedošlo k selhání datového úložiště tabulky, ke kterému dojde např. při odstraňování prázdných položek v reakci na událost zrušení jejich výběru. Protože tabulka není běžné formulářové políčko a formulář je zde využit k jinému účelu, než k jakému je původně určen (uložení zde není spojeno s odesláním informací na server), musel jsem implementovat také vlastní metody pro validaci obsahu formuláře, zjištění, zda byl jeho obsah od posledního uložení změněn, a navrácení změn (resetování formuláře). Ukázku výsledného vzhledu komponentu s formulářem, ve kterém jsou umístěny všechny implementované typy formulářových polí (textový vstup, zaškrtávací políčko, výběr ze seznamu hodnot a seznam uživatelem poskytovaných hodnot), naleznete na obrázku 8.1.

Pro přehlednost mají být ve výchozím zobrazení po inicializaci stránky všechny formuláře skryté. Viewport však neumožňuje vytvoření skrytého panelu. Tento problém jsem vyřešil tak, že je při sestavování konfiguračního objektu pro Viewport vytvářen i seznam panelů pro skrytí. Panely z tohoto seznamu jsou potom skryty po sestavení stránky (vytvoření Viewportu). Obdobným způsobem jsou tabulkám v uživatelských nastaveních přiřazeny dopadové oblasti pro přesouvání řádků, které lze rovněž přiřadit pouze zobrazeným tabulkám.

Google Gadgets API umožňuje, aby měl komponent jiný obsah v různých pohledech (náhled, normální, celá obrazovka apod.), přičemž mezi jednotlivými pohledy lze dynamicky přepínat. Protože ve mnou vytvořeném systému existuje pouze jeden pohled a dynamické přepínání pohledů by bylo v rozporu s požadavky na systém, pohledy (views) jsem neimplementoval. Aby však nedocházelo k chybnému zobrazení, implementoval jsem přeskakování elementů s obsahem pro nevhodná zobrazení. Pokud se tedy v komponentu vyskytuje více elementů s obsahem, je využit první z nich, který není určen pro zobrazení v náhledu či na celou obrazovku.

Při analýze obsahu komponentu je v Internet Exploreru 6 nutné zjistit, zda se před prvním skriptem v komponentu (prvním elementem `script`) nachází alfanumerický znak. Tento znak může být uveden nejenom jako prostý text, ale i jako jako součást nějakého elementu či entity. Pokud se zde žádný takový znak nenachází, Internet Explorer by následně skript nenašel v DOM. Z tohoto důvodu je v případě nepřítomnosti tohoto znaku



Obrázek 8.1: Ukázka komponentu s formulářem pro nastavení

v Internet Exploreru na začátek obsahu komponentu doplněn ve formě znakové entity pro nezalomitelnou mezeru (` `).

U textově nahrazovaných volání metod, která mají dvě varianty (pro metody s parametry a bez parametrů), je nutno nejprve nahradit volání s parametry tak, že jsou vyhledávány řetězce, ve kterých za otevírací závorkou následuje jiný znak než uzavírací závorka a potom řetězce končící prázdnými závorkami. Opačné pořadí nahrazování by způsobilo chybnou náhradu již nahrazených řetězců.

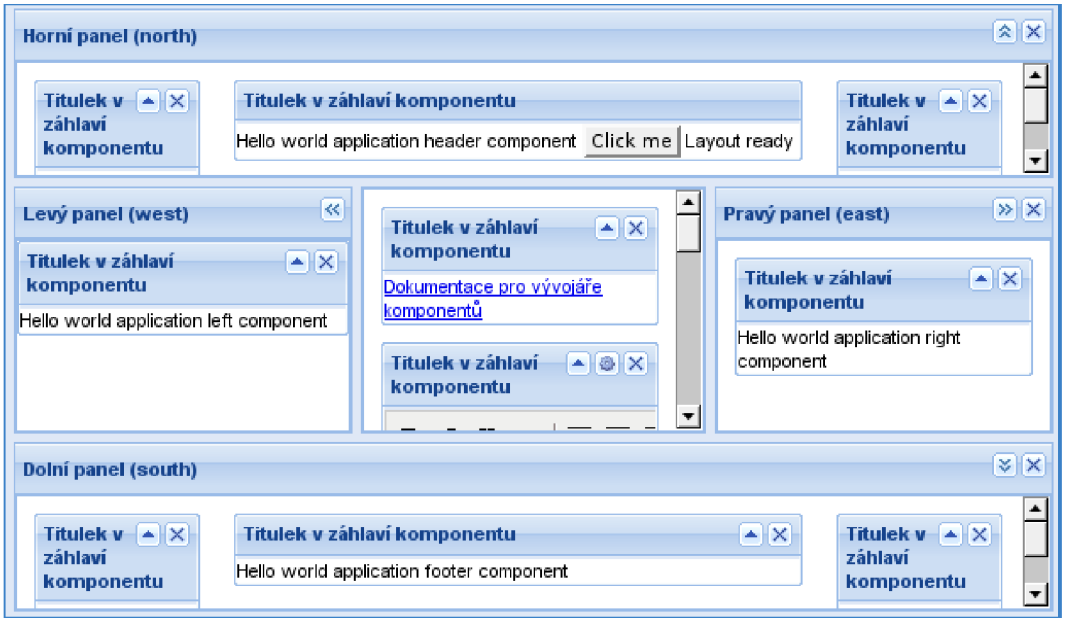
Při spouštění skriptů v komponentech pomocí jejich náhrady kopiemi elementů, které probíhá po sestavení stránky, je nutné řešit nestandardní API Internet Exploreru pro přístup k textovému obsahu tohoto elementu. Zatímco dle standardního postupu je pro textový obsah vytvořen textový uzel, který je připojen k uzlu se skriptem, v Internet Exploreru tato technika není funkční a jako náhradní řešení má v tomto prohlížeči uzel typu script vlastnost nazvanou „text“.

Po spuštění skriptů v komponentech jsou jednotlivým oblastem a portálům přiřazeny funkce pro obsluhu událostí, které aktualizují datové struktury s informacemi o stránce a komponentech a generují příslušné události. U události přesunu komponentu jsem řešil problém, kde při přesunu do prázdného sloupce není přepočítáno rozložení sloupce a komponentu tak zůstanou rozměry z předchozího umístění. Vzhledem k tomu, že událost upuštění komponentu myší (drop) je vyvolána ještě před dokončením samotného přesunu a po jeho dokončení již žádná událost vyvolána není, není možné vynutit přepočítání rozložení přímo v obsluze události. Tento problém jsem vyřešil zpožděným voláním funkce pro přepočítání rozložení sloupce po 100ms. Se zpožděním (200ms) je vyvolána i událost observeru,

na kterou může reagovat samotný komponent.

Před svým ukončením metoda pro analýzu konfiguračního XML a sestavení stránky skryje informace zobrazené v těle stránky při načítání, zaregistruje funkci pro obsluhu události dokončení inicializace stránky, ve které skryje dialogové okno zobrazené při sestavování stránky, a připojí k tělu stránky element se skriptem umístěným v externím souboru vyvolávajícím tuto událost.

Ukázka výsledného vzhledu dynamicky sestavené stránky je na obrázku 8.2.



Obrázek 8.2: Ukázka vzhledu dynamicky sestavené stránky

8.2.2 Metody tvořící API pro komponenty

U metod tvořících API pro komponenty je třeba zmínit, že jimi vracené reference nelze využít k modifikaci hodnot v příslušných datových strukturách. Modifikace musejí být vždy prováděny pomocí příslušných metod. Jednou z nich je metoda pro změnu hodnoty položky uživatelských nastavení komponentu, která aktualizuje příslušné datové struktury, hodnotu v políčku ve formuláři pro uživatele a vyvolá událost změny uživatelských nastavení. U datového typu bool rovněž zajišťuje převod na tento datový typ z řetězcové či číselné hodnoty.

Metoda pro obnovení stránky pouze volá metodu `window.location.reload()` určenou k danému účelu. Vzhledem ke složitosti dynamických manipulací s již vytvořenou stránkou je toto nejrychlejší a nejefektivnější způsob aktualizace obsahu stránky.

Metoda pro registraci posluchače událostí (funkcí pro reakci na událost), kromě samotné registrace do observeru, ukládá posluchače také do seznamu, který je procházen při zavírání komponentu, kdy jsou zaregistrované funkce odregistrovány. Metoda pro odregistraci tento seznam neprochází, protože vyhledání funkce v seznamu a její odstranění by bylo náročnější a pomalejší než případné zbytečné volání funkce pro odregistraci při zavírání, které v případě již odregistrované funkce neprovede žádnou operaci.

Metoda pro získání informací o lokalizaci pro komponent jako jediná při nedostupnosti požadovaných informací nevrací null, ale anglickou lokalizaci, čímž předchází chybovým

stavům u Gadgetů chybně upravených serverem.

8.3 Obalující třídy pro kompatibilitu s Google Gadgets

Mým úkolem bylo implementovat z API Google Gadgets pouze základní třídy, které jsou spojeny se zobrazením. Implementace tříd pro vstup/výstup a komunikaci bude provedena jiným členem projektového týmu, stejně jako implementace tříd pro pokročilé zobrazení (skins, Tab, flash apod.).

Tyto třídy ve většině případů pouze obalují API správce rozložení a knihovny Ext JS. Pro jejich provoz je nutné, aby měl správce rozložení zapnutou podporu Google Gadgets.

Ke každé třídě jsem vytvořil i alias pro zastaralou (legacy) variantu API Google Gadgets. Statická třída `gadgets.window` v zastaralé variantě nebyla a místo ní zde byly dvě samostatné funkce k danému účelu. Pro každou z odpovídajících metod třídy `gadgets.window` jsem tedy vytvořil příslušný alias. Obdobně jsem vytvořil i aliasy pro statickou třídu `gadgets.util` a jednoduché funkce, které byly pouze v zastaralé sadě funkcí nahrazené touto třídou (`_trim()`, `_uc()`, `_min()`, `_max()`, `_hesc()`, `_toggle()` a `_args()`). Následuje popis implementovaných tříd.

8.3.1 MiniMessage

Třída `gadgets.MiniMessage` slouží ke zobrazování dialogových oken se zprávami. Ty mohou být zobrazovány v konkrétním elementu nebo nad celou stránkou. Jsou definovány tři druhy oken se zprávami:

- dismissible – s tlačítkem „OK“ a zavíracím křížkem
- static – bez tlačítek, nutno zavřít programově
- timer – časované, zmizí po uplynutí časového limitu

Všechny 3 druhy je možné zavřít i programově pomocí příslušné metody. Protože metody pro vytváření oken vracejí HTML element s tělem okna (v iGoogle s elementem nahrazujícím okno), je nutné uchovávat seznam referencí na okna se zprávami, aby bylo možné okna uzavírat. Při programovém zavírání je potom dle HTML elementu s obsahem nalezena reference na okno a to je následně uzavřeno. Okna uzavřená uživatelem, nebo vypršením času, jsou ze seznamu odebírána při vytváření nových, přičemž zavřené okno je rozpoznáno dle své nepřítomnosti v DOM stránky.

Pro implementaci oken se zprávami jsem využil objekty třídy `Ext.Window`, která slouží k vytváření univerzálních dialogových oken, aby mohlo být otevřeno více oken současně (v Ext JS je také třída `Ext.MessageBox` určená přímo ke zobrazování dialogových oken se zprávami, ale ta může mít otevřeno vždy jen jedno okno). Pro lokalizaci jsem využil texty ze třídy `Ext.MessageBox`. Aby nedocházelo k zablokování uživatele větším množstvím oken od různých komponentů, implementoval jsem všechna tato okna jako nemodální.

U časovaného dialogového okna jsem pro zobrazování odpočtu uživateli využil zablokované (zašedlé, neaktivní) tlačítko a pro samotný odpočet rekurzivní funkci, která se rekurzivně volá se zpožděním 1s a aktualizuje počítadlo na tlačítku.

8.3.2 Prefs

Konstruktor třídy `gadgets.Prefs` má jako volitelný parametr id komponentu, ale narozdíl od systému iGoogle je zde tento parametr doplněn správcem rozložení a lze jej považovat za povinný, protože jeho nepřítomnost by vedla k chybovým stavům.

Metody této třídy pouze obalují příslušné metody správce rozložení a provádějí případné konverze datových typů.

8.3.3 util

Statická třída `gadgets.util` obsahuje pomocné funkce. Funkce pro práci s řetězci jsem implementoval dle specifikace API Google Gadgets. Funkce pro registraci funkce prováděné po inicializaci Gadgetu pouze obaluje volání příslušné metody observeru. Funkci pro zjištění podporovaných součástí API (`hasFeature`) jsem implementoval dle implementovaných tříd a jejich funkcionality a funkci pro zjištění parametrů vlastnosti (funkce) API obsažených v konfiguračním XML (`getFeatureParameters`) jsem pro její velmi malé využití a nedostatečnou specifikaci implementoval tak, že vždy vrací `null` (dle specifikace odpovídá situaci, kdy vlastnost nemá žádné parametry).

Obdobně jsem implementoval i samostatné funkce pro zastaralou (legacy) variantu API Google Gadgets, které byly v sadě funkcí nahrazené třídou `gadgets.util`. Jedná se převážně o jednoduché pomocné funkce pro práci s řetězci, určení minima či maxima ze 2 čísel apod. Funkci `_toggle()`, která slouží ke zobrazování a skrývání elementů, jsem implementoval dle [3] tak, že podporuje pouze práci s blokovými elementy. Pro funkci `_args()` jsem zvolil zjednodušenou implementaci, která odpovídá specifikaci z [18], ale ve výstupním poli je navíc metoda `remove()` doplněná knihovnou Ext JS.

8.3.4 window

Statická třída `gadgets.window` obsahuje metody pro nastavení titulku komponentu, výšky komponentu a zjištění jeho vnitřních rozměrů. Všechny tyto operace vyžadují znalost id komponentu, které bez vložených rámců nelze v těle metody zjistit. Z tohoto důvodu jsem těmto metodám přidal parametr s id komponentu a všechna volání metod z této třídy jsou při analýze XML textově nahrazena voláními s daným parametrem.

Všechny metody této třídy pouze obalují příslušné metody správce rozložení a knihovny Ext JS (manipulace s panelem komponentu).

8.4 Pomocné soubory pro demonstraci činnosti

Aby bylo možno systém samostatně testovat a demonstrovat jeho funkčnost, v PHP jsem vytvořil jednoduchý skript pro generování indexové stránky, který určí požadovaný jazyk dle požadavků prohlížeče a parametrů v URL, sestaví stránku a vloží do ní vazby na příslušné lokalizační soubory. Vytvořil jsem také skript, který správci rozložení zasílá konfigurační XML uvedené přímo v těle tohoto skriptu.

Pro demonstraci činnosti zahraničním členům týmu jsem vytvořil lokalizační soubory i pro skript, který provádí generování indexové stránky.

8.5 Editor TinyMCE v komponentech

Mým úkolem bylo rovněž zajistit možnost provozovat editor TinyMCE [33] v komponentech. Aby bylo možno tento editor provozovat společně s knihovnou Ext JS, je nutno využít rozšíření knihovny určené k tomuto účelu (třída `Ext.ux.TinyMCE`) [11]. Aby nedošlo ke konfliktu a zhroucení knihovny i editoru, inicializace jádra TinyMCE musí být provedena ještě před dokončením inicializace Ext JS a voláním metody `Ext.onReady`. Do skriptu pro generování indexové stránky jsem tedy přidal vazby na příslušné soubory a do skriptu pro inicializaci stránky jsem přidal inicializaci editoru TinyMCE.

Aby nedocházelo k inicializaci TinyMCE i v případech, kdy tento editor není využit v žádném komponentu, bylo by možno systém optimalizovat tak, že by server provedl vyhodnocení jeho potřeby a dle výsledku generoval indexovou stránku a skript pro inicializaci.

Pokud je editor aktivní, nemůže být manipulováno s elementy DOM, ve kterých se editor nachází. K tomuto účelu bude nutné, aby každý komponent, který editor využívá, reagoval na příslušné události svého panelu a správce rozložení a na okamžiky manipulace editor deaktivoval. Seznam příslušných událostí, včetně příkladu zprovoznění tohoto editoru, je obsažen v příloze C.

Kapitola 9

Testování a ladění

Vytvořený správce rozložení a observer jsem nejprve otestoval s jednoduchými komponenty, přičemž jsem vyzkoušel všechny konfigurační volby a jejich potenciálně problematické kombinace. Otestoval jsem také API pro skripty v komponentech, přičemž jsem mimo modifikací konfiguračního XML využil i nástroj Firebug [21], který je rozšířením pro webový prohlížeč Mozilla Firefox. Tento nástroj obsahuje i JavaScriptovou konzoli, pomocí které bylo možno v přijatelném čase velmi detailně otestovat všechny součástí API.

Po první fázi testování a ladění jsem projekt zpřístupnil ostatním členům projektového týmu, kteří při tvorbě komponentů otestovali chování systému s komplexními spolupracujícími komponenty. V této fázi jsem odladil další drobné chyby a problémy vznikající především při práci složitých spolupracujících komponentů v některých prohlížečích.

Výsledný otestovaný a odladěný systém pracoval bez problémů i s větším množstvím velmi komplexních spolupracujících komponentů.

9.1 Testování a ladění v různých prohlížečích

Správce rozložení byl otestován a odladěn v následujících prohlížečích:

- Iceweasel 2.0.0.12 (Mozilla Firefox upravený pro Debian GNU/Linux)
- Mozilla Firefox 3.0 (ve Windows XP)
- Microsoft Internet Explorer 6 (ve Windows XP bez SP a aktualizací)
- Microsoft Internet Explorer 7 (Windows XP SP3, včetně všech aktualizací)
- Opera 9.51 (Linux)
- Safari 3.1.2 (Windows XP SP3)

V IE7 není zcela správné zobrazení formuláře pro nastavení, pokud je komponent příliš úzký (dochází k nevhodnému odřádkování mezi popiskem a samotným políčkem a odsazení tohoto políčka). Tento problém nemá vliv na funkčnost a vzhledem ke stálému vývoji IE7 je možné, že některá z budoucích aktualizací tento problém vyřeší. V případě potřeby by bylo možno správce rozložení upravit a uplatnit řešení pevnou šířkou formuláře jako u IE6, kde se jedná o jediné možné řešení.

V prohlížeči Opera nemá TinyMCE kompletní podporu, pravděpodobně bude nutné důkladně testovat všechny komponenty, které jej využívají. V Konqueroru 3.5.5 projekt není funkční pro částečnou nefunkčnost ExtJS a plnou nefunkčnost TinyMCE.

Kapitola 10

Závěr

Detailně jsem prostudoval vybrané existující systémy s webovými uživatelskými rozhraními složenými z komponentů a vytvořil jsem pokročilého správce rozložení stránky pro tvorbu uživatelského rozhraní složeného z komponentů pomocí knihovny Ext JS. Správce rozložení poskytuje mnohem větší flexibilitu než systémy současně využívané ke stejnému účelu a pracuje plně na klientské straně, čímž umožňuje zjednodušení serverové části výsledného systému. U jednotlivých oblastí stránky i komponentů je možno definovat velké množství parametrů.

Pro konfiguraci správce rozložení a informace pro sestavení stránky, včetně komponentů, jsem navrhl nový formát XML, který umožňuje současné zaslání všech potřebných informací pro sestavení stránky ze serveru do správce rozložení. Systém tak efektivněji nakládá se šířkou pásma a počtem spojení se serverem.

Pro komponenty na stránce jsem vytvořil komunikační mechanismus a navrhl nové API, které má mnohem vyšší funkcionalitu a možnosti komponentů, než API jiných existujících systémů (iGoogle, Netvibes, My Yahoo! apod.). Umožňuje i dosažení částečné kompatibility s existujícími systémy, přičemž pro Google Gadgets jsem implementoval i požadované třídy potřebné pro dosažení této kompatibility.

Komponenty je možno vytvářet tak, aby byly nezávislé na správci rozložení, ale lze využít i realizované API a vytvořit vzájemně komunikující komponenty, které budou reagovat na změny v rozložení stránky a v nastavení sebe i jiných komponentů.

Vytvořený správce rozložení i komunikační mechanismus splňují všechny požadavky evropského projektu KiWi a prošly i akceptačními testy členů projektového týmu vytvářejících server a komponenty uživatelského rozhraní. Nezávislost na konkrétní implementaci serveru však umožňuje i jejich využití v jiných systémech, jako základní součást pro tvorbu uživatelského rozhraní složeného z komponentů.

Do budoucna je možno systém rozšířit o další součásti pro dosažení kompatibility s jinými systémy než iGoogle (např. vytvořit soubory prostředí pro Netvibes). Je také možno implementovat systém „vložkování“ těl komponentů vloženými rámci (elementy `iframe`) a manipulace s nimi, a eliminovat tím nutnost úprav komponentů ze systémů využívajících tuto strategii.

Literatura

[1] Achour, M.; Betz, F.; Dovgal, A.; aj.: *PHP Manual*. The PHP Documentation Group, 2008, [Online; navštíveno 3.8.2008].
URL http://www.php.net/manual/en/

[2] History of Ajax Components. 2008, [Online; navštíveno 3.8.2008].
URL http://www.ajaxwith.com/History-of-Ajax-Components-.html

[3] Shindig – an Apache incubator project for OpenSocial and gadgets. 2008, [Online; navštíveno 3.8.2008].
URL http://incubator.apache.org/shindig/

[4] Berners-Lee, T.; aj.: W3C - World Wide Web Consortium. 2008, [Online; navštíveno 3.8.2008].
URL http://www.w3.org/

[5] Darie, C.; Brinzarea, B.; Cherecheş-Toşa, F.; aj.: *AJAX a PHP - tvoříme interaktivní webové aplikace profesionálně*. Brno: Zoner Press, 2006, ISBN 80-86815-47-1, překlad: Skřivánek, R.

[6] Eich, B.; Clary, B.: JavaScript Language Resources. Leden 2003, [Online; navštíveno 3.8.2008].
URL http://developer.mozilla.org/en/docs/JavaScript_Language_Resources

[7] Everton: Personalised AJAX Homepages: Roundup and Review. In *Connected Internet*, Connected Internet, Prosinec 2005, [Online; navštíveno 3.8.2008].
URL http://www.connectedinternet.co.uk/2005/12/17/personalised-ajax-homepages-roundup-and-review/

[8] Grid Drag/Drop. 2007, [Online; navštíveno 3.8.2008].
URL http://extjs.com/forum/showthread.php?t=3423

[9] Ext 2.1 – API Documentation. 2008, [Online; navštíveno 3.8.2008].
URL http://extjs.com/deploy/dev/docs/

[10] Ext JS: Cross-Browser Rich Internet Application Framework. 2008, [Online; navštíveno 3.8.2008].
URL http://extjs.com/products/extjs/

[11] Ext.ux.TinyMCE – TinyMCE form field (v 0.2). 2008, [Online; navštíveno 3.8.2008].
URL http://extjs.com/forum/showthread.php?t=24787

[12] GridPanel – CheckboxSelectionModel – Multiple selections – Drag&Drop [SOLUTION]. 2008, [Online; navštíveno 3.8.2008].
URL http://extjs.com/forum/showthread.php?t=29797

[13] Tutorials. 2008, [Online; navštíveno 3.8.2008].
URL http://extjs.com/learn/Tutorials

[14] Fielding, R. T.; aj.: Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, Červen 1999, [Online; navštíveno 3.8.2008].
URL http://tools.ietf.org/html/rfc2616

[15] Flanagan, D.: *JavaScript Kompletní průvodce 2. aktualizované vydání*. Praha: Computer Press, 2002, ISBN 80-7226-626-8, o'Reilly, překlad: Voráček, K., Krásenský, D.

[16] Garrett, J. J.: Ajax: A New Approach to Web Applications. Únor 2005, [Online; navštíveno 3.8.2008].
URL http://www.adaptivepath.com/ideas/essays/archives/000385.php

[17] Gilmore, W. J.: *Velká kniha PHP5 & MySQL*. Brno: Zoner Press, 2005, ISBN 80-86815-20-X, překlad: Pokorný J.

[18] Gadgets. 2008, [Online; navštíveno 3.8.2008].
URL http://code.google.com/apis/gadgets/

[19] gadgets.* API Developer's Guide. 2008, [Online; navštíveno 3.8.2008].
URL http://code.google.com/apis/gadgets/docs/dev_guide.html

[20] iGoogle. 2008, [Online; navštíveno 3.8.2008].
URL http://www.google.com/ig

[21] Hewitt, J.; aj.: Firebug web development evolved. 2008, [Online; navštíveno 3.8.2008].
URL http://getfirebug.com/

[22] Janovský, D.: Jak psát web. 2008, [Online; navštíveno 3.8.2008].
URL http://www.jakpsatweb.cz/

[23] Kennedy, N.: Narrowcasting with customized homepages. Září 2006, [Online; navštíveno 3.8.2008].
URL http://www.niallkennedy.com/blog/2006/09/custom-personalized-homepage.html

[24] Krim, T.; aj.: Netvibes. 2008, [Online; navštíveno 3.8.2008].
URL http://www.netvibes.com/

[25] Krim, T.; aj.: Netvibes Developers Network. 2008, [Online; navštíveno 3.8.2008].
URL http://dev.netvibes.com/doc/

[26] Kácha, P.: Seriál HTML. 2004, [Online; navštíveno 3.8.2008].
URL http://www.linuxsoft.cz/html/

[27] Lerdorf, R.; Gutmans, A.; Suraski, Z.; aj.: php. 2008, [Online; navštíveno 3.8.2008].
URL http://www.php.net/

[28] MacManus, R.: Ajax homepages market review. In *ZDNet*, CNET Networks, Inc., a CBS Company, Únor 2006, [Online; navštíveno 3.8.2008].
URL http://blogs.zdnet.com/web2explorer/?p=127

[29] Mikle, P.: *XCSS – CSS1, CSS2, CSS2.1 – úplná přesná referenční příručka*. Brno: Zoner Press, 2004, ISBN 80-86815-13-7.

[30] Mikle, P.: *XDHTML – HTML, XHTML, DHTML – úplná přesná referenční příručka*. Brno: Zoner Press, 2004, ISBN 80-86815-01-3.

[31] Moravec, V.: Seriál CSS. 2006, [Online; navštíveno 3.8.2008].
URL http://www.linuxsoft.cz/article_list.php?id_kategory=201

[32] Mowery, J.: 14 Personalized Homepages Compared, Feature by Feature. In *Mashable Social networking news*, Mashable!, Červen 2007, [Online; navštíveno 3.8.2008].
URL http://mashable.com/2007/06/29/personalized-homepages/

[33] TinyMCE. 2008, [Online; navštíveno 3.8.2008].
URL http://tinymce.moxiecode.com/

[34] Musciano, C.; Kennedy, B.: *HTML a XHTML Kompletní průvodce*. Praha: Computer Press, 2000, ISBN 80-7226-407-9, o'Reilly, překlad: Krásenský, D.

[35] Schaffert, S.: KiWi – Knowledge in a Wiki. 2008, [Online; navštíveno 3.8.2008].
URL http://wiki.kiwi-project.eu/

[36] Swartz, A.: Raw Thought: A Brief History of Ajax. Prosinec 2005, [Online; navštíveno 3.8.2008].
URL http://www.aaronsw.com/weblog/ajaxhistory

[37] Ajax (programming). In *Wikipedia – The Free Encyclopedia*, nadace WIKIMEDIA, 2008, [Online; navštíveno 3.8.2008].
URL http://en.wikipedia.org/wiki/AJAX

[38] Ext (javascript library). In *Wikipedia – The Free Encyclopedia*, nadace WIKIMEDIA, 2008, [Online; navštíveno 3.8.2008].
URL http://en.wikipedia.org/wiki/ExtJS

[39] Hypertext Transfer Protocol. In *Wikipedia – The Free Encyclopedia*, nadace WIKIMEDIA, 2008, [Online; navštíveno 3.8.2008].
URL http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

[40] iGoogle. In *Wikipedia – The Free Encyclopedia*, nadace WIKIMEDIA, 2008, [Online; navštíveno 3.8.2008].
URL http://en.wikipedia.org/wiki/IGoogle

[41] Netvibes. In *Wikipedia – The Free Encyclopedia*, nadace WIKIMEDIA, 2008, [Online; navštíveno 3.8.2008].
URL http://en.wikipedia.org/wiki/Netvibes

[42] Yahoo! In *Wikipedia – The Free Encyclopedia*, nadace WIKIMEDIA, 2008, [Online; navštíveno 3.8.2008].
URL http://en.wikipedia.org/wiki/Yahoo

[43] Document Object Model. In *Wikipedie – Otevřená encyklopedie*, nadace WIKIMEDIA, 2008, [Online; navštíveno 3.8.2008].
URL http://cs.wikipedia.org/wiki/DOM

[44] Hypertext Transfer Protocol. In *Wikipedie – Otevřená encyklopedie*, nadace WIKIMEDIA, 2008, [Online; navštíveno 3.8.2008].
URL http://cs.wikipedia.org/wiki/HTTP

[45] Skript (program). In *Wikipedie – Otevřená encyklopedie*, nadace WIKIMEDIA, 2008, [Online; navštíveno 3.8.2008].
URL http://cs.wikipedia.org/wiki/Skript_(program)

[46] XML. In *Wikipedie – Otevřená encyklopedie*, nadace WIKIMEDIA, 2008, [Online; navštíveno 3.8.2008].
URL http://cs.wikipedia.org/wiki/XML

[47] XSLT. In *Wikipedie – Otevřená encyklopedie*, nadace WIKIMEDIA, 2008, [Online; navštíveno 3.8.2008].
URL http://cs.wikipedia.org/wiki/XSLT

[48] My Yahoo! 2008, [Online; navštíveno 3.8.2008].
URL http://my.yahoo.com/

[49] Zajíc, P.: Seriál PHP. 2006, [Online; navštíveno 3.8.2008].
URL http://www.linuxsoft.cz/php/

Přílohy

Seznam příloh

Příloha A

Dokumentace pro vývojáře vytvářející komponenty

Tato dokumentace je určena pro vývojáře komponentů a obsahuje detailní popis API observeru a správce rozložení. Obsahuje též popis struktury konfiguračního XML včetně detailního popisu významu jednotlivých elementů.

A.1 Observer

Pro observer jsem využil abstraktní bázovou třídu `Ext.util.Observable`, která řeší téměř vše potřebné. Rozšířil jsem ji zde o stavový objekt (`state`), vlastnost s informací o tom, kdo prováděl poslední modifikaci stavového objektu (`lastModifiedBy`), událost změny stavového objektu „`stateChanged`“ a následující metody:

- `update(updatedBy,variableName,newValue)`
  - Popis: Metoda pro aktualizaci stavového objektu a upozornění registrovaných funkcí na změnu stavu. Vyvolá událost, která má jako parametr objekt s následujícími vlastnostmi:
    - `lastModifiedBy` – hodnota parametru této funkce, viz níže
    - `changedVariable` – název vlastnosti stavového objektu, která se změnila
    - `oldValue` – hodnota vlastnosti před změnou
    - `newValue` – hodnota vlastnosti po změně
  - Parametry:
    - `updatedBy` – řetězcová hodnota nastavovaná do vlastnosti `lastModifiedBy` parametru události. Umožní reagovat pouze na vybrané události (od vybraných objektů). U komponentů by měla být id komponentu, aby nedocházelo k interferenci.
    - `variableName` – název vlastnosti stavového objektu, která se bude měnit
    - `newValue` – nová hodnota vlastnosti stavového objektu
- `getStateVariable(variableName)`
  - Popis: Metoda pro získání hodnoty vlastnosti stavového objektu
  - Parametry:
    - `variableName` – název požadované vlastnosti

  - Návratová hodnota:
    * vrací požadovanou vlastnost stavového objektu

- `getState()`
  - Popis: Metoda pro získání přístupu ke stavového objektu
  - Návratová hodnota:
    * vrací referenci na stavový objekt

Ze třídy `Ext.util.Observable` jsou důležité především tyto metody:

- `addEvents(object)`
  - Popis: Metoda pro definování nových událostí observeru
  - Parametry:
    * `object` – objekt s definicí událostí, kde každá událost je definována jako vlastnost s pravdivostní hodnotou true (název vlastnosti odpovídá názvu události)

- `on(eventName,handler,[scope],[options])`
  - alias pro `addListener()`

- `addListener(eventName,handler,[scope],[options])`
  - Popis: Metoda pro registraci posluchače událostí (funkce, která bude volána v reakci na událost)
  - Parametry:
    * `eventName` – název události (řetězcová hodnota)
    * `handler` – funkce, která bude volána v reakci na událost
    * `scope` – kontext volané funkce (objekt, na nějž bude ve volané funkci odkaz v proměnné this)
    * `options` – volitelný objekt pro detailnější konfiguraci volané funkce. Může mít následující vlastnosti:
      · `scope` – kontext volané funkce
      · `delay` – počet milisekund od uplynutí události, po kterém bude funkce volána
      · `single` – pokud bude tato vlastnot nabývat pravdivostní hodnoty `true`, bude funkce volána pouze při následujícím výskytu události a následně bude z observeru automaticky odregistrována
      · `buffer` – Počet milisekund, po kterém bude funkce volána pomocí objektu třídy `Ext.util.DelayedTask` (více viz manuál Ext JS)
      · další volitelné parametry volané funkce

- `fireEvent(String eventName, Object... args)`
  - Popis: Metoda pro vyvolání události.
  - Parametry:
    * `eventName` – název události v řetězci
    * `args` – argumenty pro funkce pro obsluhu události v objektech

  - Návratová hodnota:
    * Pokud některá z funkcí pro obsluhu události vrátila `false`, vrací `false`, jinak vrací `true`.

- `hasListener(eventName)`
  - Popis: Metoda pro zjištění, zda má objekt zaregistrovanou nějakou funkci pro obsluhu události.
  - Parametry:
    * `eventName` – řetězcový název události
  - Návratová hodnota:
    * Pokud má objekt zaregistrovanou funkci pro obsluhu události, vrací `true`, jinak vrací `false`.

- `un(eventName,handler,[scope])`
  - alias pro `removeListener()`

- `removeListener(eventName,handler,[scope])`
  - Popis: Metoda pro odregistrování posluchače událostí (funkce volané v reakci na událost)
  - Parametry:
    * `eventName` – řetězcový název události
    * `handler` – funkce, která se má odregistrovat
    * `scope` – volitelný kontext funkce, která se má odregistrovat (objekt)

- `relayEvents(o,events)`
  - Popis: Metoda pro nastavení přeposílání událostí mezi observery. Způsobí, že vyvolání události ve specifikovaném observeru vyvolá stejnou událost v observeru, který byl touto metodou nastaven.
  - Parametry:
    * `o` – objekt observeru, ze kterého se mají přijímat události
    * `events` – pole řetězců s názvy událostí, které se mají přijímat

Příklad použití:

```
<input type="button" value="Tlačítko"
 onClick="observer.update('__MY_ID__','color','blue');"/>
<script type="text/javascript">
  observer.addListener("stateChanged", function(event) {
    if(layoutManager.getComponent("__MY_ID__") == null)
    {  // Tato funkce je registrována přímo do observeru,
       // pokud bude komponent uzavřen, funkce nesmí
       // přistupovat ke svému rodičovskému komponentu
      return;
    }
    layoutManager.getComponentBody("__MY_ID__").setStyle(
```

```
        "backgroundColor",observer.getStateVariable('color'));
    })
</script>
```

Tento observer umožňuje jednoduše přidávat další stavové objekty, události a jejich posluchače, čehož jsem využil i ve správci rozložení. Stačí tak jeden observer pro práci se všemi událostmi na stránce a veškerou komunikaci komponentů.

Při práci s observerem je nutné ošetřit situaci uzavření komponentu. Pokud je komponent uzavřen, jsou odstraněny všechny jeho části kromě funkcí registrovaných do observeru, které při následném pokusu o přístup ke svému rodičovskému komponentu mohou vyvolat chybu. Pokud má být funkce z observeru automaticky odebrána při uzavření komponentu, je nutné ji do observeru registrovat pomocí obalující funkce správce rozložení (viz dále).

Žádný název nové události nebo stavového objektu by neměl začínat řetězcem „`layout`“, aby nedošlo ke kolizi se stavovými objekty správce rozložení. Další užitečné informace se lze dozvědět v dokumentaci API Ext JS (`Ext.util.Observable`).

A.2 Správce rozložení

Konstruktor této třídy pouze vytvoří vlastnosti objektu a přiřadí jim výchozí hodnoty. Veškerá činnost je prováděna voláním metod. Objekt má vlastnosti s řetězci určené pro lokalizaci (viz dále) a následující vlastnosti:

- `loadPageFrom` – adresa, ze které se načítá XML s obsahem stránky. Tato veřejná vlastnost musí být nastavena před voláním funkce `loadPage()` (jinak je uskutečněn pokus o načtení z example.com)
- `gadgetsSupport` – vlastnost, která umožňuje zapnout textové nahrazování některých volání metod v JavaScriptovém kódu gadgetu, který je obalen v komponentu (viz dále). Výchozí hodnota je `false` (podpora Google Gadgets vypnuta).
- `showLoadingProgress` – vlastnost umožňující deaktivaci progressbaru zobrazeného při načítání konfiguračního XML. Výchozí hodnota je `true` (progressbar bude zobrazen). V případě nastavení na `false` progressbar nebude zobrazen a element pro jeho umístění nemusí být uveden v indexové stránce.
- `viewport` – reference na objekt viewportu (obsahuje veškeré objekty tvořící stránku). Komponenty by ji neměly běžně využívat, ale může být využita při ladění.

Správce rozložení má také následující metody:

- `loadPage()`
  - metoda pro načtení stránky, provádí následující činnost:
    - Zobrazí progressbar do elementu `<div id="layoutLoadingProgress"/>` (pouze pokud není jeho zobrazení zakázáno). Tento progressbar bude animován maximálně po dobu 60s a následně animace skončí, aby bylo signalizováno podezřele dlouhé načítání.
    - Načte ze serveru XML s obsahem stránky.
    - Zobrazí dialogové okno s progressbarem, aby uživatel nemohl zasahovat do neinicializované stránky v průběhu jejího sestavování.

    - Analyzuje XML a sestaví konfigurační objekt pro viewport. Pokud XML není validní, dojde v některých prohlížečích k chybě či zamrznutí (nelze vhodně a efektivně ošetřit ve všech prohlížečích). Pokud v XML chybí nějaké komponenty, jsou nahrazeny výplňovými (při vytvoření viewportu musí být v každé vytvořené oblasti v každém sloupci nejméně 1 komponent a současně 1. komponent ve sloupci musí být v nultém řádku a mezi řádky nesmí být mezery). Při analýze XML sestaví také některé stavové objekty v observeru, které obsahují informace o rozložení stránky, komponentech a jejich nastavení.
    - Vytvoří viewport.
    - Odstraní výplňové komponenty.
    - Provede dokončení inicializace formulářů s nastaveními komponentů.
    - Spustí skripty v komponentech tak, že každý uzel DOM stromu se značkou `<script>` je nahrazen nově vytvořeným shodným uzlem (obsah značek `script` není analyzován automaticky, protože uzly se skripty nejsou do DOM stromu vloženy spravným způsobem).
    - Přiřadí oblastem funkce pro obsluhu událostí.
    - Skryje dialogové okno, progressbar a element `<div id="layoutLoading">`
    - Vloží do stránky skript, který vyvolá událost dokončení inicializace stránky („`layoutReady`“), kterou mohou komponenty využít jako náhradu za metodu `Ext.onReady()`

- `getComponent(componentId)`
  - Popis: Získá referenci na komponent s daným id. Každý komponent je objekt třídy `Ext.panel,` a pro manipulaci s komponentem lze tedy využít většinu možností této třídy (viz dále).
  - Parametry:
    - `componentId` – id komponentu, na nějž se má získat reference
  - Vrací referenci na komponent

- `getComponentBody(componentId)`
  - Popis: Získá referenci na tělo komponentu. Komponenty, které mají formulář s nastavením, mají tělo ve vnořeném panelu pro obsah, komponenty bez nastavení jej mají přímo v panelu komponentu. Tato metoda poskytuje jednotné rozhraní pro přístup k tělu komponentu.
  - Parametry:
    - `componentId` – id komponentu, na jehož tělo se má získat reference
  - Vrací referenci na tělo komponentu

- `getComponentLocation(componentId)`
  - Popis: Získá objekt s informacemi o pozici komponentu.
  - Parametry:
    - `componentId` – id komponentu, jehož pozice se má získat
  - Vrací objekt s následujícími vlastnostmi:
    - `area` – oblast, ve které se komponent nachází (north, east, south, west nebo center)
    - `col` – sloupec, ve kterém se komponent nachází (číslovány od 0)
    - `row` – řádek, ve kterém se komponent nachází (číslovány od 0)
- `getComponentPreferences(componentId)`
  - Popis: Metoda pro získání reference na pole s uživatelskými nastaveními komponentu (`UserPref`). Získanou referenci nelze využít ke změně nastavení komponentu!
  - Parametry:
    - `componentId` – id komponentu, jehož uživatelská nastavení se mají získat
  - Vrací referenci na pole objektů s uživatelskými nastaveními komponentu indexované názvy položek nastavení. Každý objekt v tomto poli má nasledující vlastnosti:
    - `dataType` – typ pole ('string', 'bool', 'enum', 'list' nebo 'hidden')
    - `displayName` – zobrazovaný název pole
    - `required` – určuje, zda je pole povinné
    - `value` – hodnota
    - `enumValues` – pouze u typu enum, obsahuje položky výběru v poli objektů s následujícími vlastnostmi:
      - `displayValue` – zobrazovaná položka výběru
      - `value` – hodnota položky výběru
- `getComponentPrefsItem(componentId, prefName)`
  - Popis: Metoda pro získání položky uživatelského nastavení komponentu.
  - Parametry:
    - `componentId` – id komponentu, jehož položka nastavení se má získat
    - `prefName` – Název položky nastavení, která se má získat
  - Vrací požadovanou položku nastavení jako objekt s následujícími vlastnostmi:
    - `dataType` – typ pole (`'string'`, `'bool'`, `'enum'`, `'list'` nebo `'hidden'`)
    - `displayName` – zobrazovaný název pole
    - `required` – určuje, zda je pole povinné
    - `value` – hodnota
    - `enumValues` – pouze u typu enum, obsahuje položky výběru v poli objektů s následujícími vlastnostmi:
      - `displayValue` – zobrazovaná položka výběru

            - `value` – hodnota položky výběru

- `setComponentPrefsItem(componentId, prefName, newValue)`
  - Popis: Metoda pro nastavení hodnoty položky uživatelských nastavení komponentu (`UserPref`).
  - Parametry:
    - `componentId` – id komponentu, jehož nastavení se má měnit
    - `prefName` – název položky nastavení
    - `newValue` – nová hodnota položky nastavení
  - Vyvolá událost změny uživatelských nastavení komponentu (popis viz níže).

- `refresh()`
  - Popis: Metoda pro obnovení stránky.
    Pouze zavolá `window.location.reload()`

- `getAreasInfo()`
  - Popis: Metoda pro získání reference na objekt s informacemi o oblastech stránky. Referencovaný objekt není dovoleno měnit.
  - Vrací referenci na objekt s informacemi o oblastech, který má následující veřejné vlastnosti:
    - `center` – objekt s informacemi o centrální oblasti, který má následující veřejné vlastnosti:
      - `presented` – udává, zda je oblast ve stránce obsažena
      - `collapsible` – udává, zda je dovoleno skrývání těla oblasti
      - `collapsed` – udává, zda je tělo oblasti skryto
      - `cols` – pole informací o sloupcích, každa položka v tomto poli je tvořena objektem s vlastností `colWidth` udávající šířku sloupce
      - `title` – titulek oblasti – tuto vlastnost má jen když je oblast ve stránce obsažena (`presented == true`, u centrální vždy splněno).
    - `north` – objekt s informacemi o horní oblasti. Má stejné vlastnosti jako objekt pro centrální oblast, a pokud je oblast ve stránce obsažena, má navíc tyto vlastnosti:
      - `defaultSize` – výchozí výška oblasti
      - `minSize` – minimální výška oblasti při změně velikosti
      - `maxSize` – maximální výška oblasti při změně velikosti
      - `size` – aktuální výška oblasti
      - `closing` – udává, zda lze oblast uzavřít
      - `closed` – tuto vlastnost má, pouze pokud byla oblast uzavřena. V tomto případě má vlastnost vždy hodnotu `true` a oblast již není obsažena ve stránce (je skryta, nebo zcela odstraněna).
    - `south` – objekt s informacemi o dolní oblasti, má stejnou strukturu jako objekt s informacemi o horní oblasti (`north`)

    * `east` – objekt s informacemi o pravé oblasti, má stejnou strukturu jako objekt s informacemi o horní oblasti, ale vlastnosti `defaultSize`, `minSize`, `maxSize` a `size` udávají šířky
    * `west` – objekt s informacemi o levé oblasti, má stejnou strukturu, jako objekt s informacemi o pravé oblasti
    * `denyClosingComponentsWithDisabledClosing` – pravdivostní hodnota určující zda lze zavírat oblast s komponentem bez zavírací ikonky a společně s ní i daný komponent.

- `getComponentPositions()`
  - Popis: Metoda pro získání reference na objekt s informacemi o pozicích komponentů na stránce. Referencovaný objekt není dovoleno měnit.
  - Vrací referenci na objekt s informacemi o pozicích komponentů na stránce, který má následující veřejné vlastnosti:
    * `north` – pole sloupců v horní oblasti. Každá položka tohoto pole obsahuje pole řádků. Každá položka pole řádků obsahuje objekt s vlastností id, což je id komponentu na daném řádku.
    * `east` – pole sloupců v pravé oblasti, struktura jako u north.
    * `south` – pole sloupců v dolní oblasti, struktura jako u north
    * `west` – pole sloupců v levé oblasti, struktura jako u north
    * `center` – pole sloupců v centrální oblasti, struktura jako u north
    * `byId` – pole indexované id komponentů. Každá položka tohoto pole obsahuje objekt s následujícími vlastnostmi:
      · `area` – umístění oblasti, ve které se komponent nachází
      · `col` – sloupec, ve kterém se komponent nachází
      · `row` – řádek, ve kterém se komponent nachází

- `getComponentPrefsPresence()`
  - Popis: Metoda pro získání reference na pole s informacemi o tom, které komponenty mají uživatelská nastavení (`UserPref`). Pole je určeno pouze pro čtení a není dovoleno jej měnit.
  - Vrací referenci na pole indexované pomocí id komponentů. Každá položka tohoto pole je objekt s následujícími vlastnostmi:
    * `havePrefs` – informace o tom, zda má komponent nějaká nastavení
    * `visiblePrefs` – informace o tom, zda má komponent nastavení, která jsou zobrazena ve formuláři (zda má formulař s nastavením)
    * `hiddenPrefs` – informace o tom, zda má komponent nějaká nastavení, která nejsou zobrazena ve formuláři (`dataType="hidden"`)

- `getComponentPrefs()`
  - Popis: Metoda pro získání reference na pole s uživatelskými nastaveními komponentů. Získanou referenci nelze využít ke změně nastavení komponentů.
  - Vrací referenci na pole indexované pomocí id komponentů. Každá položka tohoto pole je pole uživatelských nastavení daného komponentu indexované názvy proměnných uživatelského nastavení (atribut `name` elementu `UserPref`). Každá položka pole uživatelských nastavení komponentu je objekt s následujícími veřejnými vlastnostmi:
    - `dataType` – datový typ položky (`'string'`, `'bool'`, `'enum'`, `'list'` nebo `'hidden'`)
    - `displayName` – zobrazované jméno položky nastavení
    - `required` – informace o tom, zda je položka vyžadována
    - `value` – hodnota položky
    - `enumValues` – pouze u typu enum, obsahuje položky výběru v poli objektů s následujícími vlastnostmi:
      - `displayValue` – zobrazovaná položka výběru
      - `value` – hodnota položky výběru
- `getComponentLocalizedText(componentId, textName)`
  - Popis: Metoda pro získání lokalizovaného textu komponentu
  - Parametry:
    - `componentId` – id komponentu, jehož lokalizovaný text se má získat
    - `textName` – Název lokalizovaného textu, který se má získat
  - Vrací řetězec s požadovaným lokalizovaným textem
- `getComponentLocales(componentId)`
  - Popis: Metoda pro získání informací o lokalizaci (locales) pro komponent
  - Parametry:
    - `componentId` – id komponentu, jehož lokalizační informace se mají získat
  - Vrací lokalizační informace pro komponent v objektu s následujícími vlastnostmi:
    - `lang` – jazyk
    - `country` – země
- `registerEventListener(componentId, eventName, eventHandler)`
  - Popis: Metoda pro registraci posluchače událostí (listeneru) do observeru. Zaregistruje funkci pro reakci na událost pomocí metody `observer.addListener()` a zajistí, že při uzavření komponentu bude tato funkce odregistrována.
  - Parametry:
    - `componentId` – id komponentu, který posluchače registruje
    - `eventName` – název události
    - `eventHandler` – funkce, která bude zaregistrována jako posluchač událostí (tato funkce bude volána při dané události)

- `unregisterEventListener(componentId, eventName, eventHandler)`
  - Popis: Metoda pro odregistraci posluchače událostí z observeru. Odregistruje funkci pro reakci na událost pomocí metody `observer.removeListener()`
  - Parametry:
    * `componentId` – id komponentu, který posluchače odregistruje
    * `eventName` – název události
    * `eventHandler` – funkce, která bude odregistrována

Kromě skutečných metod má správce rozložení také „nahrazované“ metody, tedy metody, které v objektu neexistují, ale v komponentech je lze využívat. Tyto metody jsou při analýze XML s obsahem stránky textově nahrazeny za existující ekvivalenty (nelze je využít v dynamicky sestavovaném kódu). Jedná se o metody:

- `layoutManager.getMyComponent()`
  - nahrazeno za `layoutManager.getComponent('__MY_ID__')`
- `layoutManager.getMyComponentBody()`
  - nahrazeno za `layoutManager.getComponentBody('__MY_ID__')`
- `layoutManager.getMyComponentLocation()`
  - nahrazeno za `layoutManager.getComponentLocation('__MY_ID__')`

Zástupný symbol `__MY_ID__` je při analýze nahrazen řetězcovou hodnotou id komponentu. Pro využití je nutné jej umístit do uvozovek (příklad viz výše).

Zástupný symbol `__MODULE_ID__` je při analýze nahrazen řetězcovou hodnotou id komponentu v uvozovkách. Tento symbol je shodný se symbolem využívaným v Google gadgets.

Při analýze konfiguračního XML jsou nahrazeny také zástupné symboly za výchozí hodnoty proměnných uživatelských nastavení. Tyto zástupné symboly se skládají z řetězce "`__UP_`", názvu proměnné a řetězce "`__`". Např. element <`UserPref name="myName" default_value="jmeno"/`> definuje proměnnou uživatelského nastavení, za kterou lze využít zástupný symbol `__UP_myName__` a tento symbol bude nahrazen hodnotou "jmeno". Tyto symboly jsou nahrazeny v titulku a obsahu komponentu.

Stejně jako zástupné symboly za proměnné jsou nahrazovány i zástupné symboly za lokalizovatelné texty. Tyto mají strukturu "`__MSG_`" + název + "`__`" a náhrady jsou definovány elementy <`msg`>. Zástupné symboly za lokalizované texty jsou nahrazovány nejenom v titulku a obsahu komponentu, ale také v atributu `display_name` položek uživatelského nastavení a v atributu `display_value` hodnoty výčtu v uživatelském nastavení.

Pokud k některému ze zástupných symbolů není nalezena příslušná proměnná ani lokalizovaný text, zástupný symbol je ponechán beze změny.

Pokud je `gadgetsSupport` nastaveno na `true` (nutno nastavit obdobným způsobem, jako adresu, ze které se načítá XML s obsahem stránky) a komponent obsahuje element <`ModulePrefs`>, jsou textově nahrazena i následující volání metod, které za dané situace nelze implementovat bez parametru s id komponentu:

- `new gadgets.Prefs()`
  - nahrazeno za `new gadgets.Prefs(__MODULE_ID__)`

- `gadgets.window.adjustHeight(opt_height)`
  - ”`gadgets.window.adjustHeight()`”
    nahrazeno za ”`gadgets.window.adjustHeight(__MODULE_ID__)`”
  - ”`gadgets.window.adjustHeight(`”
    nahrazeno za ”`gadgets.window.adjustHeight(__MODULE_ID__,`”
- `gadgets.window.getViewportDimensions()`
  - nahrazeno za `gadgets.window.getViewportDimensions(__MODULE_ID__)`
- `gadgets.window.setTitle(title)`
  - ”`gadgets.window.setTitle(`”
    nahrazeno za ”`gadgets.window.setTitle(__MODULE_ID__,`”
- `new _IG_Prefs()`
  - nahrazeno za `new _IG_Prefs(__MODULE_ID__)`
- `_IG_AdjustIFrameHeight(opt_height)`
  - ”`_IG_AdjustIFrameHeight()`”
    nahrazeno za ”`_IG_AdjustIFrameHeight(__MODULE_ID__)`”
  - ”`_IG_AdjustIFrameHeight(`”
    nahrazeno za ”`_IG_AdjustIFrameHeight(__MODULE_ID__,`”
- `_IG_SetTitle(title)`
  - ”`_IG_SetTitle(`”
    nahrazeno za ”`_IG_SetTitle(__MODULE_ID__,`”

Textové nahrazování není plnohodnotnou náhradou, protože neřeší volání metod v dynamicky generovaném kódu, nebo kódu, který byl načten z externího zdroje. Vzhledem k charakteru velkého množství jednoduchých gadgetů (kód je statický, uvedený přímo v gadgetu), má tato náhrada smysl, protože umožňuje využití těchto gadgetů. Složitější gadgety, či gadgety s externími skripty je nutno provozovat buď za použití složitější knihovny, která je načte za běhu (nebudou zpracovány správcem rozložení), nebo gadgety upravit.

Správce rozložení také přidává do observeru další stavové objekty:

- `layoutAreasInfo` – obsahuje informace o oblastech na stránce. Komponenty jej mohou využít, ale nesmějí jej modifikovat. Referenci na něj lze získat výše uvedenou metodou, u jejíhož popisu jsou popsány i jednotlivé vlastnosti obsažených objektů.
- `layoutComponentPositions` – obsahuje informace o pozicích komponentů. Komponenty jej rovněž mohou využít, ale nesmějí modifikovat. Textový popis struktury a vlastností je uveden výše u popisu metody, která jej vrací. Při přesunu nebo uzavření komponentu správce rozložení automaticky upraví informace v tomto objektu a vyvolá příslušnou událost.
- `layoutComponentListeners` – je určen pro privátní použití správcem rozložení a obsahuje informace o funkcích, které mají komponenty registrované v observeru.

- `layoutComponentLocales` – obsahuje informace o lokalizačních informacích pro komponenty. Jedná se o pole indexované pomocí id komponentů, jehož každá položka je objekt s vlastnostmi `country` a `lang`.
- `layoutComponentMessagesPresence` – pole objektů indexované pomocí id komponentů obsahující informace o tom, které komponenty mají lokalizované texty. Každá položka tohoto pole je objekt s vlastností `haveMessages`.
- `layoutComponentMessages` – patří k objektu `layoutComponentMessagesPresence`, je rovněž privátní, a obsahuje lokalizované texty komponentů. Jedná se o pole indexované id komponentů, jehož každá položka je pole indexované názvy lokalizovaných textů. Každý text je uložen ve vlastnosti `content` příslušného objektu. Kromě této vlastnosti mají objekty také vlastnost `valid`, která je určena výhradně pro privátní využití správcem rozložení.
- `layoutComponentPrefsPresence` – obsahuje informace o tom, které komponenty mají uživatelská nastavení a o tom, zda mají formulář s nastavením a skrytá nastavení. Tento objekt, ani objekt se samotnými uživatelskými nastaveními, nesmějí být přímo modifikovány. Podrobný popis lze nalézt výše u metody pro získání reference (`getComponentPrefsPresence()`).
- `layoutComponentPrefs` – obsahuje uživatelská nastavení komponentů. Podrobný popis lze nalézt výše u metody pro získání reference (`getComponentPrefs()`).

Při manipulaci se všemi poli je třeba obezřetnost, protože Ext JS modifikuje konstruktor `Array()` a každé pole má metodu `remove()`. Při procházení cyklem „`for in`“ je třeba tuto metodu přeskočit.

Pro lokalizaci správce rozložení je využit obdobný princip, jako v knihovně Ext JS. Všechny vypisované lokalizovatelné textové řetězce jsou tedy ve veřejných vlastnostech třídy a pro lokalizaci se využívá změna prototypu. Aby se zde vlastnosti z prototypu nepředefinovaly, jsou definovány, pouze pokud první z nich není dostupná.

A.3 Události vyvolané správcem rozložení

Správce rozložení vyvolává různé události observeru, které mohou komponenty využít k detekci své pozice, změny rozměrů, pozice, nastavení spolupracujících komponentů apod. Následuje popis jednotlivých událostí, včetně popisu objektů, které jsou u nich předávány v parametrech:

- `layoutReady` – událost dokončení inicializace stránky, lze využít jako náhradu za metodu `Ext.onReady()`
- `layoutChanged` – událost změny rozložení stránky. Jako parametr je předán objekt, který má vždy vlastnost `action` (řetězec), udávající typ změny, a další vlastnosti s upřesňujícími informacemi dle typu změny. Následuje výpis možných hodnot vlastnosti `action` s popisy a dalších vlastností příslušejících k dané hodnotě:
  - `areaWillBeCollapsed` – tělo oblasti bude skryto (`beforecollapse`)
    - `id` – id oblasti (např. ”`northArea`”)
    - `areaLocation` – umístění oblasti (např. ”`north`”)

  - `areaCollapsed` – tělo oblasti skryto
    * `id` – id oblasti (např. "`northArea`")
    * `areaLocation` – umístění oblasti (např. "`north`")
  - `areaWillBeExpanded` – skryté tělo oblasti bude zobrazeno (`beforeexpand`)
    * `id` – id oblasti (např. "`northArea`")
    * `areaLocation` – umístění oblasti (např. "`north`")
  - `areaExpanded` – skryté tělo oblasti zobrazeno
    * `id` – id oblasti (např. "`northArea`")
    * `areaLocation` – umístění oblasti (např. "`north`")
  - `areaRemoved` – oblast odstraněna (zavřena)
    * `areaId` – id oblasti (např. "`northArea`")
    * `areaLocation` – umístění oblasti (např. "`north`")
  - `areaResized` – velikost oblasti změněna
    * `id - id oblasti` (např. "`northArea`")
    * `areaLocation` – umístění oblasti (např. "`north`")
    * `adjWidth` – skutečná nová šířka (po přizpůsobení boxu)
    * `adjHeight` – skutečná nová výška (po přizpůsobení boxu)
    * `rawWidth` – nastavená šířka (viz `Ext.panel` resize event)
    * `rawHeight` – nastavená výška
  - `componentMoved` – komponent přesunut
    * `id` – id komponentu
    * `oldArea` – oblast, odkud byl komponent přesunut
    * `oldCol` – sloupec, odkud byl komponent přesunut
    * `oldRow` – řádek, odkud byl komponent přesunut
    * `newArea` – oblast, kam byl komponent přesunut
    * `newCol` – sloupec, kam byl komponent přesunut
    * `newRow` – řádek, kam byl komponent přesunut
  - `componentRemoved` – komponent odstraněn (zavřen)
    * `id` – id komponentu
    * `oldArea` – oblast, odkud byl komponent odstraněn
    * `oldCol` – sloupec
    * `oldRow` – řádek

- `layoutComponentPreferencesChanged` – událost změny hodnot uživatelských nastavení komponentu. Jako parametr je předán objekt s následujícími vlastnostmi:
  - `id` – id komponentu
  - `oldValues` – původní hodnoty v poli objektů indexovaném názvy proměnných uživatelského nastavení. Každý objekt v tomto poli má vlastnosti:
    * `dataType` – typ pole
    * `value` – hodnota

- `newValues` – reference na pole objektů s novými hodnotami indexované názvy proměnných uživatelského nastavení. Každý objekt v tomto poli má vlastnosti:
  - `dataType` – typ pole (`'string'`, `'bool'`, `'enum'`, `'list'` nebo `'hidden'`)
  - `value` – hodnota
  - `displayName` – zobrazovaný název pole
  - `required` – informace o tom, zda se jedná o povinné pole
  - `enumValues` – pouze u typu `enum`, obsahuje položky výběru v poli objektů s následujícími vlastnostmi:
    - `displayValue` – zobrazovaná položka výběru
    - `value` – hodnota položky výběru
- `changedByFormSaving` – vlastnost, která určuje, zda bylo nastavení změněno uložením příslušného formuláře (`true`), nebo programově skriptem (`false`)
- `changedPrefName` – pouze pokud bylo nastavení změněno skriptem, obsahuje název změněné položky nastavení

Událost přesunu komponentu je vyvolána se zpožděním 200ms, čímž se obchází nedokonalost knihovny Ext JS, která způsobuje problémy při přesunu komponentu do prázdného sloupce (nedojde k přizpůsobení šířky komponentu). Po 100ms dojde k fixaci chyby v rozložení stránky a po 200ms k vyvolání události.

Příklad využití událostí od správce rozložení:

```
<script type="text/javascript">
  observer.addListener("layoutReady", function(event) {
    layoutManager.getComponentBody("__MY_ID__").dom.innerHTML
      += "Stránka inicializována! ";
  })
  layoutManager.registerEventListener("__MY_ID__",
      "layoutChanged", function(event) {
    if (event.action == "componentMoved") {
      if (event.id == "__MY_ID__") {
        layoutManager.getComponentBody("__MY_ID__").dom.innerHTML
          += "Moje nové umístění je "
          + layoutManager.getMyLocation().area
          + " oblast " + layoutManager.getMyLocation().col
          + " sloupec " + layoutManager.getMyLocation().row
          + " řádek. ";
      }
    }
  })
</script>
```

A.4 Struktura konfiguračního XML s obsahem stránky

Pro množství volitelných elementů s výchozím chováním a sémantických omezení strukturu konfiguračního XML vysvětlím na příkladech. Začnu extrémními příklady s vysvětlením chování v případě nedefinovaných informací a následně vysvětlím podrobně jednotlivé části. Minimální obsah vypadá následovně:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<pageData>
<layoutSettings>
</layoutSettings>
</pageData>
```

V případě načtení tohoto XML bude zobrazena prázdná stránka s centrální oblastí, ve které bude 1 prázdný sloupec. <pagedata> je povinný kořenový element dokumentu <layoutSettings> je povinný element, který obsahuje informace pro správce rozložení. Pokud některý z těchto elementů nebude obsažen, stránku nebude možné zobrazit.

Centrální oblast musí být vždy dostupná, pokud není v konfiguraci uvedena, bude vytvořena s 1 prázdným sloupcem.

Vložíme-li do stránky 2 prázdné komponenty, bude XML vypadat následovně:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<pageData>
<layoutSettings>
</layoutSettings>
<component/>
<component id="1"/>
</pageData>
```

Pokud komponent nemá uvedené umístění, bude umístěn na první volný řádek v nultém sloupci centrální oblasti. Jedná se o první volný řádek při současném stavu zpracování XML, je tedy možné, že bude následně nalezen komponent, který se má na řádku skutečně nacházet. Ten potom bude posunut na další volný řádek a stejně budou posunuty i další komponenty.

Pokud některý komponent nemá uvedenu oblast, nebo požadovaná oblast není dostupná, bude umístěn v centrální oblasti, pokud nemá uveden sloupec, bude v nultém sloupci dané oblasti. Pokud některý komponent ve sloupci nemá uveden řádek, skutečné umístění všech komponentů v daném sloupci lze považovat za nedefinované.

Pokud komponent nemá id, bude mu přiděleno vygenerované id. Toto id by následně mohlo dělat problémy při ukládání informací na server (je jednoznačné pouze v rámci načtení stránky). Každý komponent by tedy měl mít uvedené id. Všechna vygenerovaná id začínají řetězcem „layout“, skript pro ukládání dat na server by je měl rozeznat a neukádat nastavení komponentů s těmito id.

Nyní komponentům doplním atributy, které by všechny komponenty měly mít, a přiřadím jim i minimální obsah. Dále doplním definici oblasti se dvěma sloupci. Obsah XML bude nyní následující:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<pageData>
<layoutSettings>
  <area location="center">
    <cols>
      <col width="50%"/>
      <col width="50%"/>
    </cols>
  </area>
</layoutSettings>
<component id="2" location="center" col="0" row="0">
  <content type="text/html">
  <![CDATA[
    Text v těle komponentu.
  ]]>
  </content>
</component>
<component id="1"  location="center" col="1" row="0">
  <content type="text/html">
  <![CDATA[
    Text v těle komponentu.
  ]]>
  </content>
</component>
</pageData>
```

Pokud bude oblast definována bez sloupců, bude v ní vytvořen jeden sloupec. V oblasti je možné definovat libovolný počet sloupců, přičemž u každého je nutno definovat jeho šířku v procentech ze šířky oblasti. Pokud šířka nebude uvedena, sloupec nebude vytvořen a komponenty z něj budou přesunuty do nultého sloupce. Pokud bude součet šířek sloupců v oblasti menší než 100%, bude vedle sloupců prázdný prostor, pokud bude větší, zobrazení bude chybné.

V elementu `layoutSettings` může být také element, který způsobí, že nebude možné zavřít komponent bez zavíracího tlačítka. Pokud se uživatel pokusí zavřít oblast s takovýmto komponentem, při zavírání daného komponentu bude zobrazeno chybové hlášení a daný komponent ani oblast nebudou uzavřeny. Tento element může být prázdný, bez atributů, a jeho syntaxe je následující:

```
<denyClosingAreaWithComponentWithDisabledClosing/>
```

A.4.1 Podrobná definice oblasti

Maximálně podrobná definice oblasti je:

```
<area location="east" defaultSize="200" minSize="175"
      maxSize="400">
  <title>Pravý panel</title>
  <collapsing enabled="true" collapsed="false"/>
  <closing enabled="true"/>
  <cols>
    <col width="50%" padding="10"/>
```

```
    <col width="50%"/>
  </cols>
</area>
```

Vysvětlení atributů elementu area:

- `location`
  - umístění oblasti
  - povinný atribut, neuvedení vede k ignorování definice oblasti
  - možné hodnoty: `north`, `east`, `south`, `west` nebo `center`
- `defaultSize`
  - u severní a jižní oblasti výchozí výška v px
  - u východní a západní oblasti výchozí šířka v px
  - u centrální oblasti nemá význam
- `minSize`
  - u severní a jižní oblasti minimální výška v px
  - u východní a západní oblasti minimální šířka v px
  - u centrální oblasti nemá význam
- `maxSize`
  - u severní a jižní oblasti maximální výška v px
  - u východní a západní oblasti maximální šířka v px
  - u centrální oblasti nemá význam

`defaultSize`, `minSize` a `maxSize` slouží pro určení rozsahu změny šířky (výšky) oblasti tažením myší. Pokud jsou všechny tři hodnoty stejné, změna je zablokovaná (vůbec se nezobrazí zesílené okraje oblastí pro uchopení myší a změnu rozměrů). Výchozí hodnoty jsou:

- u severní a jižní oblasti:
  - `defaultSize = 100`
  - `minSize = 80`
  - `maxSize = 200`
- u východní a západní oblasti:
  - `defaultSize = 200`
  - `minSize = 175`
  - `maxSize = 400`

Element `title` slouží pro textový titulek oblasti. Pokud není uveden, je využita jedna mezera, pokud je uveden prázdný element <`title/`>, titulek je prázdný. Pokud je titulek prázdný a je zakázán collapsing i zavírání, vůbec nedojde ke zobrazení záhlaví oblasti.

Element `collapsing` slouží pro povolení či zakázání skrývání těla oblasti pomocí ikonky v záhlaví oblasti. Atribut `enabled` udává, zda je skrývání povoleno, a pokud je povoleno, atribut `collapsed` udává, zda má být oblast po načtení stránky skryta. Možné hodnoty jsou "`true`" (povoleno / skryta) a "`false`" (zakázáno / zobrazena). Výchozí hodnoty pro okrajové oblasti jsou:

```
enabled = "true"
collapsed = "false"
```

U centrální oblasti je výchozí hodnota:

```
enabled = "false"
```

Element `closing` slouží pro povolení či zakázání zavření oblasti. Pokud má atribut enabled hodnotu "`true`", oblast bude mít ikonku pro zavření. Při kliknutí na tuto ikonku bude zobrazen dotaz na potvrzení zavření oblasti a po případném potvrzení budou nejprve zavřeny všechny komponenty v oblasti (a vyvolány příslušné události) a následně bude uzavřena celá oblast. Centrální oblast nelze zavírat, tento element u ní bude ignorován. Výchozí hodnota je `enabled = "false"`.

Element `col` slouží k definici sloupce. Jeho atributy jsou:

- `width`
  - šířka sloupce v % (procenta oblasti zabraná sloupcem)
  - všechny sloupce v oblasti musejí mít dohromady 100%
- `padding`
  - okraje oblasti v px
  - výchozí hodnota je 10px

Element `cols` slouží pouze pro umístění elementů `col`, které byly vysvětleny výše.

A.4.2 Podrobná definice komponentu

Nejprve vysvětlím základní komponent, který je minimálně závislý na správci rozložení, a následně detailně vysvětlím komponent, včetně rozšířených možností umožňujících mimo jiné i částečnou kompatibilitu s Google Gadgets.

```
<component id="header2" location="north" col="1" row="0">
  <preferences>
    <title>Titulek v záhlaví komponentu</title>
    <collapsing enabled="true" collapsed="false"/>
    <closing enabled="true"/>
    <fixedHeight value="100" scrolling="true"/>
  </preferences>
  <content type="text/html">
  <![CDATA[
    Textový obsah komponentu
  ]]>
  </content>
</component>
```

Vysvětlení atributů elementu `component`:

- `id`
  - jednoznačné id komponentu (v rámci celého systému)
  - řetězcová hodnota
- `location`
  - umístění oblasti, ve které se má komponent nacházet
  - pokud daná oblast nebude dostupná, využije se centrální
  - možné hodnoty: `north`, `east`, `south`, `west` nebo `center`
- `col`
  - sloupec, ve kterém se má komponent nacházet
  - pokud nebude dostupný, využije se nultý
- `row`
  - řádek, ve kterém se má komponent nacházet
  - pokud nebude volný (obsadí jej jiný komponent se stejnou hodnotou, nebo dojde k posunu komponentem s nedefinovanou hodnotou), využije se první volný řádek

Element `preferences` slouží pro vizuální úpravu XML a přehlednost, jeho obsah může být beze změny významu uveden přímo v elementu `component`.

Element `title` slouží pro textový titulek komponentu. Pokud není uveden, je využita jedna mezera, pokud je uveden prázdný element <`title/`>, titulek je prázdný. Pokud je titulek prázdný a je zakázáno skrývání těla komponentu (collapsing) i zavírání, vůbec nedojde k zobrazení záhlaví komponentu.

Element `collapsing` slouží pro povolení či zakázání skrývání těla komponentu pomocí ikony v jeho záhlaví. Atribut `enabled` udává, zda je skrývání povoleno, a pokud je povoleno, atribut `collapsed` udává, zda má být tělo komponentu po načtení stránky skryto. Možné hodnoty jsou "`true`" (povoleno / skryto) a "`false`" (zakázáno / zobrazeno). Výchozí hodnoty jsou:

```
enabled = "true"
collapsed = "false"
```

Element `closing` slouží pro povolení či zakázání zavření komponentu. Pokud má atribut `enabled` hodnotu `false`, komponent nebude mít ikonu pro zavření. Při kliknutí na tuto ikonu bude zobrazen dotaz na potvrzení zavření a po případném potvrzení bude komponent uzavřen a vyvolána příslušná událost. Pokud ikona není zobrazena, stále může dojít k zavření komponentu tak, že bude přesunut do zavíratelné oblasti a následně bude uzavřena celá oblast. Pro úplné zakázání zavření komponentu je nutno zakázat tento způsob zavření komponentu výše uvedeným způsobem. Výchozí hodnota je `enabled = "true"`.

Element `fixedHeight` slouží k nastavení pevné výšky komponentu. Jeho atribut `value` udává výšku komponentu v px a atribut `scrolling` udává, zda mají být v komponentu v případě potřeby zobrazeny posuvníky (`overflow="auto"`). Pokud má atribut `scrolling` hodnotu `false`, posuvníky se nezobrazí a obsah komponentu bude oříznut na danou oblast (`overflow="hidden"`). Tato vlastnost je v Ext JS nazvána `autoScroll`. Výchozí chování

neomezuje výšku komponentu, která se přizpůsobuje aktuálním potřebám. Výchozí hodnota `scrolling` je "`true`", aby se v případě potřeby zobrazil posuvník pro horizontální posun.

Element `content` obsahuje obsah těla komponentu. Atribut `type` je v této verzi správce rozložení zcela ignorován a předpokládá se "`text/html`". Obsah komponentu musí být textový, a pokud se nejedná o prostý text bez speciálních znaků, musí být uzavřen v sekci CDATA. Obsah tohoto elementu je také analyzován a jsou v něm nahrazeny některé řetězce (viz výše).

V obsahu komponentu se nesmí vyskytovat HTML elementy s id začínajícím řetězcem „`ext`“ nebo „`layout`“, aby nedošlo ke kolizi s automaticky generovanými id a k následným chybám.

Elementy `script` obsažené v těle komponentu jsou v rámci zjednodušení analýzy omezeny tak, že smějí mít pouze atributy `type` a `src` (jiné atributy budou odstraněny).

Před 1. výskytem elementu <`script`> v obsahu komponentu by měl být nejméně jeden alfanumerický znak (písmeno nebo číslice anglické abecedy), a to buď jako text, nebo i jako součást nějaké značky či entity. Nebude-li tam, bude v IE na začátek obsahu komponentu automaticky přidána nezalomitelná mezera (` `). Toto opatření obchází chybu v IE, která způsobuje, že jsou pro prohlížeč skripty před 1. alfanumerickým znakem neviditelné.

A.4.3 Podrobná definice komponentu včetně rozšířených možností

Definice komponentu s rozšířenými možnostmi (pro větší smysluplnost uvedena bez elementů `title` a `fixedHeight`) je následující :

```
<component  id="header2" location="north" col="1" row="0">
  <preferences>
    <collapsing enabled="true" collapsed="false"/>
    <closing enabled="true"/>
    <ModulePrefs title="Titulek s hodnotou proměnné __UP_myVal__
      a lokalizovatelnou částí __MSG_myText__" height="250"
      scrolling="true">
        <Locale lang="cs" country="CZ"/>
    </ModulePrefs>
    <UserPref name="myName" display_name="Name" required="true"
      datatype="string" default_value="Jméno"/>
    <UserPref name="myEnum" display_name="Colors"
      datatype="enum">
      <EnumValue value="Red" display_value="__MSG_red__ color"/>
      <EnumValue value="Green"/>
      <EnumValue value="Yellow" display_value="Yellow color"/>
    </UserPref>
    <UserPref name="myList" display_name="__MSG_colors__"
      datatype="list" default_value="Red|Green|Blue"/>
    <UserPref name="myVal">
    <messagebundle>
      <msg name="myText">lokalizovaná část</msg>
      <msg name="colors">Barvy</msg>
      <msg name="red">Červená</msg>
    </messagebundle>
    <dragging enabled="false"/>
```

```
  </preferences>
  <content type="text/html" view="canvas">
  <![CDATA[
    Tento obsah bude přeskočen, protože je určen pouze
    pro celou obrazovku.
  ]]>
  </content>
  <content type="text/html" view="home">
  <![CDATA[
    Textový obsah komponentu.
  ]]>
  </content>
</component>
```

V atributech elementu `ModulePrefs` může být uveden titulek, výška a nastavení posuvníků komponentu. Ostatní atributy tohoto elementu budou ignorovány. Pokud není uveden element <`title`>, který má vyšší prioritu, využije se hodnota atributu `title` z elementu `ModulePrefs`. Pokud není uveden element <`fixedHeight`>, pro pevnou výšku se využije hodnota atributu `height` elementu `ModulePrefs`. Pokud není uveden element `fixedHeight` a je nastaven atribut `height` elementu `ModulePrefs`, využije se pro nastavení zobrazení posuvníků (viz atribut `scrolling` elementu `fixedHeight`) hodnota atributu `scrolling` elementu `ModulePrefs`. Není-li atribut `scrolling` uveden, využije se `scrolling="true"`.

Element `Locale` obsahuje informace o lokalizaci pro daný komponent. Atribut `lang` obsahuje kód jazyka dle ISO 639-1, atribut `country` obsahuje kód země dle ISO 3166-1 alpha-2. Pokud některý z atributů není uveden, využije se jeho výchozí hodnota (stejně jako v případě neuvedení celého elementu):

```
lang="en"
country="US"
```

Elementy `UserPref` slouží k definici proměnných pro uživatelské nastavení komponentu. Vysvětlení jednotlivých atributů těchto elementů je následující:

- `name`
  - název proměnné
  - může se skládat z písmen anglické abecedy, číslic a podtržítek (regulární výraz: ^[a-zA-Z0-9_]+$)
  - povinný atribut, není-li uveden, element bude ignorován
- `display_name`
  - zobrazovaný název proměnné
  - bude zobrazen jako popis příslušného pole ve formuláři
  - při neuvedení bude využit název proměnné
- `datatype`
  - datový typ proměnné
  - není-li uveden, využije se výchozí hodnota "`string`"

  - možné hodnoty:
    * `string`
      · řetězec
      · ve formuláři pole pro zadání textového řetězce (<`input type="text"`>)
    * `bool`
      · pravdivostní hodnota
      · ve formuláři zaškrtávací políčko
    * `enum`
      · hodnota výčtového typu
      · ve formuláři rozbalovací seznam (combobox), uživatel si vybírá jednu z možností
    * `list`
      · seznam uživatelem poskytovaných textových hodnot
      · ve formuláři dynamická tabulka s 1 sloupcem, ve které lze přidávat a mazat řádky. Prázdné řádky jsou mazány automaticky. Lze přesunovat řádky pomocí myši technikou „táhni a pusť“ (drag and drop). Lze řadit položky podle anglické abecedy.
      · dle potřeby se převádí mezi polem řetězců a řetězcem, ve kterém jsou jednotlivé hodnoty odděleny znakem ”|”
    * `hidden`
      · řetězcová hodnota, která může být využita skriptem
      · nezobrazuje se ve formuláři
      · pokud jsou všechny `UserPref` tohoto typu, negeneruje se formulář

- `required`
  - udává, zda je hodnota vyžadována
  - pokud je hodnota vyžadována, nelze uložit obsah formuláře s nastavením bez uvedení hodnoty

- `default_value`
  - výchozí hodnota proměnné
  - pokud není uveden, využije se prázdný řetězec (u datového typu `bool` hodnota `false`)

Z jednotlivých proměnných definovaných elementy `UserPref` (kromě těch, které mají datový typ `hidden`) je sestaven formulář s uživatelskými nastaveními komponentu. Po kliknutí na ikonku pro nastavení (vedle zavíracího tlačítka) se tento formulář zobrazí uživateli nad tělem komponentu. Po nastavení hodnot může uživatel obsah formuláře uložit (přičemž se formulář opět skryje), nebo formulář skrýt a uvést do stavu před jeho zobrazením. Při uložení formuláře je vyvolána příslušná událost.

Pokud je formulář zobrazen a dojde k programové změně hodnoty proměnné, změna se ihned zobrazí ve formuláři.

Struktura komponentu s vygenerovaným formulářem pro nastavení je odlišná od struktury komponentu bez tohoto formuláře. Komponent bez formuláře je tvořen objektem třídy

`Ext.Panel` a tělo tohoto panelu obsahuje obsah komponentu. Pokud má komponent nastavení, jedná se opět o objekt třídy `Ext.Panel`, ale má nastaveno vnitřní rozložení (anchor layout) a jsou v něm obsaženy další 3 panely: panel s formulářem pro nastavení, oddělovací panel (obsahuje pouze oddělovač) a panel s obsahem komponentu, v jehož těle se obsah komponentu nachází. Pro přístup k obsahu komponentu je tedy vhodné používat metodu `getComponentBody()`, která vždy vrací tělo správného panelu.

Elementy `EnumValue` slouží k definici položek výběru v uživatelské proměnné datového typu `enum`. Jejich atributy jsou:

- `value`
  - hodnota výběru (jedna z možných hodnot proměnné)
  - povinný atribut, při neuvedení bude element ignorován
- `display_value`
  - zobrazovaná hodnota výběru (zobrazí se uživateli v rozbalovacím seznamu)
  - nebude-li uvedena, využije se přímo hodnota výběru

Elementy `msg` slouží k definici lokalizovaných textů pro komponent. Povinný atribut `name` obsahuje název lokalizovaného textu (neuvedení způsobí ignorování elementu), pro jehož syntaxi platí stejná pravidla jako pro stejnojmenný atribut `UserPref`. Textový obsah elementu obsahuje lokalizovaný text. Tyto texty lze využít v titulku a obsahu komponentu, zobrazovaném názvu uživatelské proměnné a zobrazované položce výběru v uživatelské proměnné typu `enum`. Lze je také využít ve skriptech v komponentu, kde je možné získat jejich obsah pomocí metody `getComponentLocalizedText()`.

Element `Locale` a elementy `msg` mohou být uvedeny také v elementu `preferences`, nebo přímo v elementu `component`. Element `messagebundle` je zcela ignorován a je zde uveden pouze pro kompatibilitu s Google Gadgets (analyzují se pouze jeho vnořené elementy).

Pokud je v komponentu uveden element `ModulePrefs` a správce rozložení má zapnutou podporu Google Gadgets, jsou v obsahu komponentu nahrazeny také další řetězce, jejichž popis, včetně náhrad, je uveden výše. Tyto náhrady společně s třídou `KiWiGadgetsWrapper` implementují část Google Gadgets API.

Element `dragging` slouží k povolení či zakázání uchopení komponentu a tažení. Pokud má jeho atribut `enabled` hodnotu "`true`", vynutí zobrazení záhlaví komponentu. Pokud má hodnotu "`false`", komponent nebude možno uchopit a přesunout.

U elementu `content` je analyzován také atribut `view`. Pokud má tento atribut hodnotu "`canvas`" nebo "`preview`" a v komponentu se vyskytuje ještě další element `content`, daný element je ignorován a analyzuje se další element `content`. Tímto se u některých Gadgets přeskočí obsah pro celou stránku nebo náhled a využije se obsah pro Gadget v běžném zobrazení.

A.5 Rekapitulace základů rozhraní pro JavaScript v komponentech

V této podkapitole uvedu základy rozhraní pro JavaScript v komponentech vysvětlené na příkladech. Tyto příklady lze využít pro rychlé zorientování v relativně složitém API.

Na stránce jsou 2 - 3 globální objekty (dle podpory Google Gadgets):

- layoutManager
- observer
- gadgets

Každý komponent je tvořen třídou, která dědí z `Ext.Panel`, a může tedy využívat všechny metody a vlastnosti této třídy kromě těch, které by mohly nevhodně ovlivnit rozložení stránky (např. `setWidth()`).

Svoje id komponent získá pomocí zástupného symbolu `__MY_ID__`:
```
var myId = '__MY_ID__';
```

Přístup ke svému panelu komponent získá následovně:
```
var myPanel = layoutManager.getComponent('__MY_ID__');
```
nebo ekvivalentní zkratkou (textově nahrazena, nelze využít v generovaném kódu):
```
var myPanel = layoutManager.getMyComponent();
```

K obsahu těla komponentu lze přistoupit:
```
layoutManager.getComponentBody('__MY_ID__').dom.innerHTML = 'Obsah';
```
nebo ekvivalentní zkratkou (textově nahrazena, nelze využít v generovaném kódu):
```
layoutManager.getMyComponentBody().dom.innerHTML = 'Obsah';
```

Svoji pozici komponent získá metodou:
```
var myPosition = layoutManager.getComponentPosition('__MY_ID__');
```
nebo ekvivalentní zkratkou (textově nahrazena, nelze využít v generovaném kódu):
```
var myPosition = layoutManager.getMyPosition();
```
a jednotlivé vlastnosti získaného objektu jsou:
```
var myArea = myPosition.area;
var myCol = myPosition.col;
var myRow = myPosition.row;
```

Referenci na svoje uživatelské nastavení komponent získá následovně:
```
var myPreferences = layoutManager.getComponentPreferences('__MY_ID__');
```

Položku svého uživatelského nastavení (nikoliv reference, ale kopie) získá např.:
```
var myName = getComponentPrefsItem('__MY_ID__', "myName");
```
a její datový typ a hodnotu následně získá z vlastností:
```
var myNameType = myName.dataType;
var myNameValue = myName.value;
```

Hodnotu položky uživatelského nastavení může změnit pomocí:
`layoutManager.setComponentPrefsItem('__MY_ID__', "myName", "Nové jméno");`

Obdobně jako sám k sobě může komponent přistoupit i k jiným komponentům na stránce na základě řetězcové hodnoty jejich id.

Kód, který má být proveden po načtení stránky, musí být uveden ve funkci, která je registrována jako reakce na událost "`layoutReady`":

```
observer.addListener("layoutReady", function(event) {
// tělo funkce
})
```

Komponent může měnit hodnoty vlastností stavového objektu observeru:
`observer.update('__MY_ID__', 'nameOfUpdatedProperty', updatedValue);`
a může reagovat na událost jeho aktualizace:

```
layoutManager.registerEventListener("__MY_ID__","stateChanged",function(ev)
{
// tělo funkce
})
```

Komponent může reagovat také na výše uvedené události správce rozložení a může do observeru přidávat i další události, přičemž je nutno dbát na to, aby nedošlo k interferenci komponentů. Více o přidání nové události viz `Ext.util.Observable`.

Příloha B

Požadavky na server

Správce rozložení, observer a skripty pro demonstraci jejich činnosti jsou uloženy v následujících souborech a adresářích:

- index.php
  - skript, který z požadavku prohlížeče a parametru v URL zjistí požadovaný jazyk a vygeneruje indexovou stránku webu.
  - předpokládá se, že bude nahrazen skriptem, který bude sloužit ke stejnému účelu, ale bude spolupracovat s jinými částmi serveru a poskytne tak širší funkcionalitu (např. změna jazyka při přihlášení uživatele)
- locale
  - adresář s lokalizačními skripty
  - V názvu každého lokalizačního skriptu je kód jazyka dle ISO 639-1 (zde nahrazen XX). Lokalizační skripty jsou následující:
    - layout-lang-XX.js
      - skript pro lokalizaci správce rozložení
    - layout-lang-XX.php
      - skript pro lokalizaci indexové stránky
  - Skripty pro angličtinu jsou výchozí a musejí být vždy dostupné. Minimální obsah adresáře je tedy následující:
    - layout-lang-en.js
    - layout-lang-en.php
- initialization.js
  - skript pro inicializaci stránky, který deklaruje globální proměnné, inicializuje TinyMCE, a po inicializaci Ext JS skryje informace pro uživatele bez JavaScriptu a vytvoří objekty observeru a správce rozložení. Správci rozložení nastaví také zdrojovou adresu a zavolá jeho metodu pro načtení obsahu stránky.
  - před využitím je nutná jeho úprava, při které musí být aktualizována adresa, ze které bude načítáno konfigurační XML
  - server jej může generovat i dynamicky, a optimalizovat tak práci s TinyMCE apod.

- KiWiGadgetsWrapper.js
  - skript definující třídy `gadgets`, které slouží pro zajištění funkčnosti Google Gadgets
  - nyní má implementovány třídy `MiniMessage`, `Prefs` a `util`. Metody těchto tříd využívají Ext JS a ve velké míře pouze obalují metody správce rozložení.
  - pro jeho využití musí mít správce rozložení zapnutou podporu Google Gadgets (nastaveno v souboru initialization.js)
- KiWiLayoutManager.css
  - stylový předpis pro některé prvky generované správcem rozložení
- KiWiLayoutManager.js
  - třída správce rozložení
- KiWiLayoutReadyTrigger.js
  - skript pro vyvolání události `layoutReady` (jeho separace od správce rozložení řeší problémy s některými prohlížeči)
- KiWiObserver.js
  - třída observeru
- PortalColumn.js
  - sloupec portálu, plně převzato z Ext JS (ext-2.1/examples/portal/)
  - lze upravit index.php a využít verzi uloženou přímo v knihovně Ext JS (u nové verze Ext JS nutno zkontrolovat funkcionalitu).
- portal.css
  - stylový předpis portálu, plně převzato z Ext JS (ext-2.1/examples/portal/)
  - lze upravit index.php a využít verzi uloženou přímo v knihovně Ext JS
- Portal.js
  - portál – panel se sloupci umožňující přesuny komponentů. Je využit jako součást správce rozložení. Plně převzato z Ext JS (ext-2.1/examples/portal/)
  - lze upravit index.php a využít verzi uloženou přímo v knihovně Ext JS (u nové verze Ext JS nutno zkontrolovat funkcionalitu)
- Portlet.js
  - panel komponentu, plně převzato z Ext JS (ext-2.1/examples/portal/)
  - lze upravit index.php a využít verzi uloženou přímo v knihovně Ext JS (u nové verze Ext JS nutno zkontrolovat funkcionalitu)
- progress-bar.css
  - stylový předpis pro progressbar, plně převzato z Ext JS (ext-2.1/examples/sample-widgets/)

- styles.css
  - stylový předpis stránky, upraven z verze v Ext JS (ext-2.1/examples/portal/)
- test.php
  - skript, který zasílá testovací konfigurační XML správci rozložení
  - určen pro demonstraci činnosti
  - jeho funkcionalitu nahradí jiná část serveru

Na serveru musí být umístěny výše uvedené soubory a adresáře (kromě .php souborů, které pravděpodobně budou nahrazeny) a následující adresáře:

- ext-2.1
  - knihovna Ext JS, pokud bude novější verze, nutno aktualizovat obsah index.php, nebo jeho náhrady
  - nutno instalovat včetně examples (využity sdílené obrázky, lze umístit jinam a upravit KiWiLayoutManager.css)
  - lze stáhnout z: http://extjs.com/products/extjs/download.php
  - nutno rozšířit o Ext.ux.TinyMCE. Toto rozšíření lze stáhnout z http://extjs.com/forum/showthread.php?t=24787, postup jeho instalace je následující:
    - stažení souborů Ext.ux.TinyMCE_v06.zip a Ext.ux.TinyMCE_demo_v06.zip
    - vybalení obsahu Ext.ux.TinyMCE_v06.zip do ./ext-2.1/ux/
    - vybalení následujících souborů z Ext.ux.TinyMCE_demo_v06.zip:
      - lib/ext/ux/miframe.js do ./ext-2.1/ux/
      - lib/ext/ux/miframe-min.js do ./ext-2.1/ux/
- tiny_mce
  - editor TinyMCE
  - lze vybalit z Ext.ux.TinyMCE_demo_v06.zip nebo stáhnout z http://tinymce.moxiecode.com/download.php (jazykové soubory je v obou případech nutno stáhnout z uvedené adresy)
  - Rovněž je možné správce rozložení provozovat bez podpory TinyMCE. V tomto případě je nutno nejenom odebrat příslušné řádky hlavičky indexové stránky, ale také odebrat příslušný řádek z initialization.js

Server musí generovat konfigurační XML a indexovou stránku, která musí být v kódování UTF-8 a v hlavičce musí mít nejméně níže uvedený obsah. Pořadí řádků v uvedeném obsahu indexové stránky je nutné zachovat. Ke generování indexové stránky lze využít soubor index.php, který je součástí projektu, nebo k tomuto účelu vytvořit vlastní skript.

B.1 Příklad vygenerované indexové stránky s detailním popisem

```
1  <head>
2    <title>Titulek stránky</title>
3    <meta http-equiv="content-language" content="cs"/>
4    <meta http-equiv="content-type" content="text/html;
5          charset=UTF-8"/>
6    <link rel="stylesheet" type="text/css"
7          href="./ext-2.1/resources/css/ext-all.css" />
8    <!-- GC -->
9      <!-- LIBS -->
10     <script type="text/javascript"
11           src="./ext-2.1/adapter/ext/ext-base.js"></script>
12   <!-- ENDLIBS -->
13   <script type="text/javascript"
14         src="./ext-2.1/ext-all.js"></script>
15   <script type="text/javascript"
16         src="./ext-2.1/ux/miframe-min.js"></script>
17   <script type="text/javascript"
18         src="./tinymce/jscripts/tiny_mce/tiny_mce.js"></script>
19   <script type="text/javascript"
20         src="./ext-2.1/ux/Ext.ux.TinyMCE.min.js"></script>
21   <script type="text/javascript"
22         src="./ext-2.1/build/locale/ext-lang-cs-min.js"
23         ></script>
24   <script type="text/javascript" src="Portal.js"></script>
25   <script type="text/javascript" src="PortalColumn.js"></script>
26   <script type="text/javascript" src="Portlet.js"></script>
27   <link rel="stylesheet" type="text/css" href="portal.css" />
28   <link rel="stylesheet" type="text/css"
29         href="KiWiLayoutManager.css" />
30   <link rel="stylesheet" type="text/css" href="styles.css" />
31   <link rel="stylesheet" type="text/css"
32         href="progress-bar.css" />
33   <script type="text/javascript"
34         src="KiWiObserver.js"></script>
35   <script type="text/javascript"
36         src="KiWiLayoutManager.js"></script>
37   <script type="text/javascript"
38         src="./locale/layout-lang-cs.js"></script>
39   <script type="text/javascript"
40         src="KiWiGadgetsWrapper.js"></script>
41   <script type="text/javascript"
42         src="initialization.js"></script>
43 </head>
44 <body>
```

```
45    <div id="layoutForUsersWihnoutJS">
46      Text pro uživatele bez podpory JavaScriptu.
47    </div>
48    <div id="layoutLoading">
49      <h1>Nadpis stránky</h1><br/>
50      Stránka se načítá, čekejte prosím ...<br/><br/>
51      <div id="layoutLoadingProgress"></div>
52    </div>
53  </body>
```

Na řádku 6 je ke stránce připojen stylový předpis knihovny Ext JS a za ním následují jednotlivé skripty této knihovny. Na řádku 15 je rozšíření potřebné pro Ext.ux.TinyMCE, jehož skript se nachází na řádku 19. Jádro TinyMCE je vloženo na řádku 17.

Na řádku 21 je lokalizační skript pro knihovnu Ext JS. V uvedeném příkladu je česká lokalizace. Pokud by tento skript nebyl uveden, využije se výchozí jazyk, kterým je u Ext JS angličtina.

Na řádcích 24 – 27 jsou součásti rozšíření Ext.ux.Portal převzaté z příkladu Portal v knihovně Ext JS. Toto rozšíření je využíváno správcem rozložení.

Na řádku 28 je připojen stylový předpis správce rozložení, za ním následuje stylový předpis pro stránku a stylový předpis pro indikátor průběhu načítání (progressbar). Po těchto stylových předpisech následuje skript s observerem a skript se správcem rozložení.

Na řádku 37 je lokalizační skript pro správce rozložení. V uvedeném příkladu je česká lokalizace, pokud by tento skript nebyl uveden, využijí se výchozí anglické texty.

Na řádku 39 je umístěn skript s obalujícími třídami pro kompatibilitu s Google Gadgets a za ním následuje skript pro inicializaci stránky.

Tělo stránky musí vždy obsahovat nejméně 2 - 3 elementy (dle nastavení), jejichž význam je následující:

- `<div id="layoutForUsersWihnoutJS"/>`
  - element s informacemi pro uživatele bez podpory JavaScriptu
  - bude automaticky skryt ihned po inicializaci Ext JS
- `<div id="layoutLoading"/>`
  - element s informacemi zobrazenými v průběhu načítání stránky
  - bude automaticky skryt před koncem sestavování stránky (ve chvíli, kdy se bude nacházet v pozadí)
- `<div id="layoutLoadingProgress"/>`
  - element, ve kterém bude při načítání stránky zobrazen indikátor průběhu načítání (progressbar)
  - musí být umístěn v elementu `<div id="layoutLoading"/>`
  - musí být prázdný
  - pokud je zobrazení indikátoru průběhu načítání deaktivováno nastavením vlastnosti `showLoadingProgress` na `false`, nemusí být uveden

Veškeré další elementy přidané do těla stránky se musejí nacházet ve výše uvedených elementech. Mimo tyto elementy nesmí být v těle stránky žádný jiný obsah.

Příloha C

Příklad zprovoznění editoru TinyMCE v komponentu

Základ následujícího příkladu je převzat z http://extjs.com/forum/showthread.php?t=24787.

```
<component id="123460" location="center" col="1" row="1">
  <title>Titulek v záhlaví komponentu</title>
  <content type="text/html">
  <![CDATA[
    <script type="text/javascript">
    layoutManager.getMyComponent().add(new Ext.form.FormPanel({
      autoHeight: true,
      autoScroll: true,
      items: [{
        xtype: "tinymce",
        fieldLabel: "Rich text",
        hideLabel:true,
        id: "richText",
        name: "richText",
        width: 600,
        height: 400,
        tinymceSettings: {
          theme : "advanced",
          plugins: "safari,pagebreak,style,layer,table,advhr,
            advimage,advlink,emotions,iespell,insertdatetime,
            preview,media,searchreplace,print,contextmenu,paste,
            directionality,noneditable,visualchars,nonbreaking,
            xhtmlxtras,template",
          theme_advanced_buttons1 : "bold,italic,underline,
            strikethrough,|,justifyleft,justifycenter,
            justifyright,justifyfull,|,styleselect,formatselect,
            fontselect,fontsizeselect",
          theme_advanced_buttons2 : "cut,copy,paste,pastetext,
            pasteword,|,search,replace,|,bullist,numlist,|,
            outdent,indent,blockquote,|,undo,redo,|,link,unlink,
```

```
        anchor,image,cleanup,help,code,|,insertdate,
        inserttime,preview,|,forecolor,backcolor",
      theme_advanced_buttons3 : "tablecontrols,|,hr,
        removeformat,visualaid,|,sub,sup,|,charmap,emotions,
        iespell,media,advhr,|,print,|,ltr,rtl,|",
      theme_advanced_buttons4 : "insertlayer,moveforward,
        movebackward,absolute,|,styleprops,|,cite,abbr,
        acronym,del,ins,attribs,|,visualchars,nonbreaking,
        template,pagebreak",
      theme_advanced_toolbar_location : "top",
      theme_advanced_toolbar_align : "left",
      theme_advanced_statusbar_location : "bottom",
      theme_advanced_resizing : false,
      extended_valid_elements : "a[name|href|target|title|
        onclick],img[class|src|border=0|alt|title|hspace|
        vspace|width|height|align|onmouseover|onmouseout|
        name],hr[class|width|size|noshade],font[face|size|
        color|style],span[class|align|style]",
      template_external_list_url : "example_template_list.js"
    },
    value: "<h1>Demo</h1><p>Ext.ux.TinyMCE works...</p>"
  }
  ]
}),0);
// po přidání panelu s formulářem do panelu komponentu
// je třeba přepočítat rozložení (aby byl viditelný)
layoutManager.getMyComponent().doLayout();
// po inicializaci stránky se zaregistrují reakce na události
observer.addListener("layoutReady", function(event) {
  if (Ext.isIE6)
  {  // v IE6 se problémy nevyskytují, následující kód
     // fungující od verze 7 tohoto prohlížeče by je vytvořil
    return;
  }
  layoutManager.getComponent('__MY_ID__').addListener(
    'beforecollapse',function(){
      tinyMCE.execCommand('mceRemoveControl',false,
                          tinyMCE.activeEditor.id);
  });
  layoutManager.getComponent('__MY_ID__').addListener(
    'collapse',function(){
      tinyMCE.execCommand('mceAddControl',false,
                          tinyMCE.activeEditor.id);
  });
  layoutManager.getComponent('__MY_ID__').addListener(
    'beforeexpand',function(){
      tinyMCE.execCommand('mceRemoveControl',false,
                          tinyMCE.activeEditor.id);
```

```
        });
        layoutManager.getComponent('__MY_ID__').addListener(
          'expand',function(){
            tinyMCE.execCommand('mceAddControl',false,
                                tinyMCE.activeEditor.id);
        });
        layoutManager.getComponent('__MY_ID__').dd.onMouseDown
          = function(C){  // před tažením komponentu
            tinyMCE.execCommand('mceRemoveControl',false,
                                tinyMCE.activeEditor.id);
        };
        layoutManager.getComponent('__MY_ID__').dd.onMouseUp
          = function(C){  // po tažení komponentu
            tinyMCE.execCommand('mceAddControl',false,
                                tinyMCE.activeEditor.id);
        };
        layoutManager.registerEventListener("__MY_ID__",
            "layoutChanged", function(event) {
          if (layoutManager.getComponentLocation('__MY_ID__').area
            != event.areaLocation)
          {  // pokud se událost týká jiné oblasti
            return;
          }
          if (event.action == "areaWillBeCollapsed"){
            tinyMCE.execCommand('mceRemoveControl',false,
                                tinyMCE.activeEditor.id);
          }
          else if (event.action == "areaCollapsed"){
            tinyMCE.execCommand('mceAddControl',false,
                                tinyMCE.activeEditor.id);
          }
          else if (event.action == "areaWillBeExpanded"){
            tinyMCE.execCommand('mceRemoveControl',false,
                                tinyMCE.activeEditor.id);
          }
          else if (event.action == "areaExpanded"){
            tinyMCE.execCommand('mceAddControl',false,
                                tinyMCE.activeEditor.id);
          }
        });
      });
      </script>
    ]]>
    </content>
</component>
```

Pro jednoduchost a přehlednost příkladu jsem pro deaktivaci a opětovnou aktivaci editoru využil metody, které předpokládají přítomnost pouze jednoho editoru na stránce. Pro možnost využití více editorů na stránce je nutné řešení upravit.

V Microsoft Internet Exploreru verze 6 se ošetření problémů při přesunu komponentu neprovádí (na události se nereaguje). V tomto prohlížeči při přesunu komponentu problémy nevznikají a řešení pro jiné prohlížeče zde nefunguje.

Pro přehlednou rekapitulaci uvádím rovněž seznam a sémantiku událostí, na které musí komponent reagovat deaktivacemi a aktivacemi editoru:

- `beforecollapse`
  - událost panelu komponentu, ke které dojde před skrytím těla panelu
- `collapse`
  - událost panelu komponentu, ke které dojde po skrytí těla panelu
- `beforeexpand`
  - událost panelu komponentu, ke které dojde před zobrazením těla panelu
- `expand`
  - událost panelu komponentu, ke které dojde po zobrazení těla panelu
- `onMouseDown`
  - metoda uchopovacího objektu panelu komponentu, která je volána při stisknutí tlačítka myši
  - původně prázdná metoda je předefinována, nutno zachovat pouze návratovou hodnotu (žádná).
- `onMouseUp`
  - metoda uchopovacího objektu panelu komponentu, která je volána při uvolnění tlačítka myši
  - původně prázdná metoda je předefinována, nutno zachovat pouze návratovou hodnotu (žádná).
- `layoutChanged`
  - událost správce rozložení, ke které dojde při změně rozložení stránky
  - vhodné reagovat pouze na události týkající se oblasti s editorem
  - nutno reagovat na následující hodnoty vlastnosti „`action`“ objektu předaného jako parametr události:
    - `areaWillBeCollapsed` – tělo oblasti bude skryto (obdoba beforecollapse)
    - `areaCollapsed` – tělo oblasti skryto (obdoba collapse)
    - `areaWillBeExpanded` – tělo oblasti bude zobrazeno (obdoba beforeexpand)
    - `areaExpanded` – tělo oblasti zobrazeno (obdoba expand)