VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

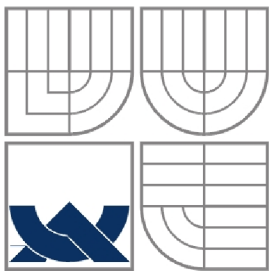
UNIVERZÁLNÍ PRACOVNÍ PLOCHA VE WEBOVÉM PROHLÍŽEČI

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

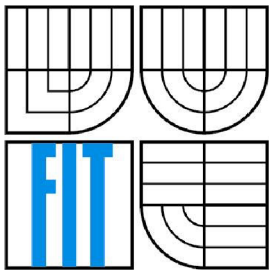
AUTOR PRÁCE
AUTHOR

Bc. LUKÁŠ MÁČEL

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

UNIVERZÁLNÍ PRACOVNÍ PLOCHA VE WEBOVÉM PROHLÍŽEČI

UNIVERSAL WORKING DESKTOP IN WEB BROWSER

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Lukáš Máčel

VEDOUCÍ PRÁCE
SUPERVISOR

prof. Ing. Tomáš Hruška, CSc.

BRNO 2009

Abstrakt

Tato diplomová práce se zabývá analýzou pracovních ploch dostupných pomocí webového prohlížeče. Popisuje technologie potřebné pro tvorbu webového rozhraní, mezi které patří DOM, událostní model dokumentu a AJAX. Součástí práce je také návrh prototypu pracovní plochy. Pracovní plocha je realizována jako webová aplikace založená na architektuře klient-server. Klientskou část aplikace reprezentuje webové rozhraní pro správu odkazů na dokumenty umístěných na pracovní ploše. Obsah pracovní plochy je členěn pomocí panelů a záložek. Uživatel/ka může měnit uspořádání odkazů prostřednictvím techniky drag & drop. Serverová část aplikace zajišťuje persistenci konfigurace pracovní plochy a vytváření nabídky odkazů. Prototyp pracovní plochy je implementován pomocí javascriptové knihovny Dojo.

Abstract

This master thesis is engaged in analyzing working desktop accessible from internet by web browser. It describes technologies required for building web interfaces such as DOM, document event model and AJAX. This master thesis contains also design of working desktop prototype. The desktop is implemented as web application based on client-server architecture. The client part of application represents web interface for managing document links placed on desktop. The desktop content is classified by panels and tabs. User can change link arrangement using drag & drop. Server part of application guarantees persistence of desktop configuration and making link offer. Working desktop prototype is implemented by Javascript library Dojo.

Klíčová slova

AJAX, Dojo, DOM, drag & drop, miniaplikace, pracovní plocha

Keywords

AJAX, Dojo, DOM, drag & drop, gadget, working desktop

Citace

Lukáš Máčel: Univerzální pracovní plocha ve webovém prohlížeči, diplomová práce, Brno, FIT VUT v Brně, 2009

Název diplomové práce v jazyce práce

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením prof. Ing. Tomáše Hrušky, CSc.

Další informace mi poskytl Ing. Michal Máčel, CSc.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Lukáš Máčel
21.5.2009

Poděkování

Na tomto místě bych chtěl poděkovat vedoucímu mojí diplomové práce prof. Ing. Tomáši Hruškovi, CSc. za zájem, připomínky a čas, který věnoval mé práci. Rovněž bych rád poděkoval Slávce a všem mým blízkým za vyjádřenou podporu.

Obsah

1 Úvod

V reálném životě představuje pracovní plocha místo, na kterém jsou uloženy dokumenty a další pomůcky potřebné pro práci. Její hlavní přínos vidím ve shromáždění pracovních podkladů, které jsou díky pracovní ploše rychle dostupné, a tak usnadňují pracovní proces.

Cíl mé práce spočívá v návrhu webové aplikace, která bude sloužit jako pracovní plocha dostupná z libovolného místa, jež je připojené k internetu. Pracovní plocha umožní uživateli vytvořit a uchovávat odkazy na zajímavé dokumenty a webové stránky. Její součástí budou také miniaplikace v podobě kalkulačky a kalendáře. Aplikace nabídne uživateli/uživatelce přehlednou organizaci obsahu a snadné přemisťování odkazů a miniaplikací pomocí techniky drag & drop.

V kapitole 2 nejdříve přiblížím pojem pracovní plocha a zmíním možnosti, které přináší přístup k pracovní ploše prostřednictvím webového prohlížeče. Dále se budu zabývat architekturou aplikace pracovní plochy. Na závěr kapitoly provedu analýzu současných webových pracovních ploch a uvedu jejich základní charakteristiku.

V kapitole 3 uvedu prostředky, které lze využít pro tvorbu webového rozhraní a reprezentaci dat. Nejdříve stručně popíši jazyky HTML, Javascript a CSS. Dále se budu zabývat rozhraním objektového modelu dokumentu (DOM). Přiblížím rovněž událostní model zahrnující vznik a šíření událostí. Dále zmíním techniku označovanou AJAX, která dovoluje kladení asynchronních požadavků. Zmíním také textový formát JSON sloužící pro výměnu dat. Závěrem kapitoly se budu zabývat problémy spojenými s implementací pod různými webovými prohlížeči.

Obsahem kapitoly 4 bude návrh prototypu pracovní plochy. Na začátku uvedu základní služby poskytované aplikací. Potom popíši schéma webového rozhraní pracovní plochy. Navrhnu také způsob rozložení prvků pracovní plochy. Dále uvedu definici akcí, které bude moci uživatel na rozhraní aplikace provádět, a podrobně rozeberu nastavení aplikace. Přiblížím rovněž princip techniky drag & drop a možnosti, které tato technika nabízí.

V kapitole 5 se budu zabývat úlohou konfiguračního serveru. Popíši způsob, jakým server vytváří nabídku odkazů pro postranní menu aplikace. Dále navrhnu strukturu konfigurace pracovní plochy a nabídky odkazů. Zaměřím se rovněž na návrh komunikačního protokolu mezi serverem a klientem.

Kapitolu 6 budu věnovat popisu javascriptové knihovny Dojo. Nejdříve uvedu hlavní důvody, které mě vedly k výběru knihovny. Poté postupně popíši základní služby knihovny. Zaměřím se především na vysvětlení konceptu widgetů a jejich využití při vytváření webového rozhraní.

V kapitole 7 se budu věnovat implementaci rozhraní pracovní plochy. Přiblížím realizaci všech důležitých komponent rozhraní. Pokusím se přitom maximálně využít prostředků, které nabízí knihovna Dojo.

Náplní kapitoly 8 bude řešení správy konfigurace. Nejprve popíši způsob načítání a ukládání konfigurace na klientovi, a potom rozeberu implementaci serverové části aplikace. Naváži tématem autentizace uživatelů/uživatelek, správy konfiguračních souborů a rovněž vytváření nabídky odkazů.

V poslední kapitole 9 shrnu návrh pracovní plochy a způsob její implementace. Zamyslím se nad přínosem předkládané práce a zmíním také další možnosti rozšíření aplikace.

1.1 Značení v textu

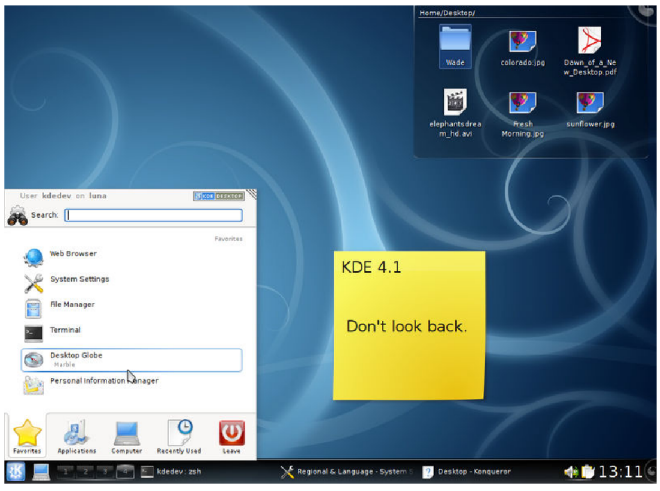
Názvy tříd spolu s jejich metodami a atributy označím kurzívou. Důležité pojmy a tvrzení zvýrazním tučným písmem. Příklady v textu budou psány neproporcionálním písmem. Pro lepší orientaci uvedu klíčová slova jazyka Javascript modrou barvou, komentáře příkladů zelenou barvou, řetězce barvou hnědou a fialovou barvou budou psány konstrukce z knihovny Dojo.

2 Univerzální pracovní plocha

2.1 Definice pracovní plochy

V případě většiny počítačů se dnes můžeme setkat s desktopovým prostředím (desktop environment), které představuje intuitivní grafické rozhraní pro interakci uživatele s počítačem [1]. Mezi známá prostředí patří například desktopové prostředí operačního systému Windows nebo desktopové prostředí GNOM a KDE pro operační systém UNIX.

Základní myšlenka desktopového prostředí spočívá v tom, že obrazovka počítače reprezentuje pracovní plochu (desktop) [2]. Stejně jako v reálném světě, kde jsou na pracovní ploše potřebné dokumenty a pomůcky (kalkulačka, poznámkový blok aj.), také obrazovka počítače nabízí prostor pro umístění odkazů na dokumenty a složky nacházející se na disku počítače. Odkazy jsou graficky reprezentovány ikonou s názvem dokumentu nebo složky. Uživatel může pomocí kurzoru myši otevřít vybraný odkaz, a zobrazit si tak obsah dokumentu nebo složky v novém okně [2]. Součástí plochy mohou být také jednoduché aplikace kalkulačky nebo poznámkového bloku.



Obr. 2-1 Desktopové prostředí KDE

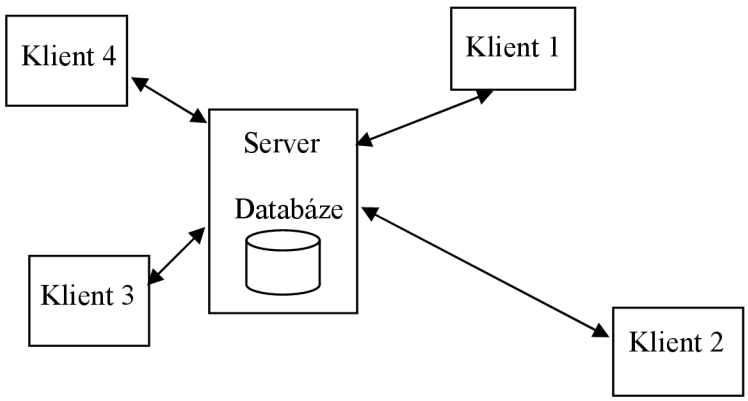
Oproti reálné pracovní ploše nabízí pracovní plocha počítače daleko větší možnosti pro její pohodlné ovládání. Mohou se zde vyskytovat odkazy pro spuštění často používaných aplikací nebo odkazy na internetové stránky, které se automaticky zobrazí pomocí webového prohlížeče. Jiným příkladem je odkaz na odpadkový koš (trash bin) nebo pomocná menu odkazů a lišty zobrazující právě spuštěné aplikace.

Přestože je obsah pracovní plochy počítače odlišný od toho, co lze najít na reálné pracovní ploše, princip využití pracovní plochy zůstává stejný. Uživatel má možnost shromáždit na jednom místě všechny potřebné dokumenty a další nástroje, které ke své práci potřebuje, za účelem jejich okamžité dostupnosti. Právě díky analogii mezi skutečnou pracovní plochou a uživatelským rozhraním plochy v počítači se desktopové prostředí stalo mezi uživateli nejoblíbenější alternativou prostředí příkazové řádky (command-line environment) [1], které zajišťuje interakci s operačním systémem počítače nebo jiným softwarem pomocí psaní příkazů, jejichž smyslem je provedení požadované úlohy [8].

2.2 Pracovní plocha dostupná z webu

Pracovní plocha, kterou jsem popsal v předchozí kapitole, je použitelná pouze na lokálním počítači. Cílem mé práce je zabývat se pracovními plochami přístupnými z libovolného místa, z něhož je možné připojit se na internet, které dovolují správu dokumentů a použití aplikací pouze pomocí webového prohlížeče.

Vzdálené spuštění aplikací na lokálním počítači z jiného počítače ve veřejné síti nelze z bezpečnostních důvodů realizovat. Přesto je však možné pracovní plochu dostupnou z webu dobře využít. Na ploše mohou být umístěny zajímavé odkazy na zdroje z internetu formou hypertextových odkazů. Zajistit lze přístup k dokumentům a obrazovým nebo hudebním souborům. Nic také nebrání tomu, aby součástí pracovní plochy byly rovněž jednoduché aplikace, které nabídnou uživateli/uživatelce služby ve formě kalkulačky, kalendáře, hodin nebo například poznámkového bloku. Díky takovéto pracovní ploše má uživatel/ka neustále k dispozici shromážděné odkazy a základní aplikace potřebné k práci, i když právě s sebou nemá osobní počítač.



Obr. 2-2 Vztah klienta a serveru

Výše popsaná pracovní plocha představuje webovou aplikaci. Tato aplikace se skládá ze dvou základních částí – klienta a serveru (viz Obr. 2-2). Klientská část aplikace nabízí kompletní uživatelské rozhraní pro interakci s pracovní plochou. Pro její spuštění není nutné instalovat další software. Uživatel/ka potřebuje pouze webový prohlížeč. Pracovní plochu může nastavit jako výchozí stránku webového prohlížeče, čímž získá rychlý přístup ke všem odkazům a aplikacím, které pracovní plocha nabízí.

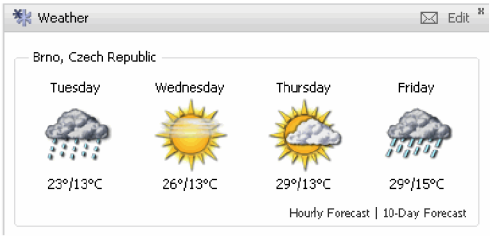
Druhou část aplikace představuje server. K jednomu serveru může být připojeno více klientů. Základní úlohou serveru je správa databáze obsahující konfigurace jednotlivých pracovních ploch. Konfigurace popisuje obsah pracovní plochy, tj. jaké odkazy a aplikace se na ploše nacházejí. Její součástí je také specifikace rozmístění obsahu a další uživatelská nastavení. Server zajišťuje, že se všechny modifikace, které uživatel na pracovní ploše provede, uloží do databáze. Při dalším otevření plochy ve webovém prohlížeči bude obsah a jeho rozmístění odpovídat poslednímu uloženému stavu.

2.3 Současné pracovní plochy

Mezi současné pracovní plochy dovolující sdílet odkazy na různé webové zdroje a služby, které jsem prostudoval, patří:

- Netvibes (http://www.netvibes.com),
- iGoogle (http://www.google.com/ig),
- Protopage (http://protopage.com),
- Pageflakes (http://www.pageflakes.com),
- Eskobo (http://www.eskobo.com).

Všechny uvedené pracovní plochy reprezentují svůj obsah prostřednictvím **miniaplikací** (gadgets). Miniaplikace představuje samostatnou komponentu, která disponuje vlastním nastavením a slouží pro zobrazení obsahu zvoleného webového zdroje nebo nabízí rozhraní jednoduché aplikace (kalkulačka, poznámkový blok, kalendář atd.). Komponentu zajišťující zobrazení předpovědi počasí ukazuje Obr. 2-3. Rozhraní miniaplikace představuje panel skládající se z titulkové lišty (caption bar) a oblasti pro zobrazení obsahu. Součástí lišty jsou kromě ikony a titulku také tlačítka zpřístupňující další volby panelu. Mezi nejčastější operace, které lze s panelem provádět, patří:



Obr. 2-3 Miniaplikace pro předpověď počasí

- Zobrazení nastavení panelu – specifikace URL zdroje, parametry ovlivňující zobrazení, další volby charakteristické pro danou miniaplikaci.
- Odstranění panelu z pracovní plochy.
- Aktualizace obsahu panelu.
- Minimalizace zobrazení panelu jenom na oblast titulkové lišty.

Výše zmíněné pracovní plochy nabízí širokou paletu miniaplikací, jejichž seznam je obsažen v menu aplikace, prostřednictvím kterého lze komponenty na plochu umisťovat. Mezi nejfrekventovanější patří miniaplikace pro zobrazení RRS kanálů, předpovědi počasí nebo miniaplikace sloužící k vyhledávání v obrazových galeriích nebo databázích obsahujících video soubory. Nechybí ani miniaplikace kalendáře, kalkulačky a poznámkového bloku. Kromě využití stávajících komponent je možné také vytvářet nové miniaplikace a sdílet je s ostatními uživateli pracovní plochy.

Pracovní plochy *Netvibes* a *iGoogle* standardizují programové rozhraní miniaplikace. Podle [3] a [8] miniaplikace představuje obecnou webovou komponentu, která je založena na HTML, CSS a Javascriptu. Pro její definici slouží konfigurační soubor ve formátu XML, na jehož základě potom aplikace pracovní plochy vytvoří grafickou reprezentaci formou panelu. Při porovnání specifikace

miniaplikace podporované v pracovních plochách *Netvibes* a *iGoogle* uvedených na [3] a [8] je patrné, že konfigurační soubor v obou případech obsahuje tři základní části:

1. metainformace,
2. uživatelské volby,
3. obsah miniaplikace.

Metainformace zahrnují údaje o komponentě, tj. její název, stručný popis, popřípadě jméno autora. Jejich využití spočívá ve vyhledávání miniaplikací v rámci katalogu nabízených komponent. Uživatelské volby potom specifikují, jaká nastavení může uživatel/ka ovlivnit. Pracovní plocha, která chce takovou miniaplikaci používat, musí zajistit zobrazení příslušného dialogu, pomocí něhož může uživatel/ka volby nastavit. Poslední a nejdůležitější částí miniaplikace je její obsah. Definuje rozhraní miniaplikace vytvořené pomocí jazyka HTML a také její chování implementované v jazyce Javascript.

Kromě vložení/odstranění miniaplikace nabízejí uvedené pracovní plochy také **správce rozložení**, který dovoluje komponenty přehledně uspořádat. Obr. 2-4 zobrazující webové rozhraní pracovní plochy *Protopage* demonstruje typický způsob organizace obsahu, se kterým jsem se setkal také u ostatních pracovních ploch.

Obr. 2-4 Rozhraní pracovní plochy *Protopage*

Pracovní plocha dovoluje definovat záložky (tabs), prostřednictvím kterých lze panely rozdělit do skupin, například podle toho, jaký obsah zobrazují. Standardně jsou dostupné operace přidání a odebrání záložky. Tlačítko pro výběr záložky obsahuje titulek záložky, případně ikonu. Obsah záložky je pak organizován do sloupců, jejichž počet se nastavuje prostřednictvím voleb záložky. Pracovní plochy *Protopage, Pageflakes* a *iGoogle* umožňují umístit panel přes více sloupců (viz Obr.

2-4). U pracovních ploch *Netvibes* takové uspořádání není podporováno. Místo toho lze však přizpůsobit šířku sloupce pomocí posuvníku nacházejícím se mezi dvěma po sobě jdoucími sloupci.

Důležitou operací pracovní plochy je reorganizace obsahu pomocí techniky drag & drop. Ta spočívá v možnosti uchopit panel prostřednictvím kurzoru myši a přemístit celou komponentu na novou pozici v rámci pracovní plochy. Panely se dají přesouvat mezi sloupci, ale také mezi záložkami. Techniku drag & drop lze uplatnit rovněž na tlačítka záložek, a tak změnit pořadí jejich zobrazení.

Veškerá nastavení a způsob organizace plochy se perzistentně ukládá na server. Pracovní plochy dovolují zřídit uživatelský účet, prostřednictvím kterého má uživatel/ka přístup ke své soukromé ploše.

Pracovní plochy, které jsem analyzoval, kladou velký důraz na grafickou prezentaci výsledného rozhraní a nabízí možnost upravit vzhled pracovní plochy prostřednictvím grafického tématu. Grafické téma představuje kolekci barev, obrázků a fontů písma definujících zobrazení výsledné aplikace. Diskutované pracovní plochy specifikují buď několik předdefinovaných grafických témat, nebo dovolují vytvářet na základě výběru barev témata nová.

Mezi další charakteristické služby pracovních ploch patří také podpora internacionalizace. Aplikace *Netvibes* a *iGoogle* nabízí lokalizaci rozhraní do více jazyků, mezi kterými je možné prostřednictvím dialogu globálního nastavení přepínat.

Porovnávané webové aplikace realizující pracovní plochu jsou si z hlediska návrhu rozhraní a základních služeb velmi podobné. Základ rozhraní vždy tvoří kontejner pro umístění miniaplikací, které jsou reprezentovány formou panelu. Každá miniaplikace má svoje vlastní nastavení a zobrazuje určitý obsah. Aplikace realizuje správce rozložení, který organizuje obsah pomocí záložek a dále rozmisťuje jednotlivé panely do sloupců. Celkový vzhled aplikace je možné upravit pomocí grafických témat a výsledné rozhraní lze přeložit do vybraného jazyka. Zjištěné poznatky popsané v předcházejícím textu využiji při návrhu prototypu vlastní pracovní plochy, který bude obsahem kapitoly 4.

3 Prostředky pro tvorbu webového rozhraní a prezentaci dat

3.1 HTML, Javascript a CSS

Pro tvorbu webového rozhraní lze využít tři základní jazyky, mezi které patří:

1. HTML pro definici hypertextového dokumentu,
2. Javascript – pro dynamickou manipulaci s dokumentem,
3. CSS – pro definici vzhledu dokumentu.

Uvedu pouze stručnou charakteristiku každého jazyka a připojím odkaz na jeho kompletní specifikaci.

HTML (HyperText Markup Language) představuje značkovací jazyk, který je aplikací jazyka SGML (Standard Generalized Markup Language). HTML nabízí množinu značek, prostřednictvím kterých lze označit část textu dokumentu a přidělit jí určitý význam. Část složená ze startovací značky, příslušného obsahu a ukončovací značky reprezentuje prvek dokumentu. Používání značek se řídí předepsanou strukturou. Celý dokument je reprezentován prvkem *html*, který obsahuje prvek hlavičky *title* a prvek těla *body*. Z dalších prvků důležitých z pohledu tvorby webového rozhraní jmenuji univerzální prvek *div* pro definici rozložení obsahu, prvek tabulky *table* nebo dialogové prvky jako tlačítko (*button),* textové pole (*input*) či roletovou nabídku (*select*). Specifikace jazyka HTML se nachází na [4].

Další formální jazyk, který lze využít pro dynamickou manipulaci s dokumentem, představuje skriptovací jazyk **Javascript.** Javascript je obecný interpretovaný jazyk s objektovou orientací, který podporují všechny moderní prohlížeče jako Internet Explorer, Firefox nebo Opera. Jádro jazyka se syntakticky podobá jazyku C, má ale potlačenou typovou kontrolu. Javascript ve spojení s objektovým modelem dokumentu, který popíši v kapitole 3.2, dovoluje dynamicky upravovat obsah a strukturu dokumentu. Jazyk Javascript je také někdy označován jako ECMAScript, jehož specifikace je obsažena v [5].

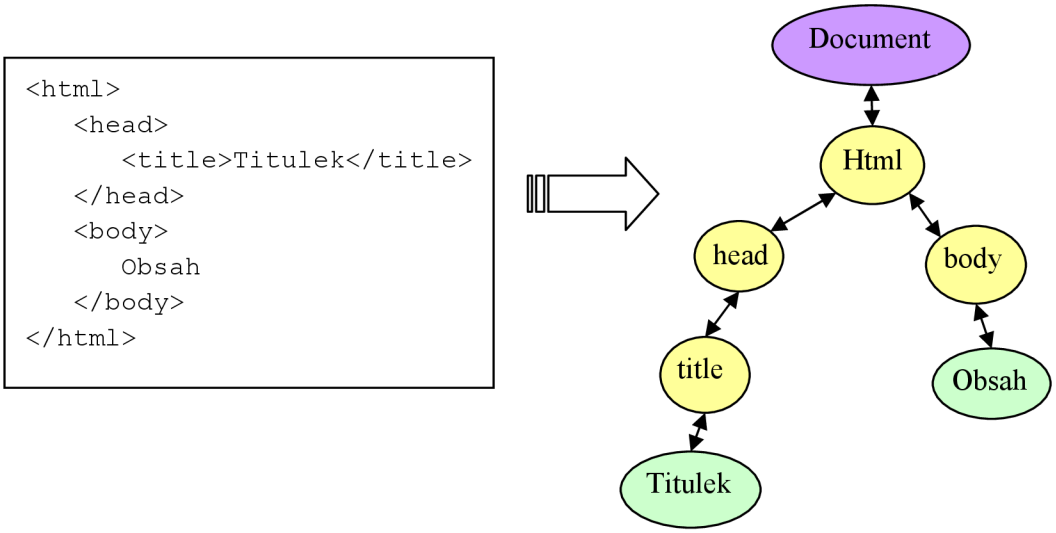
Posledním jazykem je **CSS** (Cascading Style Sheet). Tabulka kaskádových stylů dovoluje odděleně definovat grafické zobrazení jednotlivých prvků dokumentu. Tabulka je tvořena množinou pravidel, které popisují vzhled jednoho prvku nebo jejich skupiny. Každé pravidlo obsahuje selektor, na základě něhož se potom určí, na jaké prvky dokumentu bude aplikováno. Mezi základní selektory patří název prvku, třída prvku (atribut *class*) a pojmenovaný prvek (atribut *id*). Za selektorem potom následuje seznam vlastností, jejichž hodnoty definují výsledný styl. Kompletní specifikace CSS je dostupná na [7].

3.2 DOM

Důležitým prostředkem pro dynamickou úpravu dokumentu představuje objektový model dokumentu (Document Object Model) označovaný zkráceně jako DOM. Podle [21] představuje DOM jazykově neutrální rozhraní, které lze využít pro změnu obsahu a vzhledu dokumentu. Javascript, stejně jako libovolný jiný skriptovací jazyk tedy může manipulovat s modelem dokumentu popsaného pomocí jazyka HTML nebo XML.

Během interpretace dokumentu webovým prohlížečem vzniká množina objektů, které jsou hierarchicky uspořádány ve formě stromu, jehož vrcholem je samotný objekt dokumentu. Jednotlivé uzly stromu jsou provázány ukazateli, prostřednictvím kterých lze stromem procházet. DOM model umožňuje rovněž vytvářet dynamicky nové uzly a libovolný uzel zrušit, přičemž rušení rodičovského uzlu má za následek zrušení všech jeho dětí.

Následující obrázek ukazuje převod definice dokumentu v HTML na jeho objektový model.



Obr. 3-1 Transformace dokumentu na objektový model

Podle standardu existuje celkem 12 typů uzlů, jejichž seznam je možné nalézt na [21]. Mezi často využívané patří typ ***Document*** (fialová barva) reprezentující celý dokument, typ ***Element*** (žlutá barva) pro vyjádření libovolného elementu a typ ***Text*** (zelená barva) pro zobrazení textového obsahu elementu. Tab. 3-1 shrnuje důležité vlastnosti pro navigaci dokumentem a metody sloužící k manipulaci s jeho uzly.

Vlastnost/metoda	Popis
childNodes	Pole ukazatelů na nejbližší potomky.
parentNode	Ukazatel na rodičovský uzel.
fisrtChild	Ukazatel na uzel prvního přímého potomka.
lastChild	Ukazatel na uzel posledního přímého potomka.
nextSibling	Ukazatel na následující uzel z rodičovského seznamu potomků.
previousSibling	Ukazatel na předcházející uzel z rodičovského seznamu potomků.
document.getElementById(id)	Vrátí ukazatel na element, jehož id odpovídá parametru metody *id*.
<element>.createElement(tagName)	Vytvoří objekt reprezentující nový element odpovídající názvu značky *tagName* a vrátí na něj ukazatel.
<element>.createTextNode(text)	Vytvoří textový uzel, naplní ho textem předaným v parametru *text* a vrátí na něj ukazatel.
<element>.appendChild(child)	Vloží uzel *child* na poslední pozici v seznamu dětí uzlu <element>.
<element>.removeChild(child)	Odstraní uzel *child* ze seznamu dětských uzlů uzlu <element>.

Tab. 3-1 Důležité vlastnosti a metody objektu uzlu

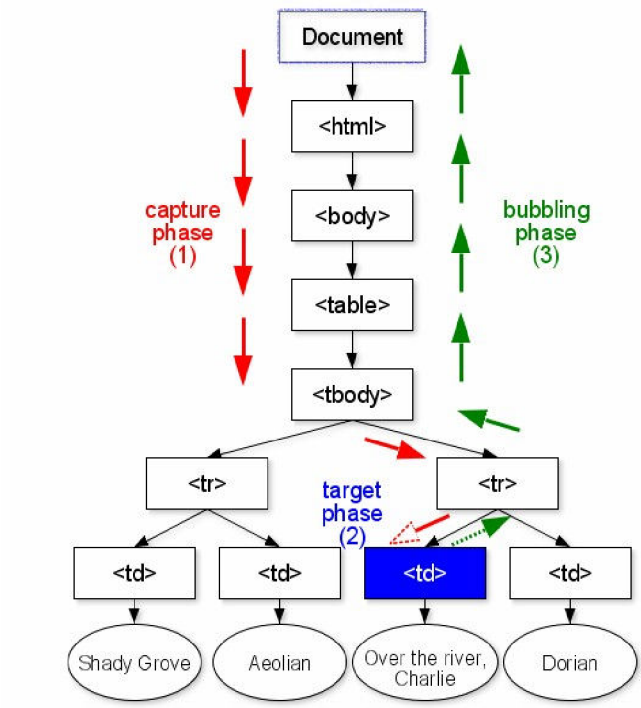
Zavedení jednotného rozhraní prostřednictvím DOM přispělo významně ke standardizaci způsobu, jakým skripty zacházejí s HTML a XML dokumenty. Díky objektovému modelu je možné konsolidovaným způsobem procházet a modifikovat obsah dokumentu. V kombinaci s jazykem HTML a CSS patří DOM k základním prostředkům pro tvorbu webového uživatelského rozhraní.

3.3 Událostní model

Pokud má webová aplikace reagovat na podnět uživatelů, musí umět rozpoznat, že uživatel/ka provedl/a na jejím rozhraní nějakou akci. V modelu dokumentu je takováto akce (změna polohy kurzoru myši, stisknutí tlačítka, dokončení natažení dokumentu, atd.) popsána pomocí objektu události. Vlastnosti události, její šíření modelem dokumentu a způsob, jakým může být registrována obsluha vzniklé události, popisuje událostní model dokumentu, který specifikuje W3C ve [12].

Proces vzniku a šíření události se podle [20] označuje jako tzv. **tok události** (DOM Event Flow). Na začátku procesu je vytvořen objekt události, který je potom vyslán stromem dokumentu.

Každá událost má svůj cílový element – uzel stromu, k němuž se postupně šíří. Proces šíření začíná od kořene stromu, postupně je událost vysílána do jeho nižších pater, až dospěje k samotnému cílovému uzlu. Odtud se potom může šířit opět vzhůru ke kořenu. Obrázek níže názorně ukazuje tok události modelem dokumentu.



Obr. 3-2 Grafická reprezentace toku událostí převzatá z [20]

Specifikace [20] uvádí následující tři fáze toku událostí:

1. **Capture phase** – událost je vysílána od kořene stromu, přes všechny předky cílového uzlu, až k jeho přímému předku.
2. **Target phase** – událost je vyslána k cílovému uzlu.
3. **Bubbling phase** – událost „probublává" zpět od přímého předka cílového uzlu ke kořenu stromu.

Proces šíření události však nemusí obsahovat všechny etapy. U některých typů událostí (např. *focus* nebo *blur*) nedochází vůbec k poslední fázi. Informace o tom, zda se bude událost šířit zpátky vzhůru stromem je uložena ve vlastnosti ***bubbles***. Další důležité vlastnosti a metody objektu události shrnuje Tab. 3-2.

Vlastnost/metoda	**Popis**
type	Identifikátor typu události.
target	DOM uzel, který je cílem procesu šíření události.
currentTarget	Aktuální uzel, do kterého dosud událost dorazila. Během druhé fáze odpovídá hodnotě vlastnosti *target*.
eventPhase	Indikace fáze procesu šíření vyjádřená číslem 1, 2 nebo 3.

bubbles	Příznak, jestli provede třetí fáze šíření události.
cancelable	Příznak, jestli je možné zrušit výchozí akci spojenou s daným typem události.
stopPropagation()	Zamezí volání funkcí obsluhy událostí vyjma těch, které jsou registrovány pro aktuální DOM uzel.
preventDefault()	Zruší výchozí akci spojenou s daným typem události.

Tab. 3-2 Vlastnosti a metody objektu události

Vzniklou událost je potřeba nějakým způsobem odchytit, a proto DOM model umožňuje na uzel stromu dokumentu připojit tzv. **posluchač událostí** (Event Listener). Prostřednictvím něho se určí, na jaký typ události se bude reagovat a jak bude tato událost zpracována. Pokud je k elementu vyslána událost správného typu, posluchač automaticky zavolá příslušný kód obslužné funkce. Uzel může mít více posluchačů. DOM model nabízí i prostředek pro jejich zrušení.

K vytvoření posluchače na příslušném elementu specifikuje [20] metodu elementu ***addEventListener()***. Její podpora je omezena pouze na prohlížeče s jádrem Gecko (Mozzila, Firefox). Internet Explorer používá pro registraci obsluhy události metodu elementu ***attachEvent(sEvent, fpNotify)***. První parametr metod určuje typ události. Internet Explorer navíc přidává ke každému typu prefix „on". Jako druhý parametr je u obou metod vyžadován ukazatel na obslužnou funkci. Metoda *addEventListener()* nabízí ještě možnost zachytit událost před tím, než je šířena dále. Pro tuto potřebu slouží třetí parametr, který je typu *boolean*. Regitraci obsluhy události podporovanou většinou současných webových prohlížečů obsahuje Př. 3-1.

Př. 3-1 Registrace obsluhy události fungující ve většině webových prohlížečů

```
function addEvent(el, eventType, func, capture) {
      if (el.addEventListener) {
         el.addEventListener(eventType, func, capture);
      } else {
         el.attachEvent("on" + eventType, func);
      }
 };
```

Existuje ještě jeden způsob definice obsluhy události určitého typu. Při definici dokumentu pomocí jazyka HTML lze u příslušného elementu uvést atribut s názvem odpovídajícím typu události. V tomto případě má ovšem typ události prefix „on". Hodnotou atributu je potom obslužná funkce události (viz Př. 3-2).

Př. 3-2 Registrace obsluhy události pomocí HTML

```
<input type="button" value="tlačítko" onclick="obsluha_stisknuti();" />
```

Výhodou je nepochybně kompatibilita u různých typů internetových prohlížečů. Tento způsob přihlášení obsluhy však nedovoluje registrovat více obslužných funkcí reagujících na stejný typ události a registrace samotná je omezena pouze na HTML elementy.

Závěrem uvedu přehled často využívaných typů událostí (viz Tab. 3-3). Jejich kompletní seznam je dostupný na [20].

Typ události	Popis vzniku události
click	Stisknutí a okamžité uvolnění tlačítka.
mousedown	Stisknutí tlačítka.
mouseup	Uvolnění tlačítka.
mouseover	Vstup kurzoru myši do oblasti uzlu .
mouseout	Opuštění kurzoru myši oblasti uzlu.
keydown	Stisknutí klávesy.
keyup	Uvolnění klávesy.

Tab. 3-3 Často využívané typy událostí

Událostní model dovoluje prostřednictvím událostí a jejich posluchačů reagovat na akce, které provedl/a uživatel/ka na dokumentu. Bohužel webové prohlížeče přesně nedodržují výše popsanou specifikaci. Vyskytují se nestandardní události (např. *onmouseenter* nebo *onmouseleave* v IE), odlišnosti se objevují ve vlastnostech objektu události (v IE je reference na uzel dokumentu, na kterém došlo k události, uložena ve vlastnosti *srcElement*, naproti tomu FF ji ukládá do vlastnosti *target*) a konečně se liší také způsob registrace obslužné funkce (v IE je nutné použít metodu uzlu *attachEvent()*, zatímco v FF *addEventListener()*). Tvorba aplikací podporující více webových prohlížečů proto vyžaduje vytvoření programové vrstvy, která by odstínila uvedené rozdíly a nabídla normalizovaný událostní model odpovídající standardu W3C.

3.4 AJAX

V předchozích kapitolách jsem popsal důležité prostředky, prostřednictvím kterých lze vytvořit webové rozhraní a dynamicky jej modifikovat. Poslední překážka na cestě za opravdu interaktivní aplikací spočívá v samotném způsobu komunikace mezi klientem a serverem založeném na modelu žádost-odpověď, která se skládá ze tří základních fází:

1. Uživatel/ka provede prostřednictvím rozhraní akci, v důsledku které klient posílá HTTP žádost (request) na server o změnu webové stránky.
2. Server obdrží požadavek od klienta a na jeho základě provede příslušné operace. Ověří například, jestli uživatel/ka má na data právo, vybere samotná data z databáze a sestaví novou stránku, kterou odešle zpět na klienta. Tyto operace trvají určitou nezanedbatelnou dobu závisející na složitosti operací, které musí server při vytváření odpovědi provést.
3. Klient přijímá novou stránku a zobrazí ji uživateli v prohlížeči.

Tento model bude vyhovovat, pokud má být na základě odpovědi serveru provedena změna významné části obsahu stránky a přitom uživateli nevadí, že tato změna bude trvat určitou dobu,

během které nebude moci webovou aplikaci využívat. V případě statické prezentace je všechno v pořádku. Uživatel/ka počká, než server splní požadavek klienta. Pro webovou aplikaci však čekání uživatele/uživatelky představuje významný problém. Je nutné okamžitě reagovat na akci, kterou uživatel/ka provedl/a. Cílem je co nejlépe odstínit model žádost-odpověď, aby nevznikaly časové prodlevy mezi jednotlivými akcemi a uživatel/ka mohl/a aplikaci využívat tak, jak je zvyklý/á z desktopového prostředí.

Nevyhovuje ani množství dat, které se při reakci na odpověď zasílá. Aplikace ve většině případů nepotřebuje vyměnit celý obsah rozhraní, ale pouze upravit jeho část. Pokud má např. dojít k aktualizaci dat zobrazených v tabulce, není potřeba znovu zasílat všechny obrázky a texty, které byly potřeba při inicializaci rozhraní.

Řešení výše popsaných problémů přináší technika označovaná jako **AJAX** (**A**synchronous **J**avascript **A**nd **X**ML). Podle [9] se nejedná o žádnou novou technologii, ale pouze o nový přístup, jak využít již ověřené technologie pro tvorbu webových stránek, které zahrnují:

- HTML a CSS pro zobrazení webové stránky,
- DOM dovolující dynamické změn,
- XML pro výměnu dat,
- **XMLHttpRequest** pro asynchronní komunikaci se serverem,
- JavaScript pro propojení předešlých částí.

Základ techniky AJAX spočívá ve využití objektu ***XMLHttpRequest*** (dále XHR), prostřednictvím něhož je možné volat server v pozadí a získat potřebná data [10]. Díky tomu není nutné aktualizovat celé rozhraní webové aplikace. Velikost dat, která posílá server, se výrazně zmenší, a v důsledku toho se také zkrátí doba potřebná pro odpověď serveru. Technika AJAX úplně zastiňuje klasický model komunikace klient-server. Server pouze dodává nová data, výchozí webová stránka zůstává stále stejná. Pouze pokud by akce vyvolaná uživatelem vyžadovala zásadní změnu rozhraní, potom dojde ke standardní žádosti a server vytvoří požadovanou stránku.

Čekání uživatele/uživatelky po dobu, kdy server zpracovává žádost klienta, vyplývá opět z modelu žádost-odpověď, který je synchronní. Uživatel/ka provede akci, počká na zpracování (synchronizuje se), a potom provede další akci. Objekt XHR ovšem dovoluje poslat **asynchronní požadavek**. Uživatel/ka může provádět akce jednu za druhou, aniž by musel/a počkat na odpověď serveru. Díky tomu přístupu je možné pak vytvořit skutečně interaktivní webové rozhraní, které rychle reaguje na podněty od uživatele.

XHR byl vytvořen pro Internet Explorer jako prvek *ActiveX* již v roce 1999 firmou Microsoft [19, s. 29]. Nejdříve byla jeho podpora omezena pouze na Internet Explorer. Postupně se však využití XHR rozšířilo a v době psaní této práce je již dostupný v řadě webových prohlížečů (Mozilla, FireFox, Opera nebo Safari). Tento fakt je velmi důležitý, protože objekt XMLHttpRequest není součástí standartu W3C [19, s. 39]. Způsoby, jak objekt vytvořit, se kvůli jeho historickému vývoji

v různých typech webových prohlížečů liší, ale API všech jeho instancí je shodný. Množinu vlastností a metod XHR přehledně shrnuje následující tabulka.

Vlastnost/metoda	**Popis**
open(method, url, async, username, password)	Inicializuje Http požadavek.
setRequestHeader(label, value)	Nastaví hlavičku požadavku se jménem *label* a hodnotou *value*.
send(content)	Pošle požadavek na server .
onreadystatechange	Obsahuje odkaz na funkci, která je volána při změnách stavu požadavku.
readyState	Obsahuje číslo stavu požadavku.
responseText	Vrátí odpověď serveru jako řetězec.
responseXml	Vrátí odpověď serveru jako dokument XML.
status	Vrátí stavový kód požadavku.
statusText	Vrátí zprávu stavu požadavku.
getResponseHeader(headerLabel)	Vrátí konkrétní hlavičku odpovědi podle jejího názvu.
getAllResponseHeaders()	Vrátí pole řetězců reprezentující všechny hlavičky odpovědi.
abort()	Zruší aktuální požadavek.

Tab. 3-4 Přehled vlastností a metod XHR

Vytvoření XHR je odlišné podle typu prohlížeče. Internet Explorer implementuje XHR jako objekt ActiveX, ostatní prohlížeče (Firefox, Opera, Safari) zvolily implementaci formou nativního objektu Javascriptu [19, s.39]. Př. 3-3 ukazuje kód potřebný pro vytvoření instance ve všech uvedených webových prohlížečích.

Př. 3-3 Vytvoření objektu XHR

```
var xhr = null;
if (window.ActiveXObject) {
   xhr = new  ActiveXObject("Microsoft.XMLHTTP");
} else if (window.XMLHttpRequest) {
  xhr = new XMLHttpRequest();
}
```

Pokud je objekt XHR k dispozici, realizaci samotného požadavku zajistí volání metod *open()* a *send()*, které obsahuje následující příklad.

Př. 3-4 Odeslání asynchronního požadavku pomocí XHR

```
xhr.open("GET", url, true); //inicializace
xhr.onreadystatechange = zmenaStavu; //obslužná funkce
xhr.send(null); //odeslání požadavku
```

Nejdříve je nutné inicializovat volání serveru pomocí metody ***open()*** objektu XHR. Jako první parametr se předá metoda, kterou se požadavek provede (GET, POST). Druhý parametr specifikuje URL serveru. Třetí parametr dovoluje nastavit asynchronní zpracování odpovědi serveru. Výchozí hodnota je *false*, tj. synchronní volání serveru, které způsobí pozastavení funkčnosti stránky, dokud není k dispozici odpověď. Požadavek se potom odešle metodou ***send()***. Jako jediný parametr se předá obsah odesílané zprávy.

Při asynchronním zpracování HTTP požadavku provádí klient po odeslání dotazu na server další operace. Informace o stavu zpracování požadavku získává od serveru průběžně prostřednictvím obslužné funkce, jejíž referenci je potřeba uložit do vlastnosti ***onreadystatechange.*** Objekt XHR potom zajistí volání funkce vždy, kdy dojde ke změně stavu. Klient tak může reagovat na vznik případné chyby a požadavek zrušit pomocí metody ***abort()*,** nebo naopak začít s analýzou odpovědi, jejíž zpracování bylo na serveru úspěšně dokončeno. Stavy požadavku popisuje Tab. 3-5.

kód	stav	Popis
0	uninitialized	Objekt XMLHttpRequest je vytvořen, ale nebyl ještě inicializován požadavek.
1	loading	Metodou *open()* byl inicializován požadavek a je připraven k odeslání.
2	loaded	Požadavek byl odeslán metodou *send()*. K dispozici jsou hlavičky odpovědi, tělo zprávy dosud ne.
3	interactive	Část těla zprávy byla přijata a její nekompletní podoba je ve vlastnosti *responseText*.
4	completed	Všechna data byla přijata a jsou k dispozici pro další zpracování.

Tab. 3-5 Stavy požadavku XHR

Samotný stav požadavku ***completed*** ještě nezaručuje, že server poslal data, o které klient zažádal. Server pouze sděluje, že se mu podařilo zaslat kompletní tělo zprávy a klient s ním může dále pracovat. Pro ověření, že během zpracování nedošlo k chybě, je potřeba otestovat kód odpovědi, který se nachází ve vlastnosti ***status*.** Proběhne-li operace na serveru v pořádku, bude vlastnost status obsahovat kód **200**. Jiné stavové kódy představují neúspěch dotazu. Jejich popis je uložen ve vlastnosti ***statusText***, která se dá využít při výpisu chyby. Př. 3-5 ukazuje kód funkce obsluhující změnu stavu. Číslo stavu požadavku je k dispozici ve vlastnosti ***readyState.***

Př. 3-5 Obslužná funkce XHR

```
function zmenaStavu() {
   if (xhr.readyState ==  4 &&
       xhr.status == 200) {
           //
           //zpracování odpovědi
           //
   } else {
      alert("Při zpracování na serveru došlo k chybě: " +
            xhr.statusText);
   }
}
```

Pokud nedojde k žádným chybám, následuje zpracování získané zprávy přístupné prostřednictvím vlastnosti ***responseText***. Alternativu představuje vlastnost ***responseXml***, kterou lze použít, jestliže server odeslal data zprávy jako XML dokument. Název objektu XMLHttpRequest je proto zavádějící. Server může odeslat zprávu **v libovolném textovém formátu vhodném pro přenos dat**.

Z předcházejícího textu vyplývá, že využití XHR není triviální a vyžaduje implementaci kompletního zpracování požadavku, přičemž je potřeba rozlišit, jestli byla odpověď serveru doručena v pořádku, nebo došlo k chybě a umožnit registraci příslušné obsluhy. Chybí také logika pro automatické ověření a transformaci samotné odpovědi. Současné webové prohlížeče tedy dovolují komunikovat se serverem asynchronně, realizace samotného požadavku a zpracování odpovědi s sebou přináší řadu komplikací.

3.5 JSON

V předchozí kapitole jsem uvedl, že odpověď serveru na XHR požadavek nemusí být zapsána pouze ve formátu XML. Jinou alternativou je využití textového formátu pro výměnu dat označovaného jako **JSON (Javascript Object Notation)**. Podle [6] představuje JSON jednoduchý, dobře čitelný formát umožňující popis hierarchicky uspořádaných dat pomocí dvou základních struktur:

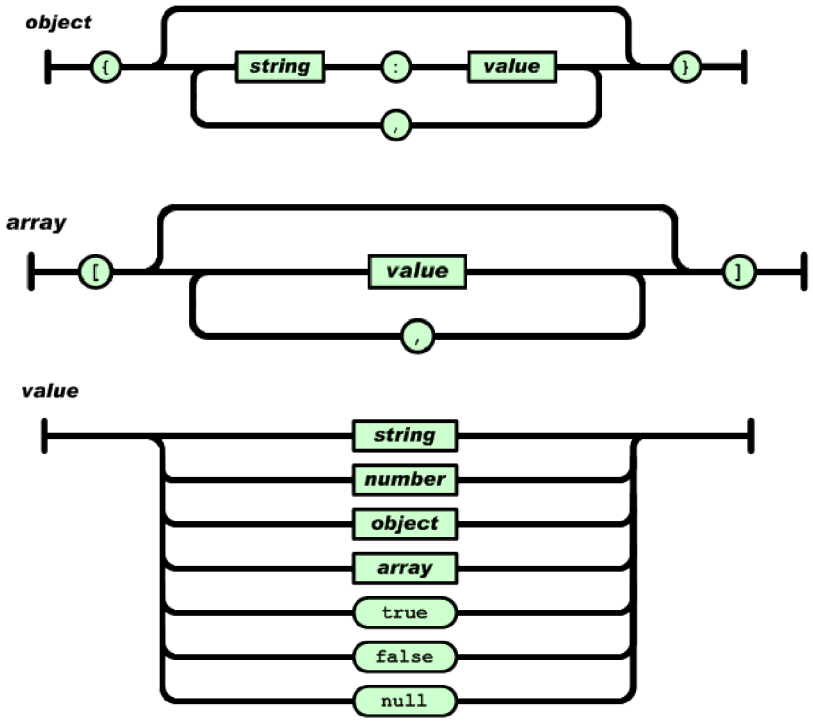
1. kolekce dvojic jméno/hodnota (objekt),
2. uspořádaného seznamu hodnot (pole).

Tyto struktury jsou podporovány všemi moderními programovacími jazyky, a proto představuje JSON vhodnou volbu pro výměnu dat mezi heterogenními systémy [19, s. 81]. Díky tomu, že JSON je založen na podmnožině jazyka Javascript [6], lze ho navíc efektivně interpretovat na klientu s využitím javascriptové funkce ***eval*** (viz Př. 3-6). Výhoda spočívá také v jednotné interpretaci JSON v různých prohlížečích.

Př. 3-6 Vyhodnocení dat ve formátu JSON na klientu

```
var data = eval("(" + json_data + ")");
```

Následující specifikaci formátu JSON jsem čerpal z [6]. Objekt JSON reprezentuje neuspořádaný seznam dvojic název/hodnota. Zápis začíná znakem *{* a končí znakem *}*. Jednotlivé dvojice jsou odděleny znakem čárka a identifikátor názvu od identifikátoru hodnoty odděluje znak dvojtečka. Pole JSON začíná znakem *[* a končí znakem *]*. Jako oddělovač jednotlivých prvků se použije znak čárka. Hodnota JSON potom může být vyjádřena řetězcem ohraničeným pomocí uvozovek, číslem nebo pomocí objektu či pole definovaného výše. Speciální povolené hodnoty jsou *true*, *false* a *null*. Popsanou syntaxi formátu JSON demonstruje Obr. 3-3.



Obr. 3-3 Povolená struktura formátu JSON převzatá z [6]

Prostřednictvím formátu JSON lze tedy popsat libovolnou hierarchicky uspořádanou strukturu. Na rozdíl od dat vyjádřených pomocí jazyka XML však poskytuje JSON mnohem kompaktnější způsob jejich reprezentace. Následující příklad porovnává zápis v obou zmíněných formátech.

Př. 3-7 Porovnání zápisu dat pomocí XML a JSON

```
<?xml version="1.0"
      encoding="UTF-8"
?>
<cars>
   <car>
      <color>čerevná</color>
      <type>1</type>
      <diesel>true</diesel>
   </car>
   <car>
      <color>modrá</color>
      <type>2</type>
      <diesel>fale</diesel>
   </car>
</cars>
```

```
{ cars: [
   {
     "color": "čerevná",
     "type": 1,
     "diesel": true
   },
   {
     "color": "modrá",
     "type": 2,
     "diesel": false
   }
}
```

Data popsaná v XML musí dodržovat striktní uzavírání jednotlivých značek. Formát JSON dovoluje kratší způsob zápisu a navíc se nemusí použít identifikátor *car* (viz Př. 3-7), protože jednotlivé záznamy o autech lze vyjádřit pomocí objektu.

JSON tedy představuje jednoduchý, dobře čitelný formát dovolující popsat obecnou datovou strukturu kompaktním způsobem. Takový popis lze potom efektivně interpretovat klientským prohlížečem.

3.6 Implementace pod různými webovými prohlížeči

Implementace webové aplikace v prostředí současných prohlížečů představuje nelehký úkol. Přestože existují standardy vydané konsorciem W3C popisující objektový model stránky (dostupný na [11]), událostní model (popsaný na [12]), současné webové prohlížeče (Internet Explorer, Firefox, Opera, aj.), vykazují odlišnosti v interpretaci kódu stránky. Rozdílná interpretace potom vede ke změnám v zobrazení webového rozhraní, a v horším případě dokonce k chybám ve funkcionalitě webové aplikace. Před zahájením implementace samotné aplikace je proto potřeba vytvořit programovou vrstvu, která by skryla výše zmíněné rozdíly a poskytla **jednotné programové prostředí**. V opačném případě totiž vzniká velké množství kódu závislého na konkrétním typu prohlížeče, který způsobuje zvýšení nepřehlednosti implementace, přispívá ke vzniku chyb a horší udržovatelnosti kódu jako celku.

Další problém spojený s realizací webové aplikace souvisí s tvorbou uživatelského rozhraní. Pomocí jazyka HTML lze definovat základní dialogové prvky (textové pole, tlačítko, roletová nabídka atd.). Manipulaci s těmito prvky, získání a nastavení hodnoty prvku, stejně jako ošetření událostí, které na něm nastaly, je nutné nejdříve implementovat za použití Javascriptu a DOM modelu

dokumentu. Vytvoření základní sady komponent tudíž vyžaduje značné úsilí. Náročnost tohoto úkolu zvyšuje rovněž rozdílné zacházení s dialogovými prvky napříč různými webovými prohlížeči. Výchozí prostředky poskytované prohlížeči pro tvorbu uživatelského rozhraní jsou z tohoto pohledu značně omezené a je potřeba je rozšířit.

Typickou operací, kterou nabízí desktopové aplikace, je reorganizace rozhraní technikou drag & drop. Webové prohlížeče ovšem dovolují pouze obsluhu základních událostí spojených se stisknutím tlačítka myši a jejím pohybem. Neexistuje žádná nativní podpora řízení drag & drop, a proto představuje implementace této operace další významnou komplikaci při vytváření interaktivní webové aplikace.

Před zahájením vývoje komplexní webové aplikace je proto podle mého názoru vhodné zvážit využití javascriptové knihovny, která by zajistila jednotné programové prostředí, nabídla hotovou sadu komponent pro budování rozhraní a přinesla také podporu pro operaci & drop nebo kladení asynchronních požadavků.

4 Návrh pracovní plochy

4.1 Základní služby

Cílem této práce je navrhnout a implementovat prototyp pracovní plochy dostupné z webu, která bude sloužit jako **úvodní stránka portálu**. Pracovní plocha zajistí přehlednou organizaci odkazů na další stránky v rámci portálu, zároveň dovolí uchovávat také odkazy na stránky externí. Na rozdíl od pracovních ploch popsaných v kapitole 2.3 nebudu usilovat o širokou paletu miniaplikací. Zaměřím se především na vytvoření základní komponenty pro správu odkazů, kterou doplním miniaplikacemi kalkulačky a kalendáře. Aplikace bude dále poskytovat nabídku zajímavých odkazů, které lze na plochu umístit.

Na základě analýzy rozhraní současných webových pracovních ploch navrhnu vlastní uživatelské rozhraní podporující vkládání miniaplikací a jejich následné rozmístění po ploše prostřednictvím správce rozložení. Návrh se pokusím provést tak, aby aplikace do budoucna umožnila přidávat nové miniaplikace. Za velmi důležitou považuji také podporu nejvíce rozšířených webových prohlížečů. Pracovní plocha půjde spustit v prohlížečích Internet Explorer (ver. 6, 7), Firefox (ver. 2.x) a Opera (ver. 9+).

Mezi základní služby, které bude prototyp pracovní plochy nabízet svým uživatelům, patří:

- autentizace a správa uživatelských účtů,
- přidávání, modifikace a mazání obsahu,
- nabídka dostupných odkazů a miniaplikací,
- správce rozložení pro snadné rozmístění obsahu,
- změna grafického vzhledu,
- podpora více jazyků,
- konfigurace a její perzistetní uložení.

Vzhledem k tomu, že pracovní plocha je dostupná z veřejné sítě, musí aplikace zajistit, aby k pracovní ploše mohl/a přistupovat pouze oprávněný/á uživatel/ka. Jedna ze služeb aplikace proto bude ověření identity uživatele. Autentizace proběhne na základě jednoznačného přihlašovacího jména a hesla.

Další důležitou službu představuje podpora správy obsahu zahrnující vytváření nových odkazů a jejich umístění na pracovní plochu. Aplikace uživateli dovolí přidávat také miniaplikace, které potřebuje pro svoji práci. Veškerý obsah bude možné upravit, v případě potřeby také odstranit. Cílem je poskytnout uživateli naprostou volnost při vytváření obsahu.

Vytvářet nové odkazy a miniaplikace vyžaduje hodně práce. Proto aplikace automaticky zobrazí nabídku zajímavých odkazů a miniaplikací, které si uživatel/ka může přidat na svoji plochu.

Výchozí nabídka odkazů bude změnitelná nastavením zdrojových serverů. Tuto problematiku podrobněji rozvedu v kapitole 5.1.

Rozmístění odkazů a miniaplikací na pracovní ploše bude zajišťovat správce rozložení. Pro pohodlné přemisťování prvků pracovní plochy jsem zvolil techniku drag & drop. Díky této technice může uživatel/ka prostřednictvím kurzoru myši uchopit libovolný prvek na pracovní ploše a pohybem myši změnit jeho polohu. Obsah pracovní plochy musí být přehledně uspořádán tak, aby odkazy a miniaplikace, které jsou si sémanticky blízké, byly umístěny u sebe. Pracovní plocha proto nabídne hierarchické členění jejich prvků.

Při návrhu jsem se snažil, aby webová aplikace pracovní plochy byla flexibilní a uživatel/ka si ji mohl/a přizpůsobit podle svých představ. Kromě volnosti, kterou má při vytváření obsahu plochy, jsem se rovněž zaměřil na možnosti přizpůsobení grafického vzhledu pracovní plochy. Aplikace nabídne uživateli několik předpřipravených grafických témat (themes). Po výběru grafického tématu zajistí aplikace změnu všech prvků rozhraní a vytvoří požadovaný design.

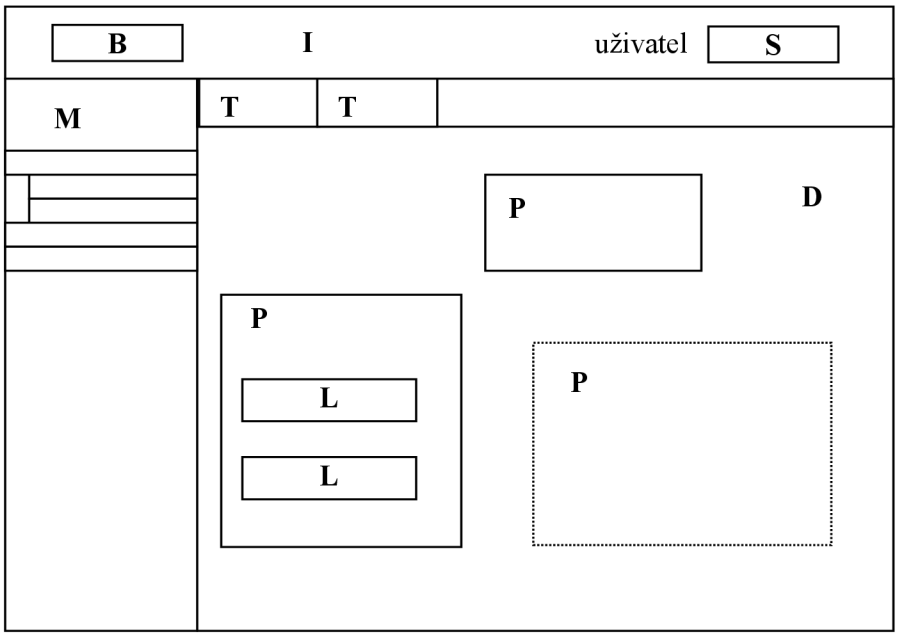
Vzhledem k tomu, že je aplikace dostupná z internetu, je možné předpokládat, že uživatelé a uživatelé pracovní plochy mohou být nejen z České republiky, ale také ze zahraničí. Považuji proto za nutné, aby aplikace poskytovala podporu jazykové lokalizace a umožňovala alespoň překlad rozhraní pracovní plochy do anglického jazyka. Uživatel/ka bude mít na výběr ze dvou jazyků, českého a anglického. Nabídku jazyků bude možné v budoucnu nadále rozšiřovat.

Poslední důležitou službou pracovní plochy je uložení konfigurace plochy. Pokud by uživatel/ka nemohl/a provést uložení, veškeré úpravy pracovní plochy by se ztratily. Jak už jsem uvedl v kapitole 2.2, perzistenci konfigurace zajišťuje konfigurační server, který ukládá veškeré informace o pracovní ploše do databáze.

4.2 Rozhraní pracovní plochy

4.2.1 Schéma rozhraní

Uživatel/ka bude používat pracovní plochu prostřednictvím webového rozhraní, a proto pokládám za velmi důležité uspořádat jednotlivé prvky rozhraní takovým způsobem, aby se v něm dobře orientoval/a a dokázal/a s plochou intuitivně pracovat. Při návrhu jsem bral v úvahu uživatelská rozhraní již implementovaných pracovních ploch popsaných v kapitole 2.3. Obr. 4-1 představuje schéma rozhraní pracovní plochy, které jsem navrhnul.



Obr. 4-1 Schéma rozhraní pracovní plochy

Základní funkcí pracovní plochy je přehledná organizace odkazů. Za vhodné považuji jejich členění do skupin podle zaměření. Počet odkazů však může být velký, a proto jsem zavedl ještě jednu úroveň hierarchie a sdružil celé skupiny odkazů. Dohromady jsem tak získal tři úrovně klasifikace, které bude pracovní plocha nabízet. Konkrétně se jedná o samotné odkazy, skupiny odkazů a soubory skupin odkazů.

Dalším požadavkem, který jsem stanovil v kapitole 0, bylo přemisťování odkazů po pracovní ploše pomocí techniky drag & drop. Pro tuto potřebu jsem umístil odkazy podobného zaměření do panelu *P* za účelem jejich společného přesunutí na jinou pozici. Panel bude představovat kontejner obdélníkového tvaru pro odkazy *L*. Plocha dovolí změnu polohy odkazu v rámci panelu a také posouvání samotných panelů v prostoru pracovní plochy *D*. Panel poslouží také jako kontejner pro miniaplikaci. Pro nejvyšší úroveň hierarchie odkazů jsem využil záložky (tabs) *T*. Každá záložka sdružuje soubor panelů a bude obsahovat ikonu a název.

Pracovní plocha nabídne systémové nastavení a též údaje o uživateli, který s pracovní plochou právě pracuje. Nevyužiji proto celou plochu pro panely. Ponechám horní pruh plochy rozhraní *I* pro zobrazení příslušných údajů a pro odkaz do systémového menu *S*. Tuto část rozhraní označím jako informační lišta.

Po levé straně rozhraní umístím menu s nabídkou odkazů a miniaplikací, které může uživatel/ka přidat na svoji pracovní plochu. Obsah menu se bude dynamicky měnit podle toho, jaké zdroje si uživatel/ka zvolí v nastavení pracovní plochy. Vzhledem k tomu, že menu může být široké, pokládám za vhodnější jej ponechat schované, aby prostor pro panely byl co největší. Menu se zobrazí/skryje pomocí tlačítka *B* umístěném na informační liště *I*.

Nabízí se také možnost využít automatického vysouvání menu v okamžiku, kdy se kurzor myši přiblíží k levému okraji rozhraní. Tento přístup však skrývá nebezpečí v tom, že pokud bude

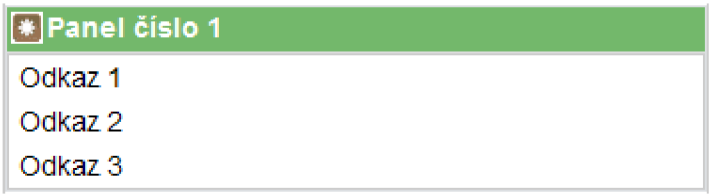
uživatel/ka přesouvat panel nebo odkaz pomocí techniky drag & drop a dostane se kurzorem myši příliš vlevo, dojde k nepříjemnému vysunutí menu. Z tohoto důvodu jsem zobrazení menu nabídky *M* navrhnul pomocí tlačítka.

4.2.2 Panel

Pro sdružení souvisejících odkazů jsem použil panel. Nabízí se možnost vytvořit panely pomocí dětských oken prohlížeče, a tím vyřešit jednoduše jejich přesouvání, které zajistí samotný webový prohlížeč. Tento přístup však není vhodný ze tří důvodů:

1. Panely musí být vázány na hlavní okno aplikace. Není vhodné, aby se společně se spuštěním pracovní plochy otevřela další okna, se kterými by se dalo manipulovat nezávisle.
2. Není potřeba, aby se panely překrývaly, protože plocha by potom působila nepřehledně.
3. Tlačítka pro maximalizaci/minimalizaci okna nemají využití. Nutné bude naopak přidat tlačítka pro volby panelu (viz. kapitola 4.5.3).

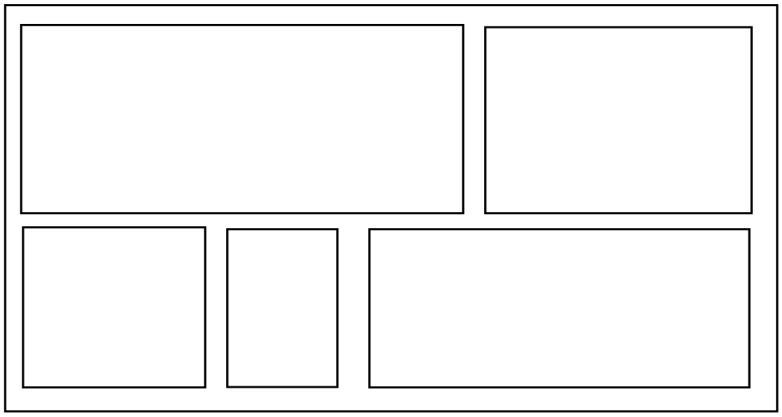
Panel proto bude představovat standardní element stránky s absolutní pozicí nabízející možnost přesunu pomocí kurzoru myši. Z grafického vzhledu okna využiji jeho rozdělení na titulkovou lištu (caption bar) a tělo panelu s obsahem. Obr. 4-2 znázorňuje panel se třemi odkazy. Titulková lišta obsahuje ikonu a název skupiny odkazů reprezentované panelem. Dále nabídne přístup k systémovému a uživatelská nastavení panelu. Tělo panelu tvoří buď uspořádaná množina odkazů, nebo rozhraní miniaplikace, pro kterou je panel kontejnerem.



Obr. 4-2 Panel s odkazy

4.3 Způsob rozložení panelů

Doposud jsem uvažoval, že panely mohou být na ploše rozmístěny libovolným způsobem, pouze jsem vyloučil překryv. Takovéto rozložení je vhodné pro ikony na pracovní ploše operačního systému. Pro panely, které jsou mnohem větší než ikony, působí plocha chaoticky.

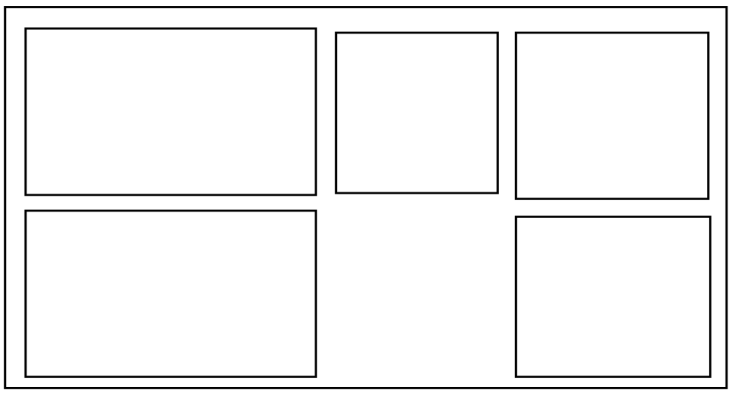


Obr. 4-3 Panely rozložené pomocí mřížky

Vzniklou situaci jsem se pokusil řešit pomocí správce rozložení, který by rozmístil panely podle neviditelné pravoúhlé mřížky. Uživatel/ka tedy může ukončit přemisťování panelu na libovolném místě plochy. Správce však panel posune na takovou pozici, aby jeho levý horní roh korespondoval s nejbližším průsečíkem mřížky. Obr. 4-3 znázorňuje seřazené panely pomocí správce rozložení. Obsah pracovní plochy působí nyní mnohem přehledněji.

Jednotlivé panely však mohou být různě široké a vysoké. V momentě, kdy uživatel/kas některý z nich posune na jinou pozici v mřížce, je potřeba přeskupit všechny panely, které zabírají místo přesouvanému panelu, protože se panely nesmí překrývat. Řetězovitě tak dojde ke složitému přeskupování celé plochy. Algoritmus správce rozložení, který zajišťuje korektní posuny, je náročný, a proto jsem navrhnul jednodušší uspořádání panelů.

Použil jsem sloupcové uspořádání, které využívají také pracovní plochy popsané v kapitole 2.3. Šířku plochy jsem rozdělil na určitý počet sloupců, přičemž šířka panelů koresponduje se šířkou sloupce.



Obr. 4-4 Sloupcové rozložení

Výška panelu nezpůsobuje žádný problém, jelikož další panel může správce rozložení umístit až za konec předcházejícího. Pokud dojde k přetečení prostoru plochy s panely, správce nechá prostřednictvím prohlížeče vygenerovat po pravé straně pracovní plochy posuvník (scrollbar). Výsledné rozmístění panelů odpovídá rozložení, které je známé z novin (viz Obr. 4-4). Jinou možností bylo využít řádkové uspořádání. To ovšem není přirozené, protože čteme vždy shora dolů a vznik horizontálního posuvníku je navíc uživateli často chápán jako negativní jev.

Zatím jsem předpokládal, že správce rozdělí šířku plochy do sloupců rovnoměrně. Uživatel/ka však může chtít některé sloupce zúžit, protože do nich umístil/a panely s krátkými odkazy. Rozšířím proto správce rozložení o možnost dynamicky měnit šířku sloupce a jeho panelů pomocí kurzoru myši. V případě, že se uživatel/ka kurzorem myši přiblíží k hranici mezi dvěma sloupci, objeví se dělící přímka, kterou uživatel/ka může posunout, čímž dojde ke změně šířky sloupce.

Pro rozložení panelů použiji tedy sloupcové uspořádání. Uživatel/ka bude moci měnit šířku sloupců a pomocí techniky drag & drop přesouvat panely z jednoho sloupce do druhého. Pokud bude přesouvaný panel širší/užší než sloupec, do kterého má být zařazen, zajistí správce jeho rozšíření/zúžení tak, aby odpovídal šířce příslušného sloupce.

4.4 Uživatelské akce

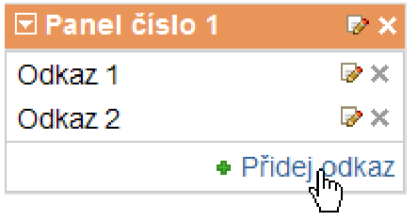
4.4.1 Akce přidání

Pracovní plocha nabídne uživateli vytváření odkazů, panelů a také záložek. Do uživatelského rozhraní jsem přidal prvky, které uvedené operace umožní. Popíši nyní, jak bude realizováno vytvoření:

- nového odkazu v rámci panelu,
- samotného panelu – pro odkazy nebo panelu miniaplikace,
- nové záložky.

Uživateli je dána naprostá svoboda při přidávání prvků na pracovní plochu. Může je umísťovat na plochu a libovolně přesouvat. Zároveň má předpřipravené panely odkazů a miniaplikací, ze kterých může vybírat.

4.4.2 Přidání odkazu



Obr. 4-5 Tlačítko pro vložení odkazu

Vytváření odkazů jsem ponechal v kompetenci miniaplikace. Součástí těla panelu bude tlačítko, po jehož stisknutí se otevře dialog pro vložení nového odkazu. Uživatel/ka vyplní jeho název, URL popisující umístění dokumentu nebo webové stránky, volitelně také popis. Po potvrzení dialogu se vytvoří nový odkaz, který se zařadí na konec seznamu odkazů v těle panelu. Uživatelúka bude moci díky technice drag & drop změnit pozici odkazu v seznamu odkazů. Další možností, jak přidat odkaz do panelu, je vložit do jeho těla odkaz z jiného panelu nebo přímo z nabídky v postranním menu.

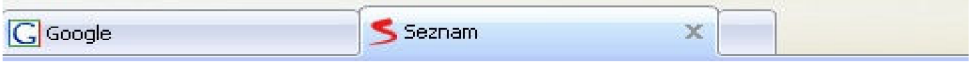
4.4.3 Přidání panelu a záložky

Pro přidání panelu jsem navrhnul tlačítka umístěná v menu. Po stisknutí tlačítka dojde k vytvoření panelu miniaplikace a ten se automaticky vloží do prvního sloupce záložky, která je aktuálně vybraná.

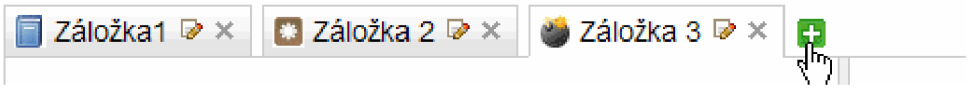
Členění obsahu prostřednictvím záložek představuje techniku, kterou využívá řada aplikací. Tento přístup používají také pracovní plochy uvedené v kapitole 2.3. Setkal jsem se s ním rovněž u webových prohlížečů, jako jsou Internet Explorer (od verze 7), FireFox nebo Opera. Po prozkoumání těchto aplikací jsem zjistil, že přidávání záložek je řešeno nejčastěji následujícími způsoby:

- pomocí pseudo-záložky (Internet Explorer 7),
- s využitím kontextového menu záložky (např. Firefox),
- připojením odkazu nebo tlačítka za poslední záložku (viz kapitola 2.3).

První způsob (viz Obr. 4-6) je založen na malé pseudo-záložce bez popisku, která je obvykle umístěna za poslední standardní záložkou. Při kliknutí na pseudo-záložku dojde k jejímu zvětšení na novou záložku. Tento způsob není podle mého názoru příliš estetický. Použití kontextového menu záložky zase vyžaduje přidání celého menu pro záložku. Vzhledem k tomu, že jsem navrhnul nastavení záložky pomocí dialogu, menu by obsahovalo pouze jednu položku, a proto jsem se rozhodl vyžít třetí způsob.



Obr. 4-6 Záložky v prohlížeči Internet Explorer 7



Obr. 4-7 Návrh záložek pracovní plochy

Za poslední záložku umístím tlačítko se symbolem „+“ (viz Obr. 4-7), aby byl ponechán co největší prostor pro ostatní záložky. Po jeho stisknutí se objeví dialog záložky, ve kterém uživatel/ka vyplní název a zvolí si ikonu. Po potvrzení se záložka přidá na poslední pozici v seznamu záložek. Kompletní obsah dialogu záložky uvedu v kapitole 4.5.2.

4.4.4 Akce odstranění

Pokud aplikace nabízí vytváření odkazů, panelů a záložek, měla by také uživateli umožnit odstranit zastaralé odkazy nebo jejich skupiny. Skrytí odkazů se provede pomocí tlačítka se symbolem křížku umístěného za příslušným odkazem. Pro vyřazení panelu využiji rovněž tlačítka se symbolem křížku na titulkové liště. Zrušení záložky bude podmíněno tím, že obsah záložky je prázdný. Stejně jako v případě odkazu a panelu bude rušení provedeno pomocí tlačítka se symbolem křížku.

4.4.5 Změna počtu sloupců

Vybral jsem sloupcové rozložení panelů. Při první inicializaci rozhraní pracovní plochy se zvolí optimální počet tří sloupců a využije se celá šířka plochy. Tři sloupce jsou nastaveny jako výchozí také u pracovních ploch zmíněných v kapitole 2.3. Uživatel/ka však může požadovat jiný počet sloupců. V případě panelů s dlouhými názvy odkazů budou výhodnější pouze dva sloupce, aby se názvy odkazů nezalamovaly. Jiným zadáním je naopak rozložení se čtyřmi sloupci, protože je potřeba umístit na plochu větší počet panelů, jejichž názvy mohou být naopak kratší. Zužování nepotřebných sloupců na minimální šířku představuje možné řešení, které ovšem není optimální. Navrhl jsem proto, aby pracovní plocha dovolila změnit počet sloupců libovolné záložky.

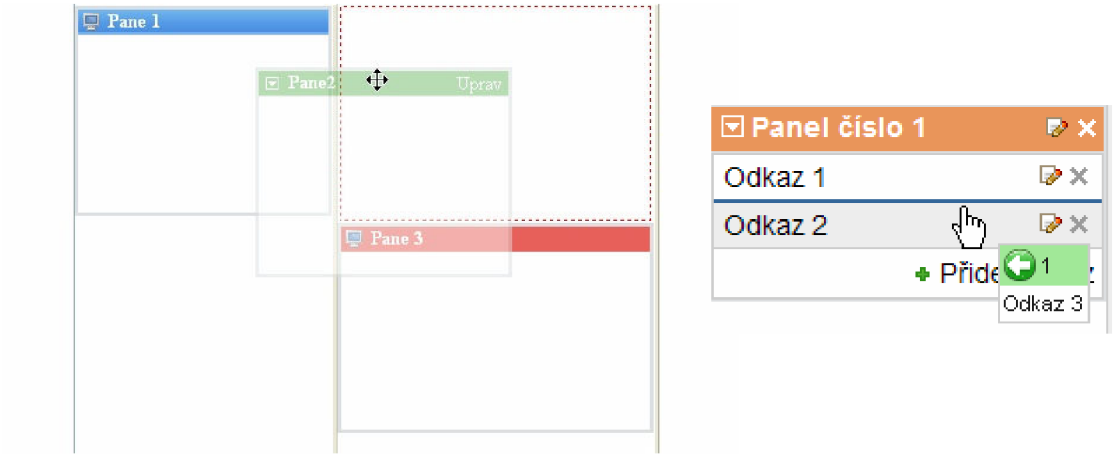
Nejdříve stanovím minimální a maximální počet sloupců. Rozložení s jedním sloupcem se jeví jako zbytečné, avšak v extrémních případech by mohlo být užitečné. Minimální počet sloupců bude tedy jeden. Příliš velký počet sloupců způsobí, že jejich šířka bude malá a názvy odkazů se budou zalamovat. Vzhledem k šířce obrazovky vyberu jako maximum osm sloupců. Nastavení počtu sloupců umístím do dialogu záložky. V případě, že je záložek více, v každé lze nastavit jiný počet sloupců, který nejlépe odpovídá obsahu záložky.

Přidávání sloupců způsobí zúžení stávajících sloupců a jejich panelů. Při ubírání sloupců se naopak stávající sloupce a panely rozšíří, aby využily celou šířku plochy. Navíc je potřeba vyřešit, co se stane s panely v odstraněných sloupcích. Bez vědomí uživatele není možné panely odstranit, a proto provede správce rozložení jejich přesun do posledního sloupce, kterého se odstranění již netýká.

4.4.6 Drag & drop

Jednou z nejdůležitějších akcí, kterou bude uživatel/ka pracovní plochy provádět, je modifikace rozložení odkazů, panelů a záložek. Jak jsem již uvedl, využiji pro ni moderní techniku označovanou jako drag & drop.

Princip je velmi jednoduchý. Uživatel/ka přesune kurzor myši nad objekt, který chce přesunout. Stisknutím levého tlačítka dojde k jeho uchopení, což je obvykle znázorněno změnou kurzoru myši. Při posouvání myši potom vybraný objekt mění polohu podle pozice kurzoru. Při uvolnění tlačítka dojde k jeho upuštění na poslední pozici kurzoru.



Obr. 4-8 Přesun panelu (vlevo) a odkazu (vpravo) technikou drag & drop

Odkazy a panely nelze přemístit kamkoliv. Aplikace musí zajistit, že se přetahovaný prvek automaticky zařadí na správné místo. Pro snazší orientaci bude rozhraní uživateli znázorňovat obdélník ohraničený přerušovanou čarou o rozměru přetahovaného objektu, který se objeví na pozici, jež je vzhledem k sloupcovému rozložení korektní. Situaci zachycuje Obr. 4-8. Uživatel/ka má díky tomu informaci, kam se umístí přetahovaný objekt, pokud uvolní tlačítko myši. U přetahování samotných odkazů uvažuji nulovou výšku objektu, a proto se místo obdélníku zobrazí jenom čára (viz Obr. 4-8).

4.5 Nastavení pracovní plochy

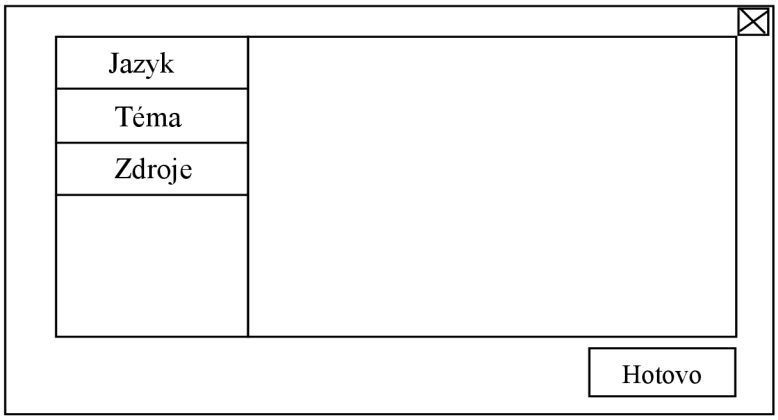
Během návrhu pracovní plochy jsem zmínil dialog nastavení pro odkazy, panely, záložky a také celou aplikaci. Nyní se podrobněji zaměřím na obsah jednotlivých dialogů a popíši, jaké volby může uživatel jejich prostřednictvím provádět. Nastavení navrhované pracovní plochy bude zahrnovat:

- globální nastavení celé aplikace,
- nastavení záložek pomocí dialogu záložky,
- volby panelu prostřednictvím:
    - systémového menu,
    - uživatelského dialogu,
- dialog pro odkazy.

Hlavním cílem je vytvořit intuitivní rozhraní aplikace. Tlačítka, která slouží pro nastavení/rušení prvků pracovní plochy, obsahují jednoduché ikony  , protože jsou výstižnější a jednodušší než titulek. Stejné ikony jsem použil u odkazů, panelů a také záložek.

4.5.1 Globální nastavení

Při návrhu schématu rozhraní jsem do informační lišty umístil odkaz na systémové menu. Po kliknutí na tento odkaz se objeví dialog pro globální nastavení, který ukazuje Obr. 4-9.



Obr. 4-9 Dialog globálního nastavení

Po levé straně je umístěno menu pro jednotlivé kategorie voleb. Zbývající část plochy tvoří formulář pro příslušné nastavení. V pravém horním rohu se nachází tlačítko se symbolem křížku sloužící k uzavření dialogu a vpravo dole je umístěno tlačítko pro potvrzení změn. Menu globálního nastavení obsahuje tři položky – jazyk, grafické téma a zdroje.

Nastavení jazyku zobrazí seznam jazyků, pro které je pracovní plocha přeložena. Potvrzení výběru způsobí přeložení všech textů do zvoleného jazyka. Lokalizace se týká pouze textů aplikace,

jako jsou položky menu, popisky formulářů nebo titulky tlačítek. Názvy zadané uživatelem zůstávají v původním jazyku.

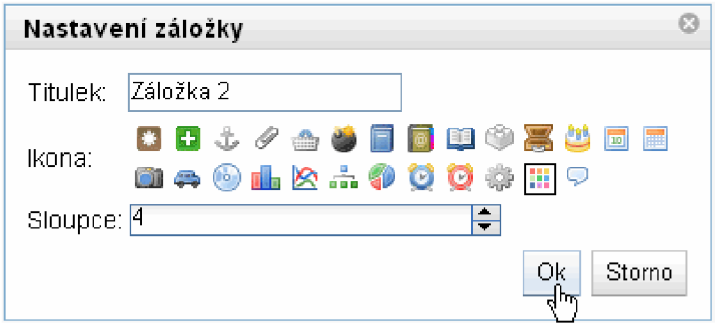
Druhé nastavení se týká grafického vzhledu pracovní plochy. Uživatel/ka bude mít k dispozici seznam grafických témat. Potvrzení dialogu způsobí překreslení aplikace, aby splňovala požadovaný vzhled.

Poslední nastavení zahrnuje seznam zdrojů, ze kterých se bude vytvářet nabídka odkazů a miniaplikací zobrazená v postranním menu. Uživatel/ka zadá URL specifikující adresu zdrojového serveru nebo vybere jeden z předvolených. Po potvrzení pracovní plocha ověří komunikaci se zdrojovým serverem. Pokud dojde k navázání spojení, automaticky se aktualizuje postranní menu. Uživatel/ka bude moci přidávat na svoji plochu odkazy a miniaplikace dodávané nově vybraným zdrojem.

4.5.2 Nastavení záložek

Na následujících řádcích vysvětlím návrh dialogu záložky (viz Obr. 4-10), který se zobrazí po stisknutí tlačítka s ikonou nastavení či při přidání nové záložky. První pole formuláře slouží pro zadání názvu záložky. Uživatel/ka si tak může pojmenovat skupinu panelů, které obsahují např. odkazy podobného zaměření.

Dialog dále nabízí volbu ikony záložky. Výběr se provádí klepnutím na jednu z předvolených ikon, jež se označí černým rámečkem. Nabídka ikon závisí na zvoleném grafickém tématu. V budoucnu je možné dialog dále rozšířit o pole pro specifikaci externí ikony. Tuto volbu jsem našel u pracovní plochy *Netvibes*.



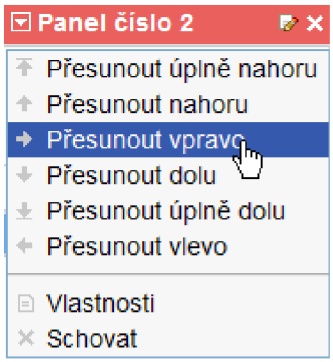
Obr. 4-10 Dialog záložky

Poslední nastavení dialogu se týká počtu sloupců rozdělujících plochu příslušné záložky. Pro jednodušší zadávání bude využito speciální vstupní pole (number spinner), které dovolí vložit pouze číslo v rozsahu 1 až 8. Počet sloupců lze inkrementovat nebo dekrementovat pomocí šipek umístěných na pravé straně daného pole.

4.5.3 Volby panelu

Všechna nastavení panelu budou přístupná přes titulkovou lištu. Panel má systémové menu (Obr. 4-11) dostupné přes ikonu šipky. Tato ikona se zobrazuje pouze v tzv. editačním režimu, který

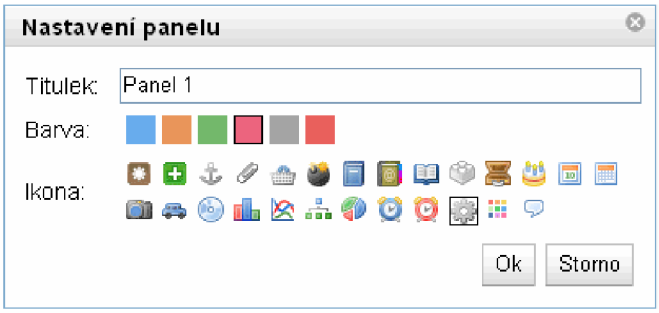
definuji v kapitole 4.5.4, jinak se místo ní zobrazí ikona panelu. Uživatelská nastavení (Obr. 4-12) obsahuje dialog vyvolaný standardním způsobem - stisknutím tlačítka s ikonou nastavení.



Obr. 4-11 Menu panelu

Nejdříve se budu věnovat návrhu systémového menu. Pro přemísťování panelů jsem využil techniku drag & drop. Pokud je však v jednom sloupci větší počet panelů a uživatel/ka potřebuje vložit panel až na poslední pozici, musí během drag & drop rolovat plochu pomocí myši. Takové přemisťování není pohodlné, a proto menu obsahuje položky pro okamžitý přesun odpovídajícího panelu na první/poslední pozici ve sloupci („Přesunout úplně nahoru“/ „Přesunout úplně dolu“). Pro úplnost jsem navrhl také změnu polohy ve zbývajících směrech. Přesun je realizován vždy o jeden sloupec vlevo/vpravo a jednu pozici v rámci sloupce nahoru/dolu. Uživatel/ka jednoduše vybere příslušnou položku v menu a panel se automaticky přesune.

Pracovní plocha musí vždy nabídnout pouze směry, v nichž je přesun korektní. Pokud se například panel, který chce uživatel/ka přesunout, nachází na první pozici v prvním sloupci a pod ním jsou ještě dva panely, nesmí mít uživatel/ka možnost přesouvat „úplně nahoru“, „nahoru“ a „vlevo“. Kromě toho je součástí menu také položka pro otevření dialogu uživatelských voleb a pro schování panelu.



Obr. 4-12 Dialog uživatelského nastavení panelu

Podobně jako u záložky, také panel bude mít uživatelský dialog, který představuje Obr. 4-12. První pole slouží pro nastavení názvu panelu. Další volbou je podpora změny barevného podkladu (background color) titulkové lišty. Kromě estetické funkce lze barvu asociovat s obsahem panelu a využít ji pro klasifikaci obsahu. Nabídka barev bude závislá na vybraném grafickém tématu. V dialogu panelu se může dále vyskytovat nastavení miniaplikace, pro kterou je panel kontejnerem.

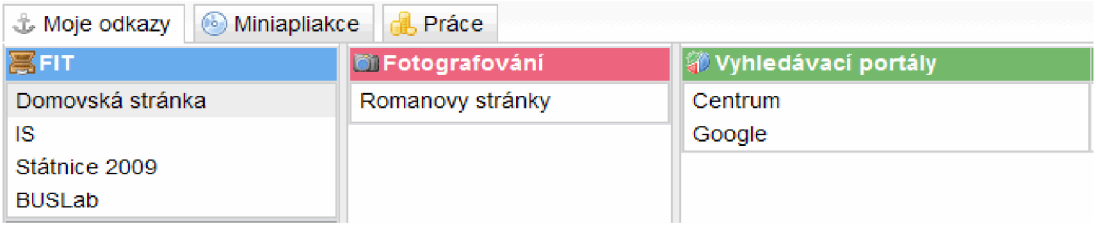
4.5.4 Editační režim

Doposud jsem předpokládal, že modifikaci pracovní plochy provádí uživatel/ka v libovolném okamžiku. Tato hypotéza ovšem přináší následující problémy:

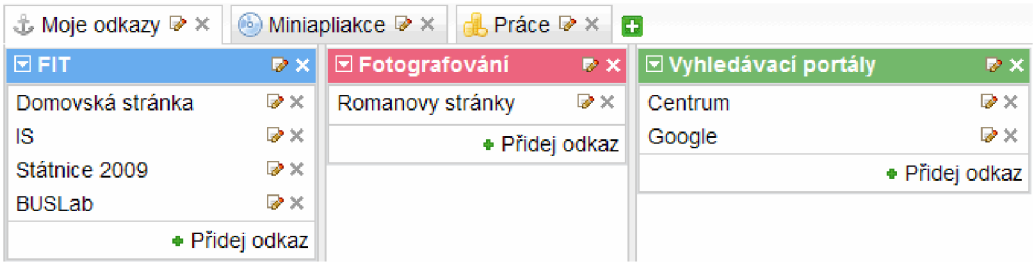
- Dochází ke kolizi mezi drag & drop a kliknutím myši na stejném prvku pracovní plochy (odkaz, panel, záložka).

- Neustále se zobrazují tlačítka pro přístup k nastavení a rušení prvků. To vede k přeplnění plochy (každý prvek má svoje vlastní) a zvyšuje riziko, že uživatel/ka omylem provede jinou akci, než zamýšlel/a.
- Pracovní plocha musí zajistit, aby každá změna provedená uživatelem/uživatelkou byla okamžitě zaznamenána do konfigurace a uložena na server.

Konflikt mezi událostí pro otevření odkazu a uchopením položky odkazu pro přemístění představuje v současném návrhu významný problém. Na rozdíl od panelu nebo záložky, kde se nachází volný prostor pro jejich uchopení pomocí techniky drag & drop, u položky odkazu takováto možnost neexistuje. Řešením by mohlo být využití dvojitého kliknutí (double click). Odkazy je ovšem obvyklé otevřít pouze pomocí jednoho kliknutí. Pokusil jsem se proto situaci řešit jiným způsobem.



Obr. 4-13 Pracovní plocha v normálním režimu



Obr. 4-14 Pracovní plocha v editačním režimu

Navrhl jsem tzv. **editační režim rozhraní**. Všechny změny spojené s akcí přidání, odebrání nebo drag & drop se budou provádět pouze v tomto režimu. Také veškerá nastavení prvků pracovní plochy včetně globálních voleb aplikace budou přístupné jenom v režimu editace. Prakticky to znamená, že uživatel/ka musí nejdříve vstoupit do editačního režimu, což učiní pomocí nového **tlačítka editace**, které umístím na informační lištu. Po stisknutí tlačítka se zobrazí všechna tlačítka sloužící pro přístup k volbám prvků pracovní plochy. Odkazy bude možné pouze přesouvat, nikoliv je otevírat. Editační režim umožní dále změnu umístění odkazů, panelů, záložek a operace přidání a skrytí. Po provedení akcí uživatel/ka ukončí editační režim opět pomocí tlačítka editace.

Editační režim řeší rovněž další dva body uvedené výše. Pokud se pracovní plocha nenachází v editačním režimu, nejsou zobrazena všechna tlačítka sloužící pro uživatelské akce, a pracovní

plocha je tak přehlednější. Rozdíl ve složitosti rozhraní ukazuje Obr. 4-13 a Obr. 4-14. Provedené změny (může jich být hodně) nemusí být navíc průběžně ukládány, ale stačí reflektovat pouze výslednou změnu po ukončení editačního režimu. Značně se tím zjednoduší algoritmus sloužící pro ukládání konfigurace.

5 Návrh konfiguračního serveru

5.1 Úloha konfiguračního serveru

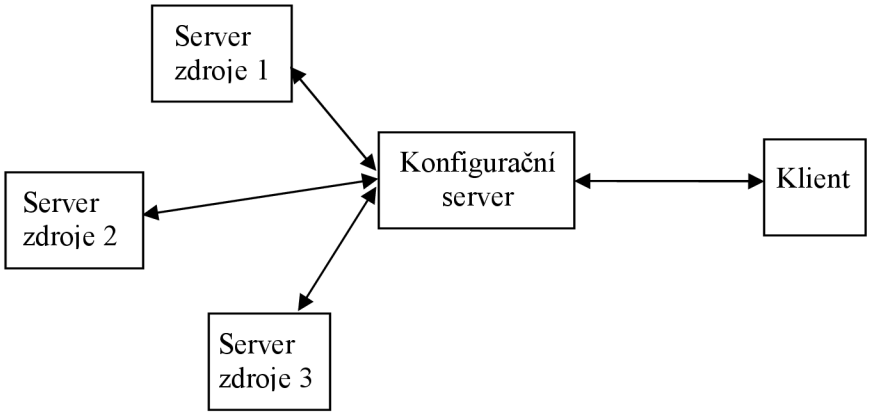
Doposud jsem se zabýval návrhem klientské části aplikace. Popsal jsem rozhraní pracovní plochy, akce, které na ní uživatel/ka může provádět, a dostupné volby. Nyní se zaměřím na neméně důležitou část aplikace - konfigurační server. Server bude spravovat databázi obsahující záznamy o jednotlivých uživatelích a též konfigurace pracovních ploch. Jeho úlohou bude zajišťovat následující služby:

1. autentizaci uživatelů/uživatelek,
2. natažení a uložení konfigurace pracovní plochy,
3. poskytnutí nabídky položek, které lze umístit na pracovní plochu.

První akcí, kterou musí uživatel/ka provést, pokud chce využívat svoji pracovní plochu, je přihlášení. Server musí provést ověření identity, aby zamezil přístup k ploše neautorizované osobě. Podle zadaného přihlašovacího jména vyhledá záznam o uživateli/uživatelce. V případě, že je záznam nalezen, provede se porovnání hesla záznamu se vstupním heslem. Hesla v databázi budou šifrována, aby v případě zcizení nedošlo k jejich prozrazení.

Další úlohou konfiguračního serveru je správa konfigurací. Navrhl jsem dvě základní operace, operaci natažení a operaci uložení. Operace natažení spočívá v přečtení a odeslání konfigurace pracovní plochy na klienta, který ji potom interpretuje a na jejím základě vytvoří a rozmístí odkazy, panely a záložky. Natažení se provede v rámci inicializace pracovní plochy po úspěšném přihlášení uživatele. Operace uložení zajistí, aby změny konfigurace byly do databáze uloženy. Provede se vždy, pokud uživatel/ka opustí editační režim.

Třetí úloha konfiguračního serveru spočívá ve zprostředkování nabídky položek zobrazené v postranním menu pracovní plochy. Konfigurační server bude komunikovat se zdrojovými servery položek, jejichž URL specifikuje uživatel/ka v konfiguraci. Situaci znázorňuje Obr. 5-1.



Obr. 5-1 Komunikace se zdrojovými servery

Uživatel si zvolil například tři zdrojové servery položek. Při prvním otevření postranního menu pošle klient konfiguračnímu serveru požadavek, aby dodal obsah menu. Konfigurační server postupně klade dotaz na zdrojové servery položek, dokud nezíská konečnou nabídku, kterou vrátí jako odpověď klientovi.

5.2 Struktura konfigurace pracovní plochy

Konfigurace popisuje prvky pracovní plochy, jejich rozmístění a uživatelská nastavení. Pro její reprezentaci jsem vybral formát **JSON** popsaný v kapitole 3.5. Podrobněji se teď zaměřím na strukturu konfigurace, kterou jsem navrhnul. Její fragment popisuje kód Př. 5-1.

Př. 5-1 Konfigurace pracovní plochy

```
{
 "theme": "theme1",
 "lang": "cz",
 "sources": [{s1: http://src1}, ...],

 "desktop": {
    "tab": [
       { "id": "t1",
         "cols": 3,
         "colSizes": "33,33,33",
         "title": "Záložka 1",
         "icon": "ico-1",
         "pane": [
            { "id": "p1",
              "type": "linkPane",
              "title": "Název panelu 1",
              "icon": "ico-2",
              "style": "paneStyle-1",
              "x": 1,
              "y": 1,
              "link": [
                 { "id": "l1",
                   "title": "Portál seznam",
                   "path": "http://www.seznam.cz",
                   "desc": "Moje pošta a vyhledávání"
                 }
              ]
            }
          ]
         },
      ...
   }
}
```

Celá konfigurace je vyjádřena jako objekt. První tři vlastnosti popisují globální nastavení aplikace, mezi které patří vybrané grafické téma (*theme*), zvolený jazyk aplikace (*lang*) a seznam s URL adresami zdrojových serverů poskytujících nabídku položek pro postranní menu pracovní plochy (*sources*). Vlastnost *desktop* udává popis obsahu pracovní plochy, který je hierarchicky uspořádaný.

Na nejvyšší úrovni se nachází specifikace seznamu záložek (*tab*). Každá záložka má jednoznačný identifikátor (*id*). Dále je zde definován počet sloupců (*cols*) a procentuálně vyjádřena šířka sloupců vzhledem k celkové šířce plochy (*colSizes*). Nechybí ani titulek záložky (*title*) a její ikona (*icon*). Záložka potom obsahuje specifikaci panelů.

Panel je charakterizován opět jednoznačným identifikátorem (*id*). Důležitou vlastnost představuje typ panelu (*type*). V příkladu používám hodnotu *linkPane*, která udává, že se jedná

o panel odkazů. Může zde být ale uveden jiný typ, jakým je například miniaplikace kalkulačky. Dále se nachází nastavení titulku (*title*), ikony (*icon*) a barevného podkladu titulkové lišty (*style*). Další dvě vlastnosti specifikují pozici panelu v ploše záložky. Vlastnost *x* udává číslo sloupce a vlastnost *y* pozici panelu v rámci tohoto sloupce. Součástí konfigurace panelu je potom seznam odkazů.

Odkaz má uveden jednoznačný identifikátor (*id*). Vlastnost *title* obsahuje název odkazu. Cíl odkazu popisuje vlastnost *path* a poslední vlastnost *desc* udává bližší charakteristiku odkazu.

5.3 Nabídka položek zdroje

Jednotlivé položky nabídky mohou reprezentovat samotné odkazy, ale také celé panely odkazů nebo miniaplikací. Pro vyjádření struktury nabídky využiji formát JSON jako u konfigurace pracovní plochy. Př. 5-2 demonstruje část nabídky položek získaných ze zdrojového serveru *s1*.

Př. 5-2 Nabídka položek

```
[
   {  id: "p1",
      title: "Programování",
      desc: "Zajímavé odkazy a nástroje pro programátora"
      type: "space",
      children: ["p2"]
      tags: "VB, C#, Python",
      source: "s1",

   },
   {
      id: "p2",
      title: "Python",
      source: "s1",
      desc: "Zajímavé odkazy pro programování v Pythonu",
      tags: ["Python", "Code"],
      type: "linkPane",
      children: ["l1"]
   },
   {
      id: "l1",
      title: "Článek o Python",
      source: "s1",
      desc: "Zajímavý článek",
      tags: ["Python", "Code"],
      type: "link",
      children: []
   }, ...
]
```

Struktura nabídky netvoří hierarchii, ale je reprezentována jako seznam (pole), jehož prvky představují jednotlivé položky. Takovéto uspořádání jsem zvolil záměrně, aby se položky zdrojů daly

uspořádat nejenom podle jejich vazeb (např. panel obsahuje určité odkazy), ale také na základě jiných atributů položky, jako jsou zdroj, z něhož pochází, klíčová slova nebo její název.

Každá položka bude charakterizována pomocí svého jednoznačného identifikátoru (id). Dále bude obsahovat název (*title*) a popis položky (*desc*). Důležitým atributem položky zdroje je typ (*type*), který určuje, zda položka bude na pracovní plochu vložena jenom jako odkaz, nebo se zde vytvoří celý panel miniaplikace. Speciálním typem je typ *space* sloužící pro sdružení odkazů a miniaplikací podobného zaměření. Seznam položek, které jsou součástí jiné položky, specifikuje atribut *children.* Klíčová slova jsou uvedena ve vlastnosti *tags.* Zdroj, ze kterého položka pochází, udává vlastnost *source*.

5.4 Komunikační protokol

Zbývá definovat, podle jakého protokolu bude komunikovat klientská část aplikace s konfiguračním serverem. Komunikace bude probíhat formou požadavku a odpovědi. Klient vytvoří spojení se serverem a zašle mu zprávu obsahující požadavek. Server ji zpracuje a odešle odpověď. Jako formát zprávy jsem vybral JSON. Důvodem je snadné sestavení zprávy a interpretace odpovědi zaslané serverem na klientu.

Nejdříve definuji strukturu zprávy požadavku. Zpráva bude obsahovat dvě základní pole, a to ***request*** a ***params***. Pole *request* specifikuje operaci, kterou má konfigurační server provést. V poli *params* jsou uvedeny parametry operace. Následující tabulka shrnuje operace podporované konfiguračnín serverem a příslušné parametry, které musí zpráva obsahovat.

Operace	**Parametry**	**Popis**
LOGIN/LOGOUT	USER, PASS/ USER	Přihlásí/odhlásí uživatele USER s heslem PASS.
LOAD/SAVE	USER/USER, CONFIG	Natáhne/uloží konfiguraci pracovní plochy CONFIG pro uživatele USER.
GET_ITEMS	SOURCE	Získá položky nabídky ze zdroje SOURCE.
EXPAND	ITEM	Získá sub-položky zadané položky ITEM.

Tab. 5-1 Operace a její parametry podporované konfiguračním serverem

Zpráva odpovědi serveru je specifikována pomocí pole ***status***, které indikuje stav odpovědi, a pole ***content*** obsahujícího samotnou odpověď korespondující s požadavkem. V případě, že došlo k chybě, obsahuje pole *content* chybové hlášení. Pro úplnost uvedu ještě stavové kódy, které se mohou vyskytovat jako hodnota pole *status* (viz Tab. 5-2).

Kód	Popis
1	Operace úspěšně provedena.
2	Neautorizovaný přístup.
3	Chyba databáze konfigurací.
4	Chyba komunikace se zdrojem nabídky položek.

Tab. 5-2 Stavové kódy a jejich význam

6 Javascriptová knihovna Dojo

6.1 Výběr knihovny

Pro implementaci pracovní plochy jsem se rozhodl využít **javascriptovou knihovnu Dojo**, která poskytuje řešení problémů uvedených v kapitole 3.6, a tak dovoluje soustředit se na samotnou aplikaci. Dojo je otevřená knihovna určená pro pohodlný a rychlý vývoj webových aplikací [14]. Mezi hlavní důvody, které mě vedly právě k této volbě, patří zejména:

1. Zajištění jednotného programového prostředí dovolujícího implementovat aplikaci pro různé webové prohlížeče.
2. Široká paleta hotových komponent pro tvorbu webového rozhraní.
3. Podpora techniky drag & drop.
4. Nadstavba jazyka Javascript (modularizace kódu, deklarace tříd, AJAX, aj.).

Knihovna Dojo nabízí překlenutí rozdílů mezi širokou škálou prohlížečů zahrnující Internet Explorer verze 6 a 7, Firefox verze 2 nebo prohlížeč Opera od verze 9. Splňuje tedy požadavek podpory prohlížečů, se kterými bude pracovní plocha kompatibilní. Dále nabízí ošetření řady chyb prohlížečů (např. hospodaření s pamětí) a přidává funkcionalitu tak, aby se vytvořila ucelená množina funkcí dostupných ve všech webových prohlížečích [13]. Mezi výhody knihovny patří rovněž udržování kompatibility s nově vznikajícími verzemi prohlížečů. V době psaní této práce se začíná používat Internet Explorer verze 8 a také nový webový prohlížeč Google Chrome. Dojo v poslední publikované verzi (ver. 1.3) již podporuje oba zmíněné prohlížeče. Použití knihovny tedy zajišťuje jednotné programové prostředí.

Velký přínos knihovny Dojo spočívá v návrhu rozhraní pomocí samostatných komponent označovaných v terminologii Dojo jako tzv. **widgets.** (V dalším textu budu užívat termín widget jako synonymum k pojmu komponenta). Knihovna nabízí komponenty pro dialogové prvky (tlačítka, textové vstupy, přepínače, výběry …), nechybí také widgety zajišťující rozložení ostatních komponent na stránce (záložky, kontejnery) nebo pokročilé komponenty (datová mřížka, strom, textový editor). Jednotlivé widgety lze upravovat, vytvářet nové a jejich kombinací sestavit i velmi složité rozhraní. Knihovna Dojo přináší paletu komponent pro tvorbu webového uživatelského rozhraní, které patří v současné době mezi jednu z největších [13].

Knihovna obsahuje implementaci techniky drag & drop. Definuje objekt manažera, který koordinuje celou operaci a vyvolává události informující o začátku, průběhu a ukončení přetahování položky. Dojo umožňuje definovat zdroje přetahovaných položek, jejich vzájemnou kompatibilitu a ošetřuje rovněž grafické vykreslení během přetahování. Na druhou stranu zůstává možnost upravit si chování objektu manažera a jednotlivých zdrojů podle potřeby. Díky této podpoře je možné velmi jednoduše realizovat drag & drop.

Knihovna přináší také rozšíření jazyka Javascript (language extensions) o řadu přínosných funkcí a technik. Kód aplikace lze rozčlenit na menší celky – moduly, ty potom podle potřeby dynamicky natahovat. Mezi další výhody patří jednoduchá definice tříd, registrace obsluhy událostí nebo manipulace s objektovým modelem dokumentu. Nechybí také podpora techniky AJAX, která dovoluje pomocí několika jednoduchých funkcí komunikovat asynchronně se serverem. Detaily způsobu zaslání požadavku a asynchronní přijetí odpovědi přitom zůstávají skryté, stejně jako rozdíly ve vytváření objektu XMLHTTPRequest v různých prohlížečích.

Dojo je otevřená knihovna dostupná zdarma pod modifikovanou BSD licencí [13]. Jednotlivé verze knihovny jsou uchovávány v SVN skladišti a jsou v současné době zpětně kompatibilní, což umožňuje přechod na novou verzi knihovny bez nutnosti významných změn v kódu. Pro potřeby prototypu pracovní plochy využívám **verzi 1.2 knihovny Dojo.**

6.2 Základní služby knihovny

Dříve než se zaměřím na popis implementace prototypu pracovní plochy, uvedu alespoň základní služby knihovny, které poté v rámci implementace často užívám. Jejich výčet není úplný. Kompletní přehled všech služeb se nachází na [14].

Knihovna Dojo se skládá ze tří projektů:

1. **Dojo** – základní služby knihovny zahrnující funkce pro manipulaci s DOM, registraci událostí, podporu AJAX, drag & drop a další.
2. **Dijit** – funkcionalita souvisejí s tvorbou webového rozhraní. Obsahuje widgety, které je možné ihned použít.
3. **Dojox** – rozšíření Dojo o další speciální komponenty a grafické knihovny.

V následujících podkapitolách se budu nejdříve zabývat projektem Dojo, potom uvedu základy tvorby uživatelského rozhraní a přiblížím projekt Dijit. Dojox obsahuje řadu užitečných rozšíření v oblasti grafiky a pokročilých komponent, které však nejsou zatím zpětně kompatibilní, a proto je v prototypu pracovní plochy nepoužívám.

6.2.1 Vložení knihovny

Knihovnu Dojo lze stáhnout z webových stránek knihovny. Aby bylo možné knihovnu využívat, musí být do hlavičky dokumentu stránky vložen inicializační skript ***dojo.js*** nacházející se v projektu Dojo (viz Př. 6-1).

Př. 6-1 Inicializační skript knihovny

```
<script type="text/javascript" src="dojoRoot/dojo/dojo.js"
   djConfig="isDebug: false, parseOnLoad: false"></script>
```

Identifikátor *dojoRoot* označuje adresář, do kterého je rozbalen stažený archiv knihovny. Prostřednictvím atributu elementu *script* se potom nastavuje výchozí konfigurace knihovny. Společně s inicializačním skriptem je potřeba přidat do hlavičky dokumentu také výchozí stylové předpisy knihovny uvedené v Př. 6-2.

Př. 6-2 Stylové předpisy pro Dojo

```
<style type="text/css">
   @import "dojoRoot/dijit/themes/tundra/tundra.css" ;
   @import "dojoRoot/dojo/resources/dojo.css";
</style>
```

Pro správné zobrazení komponent musí být přidán stylový předpis grafického tématu. V Př. 6-2 se jedná o *tundra.css*. Společně s ním je nutné přidat třídu *tundra* elementu body. Více ke grafickým tématům a jejich změně uvedu v kapitole 7.8. Dále se vloží také výchozí styly knihovny specifikované ve stylovém předpisu *dojo.css*.

6.2.2 Moduly

Knihovna Dojo je velmi rozsáhlá, aby poskytla co nejobecnější prostředky pro vytváření webové aplikace. Verze 1.2, kterou využívá prototyp pracovní plochy, obsahuje 1298 skriptů. Obvykle však není nutné využít všechny služby knihovny, a proto je její kód rozdělen podle funkcionality do menších celků – tzv. **modulů**. Modularizace se neomezuje jenom na samotnou knihovnu, ale lze jednoduše vytvářet i vlastní moduly. Kód se tak stává přehlednější a díky tomu, že se při natažení stránky nestahují všechny dostupné skripty, je aplikace používající moduly rovněž rychlejší.

Kód modulu je umístěn ve vlastním souboru se sufixem *.js*. První příkaz modulu musí obsahovat volání metody ***dojo.provide(module)***, které se předá jako parametr řetězec s jednoznačným identifikátorem modulu. Ten zároveň popisuje umístění souboru modulu v souborovém systému. Natažení modulu se potom provede dynamicky pomocí metody ***dojo.require(module).*** Argumentem metody je odpovídající identifikátor modulu.

Aby bylo možné natáhnout vlastní modul, musí se nejprve specifikovat jmenný prostor. K tomuto účelu slouží funkce ***dojo.registerModulePath(namespace, path)***, kde *namespace* udává identifikátor jmenného prostoru a *path* potom jeho mapování na cestu v souborovém systému. Cesta se zadává relativně vzhledem k bázovému skriptu *dojo.js*, který se vkládá během inicializace knihovny. V okamžiku, kdy je definován jmenný prostor, lze již jednoduše sestavit identifikátor modulu následujícím způsobem:

```
JmennyProstor.relativniCesta.NazevSouboru
```

Identifikátor *relativniCesta* představuje relativní cestu vzhledem k jmennému prostoru. Skládá se ze složek oddělených znakem tečka a každá další složka identifikátoru reprezentuje zanoření se o jednu úroveň níže v adresářové struktuře souborového systému. *NazevSouboru* odpovídá názvu souboru modulu bez sufixu *.js*. Pokud tedy zaregistrujeme jmenný prostor A s adresou *../../A* a modul nazveme *A.B.Modul*, potom bude vkládaný soubor modulu hledán na adrese *../../A/B/Modul.js* relativně k umístění *dojo.js*. Moduly knihovny Dojo se natahují stejným způsobem a jmenné prostory *dojo*, *dijit* a *dojox* se registrují automaticky při inicializaci knihovny.

Zatím jsem neuvedl, jakým způsobem je následně kód modulu dostupný ve skriptu, který ho využívá. Knihovna Dojo při vložení modulu vytvoří tzv. ***modulový objekt***. Jedná se o javascriptový objekt, jehož členy jsou třídy, funkce a proměnné definované v rámci modulu. Př. 6-3 demonstruje vytvoření modulu a volání jeho kódu.

Př. 6-3 Definice modulu a jeho použití

```
//Skript využívající modul

dojo.require("A.B.Modul");
A.B.Modul.funkce();
var tmp = A.B.Modul.promenna;
```

```
//Skript modulu  v souboru
  Modul.js
dojo.provide("A.B.Modul");
A.B.Modul.funkce = function(){…}
A.B.Modul.promenna = 1;
```

A.B.Modul představuje modulový objekt s globální platností, prostřednictvím kterého se přistupuje k funkci a proměnné definované v modulu. Alternativou je využít modulový objekt jmenného prostoru *A*, ze kterého je potom kód modulu přímo dostupný (viz. Př. 6-4).

Př. 6-4 Alternativní definice modulu a její použití

```
//Skript využívající modul

dojo.require("A.B.Modul");
A.funkce();
var tmp = A.promenna;
```

```
//Skript modulu  v souboru
  Modul.js
dojo.provide("A.B.Modul");
A.funkce = function(){…}
A.promenna = 1;
```

6.2.3 Definice třídy a dědičnost

V kapitole 6.1 uvádím, že knihovna Dojo přináší také praktické rozšíření jazyka Javascript. Jedním z nich je zjednodušení definice tříd a dědičnosti. Místo komplikovaného používání vlastnosti *prototype* postačuje zavolat metodu ***dojo.declare()*** (viz Př. 6-5).

Př. 6-5 Deklarace třídy v knihovně Dojo

```
dojo.declare("A.Trida", [Predek1, Predek2,…], {
   vlastnost: 2,
   statics: { counter: 0, pozdrav: "hello" },

   constructor: function(par1, par2) {
      this.komplikovanyObjekt = new mujObjekt();
      this.dalsiVlastnost = par1;
      this.statics.counter += 1 //statické počítadlo instancí
   },

   metoda: function(par) {
      return par + 10;
   }
});
```

První parametr *dojo.declare()* reprezentuje jméno deklarované třídy a jde vždy o řetězec. Druhý parametr je buď *null*, nebo představuje odkaz na jeden nebo více předků, jejichž vlastnosti a metody budou přidány do jmenného prostoru objektu nově vytvářené třídy. Dojo tedy umožňuje **vícenásobnou dědičnost**. V případě více předků je nutné předat jejich pole. Třetí parametr představuje objekt popisující členy třídy. Pro jeho deklaraci se využívá zkráceného zápisu objektu pomocí složených závorek.

Vlastnosti, které budou odkazovat na pole nebo objekt, je nutné inicializovat v konstruktoru třídy. Inicializace provedená už v rámci deklarace vlastností způsobí, že všechny instance třídy budou tyto vlastnosti sdílet (budou statické). Primitivní vlastnosti, které mají být naopak statické, musí být deklarovány pomocí vlastnosti *statics* (viz Př. 6-5). Jednotlivé statické proměnné vystupují jako složky objektu přístupného přes tuto vlastnost.

Pro definici konstruktoru se použije identifikátor ***constructor***. Knihovna Dojo dovoluje také jednoduché volání metody předka, která byla novou třídou přetížena. Postačuje použít metodu třídy ***inherited*()**. Vytvoření instance výše definované třídy ukazuje kód Př. 6-6.

Př. 6-6 Vytvoření intance třídy

```
var instance = new A.Trida(par1, par2);
```

6.2.4 Událostní model

Knihovna Dojo poskytuje konsolidované API pro práci s událostním modelem. Základ standardizace spočívá v normalizaci objektu události, který je podle [15, s. 68] upraven vždy tak, aby splňoval specifikaci DOM úrovně 2 dostupné na [12]. Kromě sjednocených vlastností objektu události nabízí knihovna dále standardizované kódy kláves pro obsluhu vstupu z klávesnice počítače a metody týkající šíření událostí modelem dokumentu. Nejdůležitější přínos knihovny však podle mého názoru spočívá v API pro registraci obslužné funkce události. Knihovna Dojo dovoluje přihlásit obsluhu v následujících situacích:

1. při dokončení natažení/destrukce DOM,
2. při události, která nastala na DOM uzlu, nebo při zavolání metody libovolného objektu,
3. při publikování uživatelské události.

V [16] se uvádí, že manipulaci s modelem dokumentu nebo registraci obsluhy událostí je vhodné provádět až v momentě, kdy je kompletně natažen celý model dokumentu, jinak dochází k nepředvídatelným chybám. Rovněž [15, s. 70] doporučuje, aby se veškerá registrace obsluhy událostí prováděla až v tento okamžik. Knihovna Dojo proto poskytuje metodu ***dojo.addOnLoad()***, které se předá jako jediný parametr funkce obsahující kód skriptu, jež se má provést. Knihovna podle [16] zajistí, že se skript spustí po natažení modelu dokumentu, ale před samotným stažením obrázků a dalších zdrojů, které by jinak jeho provedení zdržovaly. Podobně pomocí metody knihovny *dojo.addOnUnload()* je možné ve správný okamžik provést akce spojené s destrukcí dokumentu.

Registrace obsluhy události se podle [15, s. 70] provádí pomocí metody knihovny ***dojo.connect()***. První dva parametry metody specifikují buď DOM uzel a název události, která na něm nastala, nebo libovolný objekt a název metody tohoto objektu, která se provedla. Jako třetí parametr se předává obslužný objekt a čtvrtým parametrem je potom název metody tohoto objektu. Pokud vykonává obsluhu pouze funkce, jako třetí parametr se předá *null*. Knihovna Dojo tímto způsobem zobecňuje celý proces registrace, protože umožňuje reagovat nejenom na událost vzniklou na DOM uzlu, ale rovněž na provedení libovolné metody objektu. Knihovna dovoluje registrovat neomezený počet obsluh, které se potom volají v pořadí přihlášení. Metoda *dojo.connect()* vrací záznam o registraci. Ten je nutný předat jako parametr metodě ***dojo.disconnect()***, pokud se má reakce na událost nebo provedení metody zrušit.

Poslední významnou službu knihovny Dojo v kontextu událostního modelu představuje implementace návrhového vzoru **vydavatel-odběratel** (publisher-subscriber), který popisuje [15, s. 76]. Návrhový vzor rozlišuje objekty, které vzájemně komunikují, na dvě skupiny. První skupina zahrnuje vydavatele (publishers), kteří informují, že došlo ke vzniku události. Druhá skupina potom obsahuje odběratele (subscribers). Ti se registrují u vydavatele, na jehož události potřebují reagovat. Výhodou takovéto komunikace je, že vydavatel nepotřebuje znát svoje odběratele, vysílá pouze informaci. Každý odběratel si potom sám rozhoduje, jestli bude informaci odebírat. Na základě tohoto návrhového vzoru lze potom definovat komunikaci mezi anonymními objekty, např. mezi komponentami webového rozhraní.

Pro realizaci návrhového vzoru nabízí knihovna Dojo tři metody, jak uvádí [13, s. 121]. Metoda ***dojo.publish()*** dovoluje publikovat novou událost. Jako první parametr se předává řetězec popisující událost. V terminologii knihovny bývá označován jako téma (topic) komunikace. Druhý parametr potom představuje vysílanou informaci ve formě pole hodnot. Metody ***dojo.subscribe()*** a ***dojo.unsubscribe()*** umožňují přihlášení/odhlášení odběru informace. První parametr metody *dojo.subscribe()* obsahuje identifikátor události, druhý specifikuje objekt odběratele a třetí potom název jeho metody. Odběratelem může být i samotná funkce. V takovém případě je druhý parametr

null. Metoda *dojo.subscribe()* podobně jako metoda *dojo.connect()* vrací záznam o registraci, který je potřeba při odhlášení odběru pomocí metody *dojo.unsubscribe()*.

6.2.5 Podpora AJAX

Knihovna Dojo zapouzdřuje objekt XMLHttpRequest a nabízí jednoduchou realizaci asynchronních požadavků prostřednictvím sady metod uvedených v Tab. 6-1 [15, s. 84].

Metoda	**Popis**
dojo.xhrGet(args)	Zašle http požadavek metodou GET.
dojo.xhrPost(args)	Zašle http požadavek metodou POST.
dojo.rawXhrPost(args)	Zašle http požadavek metodou POST a dovoluje připojit data do těla zprávy.
dojo.xhrPut(args)	Zašle http požadavek metodou PUT.
dojo.rawXhrPut(args)	Zašle http požadavek metodou PUT a dovoluje připojit data do těla zprávy.
dojo.xhrDelete(args)	Zašle http požadavek metodou DELETE.
dojo.xhr(method, args, hasBody)	Zašle požadavek metodou specifikovanou v parametru *method*. Parametr *hasBody* udává, jestli bude obsah připojen do těla zprávy.

Tab. 6-1 Metody knihovny pro realizaci asynchronního požadavku

Všechny metody výše mají stejný argument *args*, který obsahuje objekt (slovník) s parametry požadavku. Jejich výčet přehledně shrnuje následující tabulka.

Název	**Typ**	**Význam**
url	String	URL požadavku.
content	Object	Obsah požadavku jako objekt.
handleAs	String	Formát přijaté zprávy (text, xml, json, …).
timeout	Number	Timeout dotazu v milisekundách.
sync	Boolean	Má se provádět synchronní dotaz. Výchozí je false.
load	Fun	Zpracování úspěšné asynchronní odpovědi.
error	Fun	Obsluha v případě neúspěšného požadavku.
form	String	Id formuláře, jehož pole (musí mít atribut name) se přidají do obsahu požadavku.

Tab. 6-2 Parametry asnychornního požadavku

Výchozí formát odpovědi je nastaven vždy na *text*. Pokud má být odpověď zpracována např. ve formátu JSON, musí se nastavit parametr ***handleAs*** na hodnotu *json*. Knihovna potom zajistí správnou interpretaci odpovědi serveru. Požadavek je kladen asynchronně, pokud není parametrem

sync explicitně nastaveno synchronní provedení. Z tohoto důvodu je potřeba registrovat obslužnou funkci, která se zavolá v momentu, kdy je již znám výsledek požadavku. V případě úspěchu se volá funkce definovaná parametrem dotazu ***load***, při neúspěchu funkce udaná parametrem dotazu ***error***. Důležitým parametr představuje také parametr ***timeout*** specifikující čas, po jehož uplynutí vyhodnotí knihovna požadavek za neúspěšný.

6.2.6 Drag & drop

Knihovna Dojo přináší také API pro realizaci techniky drag & drop. Nejdříve uvedu důležité pojmy související s abstrakcí této operace. Informace jsem čerpal z [18].

Zdroj (source) představuje oblast rozhraní aplikace z/do níž lze uchopit/upustit položku (předmět přetahování). Na tomto místě bych rád zdůraznil, že zdroj je zároveň cílem (target) operace drag & drop. Zdroj uchovává aktuální seznam položek a nabízí základní operace jako přidání, smazání položky a iteraci přes všechny položky. Libovolný počet položek může být označen jako **vybrané** (selected). Během operace drag & drop je potom přesouvána celá skupina označených položek. Důležitou vlastností zdroje je seznam přípustných **typů** položek, které může přijmout. V případě, že je použito více zdrojů, lze stanovit kompatibilitu zdrojů a povolit přesun položek např. pouze mezi některými z nich.

Položka (item) reprezentuje předmět přetahování. Knihovna Dojo striktně odděluje data položky od její vizuální podoby. Způsob, jakým se položka zobrazí ve zdroji, je definován pomocí speciální **vytvářecí funkce** (creator function). Objekt zdroje uchovává tedy pouze data a kdykoliv je potřeba vytvořit grafickou reprezentaci položky, zavolá vytvářecí funkci a ta vrátí uzel reprezentující položku v objektovém modelu dokumentu. Knihovna tedy dovoluje odděleně definovat vzhled položky, a dokonce ho podle potřeby dynamicky měnit.

Oddělení dat a vzhledu vychází z myšlenky, že webová aplikace obvykle získá nejprve data pomocí asynchronního požadavku a potom teprve tyto data zobrazuje. Díky tomuto přístupu jsou data při ukládání ihned k dispozici (seznam položek zdroje) a mohou být odeslány nebo umístěny do datového skladu. Položka obsahuje stejně jako zdroj **seznam kompatibilních typů zdrojů**, do kterých může být umístěna.

„Ztělesnění“ (avatar) položky představuje grafické zobrazení přesouvané položky. Jedná se o uzel DOM vytvořený prostřednictvím vytvářecí funkce zdroje, který se přemisťuje se změnou pozice kurzoru a signalizuje uživateli aktuální polohu přesouvané položky.

Manažer (manager) je objekt zajišťující koordinaci operace drag & drop. Zpracovává události vzniklé na uzlech dokumentu a na jejich základě vyvolává nové události signalizující aktuální stav operace. Mezi události drag & drop patří následující:

1. zahájení drag & drop (DndStart),
2. vstup do oblasti zdroje (DnDOver),
3. upuštění položky na zdroji (DnDDrop),

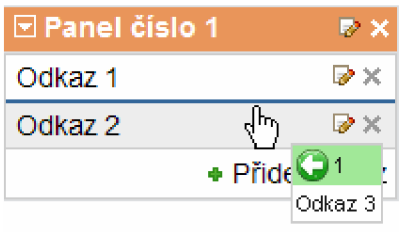
4. zrušení přetahování (DndCancel).

Druhý důležitý úkol objektu spočívá v uchovávání kontextu operace drag & drop. Manažer ukládá informaci o zdroji, ze kterého pochází přetahované položky, dále seznam přetahovaných DOM uzlů reprezentující položky a aktuální cílový zdroj.

Nyní je už možné přistoupit k popisu jednotlivých tříd, které realizují uvedenou abstrakci. **Zdroj** je podle [15, s. 156] reprezentován instancí třídy ***dojo.dnd.Source***. Jako první parametr konstruktoru se předává reference na uzel dokumentu, pomocí něhož se vymezí hranice oblasti zdroje. Druhý parametr konstruktoru je potom slovník parametrů zdroje. Jejich výčet uvádí Tab. 6-3.

Parametr	**Typ**	**Popis**
creator	function	Vytvářecí funkce, která se použije pro vizualizaci položek zdroje.
horizontal	Boolean	Položky zdroje jsou orientovány horizontálně (vedle sebe). Důležité pro správné umístění indikátoru pozice přetahované položky. Výchozí je false, tj. položky orientované vertikálně (pod sebou).
copyOnly	Boolean	Položka je po ukončení drag & drop ze zdroje zkopírována. Výchozí hodnota je false, tj. dojde k přesunu položky.
withHandles	Boolean	Pokud je nastaven, vizuální podobu položky lze uchopit pouze za oblast (DOM uzel), který má třídu **dojoDndHandle**. Výchozí je false, tj. lze zahájit drag & drop v místě oblasti uzlu položky.
accept	Array	Pole identifikátorů reprezentujících povolené typy pro test kompatibility vkládané položky. Jako výchozí je pole [“text”], které signalizuje, že položku lze vložit do libovolného zdroje.

Tab. 6-3 Parametry zdroje drag & drop



Během pohybu přetahované položky nad zdrojem je potřeba označit pozici, na kterou se v případě uvolnění tlačítka položka umístí. Objekt zdroje zajistí, že uzlu reprezentujícím položku před/za místem, kam se vloží přemisťovaná položka, připojí příslušnou třídu dojoDndItemBefore/ *dojoDndItemAfter*, která zajistí vykreslení indikátoru. Výpočet pozice indikátoru se provádí na základě pozice kurzoru myši a znalosti uzlu položky, nad kterou se myš právě nachází. Podle nastavené orientace (vlastnost *horizontal*) se potom na základě x-ové nebo y-ové souřadnice spočte, jestli se položka vkládá před/za uzel referenční položky.

Položka nemá specifikovanou žádnou speciální třídu. Existují dva způsoby, jak vložit do zdroje výchozí položky. První možností je dynamicky vytvořit instanci třídy *dojo.dnd.Source* a prostřednictvím metody ***insertNodes()*** vložit pole s datovým popisem jednotlivých položek. Vložení dvou položek ukazuje Př. 6-7.

Př. 6-7 Vložení položek do zdroje

```
source.insertNodes(false,
   [{data: "položka 1", type: ["type1"]},
    {data: "položka 2", type: ["type2"]}]);
```

Položka je reprezentována objektem se dvěma důležitými vlastnostmi. Vlastnost *data* obsahuje samotný datový popis, vlastnost *type* potom specifikuje pole typů zdrojů, se kterými je položka kompatibilní.

Druhou možností je využít statickou definici pomocí jazyka HTML (viz Př. 6-8). Položka je vyjádřena jako dětský uzel uzlu zdroje a musí mít třídu *dojoDndItem*. Typ se specifikuje atributem *dndType* a datová část potom pomocí atributu *dndData*. V případě, že není atribut *dndData* uveden, data se nastaví jako hodnota vlastnosti *innerHTML* elementu položky.

Př. 6-8 Statická definice zdroje a jeho položek

```
<div dojoType="dojo.dnd.Source">
   <div class="dojoDndItem">položka 1</span>
   <div class="dojoDndItem">položka 2</span>
</div>
```

Manažer představuje instanci třídy ***dojo.dnd.Manager***. Existuje vždy maximálně jeden manažer, který je přístupný voláním funkce *dojo.dnd.manager (),* a jeho vytvoření zajistí knihovna automaticky. Objekt manažeru využívá k publikaci událostí metodu *dojo.publish()* popsanou v kapitole 6.2.4. Př. 6-9 ukazuje jednotlivé typy událostí vznikajících během drag & drop a důležité parametry, které poskytuje manažer zdrojům.

Př. 6-9 Události operace drag & drop

```
//publikace, kterou provádí manažer
dojo.publish("/dnd/start", [source, nodes, copy]);
dojo.publish("/dnd/source/over");
dojo.publish("/dnd/drop/before", [source, nodes, copy]);
dojo.publish("/dnd/drop", [source, nodes, copy]);
dojo.publish("/dnd/cancel");
```

Parametr *source* obsahuje referenci na zdroj, ze kterého pochází přetahované položky. Parametr *nodes* potom specifikuje pole jejich DOM uzlů a konečně parametr *copy* udává, jestli je operace prováděna v kopírovacím režimu.

6.3 Tvorba webového grafického rozhraní

V této kapitole se budu zabývat projektem Dijit a pokusím se nastínit základy tvorby webové rozhraní prostřednictvím widgetů, které potom použiji pro implementaci prototypu pracovní plochy. V [15, s. 245-246] se uvádí, že projekt Dijit využívá funkcionalitu projektu Dojo pro vytvoření znovupoužitelných a rozšiřitelných widgetů. Mezi hlavní cíle projektu patří podle [tamtéž] zejména:

- nabídka široké palety standardních widgetů pro webový vývoj,
- podpora jazykové lokalizace widgetů a obousměrného textu,
- jednotné API dovolující využít znalostí z jiných widgetů pro tvorbu nových,
- umožnit jednoduše definovat widgety deklarativně pomocí jazyka HTML,
- podpora nejvíce rozšířených prohlížečů jako Internet Explorer, Firefox nebo Safari.

6.3.1 Koncept widgetů

Základní myšlenkou widgetů je nabídnout kolekci komponent, které lze libovolně kombinovat, a vytvořit tak i složitá webová rozhraní. Widgety lze zanořovat, libovolně rozšiřovat, nebo vytvářet úplně nové.

Ve skriptu je podle [15, s. 272] widget reprezentován instancí třídy ***dijit._Widget*** (Jmenný prostor třídy se nyní změnil na *dijit*). Tato třída definuje bázové vlastnosti a metody, které musí mít každá komponenta webového rozhraní. Objekt widgetu zapouzdřuje všechna data a chování komponenty. V objektovém modelu dokumentu je potom reprezentován uzlem, jehož reference je uložena ve vlastnosti ***domNode***. Knihovna Dojo tak odděluje funkcionalitu komponenty od jejího vzhledu. Další důležitou vlastností je vlastnosti ***id*** obsahující jednoznačný identifikátor objektu widgetu. Pod ním je nově vytvořená komponenta uložena do registru a díky tomu jí lze později vyhledat pomocí metody knihovny ***dijit.byId()***.

Konstruktor třídy *dijit._Widget* přijímá dva parametry. Prvním z nich je objekt (slovník) specifikující výchozí vlastnosti widgetu. Druhý parametr představuje referenční uzel modelu dokumentu, který se nahradí uzlem komponenty uchovávaným ve vlastnosti *domNode*. Př. 6-10 obsahuje kód pro vytvoření jednoduchého widgetu tlačítka.

Př. 6-10 Vytvoření widgetu tlačítka

```
var ref = dojo.byId("placeHolder");
var b = new dijit.form.Button({label: "Ok", type: "submit"}, ref);
```

Hodnoty vlastností *label* a *type* se uloží jako hodnoty stejně pojmenovaných vlastností widgetu. Dojde tedy ke sloučení objektu předaného v prvním parametru s objektem komponenty. Referenční uzel dokumentu *ref* je potom nahrazen uzlem widgetu. Alternativou je ponechat druhý parametr nevyplněn a připojit uzel widgetu do modelu dokumentu pomocí standardní metody *appendChild()*.

Od okamžiku vytvoření po jeho destrukci prochází widget tzv. **životním cyklem**. Ten se skládá z **osmi** základních fází. Pro každou fázi specifikuje třída *dijit._Widget* metodu, kterou lze přetížit, a ovlivnit tak výsledný vzhled a chování widgetu [15, s. 275]. Mezi metody životního cyklu widgetu podle [tamtéž s. 277-278] patří:

1. ***preamble()***

   Jedná se o speciální metodu volanou při vytváření instance widgetu ještě před standardním konstruktorem, která dovoluje upravit vstupní parametry. Podle [tamtéž] není její modifikace příliš častá.

2. ***constructor()***

   Tato metoda zahajuje vytváření instance třídy *dijit._Widget* a zpracovává její vstupní parametry. V [tamtéž] se uvádí dvě důležité operace probíhající ve fázi konstrukce. První je inicializace vlastností neprimitivních typů (objekty, pole). Druhá potom zajišťuje přidání vlastností potřebných pro další metody životního cyklu.

3. ***postMixInProperties()***

   Metoda je zavolána v okamžiku, kdy jsou do objektu widgetu přidány všechny vlastnosti definované ve třídách, ze kterých třída widgetu dědí. Přetížením metody lze tyto vlastnosti upravit nebo doplnit vlastnosti nové.

4. ***buildRendering()***

   Vytvoření grafické reprezentace widgetu pomocí stromu DOM uzlů a uložení reference na kořen stromu do vlastnosti *domNode* je úkolem této metody. Bázová třída *dijit._Widget* využívá referenční uzel dokumentu předaný v konstruktoru, jehož odkaz pouze dosadí do vlastnosti *domNode*. Přetížením metody lze však realizovat definici vzhledu komponenty pomocí šablon, jejichž použití popisuje kapitola 6.3.2.

5. ***postCreate()***

   Metoda je provedena po vytvoření widgetu a umístění jeho uzlu do modelu dokumentu. V tomto momentě lze provést všechny akce, pro které jsou potřeba uzly grafické reprezentace komponenty. Modifikací metody lze např. zajistit registraci obsluhy událostí. Podle [tamtéž] **nejsou** ale zatím dostupné **dětské widgety** komponenty.

6. ***startup()***

   Poslední metoda v procesu konstrukce. Volá se po dokončení vytváření dětských widgetů. Potom, co se provede metoda *startup()* na nejvyšší úrovni hierarchie komponent, provádí se postupně rekurzivně pro všechny sub-komponenty.

7. ***destroyRecursive()***

   Podobně jako je možné kontrolovat proces vytváření widgetu, dovoluje knihovna ovlivnit jeho destrukci. Tato metoda zajistí rekurzivní zrušení komponenty a jejich dětí. Pro implementaci uvolnění specifických prostředků widgetu by se měla primárně využít metoda *uninitialized(),* která je automaticky volána během destrukce [tamtéž].

8. ***uninitialized()***

   Představuje poslední metodu, která se provede během životního cyklu widgetu a slouží pro finální destrukci komponenty.

Zatím jsem popisoval dynamické vytváření widgetu pomocí skriptu v jazyce Javascript. Krátce se ještě zmíním o druhém způsobu vytvoření komponenty – deklarativní definici pomocí jazyka HTML zmíněné v [15, s. 257-258]. Knihovna Dojo dovoluje u běžných značek HTML zapsat speciální atribut ***dojoType***, jehož hodnota obsahuje plný název třídy widgetu. Pomocí dalších atributů uzlu lze potom definovat vlastnosti komponenty, které se ve skriptu specifikovaly prostřednictvím prvního parametru konstruktoru. Statickou definici widgetu tlačítka z Př. 6-10 demonstruje Př. 6-11.

Př. 6-11 Statická definice tlačítka

```
<div id="placeHolder" dojoType="dijit.form.Button" label="Ok"
type="submit" />
```

Pokud jsou widgety definovány tímto způsobem, musí se v konfiguračním objektu knihovny nastavit vlastnost ***parseOnLoad*** na hodnotu *true*. Dále je nutné vložit modul překladače komponent a také modul popisující widget, který je v dokumentu staticky deklarován [13, s. 25]. Potřebný kód pro výše uvedený příklad tlačítka ukazuje Př. 6-12.

Př. 6-12 Kód potřebný pro statickou definici tlačítka

```
<script type="text/javascript" src="dojoRoot/dojo/dojo.js"
   djConfig="parseOnLoad: true"></script>

<script type="text/javascript">
   dojo.require("dojo.parser");
   dojo.require("dijit.form.Button");
</script>
```

V implementaci prototypu pracovní plochy budu využívat výhradně dynamickou definici widgetů, protože data potřebná pro jejich vytvoření se získají až za běhu aplikace pomocí asynchronního dotazu.

6.3.2 Definice vzhledu pomocí šablon

Další významné vylepšení widgetů přináší možnost definovat vzhled komponenty pomocí **šablony**. Šablonu reprezentuje samostatný soubor obsahující kód v jazyce HTML. Prostřednictvím HTML značek lze potom jednoduše popsat stromovou hierarchii uzlů pro grafické znázornění komponenty, na rozdíl od složitého vytváření uzlů pomocí skriptu. Velký přínos šablon spočívá rovněž v oddělení tvorby vzhledu od samotného programování komponenty [15, s. 280].

Kromě samotné podpory šablon nabízí knihovna Dojo prostředky pro propojení šablony s objektem komponenty. Mezi nejdůležitější podle [tamtéž] patří:

- podpora substituce vlastností widgetu do šablony,
- připojovací body (attach points),

- událostní body (event points).

Během konstrukce widgetu se inicializují jeho vlastnosti, a tak může vzniknout potřeba některé hodnoty těchto vlastností vložit do šablony. V Př. 6-10 se jedná o vlastnost *label* obsahující název tlačítka. Bez podpory substituce by se musel název přidat pomocí přetížení metody *buildRendering()*. Knihovna však dovoluje jednoduší způsob prostřednictvím syntaxe ***${label}***. V substituci už není nutné uvádět klíčové slovo *this*. Název vlastnosti se během překladu šablony automaticky vyhledá v objektu widgetu.

Připojovací body představují opačný mechanismus umožňující získat referenci na uzel definovaný HTML značkou šablony. Pomocí speciálního atributu ***dojoAttachPoint*** se označí uzel šablony, na něhož je potřeba získat odkaz. Hodnota atributu potom udává identifikátor vlastnosti, která se během interpretace šablony v objektu widgetu vytvoří a do které se příslušná reference uloží.

Poslední prostředek se týká registrace obsluhy událostí vznikajících na uzlech definovaných v šabloně. Přihlášení se provádí speciálním atributem ***dojoAttachEvent***. Hodnota atributu má syntax uvedenou v Př. 6-13.

Př. 6-13 Syntaxe hodnoty atributu AttachEvent

```
"udalost : metoda_obsluhy1 [, metoda_obsluhy2, …]"
```

Pokud dojde k události *udalost* na uzlu označeném atributem *dojoAttachEvent*, zavolá se metoda widgetu s názvem *metoda_obsluhy1*. Metod obsluhy může být uvedeno více a v hodnotě atributu se jejich identifikátory oddělují čárkou.

Aby bylo možné využít šablonu, musí widget obsahovat funkcionalitu, která provede její interpretaci, vyřeší substituci vlastností, správně uloží reference na označené uzly a přihlásí vybrané metody widgetu jako obsluhu příslušných událostí. Knihovna Dojo podle [15, s. 280] realizuje koncept šablon třídou ***dijit._Templated***. Třída přetěžuje metodu životního cyklu *buildRendering()* widgetu a doplňuje nové vlastnosti potřebné pro užívání šablon. Jejich přehled je uveden v Tab. 6-4.

Vlastnost	Typ	Popis
templatePath	String	URL udávající umístění souboru šablony v souborovém systému relativně k umístění inicializačního skriptu *dojo.js*. Při prvním vytvoření widgetu knihovna synchronním požadavkem stáhne šablonu a provede její interpretaci. Při dalším použití widgetu se již využije záloha (cache) šablony.
templateString	String	Šablonu lze také definovat interně jako hodnotu této vlastnosti.
widgetsInTemplate	Boolean	Udává, jestli šablona widgetu obsahuje deklaraci dětských widgetů pomocí atributu **dojoType**, které je potřeba zpracovat. Výchozí hodnota je *false*.

Tab. 6-4 Vlastnosti spojené s využitím šablony

Oddělení funkcionality spojené s využitím šablon od definice základních vlastností a metod widgetu dovoluje při vytváření nové komponenty vybrat optimální řešení. Pokud je grafická reprezentace widgetu jednoduchá, uvede se jako předek nové komponenty jenom bázová třída *dijit._Widget*. Naopak v situaci, kdy je nutné definovat složitější vzhled widgetu, stačí přidat do seznamu předků třídu *dijit._Templated* (knihovna podporuje vícenásobnou dědičnost) a zobrazení widgetu definovat pomocí šablony.

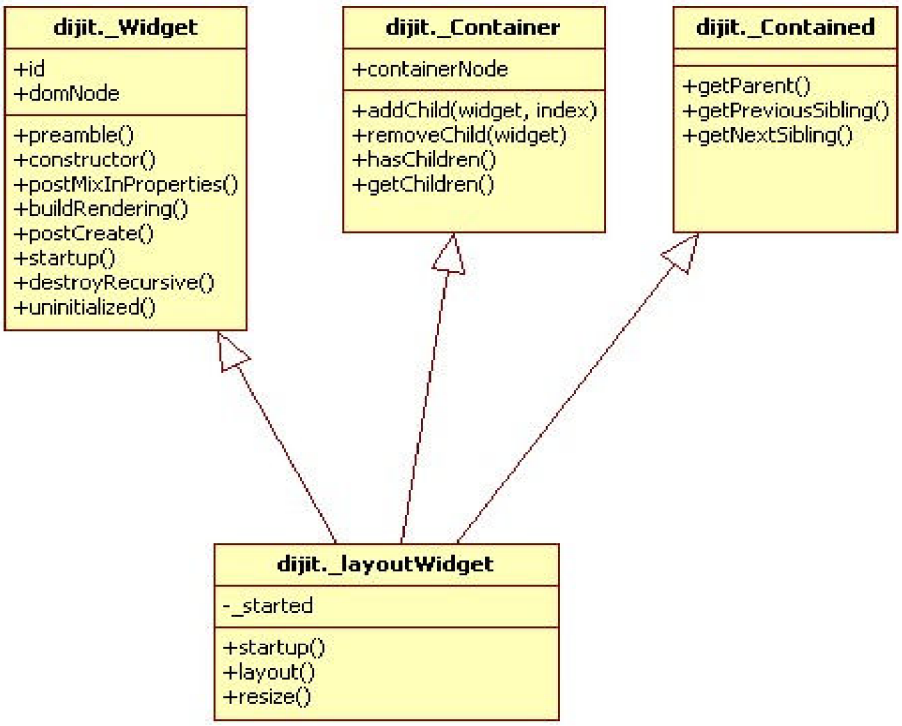
6.3.3 Layout widgety

Widgety knihovny Dojo se dělí do tří základních skupin popsaných v [15, s. 266]:

1. **Formulářové widgety (dijit.form.*)** – všechny komponenty použitelné jako prvky formuláře (textové pole, zaškrtávací políčko, tlačítko, roletová nabídka, atd.).
2. **Layout widgety (dijit.layout.*)** – widgety zajišťující rozložení ostatních komponent webového rozhraní (záložky, kontejnery).
3. **Aplikační widgety (dijit.*)** – ostatní potřebné komponenty (menu, dialogy, kalendář, textový editor, aj.).

Podrobněji se nyní budu zabývat **layout widgety**, protože prototyp pracovní plochy musí realizovat kontejner pro rozložení panelů do sloupců a bude rovněž využívat kontejner záložek.

Pro komponenty zajišťující rozložení je charakteristické, že představují kontejnery jiných widgetů a zajišťují jejich rozmístění. Knihovna Dojo definuje třídu ***dijit._layoutWidget***, která poskytuje základní funkcionalitu potřebnou pro rozložení komponent a nabízí také podporu pro změnu jejich velikosti. Následující obrázek přehledně ukazuje předky třídy *dijit._layoutWidget* a zachycuje jejich důležité vlastnosti a metody.



Obr. 6-1 Diagram tříd v UML znázorňující předky třídy dijit._LayoutWidget

Komponenty pro rozložení jsou samy také widgety, a proto třída *dijit._layoutWidget* dědí bázovou třídu *dijit._Widget* popsanou v kapitole 6.3.1. Nově se ovšem v seznamu předků objevují třídy *dijit._Container* a *dijit._Contained*, které modelují **vztah rodič-dítě** [15, s. 293].

Třída ***dijit._Container*** definuje důležité metody z pohledu rodiče. Dovoluje přidat nový widget pomocí metody ***addChild()***. Jako první parametr se předává reference na objekt vkládané komponenty. Druhým parametr specifikuje číselné vyjádření pozice v seznamu widgetů kontejneru, na kterou se nová komponenta vloží. Pro odstranění widgetu z kontejneru slouží metoda ***removeChild()***. Jediný parametr udává referenci na odstraňovaný widget. Komponenta kontejneru dále disponuje metodou ***hasChildren()*** pro otestování, jestli kontejner obsahuje nějaké sub-komponenty. Seznam dětských widgetů vrací metoda ***getChildren()***.

Třída ***dijit._Contained*** naopak přináší metody podstatné z pohledu sub-komponenty. Metoda ***getParent()*** vrací referenci na objekt kontejneru a metody ***getPreviousSibling()*** a ***getNextSibling()*** zajišťují získání reference na předcházející/následující komponentu kontejneru.

Bázová třída pro widgety rozložení *dijit._LayoutWidget* dědí výše popsané třídy a navíc přidává dvě nové metody. Metoda **resize()** zajišťuje správný výpočet šířky a výšky widgetu kontejneru a volá abstraktní metodu ***layout()***, která slouží pro definici algoritmu rozložení sub-komponent. Každý konkrétní layout widget potom tuto metodu implementuje svým specifickým způsobem. V [15, s. 341] se uvádí, že za rozložení dětských widgetů je zodpovědný vždy widget kontejneru, nikoli samotné widgety v něm obsažené. Inicializaci rozměrů widgetu kontejneru a následné rozmístění sub-komponent zajišťuje přetížení metody *startup(), která* na závěr zavolá

metodu *resize()* kontejneru vyskytujícím se na samotném vrcholu hierarchie komponent. Správné použití layout widgetu tedy spočívá v provedení tří základních kroků:

1. vytvoření widgetu kontejneru,
2. vložení dětských widgetů pomocí metody kontejneru *addChild()*,
3. zavolání metody *startup()* kontejneru.

Knihovna Dojo obsahuje několik kontejnerů (*dijit.AccordionContainer*, *dijit.BorderContainer, dijit.TabContainer,* aj.). Každý zobrazuje svoje dětské widgety jiným způsobem, avšak API pro správu dětských komponent zůstává stejné.

7 Implementace webového rozhraní pracovní plochy

Popsal jsem základní služby knihovny Dojo a koncept widgetů. V této kapitole se budu věnovat samotné implementaci prototypu pracovní plochy. Zaměřím se především na tvorbu webového uživatelského rozhraní, které bude složeno z komponent založených na widgetech knihovny Dojo. Pokusím se přitom maximálně využít prostředky, které knihovna nabízí.

Veškerý kód aplikace bude umístěn ve jmenném prostoru *Desktop*, který při inicializaci registruji pomocí metody *dojo.registerModulePath()*.

7.1 Panel

Základní komponentu pracovní plochy představuje widget panelu. Aplikace nabídne různé typy panelů, všechny však budou mít společný základ. Budou obsahovat titulkovou lištu, která kromě názvu panelu a jeho ikony zpřístupní rovněž menu panelu a dialog pro nastavení voleb. Z tohoto důvodu jsem se rozhodl vytvořit třídu ***Desktop.wgDesktopPane***, která definuje základní funkcionalitu panelu. Třídy widgetů pro jednotlivé typy panelů potom budou tuto třídu specializovat.

Komponenta panelu představuje kontejner, který může obsahovat také jiné widgety. Musí přitom zajistit jejich správné rozložení v prostoru panelu. Vzhledem k tomu, že při změně šířky sloupce pracovní plochy dojde také k upravení šířky panelu, využiji třídu *dijit._layoutWidget* popsanou v kapitole 6.3.3. Popis vzhledu panelu není triviální, vyžaduje specifikovat prostor titulkové lišty a umístění ikon, a proto definuji zobrazení panelu pomocí HTML šablony a do seznamu předků třídy *Desktop.wgDesktopPane* přidám také třídu *dijit._Templated*. Šablona panelu je uvedena v následujícím příkladu.

Př. 7-1 Šablona widgetu typu *Desktop.wgDesktopPane*

```
<div class="DesktopPane">
  <div dojoAttachPoint="captionBarNode" class="${currentStyle}">

     <div dojoAttachPoint="iconNode"
          dojoAttachEvent="onmousedown:_onIconClick" />

     <div dojoAttachPoint="captionNode">${caption}</div>

     <div dojoAttachPoint="btnsNode">
        <div class="editNode" dojoAttachPoint="editNode"
           dojoAttachEvent="onmousedown: _onEditBtnClick" />
        <div class="removeNode" dojoAttachPoint="removeNode"
           dojoAttachEvent="onmousedown:_onRemoveBtnClick" />
     </div>
  </div>

   <div dojoAttachPoint="containerNode"></div>
</div>
```

Prostřednictvím připojovacích bodů zajistím uložení referencí na DOM uzly panelu, s kterými se bude často manipulovat. Šablona panelu se skládá z uzlu *captionBasrNode* reprezentujícího titulkovou lištu a samotné oblasti obsahu označené jako *containerNode*. Tento uzel zároveň slouží pro vkládání sub-komponent metodou *addChild()*. Zbývající uzly označené v Př. 7-1 tučně potom definují ikony lišty a samotný titulek. Výchozí styl panelu *currentStyle* a název panelu *caption*, které se widgetu předají během inicializace, vložím do šablony pomocí substituce. Nakonec je ještě potřeba registrovat obsluhu událostí vzniklých při stisknutí tlačítka myši s kurzorem nad některou z ikon. Jednoduše využiji atribut *dojoAttachEvent* a uvedu metody widgetu, které zajistí provedení příslušné akce.

Následující tabulka shrnuje důležité vlastnosti a metody třídy *Desktop.wgDesktopPane*. Panel uchovává souřadnice polohy v rámci kontejneru. Za jejich aktualizaci je zodpovědný kontejner. V panelu jsou uloženy především pro snadnější získání konfigurace (viz kapitola 5.2). Kromě metod pro zobrazení menu a dialogu voleb, je součástí také metoda pro nastavení editačního režimu. Popis tvorby dialogů se budu věnovat v kapitole 7.6.

Vlastnost / metoda	**Typ**	**Popis**
caption	String	Titulek panelu.
currentIcon	String	Aktuální vybraná ikony.
currentStyle	String	Aktuálně vybraný styl lišty.
desktop	Desktop.wgDesktopPaneContainer	Reference na kontejner.
x	Integer	X-ová souřadnice polohy panelu.
y	Integer	Y-ová souřadnice polohy panelu.

set/getEditMode()	function	Nastaví/získá příznak, jestli se panel nachází v editačním režimu.
showOptionsDialog()	function	Zobrazí dialog voleb.
open/closeMenu()	function	Otevře/zavře menu panelu.
translate()	Fiction	Přeloží widget do zvoleného jazyka.

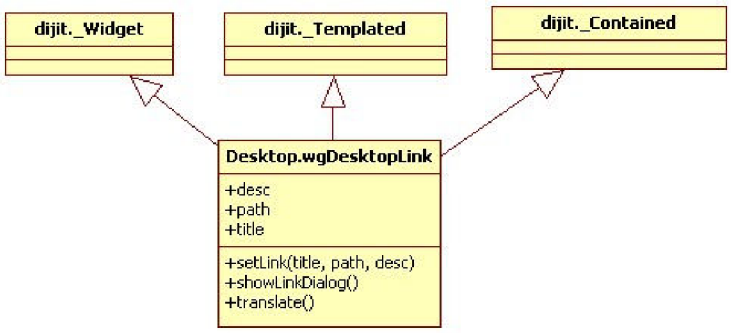
Tab. 7-1 Metody a vlastnosti třídy Desktop.wgDesktopPane

Pokud je titulek panelu příliš dlouhý, dojde k zalomení jeho textu a v konečném důsledku se rozšíří celá lišta panelu. Vhodnější způsob zobrazení dlouhého titulku spočívá ve zkrácení jeho názvu tak, aby vyplnil volný prostor lišty a na jeho konec se přidaly tři znaky tečka symbolizující, že titulek není zobrazen celý. Tento grafický efekt jsem vyřešil pomocí třídy ***Desktop.Ellipsis***, jejíž instance zajistí úpravu titulku při inicializaci a změně šířky panelu.

Implementace zkrácení titulku se bohužel odlišuje ve webových prohlížečích, které bude pracovní plocha podporovat. V prohlížeči Internet Explorer postačuje pomocí kaskádových stylů přidat DOM uzlu titulku stylové pravidlo ***textOverflow: ellipsis*** a prohlížeč automaticky zkrátí titulek a připojí na jeho konec tři znaky tečka. Prohlížeč Opera umožňuje stejný efekt za použití speciálního pravidla ***-o-text-overflow: ellipsis***. Nejkomplikovanější implementace je v prohlížeči FireFox, kde zkrácení názvu vyžaduje použití skriptu. Úpravu délky textu titulku zajišťuje metoda ***setSize()*** třídy *Desktop.Ellipsis*, které se jako parametr předá aktuální šířka uzlu titulku. Algoritmus pro získání ořezaného textu spočívá ve vytvoření nového elementu typu *span*, do kterého se v cyklu postupně přidávají znaky z názvu titulku, a takto vznikající řetězec je v každém kroku zřetězen se třemi znaky tečka. V okamžiku, kdy dosáhne pomocný element požadované šířky, se text v něm vložený jenom přesune do uzlu titulku a pomocný element se odstraní.

7.1.1 Panel odkazů

Panel odkazů rozšiřuje základ panelu a dovoluje spravovat skupinu odkazů. Dříve než popíši jeho třídu, zaměřím se na reprezentaci samotné položky odkazu. Položka obsahuje odkaz vyjádřený pomocí DOM elementu typu *a*. V editačním režimu je ale potřeba zobrazit také tlačítka pro úpravu a odstranění položky odkazu, a proto budu položku definovat jako samostatný widget, který popisuje třída ***Desktop.wgDesktopLink***. Předky této třídy zobrazuje Obr. 7-1 (jejich vlastnosti a metody už znovu neuvádím).



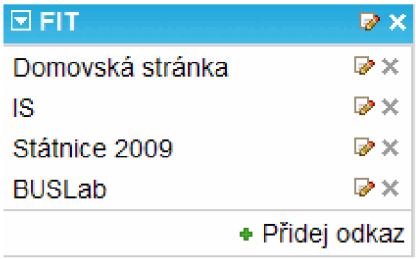
Obr. 7-1 Předkové třídy *Desktop.wgDesktopLink*

Zobrazení komponenty specifikuji opět pomocí šablony. Kromě třídy *dijit._Widget* (jedná se o widget) a *dijit._Templated* (využití šablony) zdědí třída položky odkazu také třídu *dijit._Contained*, protože odkaz bude figurovat jako prvek kontejneru. Šablonu položky odkazu uvádím v Př. 7-2.

Př. 7-2 Šablona widgetu typu *Desktop.wgDesktopLink*

```
<div class="wgDesktopLink">
   <div class="btns" dojoAttachPoint="btnsNode">
      <div class="editNode" dojoAttachPoint="editNode"
           dojoAttachEvent="onmousedown: _onEditBtnClick"></div>
       <div class="removeNode" dojoAttachPoint="removeNode"
           dojoAttachEvent="onmousedown: _onRemoveBtnClick"></div>
   </div>
   <a href="${path}" title="${desc}" target="_blank"
        dojoAttachPoint="linkNode">${title}</a>
   <div dojoAttachPoint="eLinkNode"> ${title}</div>
</div>
```

Kromě DOM uzlů pro tlačítka editace a tlačítka pro odstranění obsahuje šablona také dvě grafické reprezentace názvu odkazu. Uzel ***linkNode*** se zobrazí, pokud není položka v editačním režimu. Při stisknutí levého tlačítka myši s kurzorem nad tímto uzlem se uživateli otevře nové okno prohlížeče a zobrazí se mu obsah uložený na URL uvedeném v atributu *href*. V editačním režimu se naopak využije uzel ***eLinkNode***. Tento uzel nefunguje jako odkaz a díky tomu při operaci drag & drop položky, která je zahájena rovněž stisknutím levého tlačítka myši, nedojde k zbytečnému otevření okna.



Obr. 7-2 Panel odkazů

Třída widgetu položky odkazu definuje tři vlastnosti – název odkazu ***title***, URL odkazu ***path*** a dodatečný popis odkazu ***desc***. Z metod zmíním metodu ***setLink()*** pro nové nastavení výše uvedených vlastností a metodu ***showLinkDialog()*** zajišťující zobrazení dialogu odkazu. Metoda *translate()* slouží podobně jako u třídy *Desktop.wgDesktopPane* k přeložení widgetu podle zvoleného jazyka. Lokalizaci se budu věnovat v kapitole 7.7.

Prvním ze specializovaných panelů je panel odkazů, který definuje třída ***Desktop.wgLinkPane***. Ta dědí třídu *Desktop.wgDesktopPane* a přidává funkcionalitu potřebnou pro přidání/odebrání odkazu. Důležitou úlohou třídy je také zajištění přesunu odkazů v rámci panelu nebo mezi jinými panely odkazů.

Přidání položky odkazu je realizováno pomocí odkazu „Přidej odkaz“ (viz Obr. 7-2). Uživateli se zobrazí dialog položky odkazu a po jeho vyplnění a potvrzení se vytvoří widget položky, který se vloží jako sub-komponenta pomocí metody widgetu panelu ***addChild()***. Podobně pro odstranění se využije metoda ***removeChild()***. Obě metody jsou zděděny ze třídy *dijit._LayoutWidget* a knihovna Dojo automaticky zajistí správné vložení/odstranění.

K implementaci přesunu odkazů pomocí techniky drag & drop využiji API knihovny Dojo představené v kapitole 6.2.6. Oblast obsahu panelu odkazů bude představovat zdroj a jednotlivé odkazy potom jeho položky. Vytvoření instance třídy *dojo.dnd.Source* ukazuje Př. 7-3.

Př. 7-3 Vytvoření instance třídy *dojo.dnd.Source*

```
var dnd = new dojo.dnd.Source(this.containerNode,
{accept:['link'], creator: this._linkCreator});
```

Jako první parametr se konstruktoru zdroje předá DOM uzel reprezentující obsah panelu uložený ve vlastnosti ***cotainerNode.*** Druhý parametr obsahující objekt konfigurace zdroje nastaví pole typů položek, které bude zdroj akceptovat na ***link***. Díky tomu půjde do panelu přemístit pouze položka kompatibilní s tímto typem. Vlastnost *creator* potom specifikuje vytvářecí funkci pro grafickou reprezentaci položek zdroje. Její implementace metodou ***_linkCreator()*** je obsahem Př. 7-4.

Př. 7-4 Vytvářecí funkce pro položku odkazu

```
_linkCreator: function(link, hint) {
  var node = null;
  if (hint == 'avatar') {
     node = link.eLinkNode.cloneNode(true);
  } else {
     node = link.domNode;
  }
  node.id = dojo.dnd.getUniqueId();
  return {node: node, data: link, type: ['link']};
}
```

Metoda je volána se dvěma argumenty. Argument *link* obsahuje referenci na widget položky odkazu. Druhý argument *hint* potom udává, jestli je funkce volána za účelem vizualizace položky během operace drag & drop (hodnota *avatar*) nebo pro zobrazení položky ve zdroji. Návratová hodnota funkce potom musí být objekt, který specifikuje DOM uzel zobrazení položky (vlastnost ***node***), dále data samotné položky (vlastnost ***data***) a konečně seznam typů zdrojů, se kterými je položka kompatibilní (vlastnost ***type***).

Během operace drag & drop postačí zobrazit název odkazu, který je přemisťován. Pokud je tedy hodnota argument *hint* nastavena na *avatar*, funkce vrátí ve vlastnosti *node* kopii uzlu

eLinkNode popsaného výše. V opačném případě se do vlastnosti *node* uloží reference na grafickou reprezentaci celého widgetu položky, tj. na vlastnost ***domNode***. Jako hodnotu vlastnosti *data* předá funkce referenci na celý widget (argument ***link***) a *typ* se nastaví na pole o jedné hodnotě ***link***, aby položka odkazu šla umístit pouze do panelu odkazů.

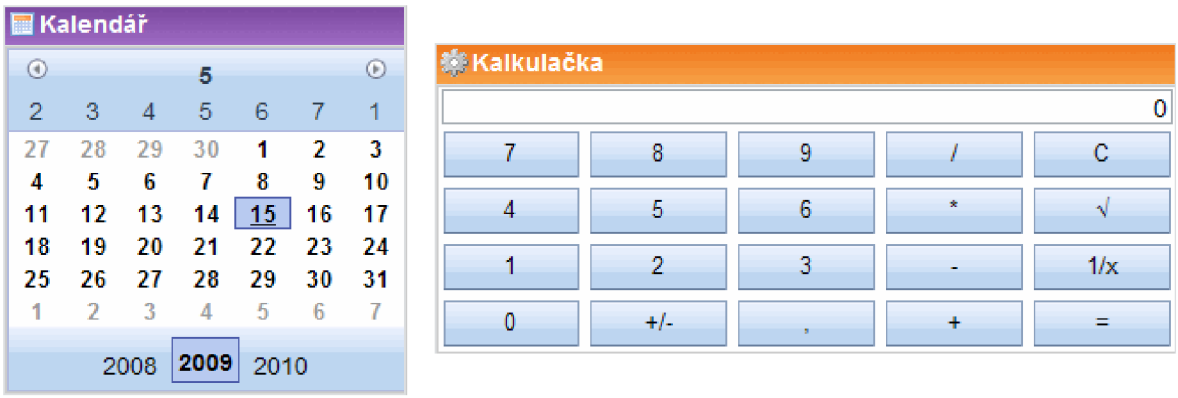
Zbývá vyřešit, jak inicializovat položky zdroje při prvním vytvoření widgetu a jakým způsobem zajistit správné vložení nově vytvořené položky odkazu pomocí dialogu. K tomuto účelu využiji metodu ***insertNodes()*** zdroje, kterou zavolám v metodě ***addChild()***. Podobně je potřeba v metodě *removeChild()* odstranit položku metodou ***delItem()*** zdroje. Díky knihovně Dojo obnáší implementace operace drag & drop jenom tři jednoduché úkony:

1. vytvoření instance třídy *dojo.dnd.Source*,
2. implementace vytvářecí funkce metodou widgetu *_linkCerator()*,
3. přetížení metody *addChild()* a *removeChild()*.

7.1.2 Ostatní panely

V návrhu pracovní plochy jsem uvedl, že uživatel bude moci využít také miniaplikace. Implementuji proto panel s jednoduchou kalkulačkou a rovněž panel kalendáře. V obou případech použiji již připravenou třídu ***Desktop.wgDesktopPane***, kterou dále specializuji.

Widget kalkulačky definuji ve třídě ***Desktop.wgCalculatorPane***. Jednotlivá tlačítka operací a numerické klávesnice kalkulačky vytvořím prostřednictvím widgetu knihovny Dojo *dijit.form.Button*. Implementace miniaplikace kalendáře je ještě jednoduší. Třída ***Desktop.wgCalendarPane*** využije již hotovou komponentu kalendáře definovanou ve třídě ***dijit.Calendar***. Výslednou miniaplikaci kalkulačky a kalendáře ukazuje Obr. 7-3.



Obr. 7-3 Widget kalendáře a kalkulačky

7.2 Kontejner pro panely

Další komponentu, která je pro realizaci prototypu pracovní plochy potřeba, představuje kontejner panelů. Kontejner zastává roli správce rozložení. Musí zajistit rozmístění panelů do stanoveného

počtu sloupců a také umožnit změnu jejich šířky. Panely půjde přitom přemisťovat technikou drag & drop nebo prostřednictvím menu panelu.

7.2.1 Realizace sloupcového rozložení

Knihovna Dojo nabízela do verze 1.0 widget *dijit.SplitContainer* zajišťující rozložení widgetů do sloupců. Bohužel v aktuální verzi knihovny byl tento widget označen za „nevhodný“ (depreciated) a byl nahrazen widgetem *dijit.BorderContainer*, který je však podle [15, s. 346] určen pro globální vymezení obdélníkových oblastí dokumentu s možností měnit šířku těchto oblastí. Z tohoto důvodu vytvořím pro potřebu pracovní plochy vlastní widget.

Implementaci sloupcového rozložení panelů jsem rozdělil na dvě části. Nejdříve realizuji obecný widget kontejneru, který dovoluje vkládat dětské widgety do sloupců, měnit počet sloupců a jejich šířku. Tento widget potom využiju jako základ komponenty zajišťující rozmístění panelů a poskytující možnost přesouvat panely pomocí techniky drag & drop.

Sloupcový kontejner definuji ve třídě ***Desktop.wgColumnContainer***. Jedná se o typický layout widget, a proto tato třída zdědí třídu *dijit._LayoutWidget*. Přehled důležitých vlastností a metod třídy komponenty shrnuje Tab. 7-2.

Vlastnost / metoda	**Typ**	**Popis**
cols	integer	Počet sloupců kontejneru.
colMinSize	integer	Minimální šířka sloupce v pixelech.
colSizes	array	Seznam procentuálně vyjádřených šířek jednotlivých sloupců vzhledem k celkové šířce kontejneru.
maxCols	integer	Maximální počet sloupců.
addChild(widget, x, y)	function	Umístí *widget* na souřadnice *x*, *y*.
removeChild(widget)	function	Odstraní *widget* z kontejneru.
setColumns(count)	function	Nastaví nový počet sloupců *count*.
setEditable(edit)	function	Povolí nebo zakáže změnu šířky sloupců.

Tab. 7-2 Důležité metody a vlastnosti třídy *Desktop.wgColumnContainer*

Při inicializaci widgetu se předá výchozí počet sloupců ve vlastnosti ***cols***. Minimální počet sloupců bude vždy jeden sloupec a jejich maximální počet udává vlastnost ***maxClos***. Kontejner dále nabídne specifikaci minimální šířky sloupce vlastností ***colMinSize***. Výchozí šířka jednotlivých sloupců bude procentuálně definována ve vlastnosti ***colSizes***. Tento údaj je velmi důležitý pro konfiguraci, protože uživatel/ka musí mít možnost si uložit aktuální šířky sloupců, aby je při dalším použití pracovní plochy nemusel/a znovu nastavovat.

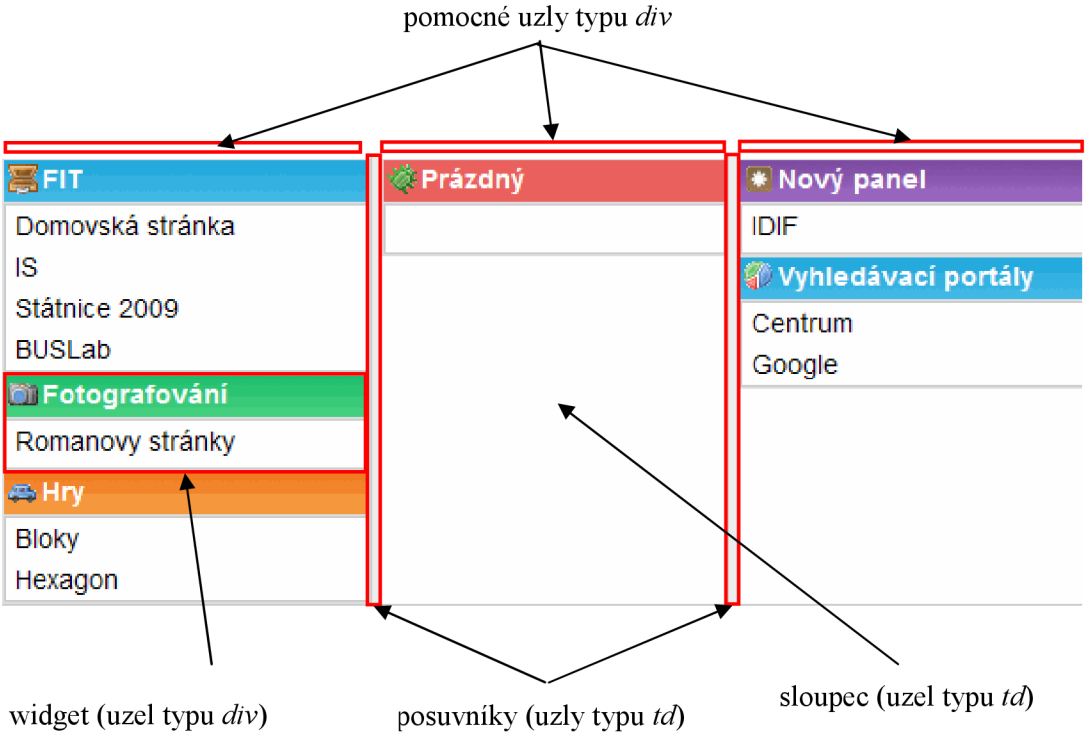
Umístění widgetů v kontejneru bude využívat dvou-dimenzionální systém souřadnic. X-ová souřadnice určí číslo sloupce a y-nová potom pozici v rámci tohoto sloupce. Přidání widgetu zajistí přetížení metody ***addChild()***. Té se jako první parametr předá reference na objekt vkládaného

widgetu a další dva parametry potom specifikují souřadnici x a y. Metoda uloží aktuální souřadnice do vlastností *x* a *y* widgetu a automaticky také zajistí jejich aktualizaci při přidání/odebrání widgetu z kontejneru. Pro odstranění sub-komponenty bude sloužit metoda ***removeChild()***. Metoda ***setColumns()*** nastaví nový počet sloupců kontejneru. Nepotřebné sloupce se odstraní ve směru zprava doleva a jejich widgety se přesunou do posledního sloupce, kterého se již odstranění netýká. Metoda potom zavolá metodu *layou()* pro výpočet nové šířky sloupců. Změny půjde provádět pouze v editačním režimu, a proto bude třída widgetu disponovat metodou ***setEditable()***.

Popsal jsem rozhraní widgetu pro sloupcové rozložení. Dále se zmíním o samotné realizaci grafického rozmístění dětských widgetů do sloupců prostřednictvím uzlů modelu dokumentu. Nabízí se dvě řešení:

1. Jednotlivé sloupce reprezentovat DOM uzly typu *div* a pomocí stylového předpisu zajistit, aby se zobrazovaly vedle sebe.
2. Využít pro rozmístění tabulku pomocí DOM uzlu typu *table*.

První způsob sice zajistí správné zobrazení, jeho nevýhodou je ale nutnost nastavit jednotlivým uzlům typu *div* správnou výšku podle sloupce, který obsahuje největší počet panelů, protože ostatní sloupce musí vyplnit obdélníkovou oblast kontejneru. Vzhledem k tomu, že se panely umístěné ve sloupcích budou přemisťovat, musí být výška celého kontejneru správně upravena při každé změně pozice panelu. Ke změně výšky přitom může docházet vícekrát během jedné operace drag & drop. Z tohoto důvodu jsem se rozhodl sloupcové rozložení implementovat prostřednictvím tabulky. Výhoda tohoto řešení spočívá v tom, že výška tabulky se automaticky upraví podle jejího obsahu. Není tedy potřeba vypočítávat výšku kontejneru skriptem. Všechny podporované prohlížeče v tomto případě zobrazí správné rozměry tabulky. Schéma tabulky potřebné pro zajištění rozložení demonstruje Obr. 7-4.



Obr. 7-4 Schéma tabulky zajišťující rozložení do sloupců

Tabulka bude mít jediný řádek a jednotlivé sloupce budou reprezentovány buňkou tabulky (uzel typu *td*). Pro *n* sloupců vloží kontejner *n – 1* buněk posuvníku sloužících ke změně šířky sloupce. Každá buňka řádku na liché pozici představuje sloupec a každá sudá posuvník. Vytvoření tabulky zajistí metoda *buildRendering()*.

Ke změně šířky sloupců využiji událostí *onmousedown* a *onmouseup*. Pokud dojde ke stlačení levého tlačítka myši s kurzorem nad buňkou posuvníku, zahájí se operace pro upravení šířky. Dokud zůstává tlačítko stisknuté, mění se na základě aktuální polohy kurzoru myši šířka sloupce před/za posuvníkem. Po uvolnění tlačítka myši operace skončí a aktuální šířky sloupců se uloží jako hodnota vlastnosti *colSizes*. Prohlížeč Opera však nedodržuje nově nastavené šířky buněk tabulky a sloupce zůstávají stále stejně široké. Chybu jsem vyřešil vložením pomocného uzlu typu *div* (viz Obr. 7-4) do každé buňky sloupce. Nová šířka se nenastavuje přímo buňce tabulky představující sloupec, ale jejímu pomocnému uzlu, který zaručí správnou šířku buňky také v prohlížeči Opera. Tento uzel má nastavenou výšku na nula pixelů, a proto se ve výsledném dokumentu nezobrazí.

7.2.2 Realizace přesunutí panelu

Widget sloupcového rozložení nyní využiji pro rozmístnění samotných panelů. Vytvořím třídu ***Desktop.wgDesktopContainer***, která bude dědit ze třídy *Desktop.wgColumnContainer*, a implementuji funkcionalitu potřebnou pro přesouvání panelů. Nové vlastnosti a metody třídy uvádí Tab. 7-3.

Vlastnost/ metoda	Type	Popis
caption	string	Titulek kontejneru, který se zobrazí jako název záložky.
Icon	string	Ikona kontejneru pro tlačítko záložky.
getDirectionsFlag(pane)	function	Vrátí seznam povolených směrů pohybu panelu.
movePane(pane, dir)	function	Posune panel v daném směru.
movePaneTo(pane, x, y)	function	Přesune panel na novou pozici v kontejneru.
set/getEditMode(mode)	function	Nastaví/získá příznak editačního režimu.
updateMenus()	function	Upraví menu všech panelů podle nového uspořádání.

Tab. 7-3 Důležité vlastnosti a metody třídy Desktop.wgColumnContainer

Metoda ***movePane()*** bude sloužit pro přesunutí widgetu panelu ve směru zvoleném prostřednictvím menu panelu. Jako první parametr se předá reference na přesouvaný panel a druhý parametr udává číslo reprezentující směr pohybu. Seznam směrů pohybu popisuje kapitola 4.5.3. Na základě aktuální pozice panelu a zvoleného směru metoda vypočítá nové souřadnice a předá je metodě ***movePaneTo()***, která potom zajistí korektní přesun widgetu. Při libovolném přemístění panelu se mění seznam povolených směrů pohybu ostatních panelů, a je proto potřeba aktualizovat jejich menu. K tomuto účelu slouží metoda ***updateMenus()***, která v cyklu prochází panely kontejneru a pomocí metody ***getDirectionsFlag()*** zjišťuje seznam povolených směrů. Ten potom předá jako argument metodě *updateMenu()* panelu. Kontejner disponuje také metodou ***setEditMode()*** pro zapnutí/vypnutí editačního režimu.

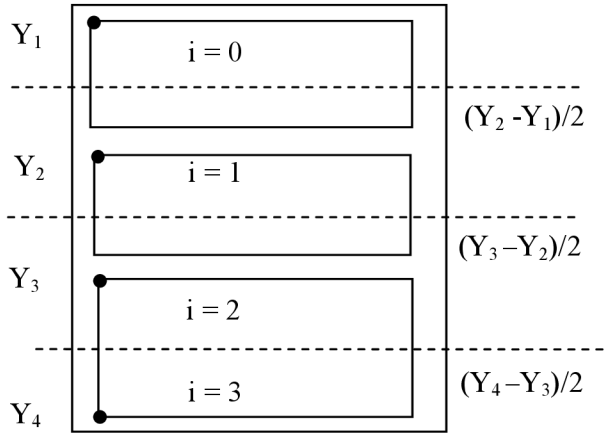
Zbývá vyřešit podporu přesunu panelů technikou drag & drop. V kapitole 4.4.6 jsem uvedl, že během přemístění se bude panel zobrazovat přímo pod kurzorem myši, aby výsledný grafický efekt působil dojmem, že uživatel panel uchopil a pohybem myši ho skutečně přesouvá. Implementace operace drag & drop v knihovně Dojo zobrazuje přesouvanou položku vedle kurzoru myši. Změna tohoto, na první pohled nepodstatného, způsobu zobrazení však vyžaduje výraznější zásah do implementace celé operace. Indikace pozice stejně jako detekce, že myš vstoupila do oblasti zdroje, je totiž založena na přijímání událostí vzniklých na uzlech nacházejících se pod kurzorem myši. Pokud se DOM uzel přetahované položky vyskytuje přímo pod kurzorem myši, zamezí se vzniku všech událostí a indikace aktuální pozice položky přestane fungovat. Vytvořím proto novou implementaci drag & drop, která by výše popsaný grafický efekt umožnila. Budu přitom vycházet ze základního konceptu drag & drop v knihovně Dojo a pojmů s ním souvisejících (viz kapitola 6.2.6).

Řešení způsobu detekce, že se přesouvaná položka nachází nad zdrojem, spočívá ve znalosti absolutní polohy DOM uzlu zdroje v rámci dokumentu a jeho rozměrů. Pokud se souřadnice pozice myši nachází v oblasti dokumentu vymezené množinou bodů *{[x, y], [x + w, y], [x, y + h], [x + w, y +h]}*, kde *x*, *y* jsou souřadnice levého horního rohu uzlu zdroje, *w* jeho šířka a *h* jeho výška, potom může manažer publikovat událost vstupu položky do oblasti zdroje (dndOver). Při zahájení operace drag & drop je proto nutné získat polohu oblastí všech zdrojů v dokumentu a kontrolovat,

jestli se kurzor myši v některé z nich nenachází. Tuto logiku jsem implementoval pomocí třídy ***Desktop.DndBoxManager***.

Dále definuji třídu zdroje ***Desktop.DndBoxContainer***, která zajistí zobrazení indikátoru aktuální pozice přesouvané položky a vypočítá pozici v rámci zdroje, na kterou se má právě přesouvaná položka umístit. Opět není možné využít událostí vyvolaných při pohybu myši nad uzly jednotlivých položek ve zdroji, a proto se musí hledaná pozice spočítat na základě jejich absolutních poloh v rámci dokumentu.

Při zahájení operace drag & drop se určí y-nové souřadnice levého horního rohu všech uzlů položek a u poslední položky také y-nová souřadnice levého dolního rohu. Tyto hodnoty se uloží do pomocného pole souřadnic. Pole se prochází v cyklu a vypočítá se přitom průměrná hodnota dvou po sobě následujících souřadnic, která se uloží do pole *limits*. V okamžiku, kdy vstoupí kurzor myši do oblasti zdroje, získá se y-nová souřadnice jeho pozice *Ym*. Postupně se potom iteruje polem *limits* a hledá se index *i*, pro který platí podmínka *limits[i]* > *Ym*. Hledaná pozice pro vložení přemisťované položky je potom dána výrazem *i + 1*, protože pozice položky je indexována od jedné. Výpočet pozice demonstruje Obr. 7-5.



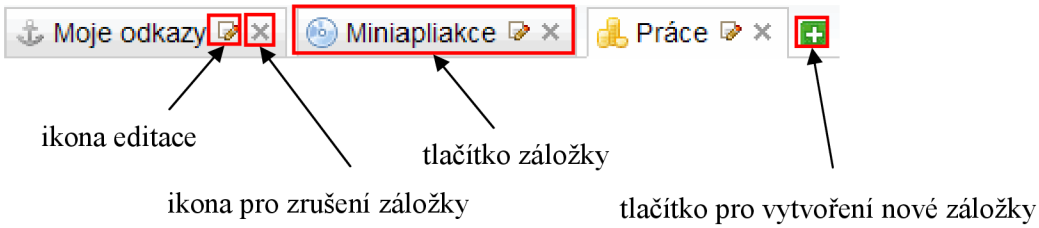
Obr. 7-5 Schéma výpočtu pozice pro vložení odkazu

Popsanou implementaci drag & drop je nyní potřeba použít pro přesun panelů. Vytvořím proto třídu ***Desktop.dndPaneContainer*** specializující třídu *Desktop.DndBoxContainer*. Konstruktoru třídy je potřeba jako první parametr předat referenci na buňku tabulky představující sloupec. Druhý parametr potom specifikuje konfiguraci zdroje a bude obsahovat index sloupce v rámci kontejneru panelů a referenci na samotný widget kontejneru. Přemístění panelu zajistí metoda ***onDndDrop()***, která se zavolá při ukončení operace drag & drop. Při jejím provedení se spustí metoda *movePaneTo()* s novými souřadnicemi panelu. X-ovou souřadnici představuje index sloupce a y-nová souřadnice je potom dána pozicí, na kterou byl panel vložen. Finální krok zajišťující podporu drag & drop spočívá ve vytvoření instance třídy *Desktop.dndPaneContainer* pro každý sloupec kontejneru panelů.

7.3 Záložky

Seskupení panelů zajistí kontejner záložek. Knihovna Dojo nabízí již připravený layout widget *dijit.TabContainer*. Jeho použití ale brání **nutnost explicitně definovat výšku widgetu**. Obsahem každé záložky bude kontejner panelů, jehož výška závisí na způsobu uspořádání panelů a může se navíc dynamicky měnit. Vzhledem k tomu, že implementace záložek není složitá, rozhodl jsem se vytvořit vlastní widget, který definuji pomocí třídy ***Desktop.wgSimpleTabContainer***.

Třída komponenty záložek zdědí třídu *dijit._LayoutWidget*. Její metoda ***addChild()*** zajistí kromě přidání nového widgetu také vytvoření odpovídající záložky. Záložku realizuji pomocí komponenty tlačítka (třída ***Desktop._wgTabButton***) zobrazujícího její ikonu a název. Využiji přitom hodnot vlastností *icon* a *caption* přidávaného widgetu. Podobně metoda ***removeChild()*** odstraní z kontejneru komponentu a tlačítko její záložky. Charakteristickou vlastností popisovaného kontejneru je, že zobrazuje pouze jednu vybranou sub-komponentu. Při inicializaci komponenty se automaticky vybere první záložka a zobrazí se její obsah. Přepnutí záložky lze potom realizovat buď stisknutím tlačítka záložky, nebo pomocí metody ***selectChild()*** s parametrem udávající vybraný dětský widget.



Obr. 7-6 Důležité elementy záložky

Prototyp pracovní plochy nabídne uživateli možnost přidávat nové záložky, smazat stávající nebo upravit text a ikonu záložky pomocí dialogu. Definuji proto třídu ***Desktop.wgDesktopTabContainer***, která rozšíří funkcionalitu třídy *Desktop.wgSimpleTabContainer*. K přidání nové záložky bude sloužit tlačítko zobrazené za poslední záložkou (viz Obr. 7-6). Po jeho stisknutí se objeví dialog pro nastavení ikony, názvu a počtu sloupců kontejneru panelů. Metoda obsluhy potvrzení dialogu potom vytvoří novou instanci třídy *Desktop.wgDesktopContainer* a vloží ji do kontejneru záložek pomocí metody *addChild()* kontejneru. Změnu nastavení záložky a její smazání zajistí ikony uvnitř tlačítka záložky označené v Obr. 7-6. Pro tuto potřebu vytvořím nový widget tlačítka záložky (třída ***Desktop._wgEditableTabButton***). Do šablony tlačítka potom doplním DOM uzly představující uvedené ikony.

7.4 Prostředí aplikace

Aplikace pracovní plochy bude obsahovat globální konfiguraci, která by měla být přístupná všem ostatním komponentám rozhraní. Z tohoto důvodu definuji speciální **widget prostředí aplikace** (třída ***Desktop.wgDesktopEnvironment***). Při inicializaci pracovní plochy se vytvoří pouze jedna instance této třídy dostupná prostřednictvím globální proměnné ***environment***. Každá komponenta tak získá přístup k informacím o aktuálním grafickém tématu nebo jazykové lokalizaci. Úlohou widgetu prostředí bude dále zprostředkovat asynchronní komunikaci se serverem a v poslední řadě také zajištění rozložení ostatních komponent rozhraní aplikace.

V seznamu předků třídy *Desktop.wgDesktopEnvironment* bude figurovat třída *dijit._layoutWidget*, protože widget musí zajistit správné rozmístění komponent při změně velikosti okna webového prohlížeče, a také třída *dijit._Templated* pro definici vzhledu pomocí šablony. Podstatné vlastnosti a metody třídy shrnuje Tab. 7-4.

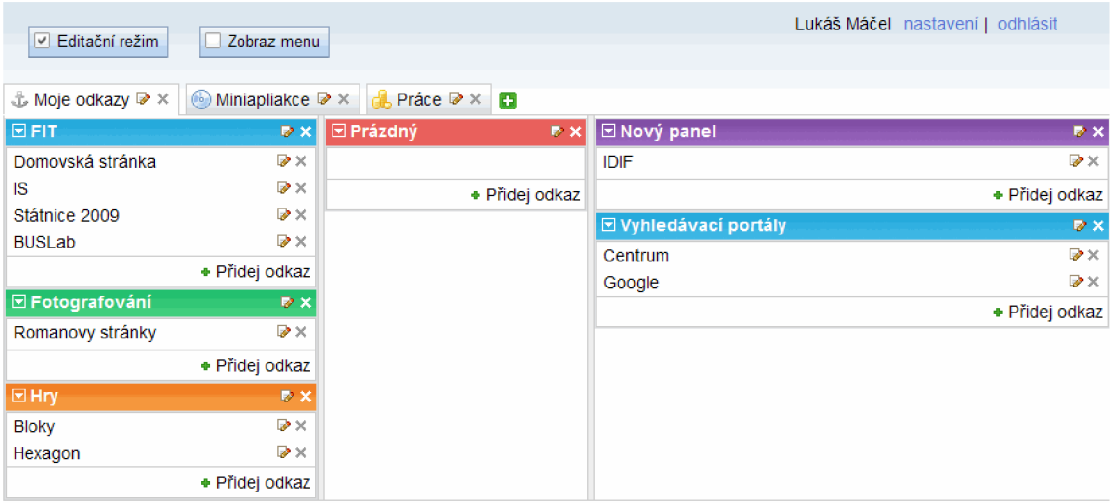
Vlastnost/metoda	Typ	Popis
editable	boolean	Příznak, jestli je možné zapnout editační režim.
lang	String	Aktuálně zvolený jazyk pro překlad aplikace.
theme	String	Aktuálně vybrané grafické téma.
url	String	URL pro komunikaci se serverem.
userName	String	Jméno uživatele pracovní plochy.
addPane(pane)	function	Přidá nový panel na pracovní plochu.
genId()	function	Vygeneruje unikátní id komponenty.
request(params, content)	function	Pošle požadavek na server.
switchEditMode()	function	Přepne editační režim pracovní plochy.
switchLanguage(lang)	function	Zajistí změnu jazykové lokalizace aplikace.
switchTheme(theme)	function	Změní grafické téma.
toggleMenu()	function	Zobrazí/skryje menu nabídky odkazů.

Tab. 7-4 Důležité vlastnosti a metody třídy *Desktop.wgDesktopEnvironment*

Metoda ***addPane()*** vloží nový panel, jehož reference je předána v prvním parametru, na souřadnice [1,1] kontejneru panelů v aktuálně vybrané záložce. Kladení asynchronních požadavků na server zajistí metoda ***request()*** s využitím podpory knihovny Dojo popsané v kapitole 6.2.5. Jako prvním argument se metodě předá objekt (slovník) parametrů požadavku. Druhý argument specifikuje potom odesílaná data. Pro vytvoření instance widgetu odkazu, panelu nebo záložky, je nutné zadat jednoznačný identifikátor komponenty v rámci celého rozhraní. Komponenta kontejneru, která zajišťuje vytváření svého prvku, ale nezná id ostatních komponent. Prostřednictvím globální proměnné *enviroment* však může zavolat metodu ***genId()*** widgetu prostředí, která zajistí přidělení

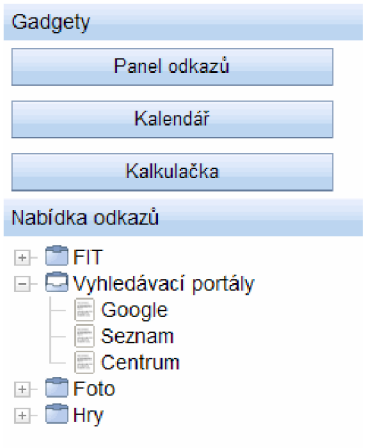
unikátního id. Změna grafického tématu aplikace se provede metodou ***switchTheme()***. Funkcionalitu potřebnou pro modifikaci vzhledu aplikace uvedu v kapitole 7.8. Metoda ***switchLanguage()*** potom poslouží pro překlad aplikace do zvoleného jazyka a podrobněji se o ní zmíním v kapitole 7.7.

V šabloně widgetu prostředí specifikuji uzly pro zobrazení přihlášeného uživatele a také odkazy pro odhlášení a přístup k dialogu globálního nastavení. Součástí informační lišty pracovní plochy bude dále widget tlačítka pro zapnutí/vypnutí editačního režimu. Použiji třídu ***dijit.form.ToggleButton***, která zobrazí vedle názvu tlačítka zaškrtávací políčko, aby uživatel jednoduše poznal, jestli se nachází v editačním režimu. Stejným způsobem vyřeším také tlačítko pro zobrazení menu nabídky odkazů. Rozhraní aplikace v editačním režimu ukazuje Obr. 7-7.



Obr. 7-7 Rozhraní aplikace pracovní plochy

7.5 Menu nabídky



Obr. 7-8 Widget menu nabídky

Důležitou komponentu rozhraní pracovní plochy představuje menu nabídky, které vytvořím jako samostatný widget popsaný ve třídě ***Desktop.wgDesktopMenu***. V kapitole 5.3 jsem uvedl, že server bude poskytovat nabídku v podobě seznamu položek. Existuje více způsobů, jak se dají takto získaná data zobrazit. V implementaci prototypu pracovní plochy budu položky miniaplikace reprezentovat pomocí tlačítek a odkazy a jejich skupiny potom pomocí **komponenty stromu** (viz Obr. 7-8 Widget menu nabídky). Jako další rozšíření je možné využít např. zobrazení formou tabulky, jejichž řádky představují samotné položky a sloupce potom jejich atributy.

Dříve než popíši použití widgetu stromu, zmíním se ještě o způsobu abstrakce dat v knihovně Dojo. Modul ***dojo.data*** nabízí implementaci API, které zajišťuje standardní prostředky pro interakci s libovolným datovým zdrojem [15, s. 196]. Prostřednictvím objektu tzv. **datového skladu** lze získat

data zdroje a dále je třídit, filtrovat nebo modifikovat. Důležité přitom je, že data jsou dostupná jednotným způsobem pro všechny komponenty rozhraní. V dalším textu budu uvádět pouze nezbytnou část API knihovny pro manipulaci s daty, která je důležitá pro získání nabídky položek a jejich další zpracování. Kompletní popis služeb nabízených modulem *dojo.data* je možné nalézt na [14].

První krok při budování obsahu menu spočívá ve vytvoření tlačítek pro vložení miniaplikace. Nejdříve je proto nutné získat ze serveru položky dostupných miniaplikací pomocí datového skladu. Knihovna Dojo nabízí pro čtení dat ve formátu JSON třídu ***dojo.data.ItemFileReadStore***. Vytvoření její instance a následné načtení popisu položek ukazuje následující příklad.

Př. 7-5 Vytvoření datového skladu a načtení zdroje

```
var store = new dojo.data.ItemFileReadStore({url: this.sourceUrl});
store.fetch(
   {query: {type: "gadget"},
    onComplete: dojo.hitch(this,"_loadData")
});
```

Konstruktoru skladu je nutné předat objekt parametrů, jehož vlastnost *url* specifikuje URL zdroje, ze kterého se data získají. Pomocí metody ***fetch()*** skladu se potom provede načtení dat. Jako argument metody se opět předává objekt parametrů. Stažení obsahu zdroje probíhá asynchronně, a proto je potřeba definovat funkci obsluhy, která se zavolá v momentě, kdy jsou data k dispozici. K tomu slouží vlastnost ***onComplete***. V Př. 7-5 zajistí zpracování položek metoda *_loadData()* třídy *Desktop.wgDesktopMenu*. Aby se metoda provedla v kontextu objektu menu, využije se metoda *dojo.hitch()*. Cílem požadavku na zdroj není získat všechna data, ale pouze položky popisující dostupné miniaplikace. Ve zdroji je označím pomocí vlastnosti ***type*** s hodnotou ***gadget***. Filtr, který zajistí výběr takovýchto položek, se specifikuje jako hodnota vlastnosti ***query*** parametru metody *fetch()*.

Pro úspěšné provedení dotazu je ještě nutné modifikovat návrh struktury dat uložených na serveru. Potřebnou úpravu obsahuje Př. 7-6.

Př. 7-6 Upravená struktura dat nabídky

```
{
  "identifier": "id",
  "label": "title",
  "items": [ … ]
}
```

Třída *dojo.data.ItemFileReadStore* vyžaduje, aby zdroj definoval pomocí vlastnosti ***identifier*** jméno atributu položky, jehož hodnota je v rámci obsahu zdroje unikátní. Tuto podmínku splňuje hodnota atributu ***id***. Dále do definice dat na serveru přidám vlastnost ***label***, která se využije při textové reprezentaci položky. Jako hodnotu vlastnosti uvedu popisek položky ***title***. Vlastnost ***items*** potom obsahuje samotné pole položek definovaných v kapitole 5.3.

Po získání dat ze serveru vytvoří metoda ***_loadData()*** tlačítka pro vkládání miniaplikací. K jednotlivým atributům položky zdroje je nutné přistupovat prostřednictvím metody skladu ***getValue(),*** které se jako první parametr předá objekt položky a ve druhém se specifikuje identifikátor atributu. Při stisknutí tlačítka miniaplikace se vytvoří nová instance příslušné třídy (vlastnost ***classType*** položky nabídky) a pomocí metody *addPane()* objektu prostředí se vloží na pracovní plochu.

Díky datovému skladu představuje zobrazení položek odkazů ve formě stromu jednoduchý úkol. Postačí vytvořit instanci třídy ***dijit.Tree*** a konstruktoru předat objekt parametrů uvedený v Př. 7-7.

Př. 7-7 Vytvoření widgetu stromu

```
var tree = new dijit.Tree({
   store: store,
   labelAttr: "title",
   query: {type: "space"},
   dndController: "Desktop.TreeLinkDndSource"
});
```

Hodnota parametru ***store*** představuje referenci na objekt datového skladu. Parametr ***labelAttr*** udává název atributu položky datového skladu, jehož hodnota se využije jako název uzlu stromu. Filtr pro omezení seznamu položek zdroje zobrazených widgetem stromu specifikuje parametr ***query***. Data zdroje jsou reprezentovány lineárně, a proto je potřeba vybrat jenom položky na vrcholu hierarchie označené atributem ***type*** s hodnotou ***space***. Transformaci data na hierarchickou strukturu zajistí komponenta automaticky na základě referencí uložených v atributu ***children*** položky. Poslední parametr v Př. 7-7 ***dndController*** udává odkaz na objekt zdroje, který umožní přesouvat uzly stromu pomocí techniky drag & drop.

Zdroj drag & drop definuji prostřednictvím třídy ***Desktop.TreeLinkDndSource*** dědící základní třídu ***dijit._tree.dndSource.*** Důvodem, proč třídu knihovny specializuji, je implementace následujících podmínek:

1. Strom odkazů nepřijímá žádné nové uzly a neumožní také přemisťování uzlů v rámci komponenty.
2. Listy stromu představují položky odkazů, které lze pomocí techniky drag & drop vložit do libovolného panelu odkazů, ostatní uzly přemístit nejde.

První podmínku lze zajistit pomocí přetížení metody zdroje ***checkAcceptance()***. Ta vrací příznak, jestli může být přesouvaná položka zdrojem přijata. Pokud metoda vrátí při každém volání hodnotu *false*, widget stromu nebude přijímat žádné položky a to ani svoje vlastní uzly. Druhou podmínku potom zajistí přetížení metody ***onMouseDown()*** zdroje, která v případě přesouvání uzlu odkazu nastaví položce vlastnost ***type*** na hodnotu ***link*** a do jejich dat uloží informaci o názvu, URL adrese a popisu odkazu. Tyto informace se potom využijí při vytváření nového odkazu v panelu odkazů.

7.6 Dialogy

Poslední komponentou, kterou jsem se zatím nezabýval, je widget dialogu. Knihovna Dojo nabízí widget typu ***dijit.Dialog***. Tato komponenta vytvoří DOM uzel reprezentující okno dialogu, které lze standardním způsobem přesouvat po dokumentu pomocí kurzoru myši. Modalitu dialogu řeší knihovna prostřednictvím pomocného DOM uzlu, jehož rozměry se nastaví tak, aby svojí plochou překrýval celý obsah dokumentu, a zamezil tak vzniku událostí, dokud nebude dialog uzavřen.

Pro potřeby prototypu pracovní plochy rozšířím popsaný widget a definuji tři základní dialogy:

1. **Dialog zprávy** (*Desktop.dlgMessageBox*) – zajistí zobrazení zprávy, varování nebo chyby.
2. **Dialog potvrzení** (*Desktop.dlgConfirmBox*) – vytvoří okno s otázkou, na kterou je možné odpovědět ano/ne.
3. **Dialog nastavení** (*Desktop.dlgOptionsBox*) – zobrazí formulář pro nastavení důležitých voleb.

Třída *Desktop.dlgMessageBox* definuje tři důležité vlastnosti. První je vlastnost ***title*** udávající titulek dialogu, druhou vlastnost ***message*** pro zadání zobrazené zprávy a poslední je vlastnost ***type*** specifikující typ zprávy (zpráva, varování, chyba), který se potom projeví změnou ikony dialogu. Pro zobrazení dialogu zprávy stačí pouze jeden dialog, a proto jsem vytvořil funkci ***Desktop.msgBox()***. Té se jako parametry předají hodnoty výše popsaných vlastností a funkce zařídí vytvoření instance widgetu a jeho zobrazení.

Třída *Desktop.dlgConfirmBox* má rovněž tři vlastnosti. Vlastnost ***title*** a **message** se shodují s předešlou třídou. Třetí vlastnost ***action*** udává referenci na obslužnou funkci, která se zavolá při stisknutí tlačítka dialogu. Funkce obdrží hodnotu *true*/*false* podle toho, jestli uživatel zvolil tlačítko „ano“ nebo „ne“. Podobně jako v předchozím případě lze zobrazení dialogu zjednodušit zavoláním funkce ***Desktop.confirBox()***.

Třída *Desktop.dlgOptionsBox* potom definuje bázovou třídu pro dialog voleb. Mezi důležité vlastnosti patří vlastnost ***title*** pro zadání titulku dialogu a vlastnosti ***save*** a ***cancel*** obsahují referenci na funkci obsluhy události stisknutí tlačítka pro potvrzení nebo zrušení dialogu. Komponenty rozhraní prototypu pracovní plochy potom tuto třídu specializují, aby vytvořily vlastní dialog nastavení. Ukázky jednotlivých dialogů demonstruje Obr. 7-9.

Obr. 7-9 Ukázka dialogů aplikace

7.7 Moduly a jazyková lokalizace

V kapitole 4.1 jsem stanovil požadavek, aby aplikace pracovní plochy byla lokalizována alespoň do dvou jazyků a uživatel si mohl zvolit preferovaný jazykový překlad rozhraní. Implementace internacionalizace aplikace vyžaduje vyčlenění všech jazykově závislých řetězců mimo zdrojové kódy skriptu a jejich následné přeložení. Prototyp pracovní plochy bude lokalizován do češtiny a angličtiny. Překladu musí podléhat:

- chybová hlášení a zprávy uživateli,
- všechny texty (titulky, názvy tlačítek, položky menu atd.) rozhraní.

K lokalizovaným textům lze přistupovat globálně prostřednictvím překladového slovníku. U chybových hlášení je však vhodné udržovat informaci o místě, kde k chybě došlo. Z tohoto důvodu definuji lokalizované texty na úrovni modulu. Každá chyba aplikace bude jednoznačně určena jménem modulu a číslem chyby. Zbývající texty podléhající překladu specifikuji rovněž na úrovni modulu.

Knihovna Dojo podporuje modularitu pomocí metod knihovny *dojo.provide()* a *dojo.require()* popsaných v kapitole 6.2.2. Modularita však není plnohodnotná, protože nedovoluje deklarovat lokální proměnné viditelné pouze v rámci modulu. Volání metody *dojo.require()* pouze přidá kód modulu do společného kódu aplikace stejně, jako kdyby se vykonalo vložení skriptu prostřednictvím značky *script*. Pokud jsou tedy ve dvou různých modulech stejně pojmenované identifikátory textové položky nebo chyby, dojde ke kolizi. Problém popsaný výše vyřeším novou funkcí ***Desktop.module()***, která nahradí funkci *dojo.provide()*. Př. 7-8 ukazuje nový způsob deklarace modulu.

Př. 7-8 Nová deklarace modulu

```
Desktop.module("Desktop.Modul", function(module) {
   //... definice modulu ....
   var txt1 = 10;
   var err2 = 2;

   module.text("txt1", param); //lokalizovaný text
   module.error("err2", param, inner); //hlášení chyby
});
```

První parametr funkce *Desktop.module()* představuje název modulu a odpovídá parametru funkce *dojo.provide()*. Jako druhý parametr se potom předá reference na funkci obsahující kód celého modulu. Všechny lokální proměnné jsou deklarovány uvnitř této funkce, a proto nebudou viditelné na globální úrovni. V jiném modulu lze použít stejně pojmenované proměnné, aniž by došlo ke kolizi. Kód funkce *Desktop.module()* je obsahem následujícího příkladu.

Př. 7-9 Funkce pro deklaraci modulu *Desktop.module()*

```
Desktop.module = function(name, init) {
   dojo.provide(name);
   init(new Desktop.Module(name));
};
```

Nejdříve se provede deklarace modulu metodou knihovny *dojo.provide()*. Druhý krok spočívá v provedení funkce ***init*** s kódem modulu. Tím se zajistí, že budou dostupné všechny funkcí a třídy definované v rámci modulu. Navíc se však funkci *init* předá instance třídy ***Desktop.Module***, která bude sloužit pro přístup k lokalizovaným textům a chybovým hlášením.

Třída *VemaeSys.Module* definuje dvě důležité metody. První metoda ***error()*** vytvoří objekt výjimky obsahující přeložený text chyby a provede její „vyhození“ (příkaz *throw*). První parametr metody představuje jednoznačnou identifikaci chyby v rámci modulu. Druhým parametrem je možné předat pole hodnot, které se substituují do textu chybového hlášení. Poslední parametr slouží pro uložení předcházející výjimky. Druhá metoda *VemaeSys.Module* nazvaná ***text()*** slouží pro získání lokalizovaného řetězce. První parametr opět představuje jednoznačný identifikátor řetězce a druhý pole hodnot pro substituci. Bude-li tedy v rámci modulu potřeba zahlásit chybu nebo použít lokalizovaný text, stačí zavolat příslušnou metodu (viz Př. 7-8).

Lokalizované texty a chybová hlášení budou umístěny v samostatných souborech ve formátu JSON pojmenovaných *jmenoModulu.error.json* a *jmenoModulu.text.json*. V momentě, kdy poprvé vznikne požadavek na získání jazykově závislého textu, dojde ke stažení obsahu souboru pomocí synchronního požadavku. V takto získaném slovníku lokalizovaných textů se potom na základě id asociativně vyhledá požadovaný řetězec.

Implementace přepnutí jazyka pomocí metody ***switchTheme()*** widgetu prostředí aplikace spočívá potom ve dvou jednoduchých krocích. Nejdříve se pro všechny natažené moduly provede aktualizace jejich lokalizovaných textů podle zvoleného jazyka. V druhém kroku se potom v cyklu

projde registr vytvořených widgetů rozhraní (*diji.registry*) a u každé komponenty se zavolá metoda ***translate()***, která zařídí její přeložení.

7.8 Grafická témata

Knihovna Dojo podporuje změnu vzhledu widgetů prostřednictvím grafických témat (themes). Každé téma je definováno v externím stylovém předpisu. Strukturu souborů grafického tématu *Tundra* a *Soria* naznačuje Př. 7-10.

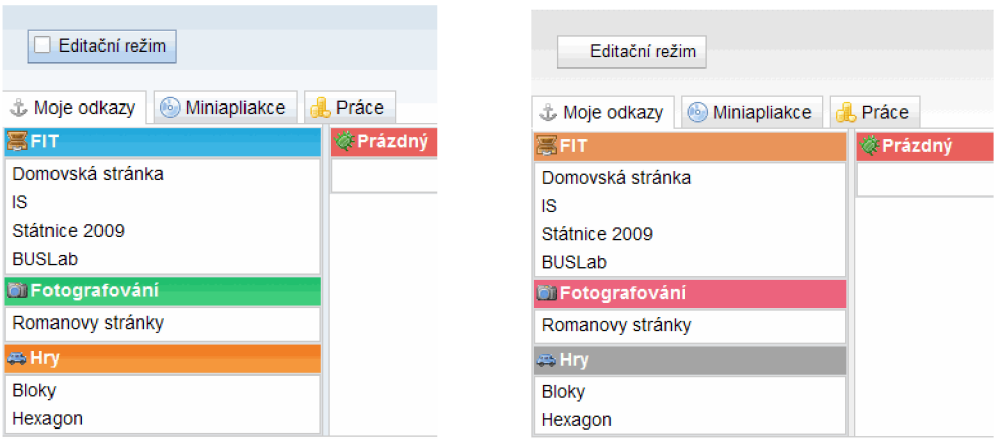
Př. 7-10 Porovnání stylových předpisů témat *Tundra* a *Soria*

```
.tundra .dijitButton {…}
.tundra .dijitTextBox {…}
.tundra .dijitCheckbox {…}
        …
```

```
.soria .dijitButton {…}
.soria .dijitTextBox {…}
.soria .dijitCheckbox {…}
        …
```

Všechna stylová pravidla v předpisu jsou definována kombinovaným selektorem složeným ze selektoru třídy tématu (*tundra/soria*) a selektoru třídy konkrétního widgetu (*dijitButton, dijitTextBox, dijitCheckbox*). Styly pravidla se aplikují jenom tehdy, pokud je v objektovém modelu dokumentu uzel widgetu dítětem uzlu, který má třídu tématu. Jestliže jsou tedy v dokumentu přítomny oba stylové předpisy z Př. 7-10 a třída tématu se připojí elementu *body*, potom se pouhou změnou třídy elementu *body* z *tundra* na *soria* zaručí překreslení všech widgetů.

Aplikace pracovní plochy nabídne dvě grafická témata založená na tématech *Tundra* a *Soria* knihovny Dojo. Přepnutí grafického tématu aplikace bude zajišťovat metoda ***swicthTheme()***, která nastaví třídu elementu *body* podle vybraného názvu tématu a dynamicky vloží do hlavičky stylu odpovídající externí stylový předpis. Změnu vzhledu rozhraní zachycuje Obr. 7-10.



Obr. 7-10 Porovnání vzhledu rozhraní pro dvě různá témata

8 Implementace správy konfigurace

V předchozích kapitolách jsem popsal implementaci komponent potřebných pro vytvoření webového rozhraní aplikace. Nyní se ještě zaměřím na správu konfigurace. Prototyp pracovní plochy musí zajistit vybudování rozhraní na základě konfigurace a také aktuální obsah pracovní plochy perzistentně uložit. Další tématem této kapitoly bude implementace autentizace.

8.1 Načítání a ukládání konfigurace

Při inicializaci aplikace se prostřednictvím asynchronního požadavku získá konfigurační objekt pracovní plochy, na jehož základě je potřeba vybudovat webové rozhraní. Naopak při ukončení editačního režimu se tento objekt vytvoří z aktuálního stavu rozhraní a odešle se na server. Načtení konfigurace widgetu a její uložení realizuji pomocí třídy ***Desktop.Configable***, která je uvedena v Př. 8-1.

Př. 8-1 Ukázka definice třídy *Desktop.Configable*

```
dojo.declare("Desktop.Configable", null, {
   loadConfig: function(data) {
   },
   saveConfig: function() {
      return {};
   },
   isModifed: function() {
      return false;
   }
});
```

Všechny widgety rozhraní, které mají být konfigurovatelné (odkaz, panel, kontejner panelů a kontejner záložek), zdědí tuto třídu a povinně implementují abstraktní metody ***loadConfig()***, ***saveConfig()*** a ***isModified()***. Tím zajistím, že widget je schopný nastavit svoje vlastnosti podle konfigurace, vytvořit konfiguraci a vrátit příznak, jestli během editačního režimu došlo ke změně jeho konfigurace. Komponenty, které stojí v hierarchii rozhraní nejvýše, zpracují svoji část konfigurace a její zbytek rozdělí svým vnořeným widgetům. Ty zase použijí svoji konfiguraci, a rekurzivně se tak vytvoří celé rozhraní. Obdobným způsobem se bude postupovat i v opačném směru a z jednotlivých konfigurací widgetů se složí výsledný objekt konfigurace.

Zatím ale chybí kořen celé hierarchie, který by zajistil získání a uložení konfiguračního objektu a inicioval vytváření, resp. ukládání konfigurace. Definuji proto ještě jednu třídu nazvanou ***Desktop.ConfigManager***. Metody třídy ***loadConfig()*** *a* ***saveConfig()*** vyřeší získání/uložení konfiguračního objektu. Tuto třídu přidám do seznamu předků třídy *Desktop.wgDesktopEnviroment*.

Widget prostředí potom jednoduše zajistí volání metody *loadConfig()/saveConfig()* vrcholové komponenty pracovní plochy, kterou představuje widget kontejneru záložek.

8.2 Implementace serverové části

Těžiště implementace prototypu pracovní plochy leží na klientovi. Přesto se však aplikace neobejde bez serverové části, která musí zajistit evidenci uživatelských účtů pro potřeby autentizace, správu konfiguračních souborů a rovněž získání nabídky odkazů. Logiku serveru implementuji ve skriptovacím jazyce **PHP verze 5.**

Pro reprezentaci serveru vytvořím třídu ***Server***. Její důležité metody shrnuje Př. 8-1.

Metoda	**Popis**
login(login, pass)	Provede přihlášení uživatele do aplikace.
logout()	Odhlásí uživatele z aplikace.
validSession(time)	Zkontroluje, jestli je platná *session* uživatele.
getLoggedUser()	Vrátí jméno přihlášeného uživatele.
getData(name)	Získá obsah souboru ve formátu JSON.
setData(name, data)	Uloží nový obsah souboru ve formátu JSON.
getConfig(login)	Načte obsah soubor konfigurace a pošle ho na klienta.
setConfig(login, cfg)	Získá nový obsah konfigurace a uloží ho do souboru.
createOffer()	Vytvoří nabídku odkazů.

Tab. 8-1 Důležité vlastnosti a metody třídy *Server*

Informace o uživatelských účtech budou uloženy v souboru ***users.json***. Obsah souboru ve formátu JSON ukazuje Př. 8-2.

Př. 8-2 Uložení uživatelského účtu ve formátu JSON

```
{
   "macel": {
      "login": "macel",
      "pass":  "098f6bcd4621d373cade4e832627b4f6",
      "name": "Lukáš Máčel"
   }, …
}
```

Jednotlivé záznamy uživatelů reprezentuji objektem a uložím je do slovníku uživatelských účtů. Přístupový klíč představuje přihlašovací jméno (login) uživatele. Hash uživatelského hesla získaný PHP funkcí ***md5()*** bude uložen ve vlastnosti ***pass*** a vlastnost ***name*** bude obsahovat jméno uživatele, které se zobrazí na informační liště rozhraní.

Přihlášení uživatele zajistí dokument **login.php** obsahující dialog pro zadání přihlašovacího jména a hesla. Odeslané informace se zpracují metodou ***login()*** třídy serveru. Ta nejdříve načte soubor s uživatelskými účty pomocí metody *getData()*. Pro zpracování formátu JSON využiji PHP funkci ***json_decode()***. V získaném registru uživatelů se potom hledá záznam uložený pod uvedeným loginem. Pokud je záznam nalezen, provede se výpočet hashe zaslaného hesla funckcí *md5()* a výsledek se porovná s hodnotou uloženou ve vlastnosti *pass*. V případě úspěchu se uloží záznam uživatele do session a dojde k přesměrování na stránku pracovní plochy. Před inicializací aplikace se vždy provede metoda ***validSession()***, které se předá délka přihlášení v minutách. V případě, že vyprší povolený čas nebo není korektně nastaven obsah session, dojde k přesměrování zpět na stránku *login.php* za účelem nového přihlášení.

Každý uživatel bude mít konfiguraci pracovní plochy uloženou v samostatném souboru s názvem ***login.json***. Jestliže proběhne autentizace v pořádku, zažádá klient o konfiguraci a server pomocí metody ***getConfig()*** načte na základě loginu uloženém v session příslušný soubor a odešle jeho obsah na klienta. Při uložení se postup opakuje s tím rozdílem, že server do souboru zapíše přijatou konfiguraci.

Poslední úloha serveru spočívá v sestavení aktuální nabídky pro menu aplikace, kterou zajišťuje metoda ***createOffer()***. Server od klienta obdrží seznam URL zdrojů a pokusí se získat jejich obsah. Pokud je úspěšný a získaná data jsou ve formátu popsaném v kapitole 5.3, provede sloučení seznamů nabídek a vygeneruje nová id všech položek tak, aby byla zaručena jejich unikátnost v rámci celé nabídky. Výsledný seznam potom odešle na klienta.

9 Závěr

V této práci jsem se zabýval pracovními plochami spustitelnými ve webovém prohlížeči. Na rozdíl od desktopových prostředí, jejichž pracovní plocha je dostupná pouze z lokálního počítače, nabízí webové pracovní plochy přístup ke svému obsahu z libovolného místa připojeného k internetu. Díky této skutečnosti získává uživatel/ka možnost využívat svoji pracovní plochu, i když u sebe právě nemá osobní počítač.

Aplikace webové pracovní plochy se skládá ze dvou základních částí – klienta a serveru. Klient zajišťuje webové uživatelské rozhraní pro zobrazení a správu obsahu plochy. Úloha serveru spočívá v perzistentním uložení konfigurace plochy, která popisuje její obsah a veškerá nastavení rozhraní.

Současné webové pracovní plochy nabízí uživateli shromáždit a spravovat odkazy na různé webové zdroje. Jejich obsah je reprezentován pomocí množiny samostatných komponent označkovaných jako miniaplikace (gadgets). Miniaplikace představují samostatnou komponentu zajišťující zobrazení příslušného webového zdroje, která je graficky zobrazena ve formě panelu. Aplikace pracovní plochy potom dovoluje přidat/odebrat miniaplikace a zajišťuje jejich rozmístění na ploše prostřednictvím správce rozložení. Ten pro organizaci obsahu pracovní plochy využívá záložek a jednotlivé miniaplikace potom uspořádává do sloupců.

V další části práce jsem se zabýval prostředky potřebnými pro tvorbu webového rozhraní a prezentaci dat. Základ webového rozhraní tvoří dokument specifikovaný pomocí jazyka HTML. Vzhled rozhraní je popsán nezávislým stylovým předpisem prostřednictvím jazyka CSS. Dynamické změny rozhraní se dají realizovat skriptovacím jazykem Javascript, který mění obsah a strukturu rozhraní skrze objektový model dokumentu (DOM). Akce, které provedl/a uživatel/ka na rozhraní, jsou popsány pomocí objektu události. Způsob vzniku události a tok události modelem dokumentu popisuje událostní model. Reakci na podněty uživatele potom zajišťuje javascriptová funkce registrovaná jako obsluha události. Všechny předešlé technologie spojuje technika označovaná jako AJAX, která dovoluje asynchronně poslat žádost o nová data na server bez potřeby aktualizovat celé rozhraní aplikace. Pro popis dat předávaných mezi klientem a serverem lze využít jednoduchý a kompaktní textový formát JSON.

Na základě analýzy současných webových pracovních ploch a výše shrnutých technologií jsem navrhnul a implementoval prototyp pracovní plochy sloužící jako hlavní stránka portálu. Pracovní plocha musí být spustitelná v prohlížečích Internet Explorer 6 a 7, Firefox 2 a Opera 9+. Aplikace umožní uživateli/uživatelce portálu vytvořit a uložit odkazy na dokumenty a stránky portálu, nebo na jiné zdroje dostupné z internetu. Kromě toho nabídne také vložení jednoduché miniaplikace kalkulačky a kalendáře.

Po úspěšném přihlášení uživatele aplikace poskytne operace pro vytváření, modifikaci a odstranění odkazů. Uživatel/ka může přidávat nové odkazy nebo využít nabídky již

předpřipravených odkazů. Z dalších služeb pracovní plochy ještě zdůrazním změnu vzhledu rozhraní založenou na grafických tématech a podporu lokalizace.

Pracovní plocha nabídne hierarchické členění obsahu prostřednictvím panelů a záložek. Panel bude sloužit pro sdružení souvisejících odkazů a také jako kontejner miniaplikace. Záložky potom dovolí seskupení panelů. Uživatel/ka bude moci přemisťovat všechny prvky pracovní plochy technikou drag & drop. Pro rozmístění obsahu jsem navrhnul sloupcové rozložení s možností měnit počet a šířku jednotlivých sloupců. Součástí webového rozhraní bude také postranní menu obsahující nabídku zajímavých odkazů, které bude možné přidat na pracovní plochu. Veškeré změny půjdou realizovat pouze v editačním režimu.

Obsah pracovní plochy, jeho rozmístění a další uživatelská nastavení tvoří konfiguraci, kterou bude poskytovat a ukládat konfigurační server. Server dále zajistí autentizaci uživatelů a vytvoření nabídky odkazů dostupných v postranním menu aplikace. Nabídku sestaví z odkazů poskytovaných zdrojovými servery, jejichž adresu si uživatel specifikuje v rámci konfigurace. Pro komunikaci mezi serverem a klienty jsem navrhnul jednoduchý protokol, který bude založen na posílání zpráv ve formátu JSON. Komunikaci realizuji technikou AJAX a server implementuji pomocí skriptovacího jazyka PHP 5.

Požadavek podpory více webových prohlížečů přináší nejednotné programové prostředí a z toho plynoucí rozdíly v interpretaci rozhraní a základních operací pracovní plochy různými webovými prohlížeči. Z tohoto důvodu jsem se rozhodl pro implementaci prototypu pracovní plochy využít javascriptovou knihovnu Dojo.

Knihovna standardizuje programové prostředí a nabízí řadu užitečných služeb pro urychlení vývoje webové aplikace. K nejdůležitějším patří podpora modularizace kódu, rozšíření jazyka v oblasti definice tříd nebo implementace operace drag & drop. Knihovna Dojo nabízí také širokou škálu již hotových komponent (widgetů) pro tvorbu webového uživatelského rozhraní. Chování widgetů lze ovlivňovat pomocí metod životního cyklu komponenty, jejich vzhled pak prostřednictvím šablony uložené v samostatném souboru.

Během implementace jsem využil především widget zajišťující rozložení (layout) obsahu rozhraní. Na jeho základě jsem definoval komponentu obecného panelu miniaplikace, kterou potom dále specializuji za účelem vytvoření panelu odkazů a také aplikací kalkulačky a kalendáře. Z dalších widgetů knihovny Dojo, které jsem použil, uvedu například komponentu stromu nebo widget dialogu.

Pro změnu grafického vzhledu využívám grafických témat specifikovaných prostřednictvím externího stylového předpisu. Vycházím ze základních témat knihovny, které dále rozšiřuji o popis grafické prezentace komponent pracovní plochy. Jazykovou lokalizaci rozhraní řeším již na úrovni modulů aplikace pomocí definice externího souboru obsahujícího všechny jazykově závislé texty uložené ve formátu JSON.

9.1 Přínos práce

Podařilo se mi vytvořit prototyp pracovní plochy, kterou lze dobře využít jako úvodní stránku portálu. Aplikace dovoluje uživatele/uživateli přehledně vytvářet a organizovat odkazy, které potom může použít při navigaci po dalších stránkách portálu. Za přínos považuji také přehlednou strukturu obsahu pracovní plochy a intuitivní uživatelské rozhraní umožňující přemisťovat prvky pracovní plochy pomocí techniky drag & drop. Pozitivum pracovní plochy spatřuji dále v nabídce zajímavých odkazů, které může uživatel/ka libovolně přidávat na pracovní plochu, aniž by musel/a pracně zadávat jejich URL adresy.

Další přínos aplikace spočívá podle mého názoru v možnostech její snadné rozšiřitelnosti. Rozhraní pracovní plochy je založeno na objektově orientovaném návrhu, který dovoluje opětovně použít jednotlivé komponenty při definici nových bez nutnosti znovu implementovat veškerou funkcionalitu. V nabídce jsou sice prozatím jenom miniaplikace kalkulačky a kalendáře, ale definice nových miniaplikací nepředstavuje složitý úkol, protože postačuje pouze specializovat základní komponentu panelu. Další vývoj aplikace ulehčuje také fakt, že aplikace používá knihovnu Dojo, která poskytuje řadu užitečných služeb a nástrojů.

Přínos vidím rovněž v jasném oddělení všech lokalizovaných textů aplikace umístěných v externích souborech. Díky tomu lze provést rychlý překlad celého rozhraní do nového jazyka. Jednoduchá specifikace grafických témat pouze prostřednictvím samostatného stylového předpisu dovoluje snadné vytvoření nového vzhledu aplikace.

Za pozitivum prototypu pracovní plochy považuji také podporu nejrozšířenějších prohlížečů, mezi které patří Internet Explorer, Firefox a Opera.

9.2 Další vývoj projektu

Další rozšíření pracovní plochy spočívá například v přidání nových miniaplikací, které může uživatel/ka využívat. Mezi zajímavé komponenty z pohledu úvodní stránky portálu patří miniaplikace vyhledávače sloužící pro nalezení potřebných dokumentů. Jinou užitečnou komponentu představuje miniaplikace zobrazující RSS kanály informující o novinkách na portálu.

Správce pracovní plochy plně zajišťuje rozmístění komponent a jejich reorganizaci technikou drag & drop. Předmětem dalšího vývoje by mohla být podpora přesunu záložek. Další užitečnou operací, kterou prototyp zatím nenabízí, je přemístění miniaplikací a odkazů mezi jednotlivými záložkami pomocí drag & drop.

Menu aplikace zatím podporuje pouze hierarchické zobrazení nabídky odkazů ve formě stromu. Do budoucna by bylo vhodné nabídnout také jiný způsob grafické prezentace, například v podobě datové mřížky, která by dovolila třídit položky nabídky podle jejich atributů, případně vyhledávat a filtrovat zobrazené položky. Struktura dat je navržena tak, aby takovéto rozšíření

umožnila. Prototyp pracovní plochy dovoluje umísťovat odkazy na pracovní plochu pouze jednotlivě. Užitečná by byla podpora přesunu celých skupin odkazů pomocí drag & drop, která by způsobila vložení panelu odkazů.

Z hlediska globálního nastavení aplikace lze například nabídnout nová grafická témata nebo přidat další jazyky, do kterých je aplikace přeložena.

V době psaní této práce jsou dostupné moderní webové prohlížeče jako Internet Explorer 8, Google Chrome nebo Firefox 3, které jsem však nestihl zahrnout do seznamu podporovaných prohlížečů. Testy, které jsem prováděl, ukazují, že pracovní plocha zatím není plně kompatibilní s těmito prohlížeči. Další vývoj projektu by měl zohlednit nově vznikající webové prohlížeče a zajistit jejich podporu.

Literatura

[1] *Desktop environment* [online]. 30 December 2008. [cit. 2009-01-01]. Wikipedia. Dostupný na URL: <http://en.wikipedia.org/wiki/Desktop_environment>.

[2] *Desktop Metaphor* [online]. 25 December 2008. [cit. 2009-01-01]. Wikipedia. Dostupný na URL: <http://en.wikipedia.org/wiki/Desktop_metaphor>.

[3] *Gadgets Specification* [online]. January 28, 2008. [cit. 2009-01-01]. Dostupný na URL: <http://code.google.com/intl/cs/apis/gadgets/docs/spec.html>.

[4] *HTML 4.01 Specification* [online]. 24 December 1999. [cit. 2009-05-21]. Dostupný na URL: <http://www.w3.org/TR/html4/>.

[5] ECMA: *ECMAScript Language Specification* [online]. [cit. 2009-05-21]. Dostupný na URL: <http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf>.

[6] *JSON* [online]. [cit. 2008-12-12]. Dostupný na URL: <http://www.json.org/>.

[7] *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification* [online]. 23 April 2009. [cit. 2009-05-21]. Dostupný na URL: <http://www.w3.org/TR/CSS2/>.

[8] *Anatomy of a UWA widget* [online]. 2008/10/24. [cit. 2009-01-01]. Dostupný na URL: < http://dev.netvibes.com/doc/uwa/documentation/anatomy_of_a_uwa_widget>.

[9] Garrett, J. J.: *Ajax: A New Approach to Web Applications* [online]. February 18, 2005. [cit. 2009-05-21]. Dostupný na URL: <http://adaptivepath.com/ideas/essays/archives/000385.php>.

[10] Darie, C. a kol.: *AJAX a PHP: tvoříme interaktivní webové aplikace profesionálně*. 1. vydání. Zoner Press, Brno, 2006. 320 s. ISBN 80-86815-47-1.

[11] *Document Object Model (DOM) Technical Reports* [online]. 2004/07/08. [cit. 2009-05-21]. Dostupný na URL: <http://www.w3.org/DOM/DOMTR#dom1>.

[12] *Document Object Model Events* [online]. 13 November, 2000. [cit. 2009-05-21]. Dostupný na URL: <http://www.w3.org/TR/DOM-Level-2-Events/events.html>.

[13] Gill, R., Riecke, C., Russell, A.: *Mastering Dojo: JavaScript and Ajax Tools for Great Web Experiences*. Pragmatic Bookshelf, 2008. 568 s. ISBN 978-1934356111.

[14] *The Book of Dojo* [online]. 04/14/2008. [cit. 2009-05-21]. Dostupný na URL: <http://www.dojotoolkit.org/book/dojo-book-1-0>.

[15] Russell, M. A.: *Dojo: The Definitive Guide*. O'Reilly, 2008. 500 s. ISBN 978-0596516482.

[16] *Page Load / Unload* [online]. 06/05/2007. [cit. 2009-05-08]. Dostupný na URL: <http://www.dojotoolkit.org/book/dojo-book-0-9/part-3-programmatic-dijit-and-dojo/event-system/page-load-unload>.

[17] *XMLHttpRequest (XHR)* [online]. 04/28/2007. [cit. 2009-05-08]. Dostupný na URL:

<http://www.dojotoolkit.org/book/dojo-book-0-9/part-3-programmatic-dijit-and-dojo/ajax-transports>.

[18] *dojo.dnd package* [online]. 2009-05-11. [cit. 2009-05-09]. Dostupný na URL: <http://docs.dojocampus.org/dojo/dnd>.

[19] Asleson, R., Schutta, N. T.: *AJAX: Vytváříme vysoce interaktivní webové aplikace*. 1. vydání. Computer Press, a. s., Brno, 2006. 262 s. ISBN 80-251-1285-3.

[20] *Document Object Model Events* [online]. 21 December 2007. [cit. 2009-05-21]. Dostupný na URL: <http://www.w3.org/TR/DOM-Level-3-Events/events.html>.

[21] *Document Object Model (DOM) Level 1 Specification (Second Edition)* [online]. 29 September, 2000. [cit. 2009-05-21]. Dostupný na URL: <http://www.w3.org/TR/2000/WD-DOM-Level-1-20000929/>.

Seznam příloh

Příloha 1. Obsah přiloženého CD

Součástí práce je CD, které obsahuje následující soubory:

- text diplomové práce ve formátu PDF,
- text diplomové práce ve formátu DOC,
- programovou dokumentaci,
- zdrojové soubory programu,
- odkaz na demonstrační stránky pracovní plochy,
- pokyny pro instalaci.