České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů

Bakalářská práce

Přehledová dokumentace Mozilla Application Frameworku

Pavel Vybíhal

Vedoucí práce: Ing. Tomáš Novotný

Studijní program: Elektrotechnika a informatika, strukturovaný, bakalářský

Obor: Výpočetní technika

28. května 2010

Poděkování

Tímto bych chtěl poděkovat ing. Tomáši Novotnému za jeho cenné rady a čas, který mi poskytl při tvorbě této práce.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 28. 5. 2010 ……

Abstract

The goal of this thesis is to provide overview of documentation of Mozilla Application Framework in TiddlyWiki application interface. The first part describes the implementation of scripts that automatically compile an outline of existing documentation, the second part deals with the integration of javascript-based WYSIWYM editor „WYMeditor“ into TiddlyWiki interface.

Abstrakt

Tato bakalářská práce se zabývá tvorbou přehledové dokumentace Mozilla Application Frameworku v aplikačním rozhraní TiddlyWiki. V první části je popisována implementace skriptů sloužících k automatizované tvorbě přehledu již existující dokumentace. V druhé části je zmapováno zaintegrování javascriptového WYSIWYM editoru „WYMeditor“ do rozhraní TiddlyWiki.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Na počátku vývoje každého software by jsme si jistě měli položit otázku, jaký programovací jazyk by se nejvíce hodil pro tvorbu naší aplikace. S tím spojené by mohlo být zamyšlení, zda při tvorbě nevyužít již hotové komponenty, které nám práci zjednoduší a zefektivní. Dnes si naštěstí můžeme vybrat z nepřeberné palety frameworků, které nám pomůžou s elementárními problémy, díky čemuž se budeme moci naplno věnovat podstatné problematice tvořící jádro našeho programu.

V této práci se budu zabývat Mozilla Application Frameworkem. Tento balík vzájemně provázaných nástrojů v sobě skrývá, vzhledem ke své platformní nezávislosti, velký potenciál, a proto se těší rostoucí oblibě. Do širšího povědomí se dostává díky použití v naprosté většině produktů rodiny Mozilla, pro které byl také vyvinut.

Na základě výše uvedeného se dá očekávat růst poptávky po dostupných informacích. Proto si tato práce klade za cíl vytvořit přehled existující dokumentace, který by posloužil vývojářům pracujícím s tímto frameworkem a tím jim ušetřil drahocenný čas, kterého při tvorbě kvalitního software jistě není nikdy nazbyt.

Jádrem mého přehledu, tvořeného v aplikačním rozhraní TiddlyWiki, bude seznam existujících interfaců frameworku spojený s konkrétními funkčními odkazy na dostupnou on-line dokumentaci. Dále zde bude předpřipraven prostor pro manuální editaci, díky kterému bude umožněno tuto databázi cíleně doplňovat. Abych i tuto činnost zjednodušil, zaměřil jsem se v druhé části této práce na integraci javascriptového editoru WYMeditor do rozhraní TiddlyWiki. Toto vylepšení není vázáno pouze na tuto přehledovou dokumentaci, a tak si zajisté najde uplatnění i u běžných uživatelů.

2 Dokumentace Mozilla Application Frameworku

Mozilla Application Framework je soubor multiplatformních softwarových komponent určených pro vývoj aplikací na různých operačních systémech. Základ tohoto frameworku tvoří `XUL` (jazyk na bázi `XML` sloužící pro tvorbu grafických rozhraní aplikací) a vykreslovací jádro `Gecko` (na `XUL` stavěné, používané produkty Mozilla pro vykreslování webových stránek) [5]. Celkově je tato vývojová platforma užívána takřka výhradně v softwarové rodině Mozilla.

2.1 Existující dokumentace

Existující dokumentace Mozilla Application Frameworku je poměrně rozsáhlá, avšak na můj vkus ne příliš přehledně uspořádaná. Existuje totiž vícero serverů, které se dokumentací zabývají, a tak je hledání určité konkrétní věci složité – ne vždy lze informaci najít na první pokus. Dokumentace určitého problému buď na serveru chybí nebo je neúplná. Proto je nutné procházet více stránek a nalezené informace následně sloučit. Z tohoto důvodu jsem si dal za cíl práci s dokumentací Mozilla Application Frameworku pokud možno zjednodušit a zpříjemnit.

V své práci jsem se zaměřil na zmapovaní dostupných informací o interfacech Mozilla Application Frameworku, zejména na jejich souhrn a roztřídění. Databázi jsem začal tvořit v rozhraní TiddlyWiki, neboť je tato aplikace svým charakterem pro mou potřebu velmi vhodná. Výhodu představuje zejména snadná rozšiřitelnost o vlastní skripty a vcelku jednoduché uživatelské úpravy vzhledu a funkčnosti. Výše uvedené vlastnosti představují prostor pro zefektivnění mé práce, neboť podle rozsahu dokumentace je patrné, že bude potřeba třídění informací co nejvíce zautomatizovat.

Jako hlavní zdroje informací jsem po konzultaci s vedoucím práce zvolil níže uvedené servery.

2.1.1 Server db48x.net

Na tomto serveru [8] se nachází velmi rozsáhlá knihovna vytvořená pomocí nástroje Doxygen. Jednotlivé stránky jsou zde uložené ve složkách `struct`, `class` a `interface`. Každý zde popisovaný objekt má svou vlastní HTML stránku, jejíž název odpovídá názvu daného interfacu. Výše uvedené vlastnosti činí tento dokumentační server vhodným pro zautomatizované dotazování dat. Ve své další práci se pravděpodobně zaměřím na kontrolu existence dokumentace na tomto serveru a v případě úspěchu na vytvoření odkazu na příslušnou stránku. Pokud to bude možné, pokusím se i odfiltrovat užitečná data.

Obrázek 1: Příklad stránky umístěné na serveru db48x.net

2.1.2 Mozilla Developer Center

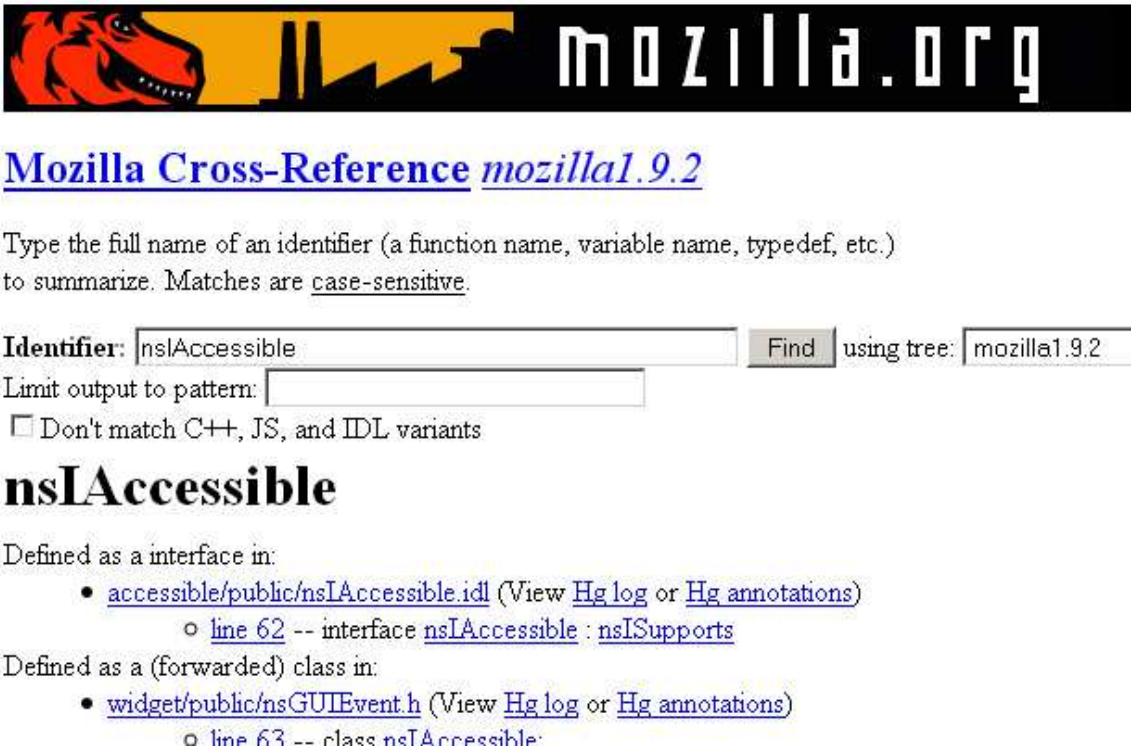
Mozilla Developer Center je server [7] patřící neziskové organizaci Mozilla Foundation, na kterém můžeme najít dokumentaci k projektům založeným na Mozilla Application Frameworku. Adresy stránek s dokumentací vycházejí vždy z adresy `https://developer.mozilla.org/en/`, případně z adresy `https://developer.mozilla.org/en/Toolkit_API/`, dané dokumenty lze vždy dohledat pod jejich názvy. Opět se zde tedy jeví prostor pro automatizaci dotazů.

Obrázek 2: Příklad stránky umístěné na Mozilla Developer Center

2.1.3 Mozilla Cross-Reference

Mozilla Cross-Reference je server [6], který vyhledává ve zdrojových kódech z verzovacích serverů mozilla.org - CVS, Mercurial a Subversion. Výsledky hledaného

dotazu jsou zobrazeny na jedné stránce ve struktuře se základními informacemi o umístění požadovaného objektu. Vyhledávač je přístupný pod adresou `http://mxr.mozilla.org/mozilla1.9.2/ident?i=`, za kterou jen stačí zadat správný pojem. Při tvorbě mého přehledu odkaz na tento server určitě využiji a pokusím se ze získaných informací extrahovat informace o souboru, ve kterém je daný interface umístěn, což využiji při dalším logickém členění.



Obrázek 3: Příklad stránky umístěné na Mozilla Cross-Reference

2.2 Rozhraní TiddlyWiki

TiddlyWiki [11] je zvláštní druh publikačního systému. V podstatě se jedná o jednu `HTML` stránku, která nám ovšem nabízí veškeré vymoženosti známého systému Wiki. Uživatel zde může vytvářet a editovat články (zvané tiddlery), třídit dle štítků – tagů, vyhledávat v databázi. Nechybí ani bohaté uživatelské nastavení jak vzhledu, tak funkčnosti. A to vše v jednom jediném souboru, který bez jakékoli instalace zpřístupníme například nahráním na libovolný webový server. Stejně bezproblémový je i například přenos na flash disku a podobně.

Celá TiddlyWiki je založená na technologiích `HTML`, `CSS` a `JavaScript`, což z ní dělá velmi přenositelnou aplikaci. K jejímu spuštění tak stačí prakticky jakýkoli internetový prohlížeč. Databáze uživatelských dat je uložena v samotném souboru aplikace, nejsou tedy nutné (naopak jsou nežádoucí) jakékoliv další soubory.

Uživatel TiddlyWiki nemusí být příliš počítačově gramotný. Obsluha je díky jednoduchému uživatelskému rozhraní velice intuitivní. Dokonce ani vlastní přidávání příspěvků nevyžaduje znalost programovacích jazyků. Podobně jako jakákoli Wiki, i tato má vlastní značkovací jazyk [12], který je názorný a dobře pochopitelný. Jeho syntaxe je velmi podobná stávajícím a dobře známým značkovacím jazykům použitým například ve Wikipedii.

Jakýkoli uživatelský obsah se vkládá do jednotlivých tiddlerů. Vedle těchto se v aplikaci nachází množství takzvaných „shaddow“ tiddlerů, kterými lze poměrně zásadním způsobem měnit funkčnost celé aplikace. Tyto tiddlery jsou v aplikaci předdefinované. Jejich změnou či doplněním se přepíše standardní vnitřní nastavení a aplikuje se nové. Takto lze třeba editovat styly vzhledu, šablony těla příspěvků, ovládací panely či hlavní menu aplikace. Vzhledem k charakteru změn, které lze tímto provést, náleží tyto úpravy spíše programátorům než běžným uživatelům. Klasické bezpečné uživatelské nastavení lze provádět v hlavním menu pod položkou options, případně v „backstage“, které je umístěné v záhlaví aplikace.

V „backstage“ najdeme nástroje pro správu celé aplikace. Nás bude dále především zajímat položka „tweak“, pod kterou se nachází pokročilé nastavení aplikace (blíže v kapitole 3.7.3.4), a „plugins“, ve které můžeme spravovat dostupné pluginy.

Obrázek 4: Náhled menu s nabídkou shadowed tiddlerů (již editované tiddlery tučným písmem)

Jako doplněk k dosud uváděným je ještě možné vytvářet vlastní systémové tiddlery, prostřednictvím kterých může uživatel vytvářet vlastní pluginy, čímž dokáže zásadním způsobem ovlivnit či doplnit funkčnost TiddlyWiki. Z obyčejného tiddleru se stane systémový pouhým přidáním tagu `systemConfig`. Tím se zaktivuje veškerý v něm obsažený javascriptový kód, který se při spuštění nebo znovunačtení aplikace provede. Tyto jednotlivé tiddlery se v aplikaci označují jako pluginy a lze je spravovat v již zmíněném backstage.

Obrázek 5: Náhled okna se správou systemConfig tiddlerů – pluginů.

Pro mou potřebu se TiddlyWiki jeví jako velmi vhodná. Základem této práce bude vytvoření databáze, její naplnění a udržování v aktuálním stavu a v neposlední

řadě její logické rozčlenění pro dobrou orientaci vývojářů. V závěrečné fázi se pokusím rozhraní TiddlyWiki vylepšit tak, aby další případné přidávání dat bylo efektivní a uživatelsky příjemné.

Pro svou práci jsem z oficiálních stránek [11] projektu stáhnul aktuálně nejnovější verzi TiddlyWiki s označením 2.6.0. Při programování budu využívat informace z hlavního dokumentačního serveru [2]. Veškeré úpravy TiddlyWiki budou probíhat v tiddlerech, jejichž seznam je uveden v příloze B. Provedené změny tak bude snadné přenést do novějších verzí této aplikace.

2.3 Generování interfaců

Vzhledem k rozsahu dostupné dokumentace Mozilla Application Frameworku je patrné, že mou snahou bude tento proces maximálně automatizovat. V aplikaci TiddlyWiki se tedy pokusím vytvořit a začlenit několik skriptů, které budou mít za úkol generovat a aktualizovat informace o interfacech. V databázi budu spravovat funkční odkazy na existující dokumentaci a další základní informace, které posléze využiji k logickému členění.

Jako první krok sem se rozhodl vytvořit tiddlery pro jednotlivé interfacy. K danému účelu jsem v TiddlyWiki vytvořil makro `InterfaceGenerator`, které jsem zpřístupnil přes tlačítko v hlavní nabídce aplikace. Seznam jmen všech interfaců jsem získal z objektu `Components.interfaces`. Tento objekt, přístupný v aplikacích využívajících Mozilla Application Framework (tedy tento skript je možné využívat pouze v prohlížeči Mozilla Firefox a v dalších produktech založených na jádře `Gecko`), v sobě obsahuje jména jednotlivých interfaců, které jsou v programu dostupné. Automatický generátor pak pro každý interface vytvoří vlastní tiddler a označí ho tagem „`Interface`“, což mi pak v budoucnu usnadní další kroky. Vlastní tělo tiddleru je prázdné, obsahuje pouze předpřipravený prostor pro generátor obsahu.

```
foreach (var tiddler in Components.interfaces) {
  if (not(existujeTiddler(tiddler)) {
    var obsah = '!! generovaný obsah\n{{genObsah{}}}';
    vytvorTiddler(tiddler, obsah, 'Interface');
  }
}
```

Pseudokód generátoru tiddlerů

Na první pohled se zdá, že daný skript slouží pouze k jednorázovému vytvoření základní databáze. Já jsem se však rozhodl jej udělat univerzálněji. Díky budoucím aktualizacím Mozilla Application Frameworku se může seznam interfaců rozšířit a tak by můj přehled nebyl úplný. Proto lze opětovným spuštěním skriptu zkontrolovat, zda existují tiddlery pro veškeré objekty z `Components.interfaces`. Případné chybějící se pak automaticky dotvoří.

Tímto způsobem je do databáze vloženo takřka 1400 tiddlerů, což jen dokazuje fakt, že bez vhodné automatizace by byla tvorba jakékoli přehledové dokumentace nemyslitelná.

2.4 Aktualizace interfaců

Mechanizmus aktualizace informací o daném interfacu spočívá ve stažení dostupných informací, jejich setřídění, úpravy pro výstup a uložení zpět do tiddleru. Pro jakékoliv stahování dat na pozadí lze velmi efektivně použít technologii AJAX [1]. V TiddlyWiki integrovaný framework jQuery [3] přímo zjednodušenou práci s AJAXem umožňuje, což je pro mé účely naprosto dostačující. AJAX slouží pro vývoj interaktivních webů, jejichž obsah je měněn bez nutnosti znovunačítání stránky. V poslední době se tato technologie právem těší velké oblibě.

2.4.1 Struktura zobrazovaných dat

Jako základní kostru těla tiddleru jsem zvolil následující:

- funkční odkazy na podrobné informace o konkrétním interfacu
- jméno a adresa souboru, ve kterém je daný interface definován
- seznam členských funkcí
- prostor pro manuální doplnění informací

Seznam přímých funkčních odkazů usnadní případnému vývojáři vyhledávání informací – zde uvedené linky míří přímo na konkrétní funkční stránku s dokumentací, odpadá tak složité vyhledávání po různých serverech. Základní odkazy směřují na výše uvedené servery doxygen.db48x.net, developer.mozilla.org a mxr.mozilla.org. Skript automaticky kontroluje existenci dokumentace na daném serveru (např. server db48x.net má stránky s interfacy pod třemi různými adresami), případně vytvoří odkaz pro vyhledání dotazu přímo v databázi serveru.

Obrázek 6: Náhled tiddleru s automaticky aktualizovaným obsahem

Jméno a adresa souboru, kde je interface definován, je získávána z dokumentační stránky na serveru mxr.mozilla.org, pokud tedy konkrétní stránka

existuje. Daná informace je odfiltrována promocí regulárních výrazů. Podobně je získán seznam členských funkcí, který ovšem informace stahuje ze serveru `db48x.net`.

Mimo generovaný blok je pak možné provádět manuální editaci. Zde se tedy nachází prostor pro práci vývojáře, který se danou problematikou hlouběji zabývá a může tak zde sjednotit své případné postřehy a poznatky.

2.4.2 Vlastní mechanizmus aktualizace

Pro vlastní aktualizaci jsem vytvořil makro `InterfaceUpdate`, které zajišťuje jak automatické stažení a uložení dat, tak začlenění funkce do rozhraní TiddlyWiki.

Základem je tedy stažení požadovaných dat. K tomuto účelu jsem vytvořil funkci `dalsiTiddlerUrl()`, která na základě parametrů pomocí `jQuery.ajax` vysílá na pozadí konkrétní dotazy. Na základě úspěchu či neúspěchu funkce upraví šablonu generovaných dat – tedy buď vloží získaná data do těla tiddleru nebo nikoliv. Po dokončení všech dotazů, zpracování výsledků a případné odfiltrování požadovaných informací je obsah tiddleru uložen. Princip dotazovací funkce je nastíněn v pseudokódu uvedeném níže.

```
var seznamUrl = [ { 'seznam URL dotazovaných serverů' } ];
function dalsiTiddlerUrl(tiddler, seznamUrl) {
  var adresa = seznamUrl.pop();
  if (adresa) {
    jQuery.ajax( //dotaz pomocí AJAX
      if (ziskejData(adresa)) {
        zpracuj(data, adresa);
        dalsiTiddlerUrl(tiddler, seznamUrl);
      }
      else {
        zpracujChybu(adresa);
        dalsiTiddlerUrl(tiddler, seznamUrl);
      }
    );
  }
}
function zpracuj(data, adresa) {
  upravSablonu(data, adresa);
  if (vsechnyDotazy() == true) {
    saveTiddler(title);
  }
}
```

Pseudokód dotazovací funkce

Součástí aktualizace je taktéž vytvoření tagu tiddleru s prefixem „`SOUBOR:`“, obsahujícího adresu složky, ve které se interface nachází. Tento tag je dále upraven z důvodů následujícího logického členění interfaců – z adresy jsou vynechány složky „public“ a „idl“, které členění znepřehledňují, protože se v adresách vyskytují často a nemají příliš vypovídající hodnotu.

Při dotazování jsem musel řešit následující problém: Z bezpečnostních důvodů není v Mozille Firefox defaultně umožněno stahování informací na pozadí z jiného

serveru, než kde se aplikace nachází. Proto jsem byl nucen toto nastavení změnit a udělat výjimku pro moje dotazy pomocí následujícího kódu:

```
netscape.security.PrivilegeManager
      .enablePrivilege("UniversalXPConnect");
```

2.4.3 Spouštění aktualizace tiddlerů

Pro spuštění aktualizace daného tiddleru jsem vytvořil v rozhraní TiddlyWiki tlačítko, které jsem umístil do ovládacích prvků každého tiddleru – do `ViewToolbaru`, který lze snadno editovat konfiguračním „shadow“ tiddlerem `ToolbarCommands`. Na základě existence výše uvedeného tagu s prefixem „`SOUBOR:`“ jsem zajistil, aby se funkce spouštěla pouze na tiddlery obsahující interface s prostorem pro automaticky generovaná data. Takto lze tedy jednoduše obnovit data vybraného tiddleru.

Obrázek 7: Začleněné tlačítko aktualizace tiddleru do rozhraní TiddlyWiki

Ne vždy by asi bylo aktualizování tiddlerů po jednom efektivní, proto jsem se rozhodl implementovat další možnosti spuštění aktualizace – a to aktualizování právě otevřených tiddlerů a aktualizování všech tiddlerů.

Aktualizaci právě otevřených tiddlerů jsem implementoval v makru `InterfaceUpdateStory`. Dané makro je spustitelné pomocí tlačítka, které je začleněno do hlavní nabídky aplikace. Vlastní kód makra získá z objektu `story` seznam právě otevřených tiddlerů, nad kterými zavolá výše uvedenou dotazovací funkci umístěnou v makru `InterfaceUpdate`.

```
var seznamTiddleru;
function dalsiTiddler() {
  var title = seznamTiddleru.pop();
  if(title != null) {
    var tags = store.getTiddler(title).getTags();
    if(akttiddler.zacina("SOUBOR:")) {
      dalsiTiddlerUrl(title, seznamUrl); //provedeme aktualizaci
    }
    else {
      dalsiTiddler(); //přeskočíme na další tiddler
    }
  }
}
```

Pseudokód aktualizační funkce

Aktualizační funkce se spouští rekurzivně, aby bylo možné využít asynchronní chování AJAXu, a zároveň aby se dotazy pro jednotlivé tiddlery nespustily všechny naráz a nedošlo tak k přetížení aplikace.

Obrázek 8: Hlavní nabídka TiddlyWiki s integrovanými tlačítky pro volání maker

Analogickým způsobem je vytvořena funkce `InterfaceUpdateAll`, která ovšem seznam tiddlerů k aktualizaci získává z úložiště všech tiddlerů – `store`. Výše zmiňovaná funkce `dalsiTiddler()` je pak zavolána na kompletní seznam všech tiddlerů v dané TiddlyWiki.

Aktualizace celé databáze tiddlerů je vzhledem k rozsahu dat velice časově náročná činnost, avšak její provedení je nutné pouze při zakládání databáze. Další spouštění již záleží na vůli uživatele. Dá se předpokládat, že budou prováděny spíše aktualizace menších celků, se kterými bude uživatel v tu danou chvíli pracovat.

Pro signalizaci probíhající činnosti je v hlavním uživatelském menu spuštěna animace představující načítání, která se, pokud je otevřen, taktéž zobrazuje v těle právě aktualizovaného tiddleru. Tímto je uživatel neustále informován o aktivitě skriptu.

2.4.4 Ukládání tiddlerů

Ukládání změn provedených aktualizační funkcí závisí na centrálním nastavení v položce options. Pokud je zde zaškrtnuta volba „AutoSave“, je každá změna provedená aktualizačním skriptem ihned uložena. V opačném případě je nutné, po doběhnutí úlohy, veškeré změny uložit naráz manuálně pomocí volby „save changes“ v hlavní nabídce.

2.5 Členění interfaců

Aby byla dokumentace přehledná, bylo nutné vymyslet způsob členění interfaců. Po dohodě s vedoucím práce jsem se rozhodl pro rozřazení dle stromové struktury souborů, ve kterých se interfacy nacházejí. Díky tomu budou pohromadě interfacy, jejichž funkce má podobné zaměření.

Obrázek 9: Ukázka generované stromové struktury souborů

2.5.1 Způsob členění

Pro tvorbu přehledové stromové struktury jsem vytvořil makro `InterfaceIndex`, které, na základě tagů s informací o uložení interfacu, sestaví strom. Za účelem zachování přehlednosti zde nejsou přímo odkazy na jednotlivé tiddlery s interfacy – seznam by byl velmi rozsáhlý a ztratil by svou názornost. Z tohoto důvodu jsem se rozhodl pro vytvoření dalších tiddlerů, které jsou odkazovány z jednotlivých listů a uzlů stromu. Odkazy na vlastní interfacy označené stejným tagem se nacházejí uvnitř.

2.5.2 Mechanizmus členění

Jak jsem již výše uvedl, vlastní tvorbu stromu zajišťuje makro `InterfaceIndex`. Toto má za úkol nejen zobrazení na základě tagů vytvořeného stromu, ale i tvorbu tiddlerů s jednotlivými podseznamy.

Základem je for cyklus, který pomocí zabudované TiddlyWiki funkce `getTags()` získá seznam všech tagů systému. Z těchto použije jen ty, které obsahují informaci o složce. Jednotlivé tagy jsou postupně procházeny a na jejich základě je vypisována stromová struktura. Pro každý tag (tj. konkrétní složku ve stromové struktuře, ve které se nachází určitý soubor s interfacem) je tedy vytvořen odkaz na tiddler, ve kterém je možné najít podseznam vlastních tiddlerů s danými interfacy.

Obrázek 10: Náhled tagů přidaných k tiddleru s interfacem

```
function zaridTiddler(nazev) {
  if (not(existujeTiddler(nazev))) {
     vytvorTiddler(nazev,  '<<InterfaceSubList>>');
  }
}
var seznam = store.getTags();
foreach (var polozka in seznam) {
  if (polozka.zacina("SOUBOR:")) {
    var adresa = polozka.urizni("SOUBOR:");
    var aktSlozka = adresa.rozdelitNaPodslozky(/[/]{1}/);
    foreach (var castAdresy in aktSlozka) {
      if (not(bylaVypsana(castAdresy))) {
        vypisOdrazky();
        if(jePosledni(castAdresy, aktSlozka)) {
          zaridTiddler(castAdresy);
          vypisOdkaz(castAdresy);
        }
        else {
          vypisSlozku(castAdresy);
        }
      }
    }
  }
}
```

Pseudokód funkce zajišťující výpis interfaců dle jejich adresářové struktury

Ve výše uvedeném pseudokódu je nastíněna funkce zajišťující výpis interfaců. Postupně jsou procházeny všechny tagy v TiddlyWiki a následně zpracovány jen ty, které začínají prefixem „SOUBOR“. Tento prefix je odstraněn a dále je zpracován zbytek tagu – adresa. Jednotlivé podsložky adresy se projdou a zkontroluje se, zda již byly vypsané. Pokud daná složka dosud zobrazena nebyla, tak je následně provedeno. V případě, že je tato složka v adrese poslední, je zobrazena i s odkazem na tiddler s podseznamem.

Jednotlivé tiddlery s podseznamy, které makro `InterfaceIndex` generuje, jsou označené tagem `InterfaceList` a pojmenované podle jména a adresy složky, jejíž interfacy zobrazují. Vlastní tělo těchto tiddlerů obsahuje pouze volání makra `InterfaceSublist`, které daný výpis tiddlerů i s odkazy na základě tagů vygeneruje při každém volání. Tímto je zabezpečena aktuálnost – případný update tiddlerů se samotnými interfacy je tak zohledněn. Zároveň zde zbývá opět prostor pro případnou ruční editaci, díky které je možné doplnit další informace, týkající se dotyčné skupiny interfaců.

2.6 Ruční editace

Domnívám se, že jsem výše uvedenými úpravami zautomatizoval podstatnou část tvorby ucelené dokumentace a připravil tak dobrý základ pro vývojáře, který by se danou problematikou hlouběji zabýval. V případě potřeby lze však zautomatizované dotazy dále lehce rozšířit. Zda to bude nutné, ukáže pravděpodobně až praxe. Nyní již v dokumentaci zbývá převážně prostor pro ruční editaci. Abych i tuto činnost mým následovníkům zjednodušil a zpříjemnil, rozhodl jsem se, po diskuzi s vedoucím práce, rozhraní TiddlyWiki rozšířit o WYMeditor.

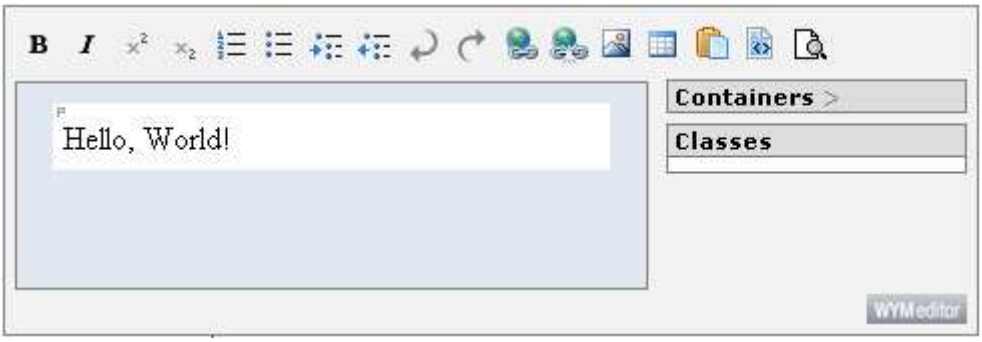
3 Rozšíření rozhraní TiddlyWiki o WYMeditor

3.1 Seznámení s WYMeditorem

WYMeditor je webově založený javascriptový `WYSIWYM` editor. Oproti editorům typu `WYSIWYG` se zaměřuje především na zachování struktury dokumentu, což je při práci na webu velmi důležité. S tím souvisí i to, že generuje XHTML strict kód, který odpovídá standardům `W3C`.

WYMeditor je velmi dobře připraven na integraci do již hotových aplikací. Ve své podstatě je potřeba zajistit načtení vlastních javascriptových souborů, dále pak stylů vzhledu a ostatních prvků grafiky. Nakonec zbývá jen zavolání editoru v určeném místě naší aplikace. Pomocí různých atributů, z nichž nejpoužívanější jsou uvedené v dokumentaci projektu [14], lze editor dále libovolně uživatelsky přizpůsobit.

Po předběžném zmapování situace se zdá jeho použití v TiddlyWiki velmi vhodné. Mimo samotného začlenění a základních úprav funkčnosti bude pravděpodobně spočívat má práce v návrhu překladače z HTML do TiddlyWiki textu.



Obrázek 11: WYMeditor ve své standardní podobě

3.2 Integrace souborů

Z oficiálních stránek projektu [15] jsem nejprve stáhnul samotný program. Zvolil jsem jeho aktuální verzi – WYMeditor 0.5. V distribučním balíčku se nachází vše potřebné pro spuštění, včetně mnoha ukázek integrace a uživatelského přizpůsobení. Mým cílem bylo tyto potřebné soubory začlenit tak, aby se všechny staly součástí TiddlyWiki a nebylo nutné načítat jakákoliv data z externích souborů. Vše samozřejmě pod podmínkou zachování maximální funkčnosti.

Balíček aplikace v sobě obsahuje 3 základní složky:

- `wymeditor` – soubory vlastní aplikace, konfigurační soubory jazykové, vzhledové, pluginy.
- `jquery` – složka se soubory javascriptové knihovny `jQuery`, která je důležitá pro základní funkčnost WYMeditoru
- `examples` – soubory s ukázkami libovolného nastavení a přizpůsobení

Nejprve bylo nutné začlenit do TiddlyWiki základní soubor aplikace. Ten se v distribuci nachází ve třech verzích: nekomprimovaná – vhodná pro vývojáře na

analýzu kódu, standardní – kterou použiji (s příponou `pack`) a minimalizovaná (s příponou `min`). Tento soubor (`jquery.wymeditor.pack.js`) jsem se rozhodl, jako jediný, ponechat externě. Případná aktualizace WYMeditoru se tak jednoduše provede jeho nahrazením. Začlenění do `DOM` TiddlyWiki jsem provedl v tiddleru `WYMeditorPlugin` pomocí následujícího skriptu:

```
var fileref = document.createElement("script")
fileref.setAttribute("type","text/javascript")
fileref.setAttribute("src", "jquery.wymeditor.pack.js")

document.getElementsByTagName("head")[0].appendChild(fileref);
```

Dále se v distribuci nacházejí soubory zajišťující další nastavení – jazykové a vzhledové. Oba tyto typy souborů bylo nutné začlenit do TiddlyWiki. Ze skinů jsem vybral složku s defaultním nastavením, neboť budu vzhled pravděpodobně dále přizpůsobovat standardu TiddlyWiki.

Přidávání stylů vzhledu do TiddlyWiki je poměrně jednoduchá činnost. Veškeré nastavení vzhledu aplikace lze změnit v shadow tiddlerech s prefixem `StyleSheet`. Přidávání vlastních stylů lze pak provést dvěma způsoby. První je přímé vložení definic `CSS` do tiddleru `StyleSheet`. Druhou metodou je uložení stylů do vlastního tiddleru, na který posléze v tiddleru `StyleSheet` vytvoříme odkaz.

Já jsem pro přehlednost zvolil druhou metodu. Kaskádové styly jsem uložil do tiddleru `StyleSheetWYMeditor`, který jsem zavoláním ze systémového tiddleru `StyleSheet` aktivoval a dané styly tak začlenil do `DOM` aplikace. Obrázek obsahující grafiku tlačítek jsem pomocí technologie `dataURI` vložil přímo do definice stylů. Obsah javascriptového souboru s dalším nastavením vzhledu jsem vložil do `systemConfig` tiddleru `WYMeditorPlugin`.

Nastavení popisů tlačítek je prováděno konfiguračními soubory lišícími se zvoleným jazykem. Já jsem se rozhodl pro standardní angličtinu - změna jazyka je však snadná. Obsah vybraného konfiguračního souboru (v mém případě `en.js`) jsem stejně, jako v případě nastavení vzhledu, vložil do `systemConfig` tiddleru `WYMeditorPlugin` a tím zajistil kompletní nastavení jazyka.

```
WYMeditor.STRINGS['en'] = {
    Strong:          'Strong',
    Emphasis:        'Emphasis',
    Superscript:     'Superscript',
    Subscript:       'Subscript',
    Ordered_List:    'Ordered List',
    Unordered_List:  'Unordered List',
    Indent:          'Indent'
```

Obrázek 12: Ukázka konfiguračního souboru nastavujícího jazyk WYMeditoru

Soubory ve složce iframe slouží k vytvoření a nastavení vlastního editačního okna. Tomuto problému se budu podrobně věnovat v kapitole 3.5.

V základní složce distribuce se ještě nacházejí složky `jquery` a `examples`. Název složky `examples` mluví sám za sebe – obsahuje pouze příklady konfigurace a použití. Tyto soubory jsem při práci samozřejmě využíval, ovšem jejich integrace by byla nelogická. Složka `jquery` obsahuje stejnojmennou javascriptovou knihovnu,

která je k běhu WYMeditoru nezbytně nutná. Vzhledem k tomu, že je `jQuery` již součástí TiddlyWiki, nemusím se začleňováním těchto souborů dále zabývat.

3.3 Vlastní volání WYMeditoru

Uvažoval jsem více možností jak WYMeditor začlenit do rozhraní TiddlyWiki. Variantu úplného nahrazení stávajícího jednoduchého textového editoru jsem však ihned zavrhl a tuto základní funkci jsem se rozhodl zachovat jako alternativu. Raději jsem se tedy zaměřil na vytvoření zvláštního tlačítka analogického původní funkci „Edit“, ovšem s funkcí WYMeditoru. Za tímto účelem jsem vytvořil `systemConfig` tiddler `WYMeditor`, do kterého jsem se rozhodl začlenit veškerou funkčnost WYMeditoru od spuštění editace, po její ukončení a uložení.

Základem bylo vlastní vytvoření tlačítka. Definice šablon nabídek a ovládacích prvků je v TiddlyWiki přístupná přes shadow tiddlery, v tomto případě přes tiddler `ToolbarCommands` (viz Obrázek 14). Zde najdeme tabulku, jejíž editací můžeme změnit `ViewToolbar` a `EditToolbar` – tedy nástrojové lišty, které nám jsou přístupné při prohlížení a editaci jednotlivých tiddlerů. Mým cílem bylo zde umístit nové tlačítko pro editaci.

Analogicky funkci Edit jsem vytvořil v `config.commands` objekt `WYMedit`, ve kterém jsem vytvořil název a popis tlačítka, spolu s volanou událostí. Tlačítko jsem začlenil do záhlaví každého tiddleru (`ViewToolbar`) editací výše zmíněného konfiguračního „shadow“ tiddleru `ToolbarCommands`. Hlavní rozdíl handleru tohoto tlačítka, oproti standardnímu „Editu“, tkví ve volání funkce `displayTiddler()`, která zajišťuje zobrazení tiddleru dle předaného parametru – zobrazovacího templatu. Ten jsem byl nucen vytvořit nový. Zde je také zavoláno zobrazení WYMeditoru.

Obrázek 13: Začleněné tlačítko pro editaci ve WYMeditoru

Šablona pro nový editační režim, kterou jsem pojmenoval `WYMTemplate`, vychází z původního `EditTemplatu`. Hlavní změna je v nastavení zobrazení vlastního toolbaru, obsahujícího v sobě tlačítko, zajišťující ukládání změn z WYMeditoru. Tento toolbar, ve kterém se nachází tlačítko `saveWYMTiddler`, jsem vytvořil v „shadow“ tiddleru `ToolbarCommands` jako WYMToolbar.

ToolbarCommands

GUEST, 18 March 2010 (created 27 October 2009)

ViewToolbar	closeTiddler closeOthers editTiddler +**WYMedit InterfaceUpdate** > fields syr
EditToolbar	+saveTiddler -cancelTiddler deleteTiddler
WYMToolbar	+saveWYMTiddler -cancelTiddler deleteTiddler

Obrázek 14: Shadow tiddler ToolbarCommands rozšířený o WYMtoolbar a další tlačítka

Další změna ve WYMTemplatu je volání nového makra `wedit`, které oproti standardnímu `edit` makru zajišťuje jiný překlad wikitextu – podrobněji se této problematice budu věnovat v kapitole 3.7.

3.3.1 WYMedit handler

Události spouštěné v handleru WYMeditu vycházejí z defaultního editu. Velkou změnou, spolu s použitím vlastního templatu, je jistě vytvoření instance WYMeditoru. Toto se provádí velmi jednoduše – zavolání funkce `wymeditor()` na daný prvek. Já jsem v tomto případě použil následující `jQuery` selektor, abych jasně vybral element na stránce a nedocházelo ke konfliktu souběžně spuštěných instancí WYMeditoru.

```
jQuery('div[tiddler='+title+'] .fieldsetFix textarea')
  .wymeditor();
```

Výše uvedený kód je tedy spouštěn při každé nově započaté editaci a samozřejmě umožňuje více editací souběžně, neboť se díky přesnému selektoru instance WYMeditoru navzájem neovlivňují. Samotné zavolání je, jak je vidět, velice jednoduché. O to zajímavější je široká paleta možností nastavení, které se provádějí přidáním parametrů funkci `wymeditor()`. Jednotlivá nastavení a přizpůsobení, která jsem provedl, budu blíže popisovat v následujících kapitolách.

3.4 Přizpůsobení vzhledu

Nejprve jsem se rozhodl WYMeditor přizpůsobit vzhledu TiddlyWiki, aby působil jako její nedílná součást. Hlavní rozvržení jednotlivých prvků editoru je nastaveno pomocí šablony, která se v nastavení nachází pod položkou `boxHtml`. Provedl jsem editaci defaultní šablony a uzpůsobil ovládací prvky tak, aby se nacházely v jedné linii v záhlaví editoru.

Obrázek 15: Vzhled editačního okna WYMeditoru po přizpůsobení vzhledu

Tímto jsem vytvořil hrubou strukturu vzhledu okna editoru, pro doladění finální podoby pak bylo nutné použít CSS. Jednotlivá volání selektorů `jQuery` jsem provedl v položce nastavení `postInit`. Je totiž nutné, aby byly tyto funkce aplikovány až na zcela načtené okno editoru, což položka `postInit` zaručuje. Poté již bylo dokončení nastavení celkového vzhledu pomocí `CSS` velice jednoduché.

3.5 Nastavení editačního okna – iframe

Další problém, na který jsem při integraci narazil, se týkal hlavního editačního okna WYMeditoru. To je standardně tvořeno iframem, jehož obsah – prázdná stránka se

základní HTML strukturou a CSS styly – je načítán z externích souborů. Bylo tedy nutné stávající řešení prostudovat a provést integraci tak, aby byla zachována funkce a přitom nebylo nutné používat externí soubory.

Prvním krokem bylo odstranění načítání stávajícího externího souboru. V nastavení editoru jsem tedy vyhledal šablonu použitou při tvorbě editačního okna a tu upravil tak, aby se v src iframu místo „IFRAME_BASE_PATH“ (tj. nastavující konstanty obsahující adresu složky, ve které byl standardně uložen obsah okna) nacházelo „about:blank“. Tím jsem zajistil vytvoření prázdné stránky přímo v iframu, bez použití jakýchkoliv externích souborů.

Dále bylo nutné zajistit integraci stylů vzhledu. Soubor s externími CSS však nebylo možné jednoduše nahrát mezi další styly TiddlyWiki – zde je nutné CSS aplikovat přímo na iframe editačního okna. Problém jsem vyřešil následujícím způsobem: Nejprve jsem vytvořil tiddler StyleSheetWYMiframe, do kterého jsem uložil veškeré styly z externího souboru použité pro iframe. Nutné obrázky (tj. několik malých ikon používaných při editaci) jsem začlenil do stylů po převedení na DataURI.

Nyní zbývalo jen styly přiřadit k danému iframu. Využil jsem k tomu následující jQuery selektor:

```
jQuery('div[tiddler='+title+'] iframe')
  .contents()
  .find("head");
```

kterým jsem přesně zacílil na hlavičku HTML stránky daného iframu. Poté jsem již jednoduše připojil styly z mého tiddleru StyleSheetWYMiframe pomocí funkce getTiddlerText():

```
.append("<style>"
  + store.getTiddlerText("StyleSheetWYMiframe")
  + "</style>");
```

Aby se výše uvedený kód provedl ve chvíli, kdy je již WYMeditor spolu s iframem kompletně načten, provedl jsem jeho zavolání opět v nastavení v postInit.

3.6 Dialog okna

Dalším krokem mé práce bylo zprovoznění dialogových oken editoru, které slouží ke vkládání odkazů, obrázků, tabulek apod. Standardně je dle šablony dialogHtml vytvořeno „vyskakovací“ okno, do kterého jsou z přiložených souborů načteny styly a jQuery knihovna. Toto řešení není příliš elegantní a je i problematické vzhledem k načítání externích souborů, kterému se potřebuji vyvarovat (zejména použití jQuery knihovny, která je již integrována v TiddlyWiki).

Obrázek 16: Náhled původního „vyskakovacího“ dialogového okna

Proto jsem se rozhodl dialog okna udělat jiným způsobem. Za ideální považuji, i vzhledem k současným trendům, dialogová okna začleněná přímo do `DOM` stránky. Pro svůj účel jsem se rozhodl použít rozšíření knihovny `jQuery` – `jQueryUI` – které nabízí nepřebernou škálu utilit pro usnadnění práce při tvorbě uživatelských rozhraní. Dalším kladem této knihovny je i možnost použití samotných dílčích částí – nejsem tedy nucen použít kompletní knihovnu i s funkcemi, které nevyužiji. Pro řešení mého problému tedy stačí použít `jQueryUI Dialog`.

Prvním krokem tedy bylo začlenění do TiddlyWiki. Ze stránek projektu [4] jsem si nechal vygenerovat na základě vlastní konfigurace jak nezbytné javascriptové knihovny, tak styly vzhledu. Integraci jsem prováděl klasickým způsobem – `CSS` jsem vložil do tiddleru `StyleSheetjQueryUI` a do `DOM` TiddlyWiki připojil nastavením ve shadow tiddleru `StyleSheet`. Javascriptový kód knihovny jsem vložil do `systemConfig` tiddleru `jQueryUIPlugin`, čímž se kód stal přístupný pro další použití.

Obrázek 17: Náhled shadow tiddleru StyleSheet

3.6.1 Dialog okna – zprovoznění

Veškeré úpravy týkající se dialogových oken jsem se pro přehlednost rozhodl provádět v `systemConfig` tiddleru `WYMeditorDialog`. Aby byl mnou vytvořený kód korektně načten a nastavení změn provedeno ve správnou dobu, veškeré následující úpravy jsem prováděl v rámci funkce `WYMeditorDialog()`, kterou volám v `postInit` v nastavení WYMeditoru.

Nejprve bylo nutné zrušení stávajících „vyskakovacích“ oken a jejich nahrazení `jQuery Dialog`em. Obsluha standardních dialogů je na základě šablon prováděna ve funkci `WYMeditor.editor.prototype.dialog()`, já jsem tedy tuto funkci řešil duplicitně, abych v ní mohl provádět úpravy. Základem bylo zrušení vytváření dialogu pomocí `window.open()` a jeho nahrazení. V `DOM` jsem tedy pomocí následujícího kódu vytvořil nový element a jemu přiřadil funkci `dialog()`:

```
var dialogDiv;
jQuery(document.body).append(dialogDiv = jQuery(dialogHtml));
dialogDiv.dialog({
  width: 600,
  minHeight: 100,
  modal: true,
  title: title,
  close: function() { jQuery('.wym_dialog').remove(); }
});
```

V parametrech funkce dialog jsem ještě provedl základní nastavení funkčnosti a ovládání.

3.6.2 Dialog okna – přizpůsobení šablon

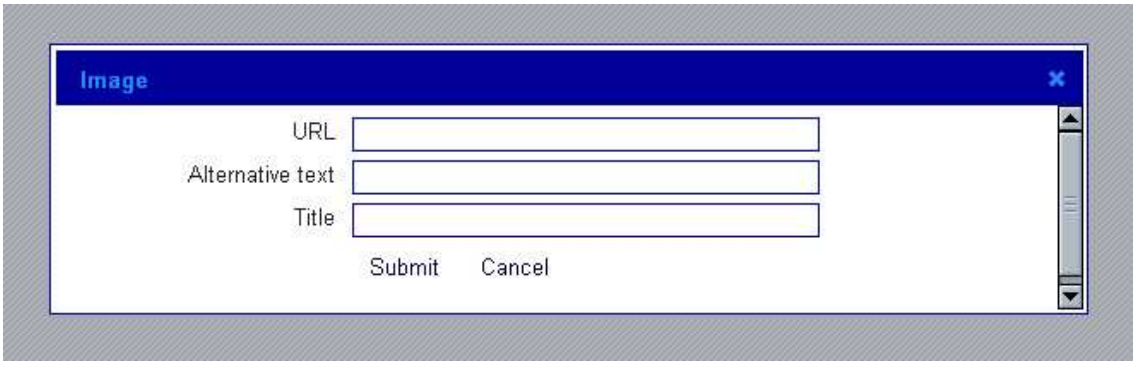
Ve chvíli, kdy již byla má dialogová okna zprovozněna, bylo nutné nastavit šablony jejich obsahu. Ty stávající v sobě totiž měly kompletní `HTML` strukturu, neboť existovaly jako samostatné stránky, což by v případě mých dialog oken ztrácelo smysl.

Úpravu šablon dialogových oken jsem opět provedl v nastavení při spouštění WYMeditoru. Nejprve jsem upravil hlavní šablonu `dialogHtml`, která v sobě obsahovala kompletní základní kostru `HTML` stránky. Dále jsem byl nucen vytvořit vlastní šablony u následujících prvků: `dialogLinkHtml`, `dialogImageHtml`, `dialogTableHtml`, `dialogPasteHtml` a `dialogPreviewHtml`.

```
dialogLinkHtml:  "<div class='wym_dialog wym_dialog_link'>"
               + "<form>"
               + "<input type='hidden' class='wym_dialog_type'
               + WYMeditor.DIALOG_LINK
               + "' />"
               + "<div class='row'>"
               + "<label>{URL}</label>"
               + "<input type='text' class='wym_href' value=''
```

Obrázek 18: Ukázka nastavení vlastní šablony pro dialogové okno

Hlavní změnou bylo výše uvedené odstranění zbytku HTML struktury (tj. ponechání obsahu samotného elementu `body`), dále jsem provedl menší změny převážně vzhledového charakteru.



Obrázek 19: Příklad zaintegrovaného dialog okna přizpůsobeného jak funkčně, tak vzhledově

3.6.3 Dialog okna – obsluha a zpracování obsahu

Nyní přišla na řadu samotná obsluha zpracování obsahu oken. Stávající dialogy byly obsluhovány funkcí `WYMeditor.INIT_DIALOG`, která byla volána z jejich těla. Já jsem se tuto funkci rozhodl začlenit do mnou již upravené funkce `WYMeditor.editor.prototype.dialog()`, která zajišťuje vytváření dialogů.

Po provázání dialog okna s jeho obsluhou jsem ještě musel vyřešit problém s jeho zavíráním. Stávající `window.close()` bylo samozřejmě nutné odstranit a nahradit. Proto jsem v mnou upravené funkci `INIT_DIALOG` musel na všech místech, kde je nutné dialog okno uzavřít, volat následující kód:

```
jQuery('.wym_dialog').remove();
```

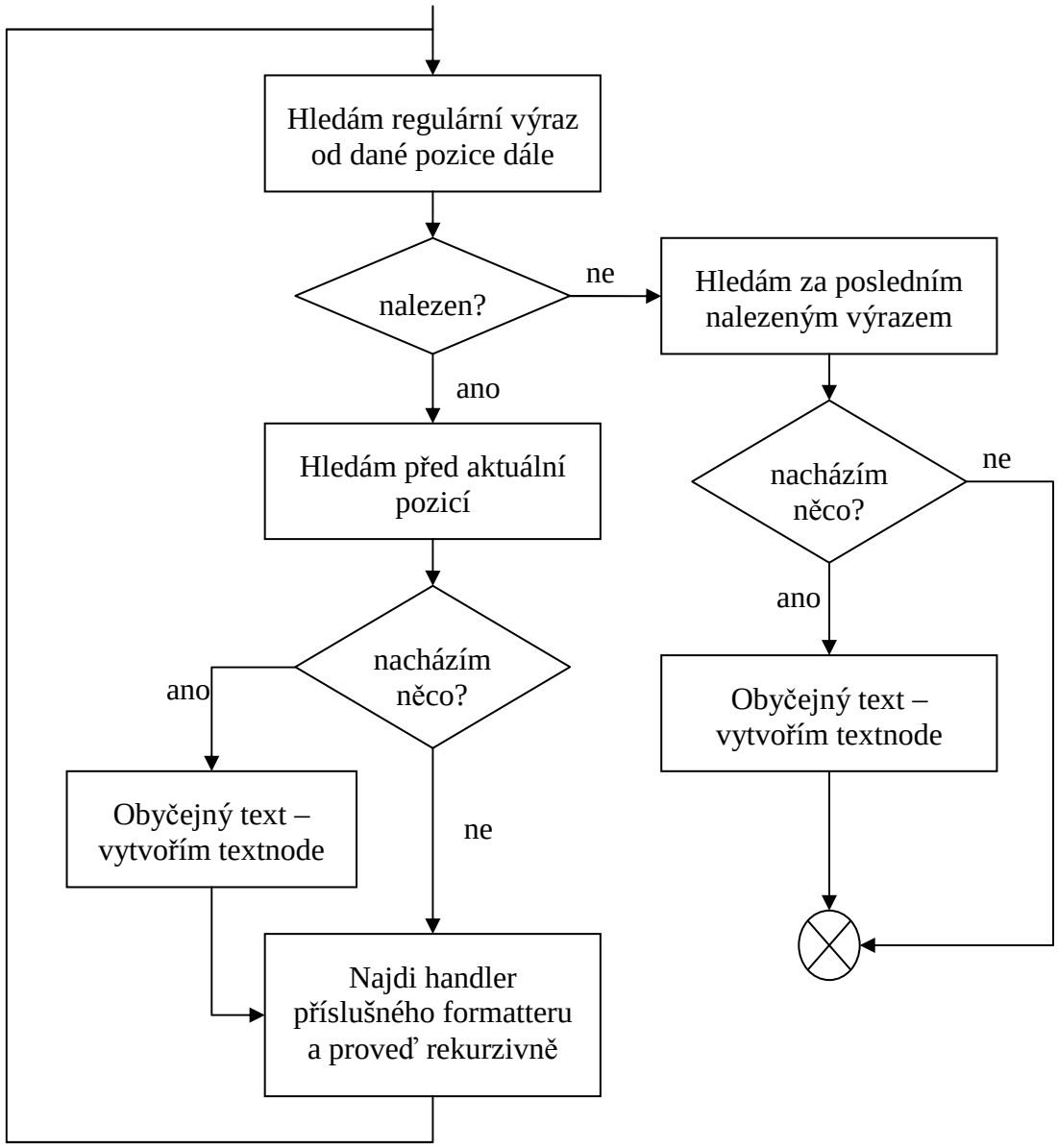
čímž jsem jednoduše odstranil dialogové okno z `DOM` stránky.

3.7 Načtení editovaného textu

Zavoláním funkce `wymeditor()` na daný objekt na stránce se jeho obsah automaticky otevře v editačním okně a text se naformátuje. WYMeditor zpracovává na vstupu `HTML` text, a tak bylo nutné zajistit převod wikitextu do `HTML`. Tento překlad samozřejmě TiddlyWiki standardně umí – je využíván při každém zobrazení tiddlerů v klasickém čtecím módu. Proto jsem tuto problematiku analyzoval hlouběji a rozhodl se tuto vestavěnou funkci využít.

Standardně jsou data tiddlerů uložena v prostém wikitextu. Při čtení je na tento obsah zavolána funkce `wikifyStatic()`, která na základě zvoleného formatteru provede překlad do `HTML`. Při klasické editaci tato funkce volána není, protože uživatel edituje pouze prostý wikitext. Při editaci WYMeditorem ovšem bude nutné tuto funkci zavolat a tím zajistit překlad do `HTML`.

Zjednodušené schéma funkce formatteru je možné prohlédnou na níže uvedeném obrázku.



Obrázek 20: Zjednodušené schéma překladu wikitextu do HTML

Spuštění funkce `wikifyStatic()` je evidentně nutné na začátku editace – v makru zajišťujícím spuštění editace. Pro tento účel jsem tedy duplikoval obsah klasického `edit` makra a jeho upravenou verzi s funkcí `wikifyStatic()` uložil zpět do `config.macros` pod názvem `wedit`. Spuštění tohoto makra při editaci WYMeditorem jsem zajistil nastavením v editační šabloně `WYMTemplate`, kterou jsem předtím již vytvořil (blíže v kapitole 3.3).

3.7.1 Úprava formatteru

Zavolání funkce `wikifyStatic()` na editovaný text zajistilo základní funkčnost editoru. Při bližším prozkoumání se však ukázalo, že překlad není zcela ideální. Proto jsem byl nucen nastudovat princip funkce formatteru a vytvořit vlastní.

V TiddlyWiki v objektu `config.formatters` se nachází překladový slovník, který na základě regulárních výrazů provádí překlad wikitextu a následnou tvorbu `DOM` stránky. Obsah slovníku je uložen v proměnné `formatter`, která je posléze předávána funkci `wikifyStatic()`. Nabízí se zde tedy řešení vytvoření vlastního formatteru a jeho následné přiřazení překladové funkci.

3.7.2 Vlastní formatter

Pro přehlednost jsem tvorbu vlastního formatteru provedl v `systemConfig` tiddleru `WYMFormatter`. Jelikož se moje úprava netýkala všech pravidel formatteru, rozhodl jsem se u nezměněných provést pouze tzv. mělkou kopii, abych zbytečně neplýtval systémovými prostředky. Tímto způsobem jsem vytvořil pole jednotlivých pravidel, z něhož jsem na konci vestavěnou funkcí `Formatter()` vytvořil objekt, který lze již jednoduše postoupit funkci `wikifyStatic()` a zajistit tak překlad textu podle vlastního předpisu.

První důvod, kvůli kterému jsem byl nucen vytvářet vlastní formatter, byl překlad maker volaných z tiddlerů. Standardně bylo totiž makro zavoláno a provedeno, což je při editaci nežádoucí. Problém jsem vyřešil upravením formatteru tak, aby bylo zachycené makro zabaleno do `HTML` objektu `span` s třídou `macro`. Tím jsem volání makra zpřístupnil k editaci. Při následném překladu zpět do wikitextu při ukládání (podrobněji v kapitole 3.8) jsem pak samozřejmě musel zajistit zpětnou konverzi.

3.7.3 Problém s překladem odstavců

Vlastní kapitolu by si zasloužilo popsání problematiky překladu odstavců. Zde bylo totiž nutné zkombinovat odlišné vlastnosti členění textů TiddlyWiki a WYMeditoru. Proto se pokusím tuto oblast popsat podrobněji.

V TiddlyWiki bychom symbol pro odstavec těžko hledali. Situace je zde velmi zjednodušena použitím prostého zalamování řádků – ve wikitextu tedy co enter, to nový řádek. Následný překlad do `HTML` je proveden jednoduše nahrazením konců řádků (znak `\n`) `BR` značkou. Toto řešení je sice jednoduché, ale pro můj účel nepoužitelné, nehledě na nevaliditu výsledného kódu.

WYMeditor oproti tomu na validitu kódu dle standardů `W3C` dbá. Proto na svém vstupu očekává veškeré texty uzavřené ve značce pro odstavec (`P`). Z tohoto důvodu jsem byl nucen vymyslet nové pravidlo formatteru pro odstavce takové, aby výstup překladu z wiki do `HTML` byl validní a splňoval svůj účel. Mým cílem tedy bylo dosáhnout „obalení“ veškerých prostých textů do elementu `P`.

3.7.3.1 Varianty řešení problému

První idea byla editovat stávající pravidlo pro překlad konců řádků. To je velmi jednoduché – funguje na principu postupného nahrazení značky `\n` nepárovou `HTML` značkou `<br />`. Ona „nepárovost“ však činí tento princip nepoužitelný – `P` značka je párová, proto je nutné detekovat při překladu (viz Obrázek 20) jak počáteční pozici (pro `<p>`), tak koncovou pozici (pro `</p>`). To je však neproveditelné, protože počátek řádku není nijak označen a využití symbolu `^` (v regulárním výrazu označujícím začátek řetězce/řádku) je vzhledem k principu funkce formatteru nemožné.

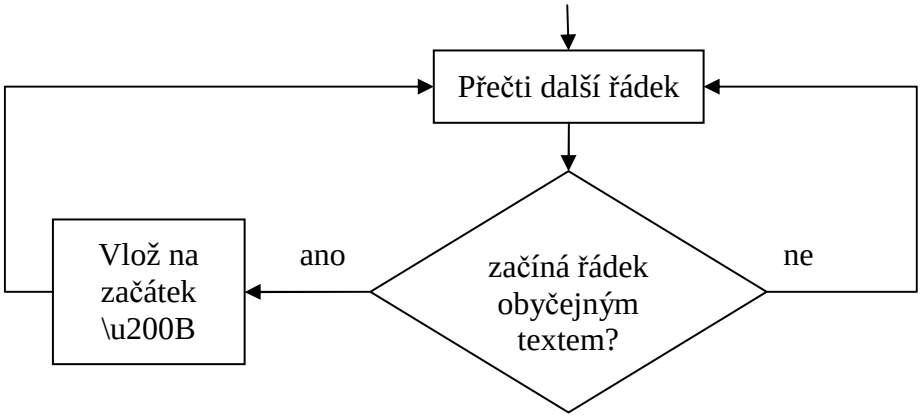
Ve druhé variantě jsem uvažoval provádět veškeré úpravy až po přeložení do `HTML` objektu. Idea by spočívala v obalení jednotlivých textnodů značkami `P`. Na první pohled funkční řešení však v sobě skrývá problémy. Textnody nelze slepě nahradit v celém kódu - značkami `P`. V tomto případě by pak byly „obaleny“ i obyčejné textové uzly umístěné např. v tabulce nebo uzly zanořené hlouběji v kódu již hotového odstavce. Další problém by byl s překladem textových řádků, na jejichž začátku by se nacházela nějaká stylizační značka jako `STRONG`, `EM`, `U`, `STRIKE` apod. – „obalen“ by byl až text uvnitř těchto řádků. Toto řešení se mi tedy, na základě výše uvedených poznatků, jeví jako problematické, i když by jistě po ošetření veškerých problémů bylo funkční.

Nakonec jsem se rozhodl do vstupního wikitextu přidat speciální značky pro odstavec tak, aby byl použitelný princip klasického formatteru popsaný výše. Toto řešení se ukázalo jako nejschůdnější, a proto jsem problém vyřešil tímto způsobem.

3.7.3.2 Řešení přidáním speciálních značek

Aby byl princip vestavěného formatteru pro správný překlad odstavců použitelný, bylo nutné na počátek každého zvoleného řádku vložit speciální značku, kterou by posléze formatter detekoval a zpracoval. Pro tento účel jsem zvolil unicode znak „zero width space“ - `\u200B` - představující mezeru nulové šířky. V případě, že by formatter tento znak někde ponechal, nebude to na závadu, protože ho uživatel neuvidí. Při následném ukládání jsou i tyto případné zbylé neviditelné znaky odstraněny, takže ve vlastních uložených datech nevznikne žádná skrytá nekonzistence.

Dalším krokem bylo vytvoření regulárního výrazu, který by značku na správné místo umístil. Zde bylo nutné odfiltrovat takové řádky, které začínají značkami pro nadpis, seznam, citaci apod., neboť by se tím jejich funkce deaktivovala.



Obrázek 21: Princip přidávání speciálních značek do textu

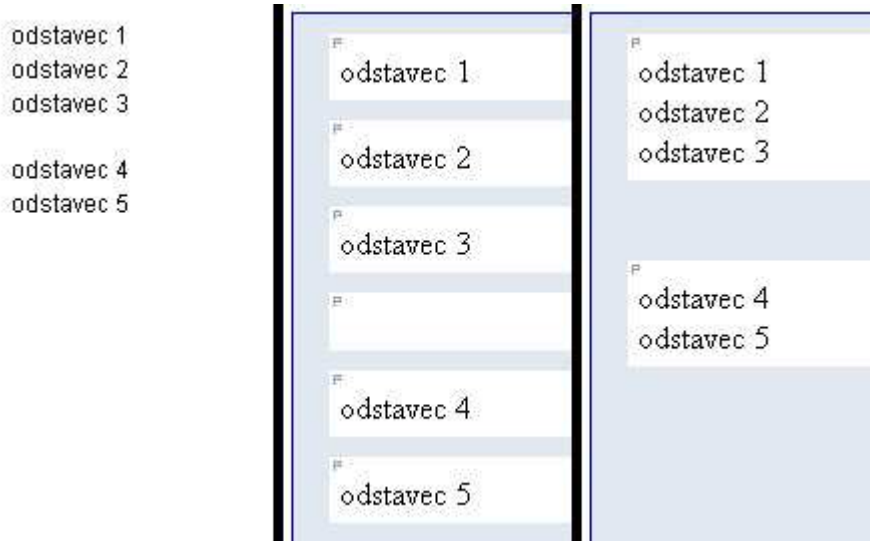
Nakonec bylo nutné rozšířit slovník formatteru o novou značku – odstavec `P`. Tento jsem nastavil tak, aby se aktivoval při nalezení značky `\u200B` a veškerý následující text až do konce řádku (`/n`) vložil do elementu `P`. Současně bylo nutné deaktivovat tvorbu elementu `BR`.

3.7.3.3 Varianty překladu

Pomocí výše uvedených úprav jsem docílil vložení každého textového řádku do elementu `P` a zároveň naprostou deaktivaci tagů `BR`. Při testování jsem však byl na

pochybách, zda je toto řešení z pohledu uživatele vhodné. Jako první jsem se pozastavil nad překladem prázdných řádků. Ve výsledku jsou zobrazeny jako prázdné odstavce. Fakticky je toto řešení správné, ale pro uživatele může být matoucí nemožnost oddělení dvou odstavců jinak než takto. Rozhodl jsem se tedy, že prázdné odstavce nahradím tagy BR, čímž provedu oddělení sousedních elementů. Domnívám se, že z uživatelského hlediska je to rozumné řešení.

Dále jsem uvažoval nad slučováním vedle sebe ležících odstavců. Zamyslel jsem se nad otázkou, zda je pro uživatele příjemnější mít jednotlivé odstavce (tj. i osamocené řádky) v elementech P zvlášť, či spojené dohromady a oddělené tagem BR. Druhá volba sice není v souladu s webovými standardy, avšak zdá se mi pro daný účel vhodnější, proto jsem ji implementoval.



Obrázek 22: Porovnání odlišného zpracování odstavců v různých editačních módech

Na výše uvedeném obrázku (Obrázek 22) lze lehce porovnat zmiňované editační módy. V prvním sloupci se nachází text ve wikiformátu. V druhém sloupci je tento text otevřený ve WYMeditoru se základním překladem. Ve třetím sloupci je ten samý text zobrazený po výše provedených úpravách. Je zde patrné jak nahrazení prázdného odstavce BR tagem, tak sloučení vedle sebe ležících odstavců.

Obojí výše uvedené problémy nejsou funkčně podstatné – týkají se jen zobrazení dat uživateli při editaci WYMeditorem, následné uložení do wikitextu je v obou případech ve výsledku stejné. Z tohoto důvodu jsem se rozhodl nechat v tomto směru uživateli volnost a obě možnosti nabídnout na výběr v nastavení, které popíšu dále.

3.7.3.4 Nastavení v options

Veškeré uživatelské nastavení TiddlyWiki lze nalézt v hlavní nabídce pod položkou `options > AdvancedOptions`, alternativně též v backstage aplikace pod položkou `tweak`. Veškeré zde provedené úpravy se ukládají do `cookies` prohlížeče a nadále tak ovlivňují chod TiddlyWiki na daném počítači. Já jsem se rozhodl paletu nastavení rozšířit o další dvě položky týkající se výše uvedeného překladu odstavců.

Definici obsahu nabídky `AdvancedOptions` jsem pro bližším prozkoumání našel uloženou v objektech `config.options` a `config.optionsDesc`.

V prvním zmiňovaném je prováděna inicializace proměnných a jejich defaultních hodnot, v druhém se nachází popisky daných voleb. Já jsem tedy tyto objekty jednoduše rozšířil o své proměnné následujícím kódem:

```
merge(config.options,{
  chkMergeP: true,
  chkAllowBr: true
});

merge(config.optionsDesc,{
  chkMergeP: "Slučovat sousedící odstavce (P) při editaci
WYMeditorem",
  chkAllowBr: "Povolit nahrazování prázdných odstavců tagy BR
při editaci WYMeditorem"
});
```

Jména proměnných jsou volena s ohledem na používaný princip – dle jejich prexifu (chk či txt) TiddlyWiki pozná, zda se jedná o proměnnou s boolean hodnotou (v nastavení vznikne zaškrtávací tlačítko – checkbox) či textovou/číselnou hodnotou (vznikne pole – input). V tomto případě jsem tedy obě proměnné volil s prefixem chk. Jak mnou upravené nastavení aplikace vypadá je možné vidět na obrázku níže.

Obrázek 23: Náhled tiddleru AdvancedOptions s novými možnostmi nastavení

3.7.4 Nedokonalosti překladu z WIKI do HTML

Při dalším testování jsem nenarazil na podstatné nedokonalosti překladu do `HTML`. Myslím, že se mi podařilo vytvořit kvalitní a dostačující formatter. Drobná vada, kterou jsem jako jedinou objevil, se týká zobrazení `span` elementu ve WYMeditoru. Následující wikikód sloužící k definování inline stylů nedokáže WYMeditor správně analyzovat:

```
@@CssProperty:value;text@@
```

Správný, formatterem vytvořený překlad vypadá takto:

```
<span style="CssProperty:value;">text</span>
```

WYMeditor tento `HTML` kód načte a zobrazí následujícím způsobem:

```
text</span>
```

Startovací tag `span` zde zcela chybí, i když ho editor v grafickém editačním módu zobrazí. Z výše uvedeného je patrné, že se jedná o chybu ve WYMeditoru, neboť mnou předávaný kód je plně validní. Dá se očekávat, že bude tento nedostatek v budoucích

vývojových verzích WYMeditoru odstraněn. Mé aplikaci by tato menší nedokonalost neměla uškodit, takže jsem se jí dále nezabýval.

3.8 Ukládání editovaného textu

Po dokončení editace bylo taktéž nutné vyřešit vlastní ukládání textu a jeho zpětný překlad z HTML do wiki, aby mohl být posléze standardně zobrazován. Nejprve bylo nutné vytvořit vlastní tlačítko, které by celý proces ukládání spustilo. To jsem provedl klasickým způsobem – rozšířením stávajícího TiddlyWiki objektu config.commands o položku saveWYMTiddler. Zde jsem nejen nastavil tlačítko, ale i jeho obsluhu a další funkce nutné k překladu HTML do wiki, kterým se budu věnovat dále.

V prvé řadě bylo nutné přenést vlastní upravený text z editačního iframu WYMeditoru zpět do DOM stránky, aby mohl být následně zpracován a uložen. Za tímto účelem bylo nutné nalézt původní WYMeditorem nahrazený objekt textarea a na něm provést update(). K tomu účelu jsem vytvořil následující kód využívající opět jQuery, abych přesně zacílil danou instanci WYMeditoru.

```
var area = document.getElementById("tiddler"+title)
  .getElementsByTagName("textarea")[0];
jQuery.wymeditors(jQuery.data(area, WYMeditor.WYM_INDEX))
  .update();
```

3.8.1 Převod HTML textu do wiki

Nyní již bylo nutné se zabývat konverzí HTML do wikitextu. Nejprve jsem se pokoušel nalézt již hotové řešení, které by mojí práci usnadnilo a zefektivnilo. Jak se ukázalo, podobný problém řešili autoři RichTextu [9], plug-inu do TiddlyWiki, který zajišťuje základní začlenění poměrně složitého javascriptového WYSIWYG editoru TinyMCE. Rozhodl jsem se inspirovat jejich řešením překladu HTML do wiki a použil jej jako kostru mého algoritmu.

Základem překladače je funkce HTMLNodeToWiki rekurzivně spuštěná na obsah editovaného textu, který je ve formě stromu složeného z jednotlivých uzlů – HTML značek. HTMLNodeToWiki použitá v RichTextPluginu zajišťovala překlad základních formátovacích značek jako jako strong, em a u, nadpisů h1-h6, seznamů ol a ul včetně kaskády jednotlivých listů. Překlad složitějších značek implementován nebyl. Proto jsem funkci přetvořil a poměrně široce rozšířil, aby byl překlad co nejúplnější.

Nejprve jsem provedl úpravu objektu matchingRules, který v sobě obsahoval „slovník“ překládaných značek. Každé zde uvedené položce jsem přiřadil nově vytvořenou funkci - parser – který podle dané HTML značky provedl vlastní překlad.

```
var saveWYMTiddler = config.commands.saveWYMTiddler = {};
merge(saveWYMTiddler, {
  parsers = { //vlastní zpracování konkrétní HTML značky
    akce: function(start, end) {
      return function(myNode) {
        return start + this.HTMLNodeToWiki(myNode) + end;
      }; //zajistí obalení a rekurzivní zpracování vnitřku
    }
  },
  matchingRules: { //zavolá na HTML značku příslušnou funkci
    značka : parsers.akce("start značka", "end značka");
    ...
  },
  HTMLNodeToWiki: function(myNode) {
    var res;
    if(myNode.nodeType == "text")
      return myNode.text; //list stromu obsahující text
    switch(myNode.nodeName) {
      //v případě uzlů OL, UL a LI zajišťuje zanoření do kaskády
      case "ul" : zpracujUl(); break;
      case "ol" : zpracujOl(); break;
      case "li" : zpracujLi();
    }
    for(i in myNode.childNodes) //pro každý childNode
      res += this.matchingRules[i.nodeName];
    return res; //vrátí přeložený obsah
  }
}
//zavolání překladové funkce
saveWYMTiddler.HTMLNodeToWiki(tiddler.text);
```

Zjednodušený pseudokód funkce zajišťující překlad HTML značek na wikiznačky

3.8.2 Překlad jednoduchých HTML značek

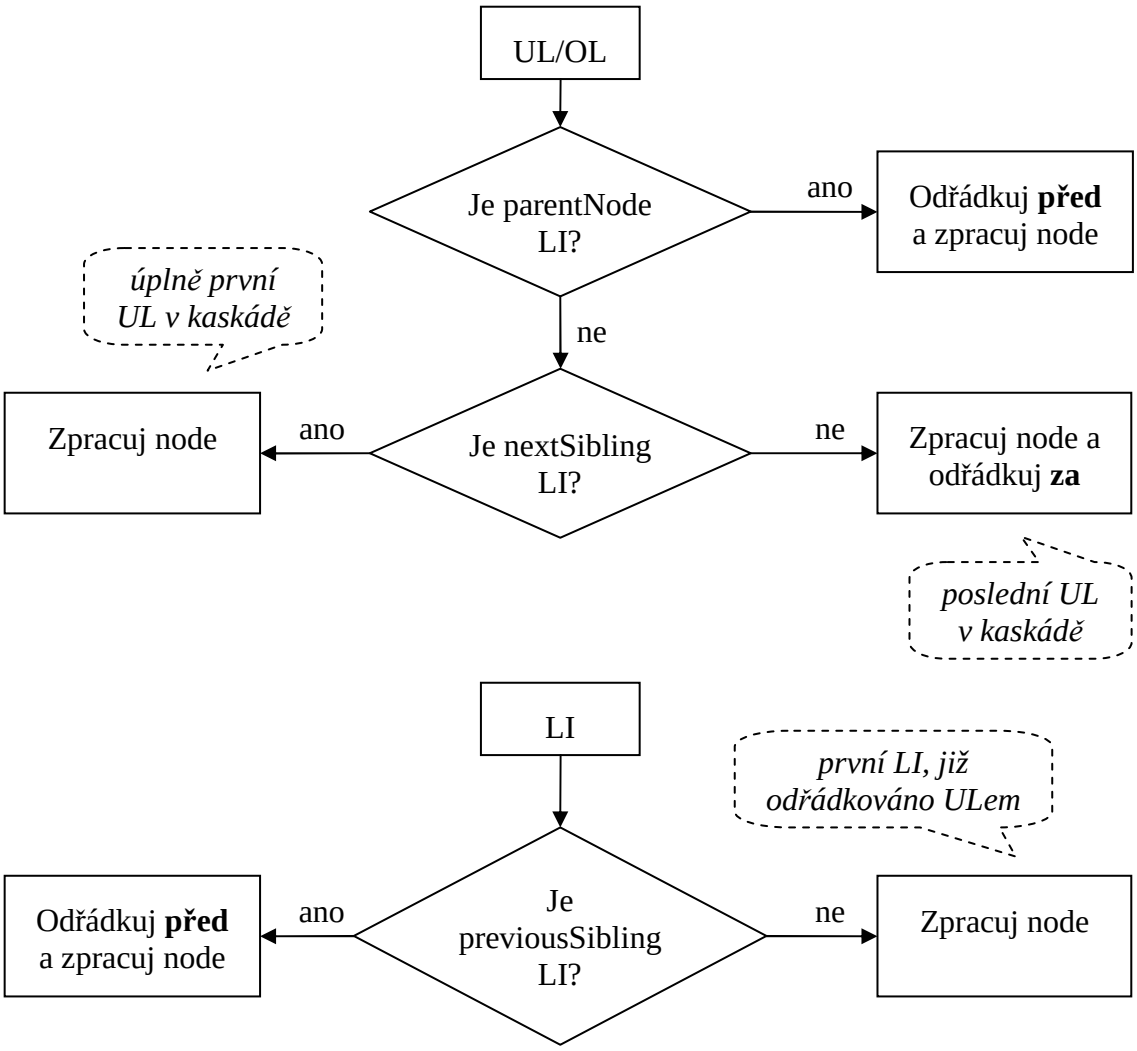
Pro jednoduché značky typu `strong`, `em` apod. jsem vytvořil univerzální parsery, které zajišťují nahrazení daného `HTML` elementu wiki značkami. Nejjednodušším je parser `simple()`, který na základě předané proměnné nahradí daný element a obsah „obalí“ stejnou wiki značkou. Tuto funkci jsem využil u `HTML` značek `strong`, `em`, `u` , `strike`, `sup`, `sub`.

O málo složitější funkce `both()` přijímá parametry dva – po nahrazení `HTML` značky totiž obsah „obalí“ dvěma různými značkami – jednou na začátek a druhou na konec řetězce. Taková funkce se využila při překladu tagů `p`, `br`, `h1-h6`, `code`, `dd`, `dt`, `blocquote` a `pre`.

3.8.3 Překlad složitějších značek

Pro složitější značky jsem musel vytvořit speciální parsery, které dané wiki značky, na základě pokročilejší logiky, vytvoří. Mezi ty jednodušší patří parsery pro odkazy a obrázky. Ty při překladu do výsledného wiki tagu ukládají, kromě adresy odkazu či obrázku, také např. popis. Parsery pro `span` a `div` zase zajišťují zachování třídy (`class`).

Složitější a zajímavější překlad bylo nutné vymyslet u seznamů - jak číslovaných, tak nečíslovaných. Základní logiku zanořování do kaskády `li` tagů jsem převzal od RichTextu. Pro vlastní překlad jsem pak nejprve použil parser `both()`, ovšem při testování se jeho funkčnost ukázala jako nedostatečná. V případě seznamů je totiž bezpodmínečně nutné dodržet zalamování řádků - znak pro odrážku se musí bezpodmínečně nacházet na začátku řádku, jinak se následný překlad z wiki do HTML provede špatně. Proto jsem musel problém prozkoumat hlouběji a navrhnout parsery tak, aby na základě zanořování do kaskády `li` tagů přidávaly značku konce řádku či nikoli. Princip je zjednodušeně zobrazen na následujícím obrázku.



Obrázek 24: Zjednodušené schéma překladu seznamů z HTML do wiki

Nejproblematičtější bylo zajistit správný překlad tabulek. Jejich struktura je totiž v HTML složitá - skládá se z mnoha různých navzájem vnořených tagů. Proti tomu jejich popis ve wikitextu je na první pohled jednoduchý, ovšem při bližším prozkoumání zjistíme, že podporuje různé složité nastavení. Mým cílem samozřejmě bylo zachovat maximální funkčnost a provést zcela dokonalý překlad.

```
<table class="tridaTabulky">
  <thead>
    <tr class="evenRow">
      <th>hlav1</th>
      <th>hlav2</th>
      <th>hlav3</th>
    </tr>
  </thead>
  <tbody>
    <tr class="oddRow">
      <td>bunka1</td>
      <td>bunka2</td>
      <td rowspan="2">bunka3</td>
    </tr>
    <tr class="evenRow">
      <td colspan="2">bunka4</td>
    </tr>
  </tbody>
  <tfoot>
    <tr class="oddRow">
      <td>pata1</td>
      <td>pata2</td>
      <td>pata3</td>
    </tr>
  </tfoot>
</table>
```

```
|tridaTabulky|k
|!hlav1|!hlav2|!hlav3|h
|bunka1|bunka2|bunka3|
|>|bunka4|~|
|pata1|pata2|pata3|f
```

Tabulka 1: Příklad vyjádření stejné tabulky pomocí HTML a wiki kódu.

Velký problém mi přineslo například slučování buněk v řádcích a ve sloupcích (colspan a rowspan). V syntaxi TiddlyWiki totiž musí být v kódu uvedené všechny buňky tabulky – a to i v případě, že díky sloučení fakticky neexistují - jen musí obsahovat speciální symbol. Oproti tomu v HTML jsou sloučené buňky fyzicky vyjádřeny pouze jednou – s příslušným atributem rowspan či colspan. Příklad tohoto problému je vidět například ve výše uvedené tabulce.

3.8.4 Překlad zbylých značek

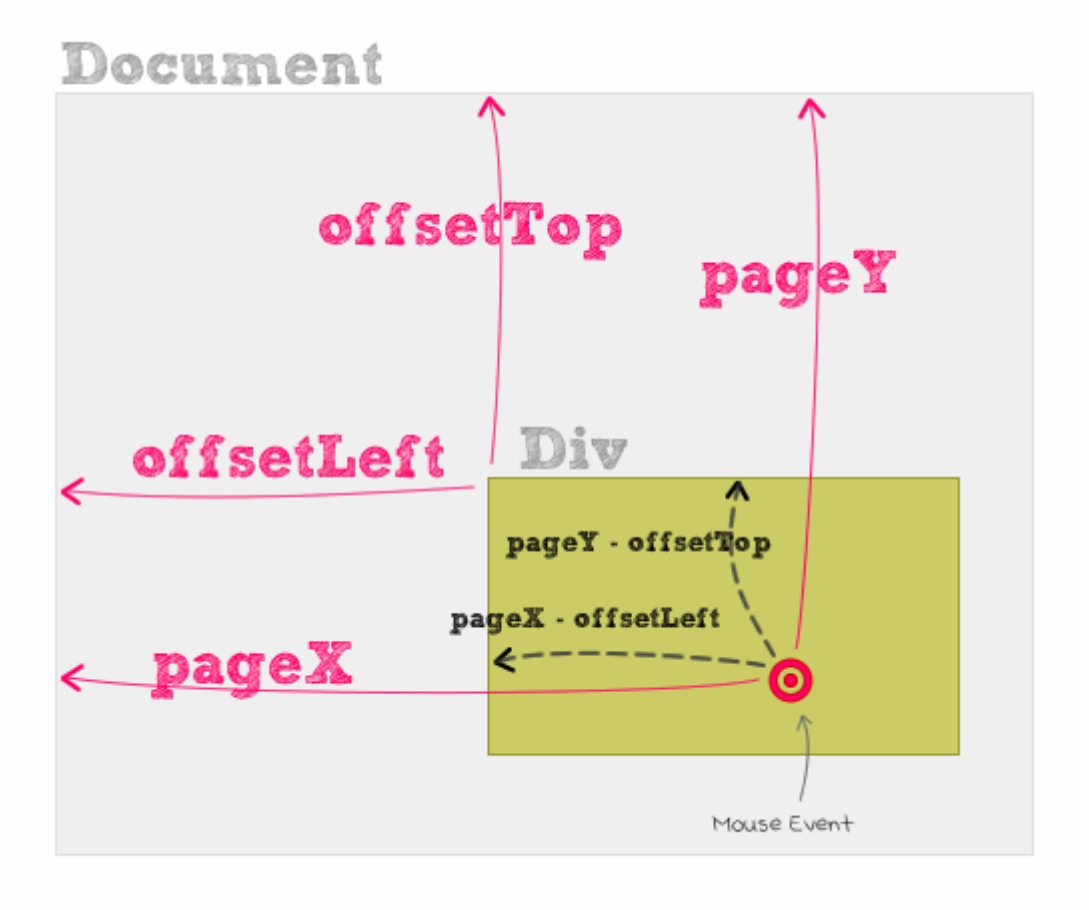
Pokud se v překládaném textu najde značka, kterou můj slovník nezná, znamená to, s největší pravděpodobností, že ani TiddlyWiki danou značku nepodporuje. Z tohoto důvodu jsou tyto symboly bez náhrady odstraněny a dále je zpracováván pouze jimi ohraničený obsah. Každopádně je úprava pravidel překladu poměrně snadná záležitost, takže případné další úpravy a rozšíření by neměly představovat větší problém.

3.9 Zachování pozice scrollbaru v editovaném textu

V TiddlyWiki lze standardně otevřít editační režim dvojklikem na daný tiddler. Při této činnosti však ztratíme polohu, na které jsme se předtím nacházeli, což může být při úpravě rozsáhlejších tiddlerů obtěžující. Proto jsem se rozhodl upravit editační režim tak, abych dvojklikem označil polohu, na kterou chci v editačním okně nascrollovat.

3.9.1 Teoretický rozbor

Prvním krokem bude pravděpodobně zachycení pozice dvojkliku. Tato pozice bude muset být absolutní vzhledem k danému tiddleru – přesněji `DIV` elementu, ve kterém je tiddler vložen a zobrazen. Zjištění souřadnic lze provést pomocí rozšíření `jQuery.scrollTo`. Na následujícím obrázku, pocházejícího ze stránek Bena Nadela [13], je zobrazen použitelný princip. V mém případě bude nutné vypočítat hodnotu `pageY – offsetTop`.



Obrázek 25: Zjištění relativní pozice v dokumentu

3.9.2 Řešení problému

V první fázi jsem `jQuery` v TiddlyWiki obohatil o plugin `ScrollTo` [10], který mi umožní na vypočtenou hodnotu na stránce nascrollovat. Kód tohoto rozšíření jsem uložil do tiddleru `jQueryUIPlugin`. Dále bylo zapotřebí zachytit absolutní pozici kliku, což jsem provedl v tiddleru WYMeditor následujícím kódem:

```
var poziceKliku;
jQuery(document).ready(function(){
  jQuery(document).mousedown(function(e){
    poziceKliku = e.pageY;
  });
})
```

Nyní už jen zbývalo vypočítat relativní pozici vzhledem k DIV elementu tiddleru. Hodnotu `offsetTop` (viz Obrázek 25) jsem získal pomocí `jQuery` selektoru

```
jQuery('div[tiddler='+title+']').offset().top;
```

a tuto hodnotu odečetl od výše získané hodnoty `pageY`. Následovalo pomocí `jQuery` zacílit na iframe editačního okna a pomocí funkce `ScrollTo` nascrollovat na vypočtenou pozici:

```
jQuery('div[tiddler='+title+'] iframe')
  .scrollTo(pozice,0,{axis: 'y'});
```

Aby funkce probíhala ve správnou chvíli, vložil jsem její kód do `postInit` nastavení WYMeditoru a navíc podmínil přítomností události „`dblclick`“. Funkce, vzhledem k rozdílnému formátování prohlíženého a editovaného textu, scrolluje pouze na přibližnou. I tak se ale domnívám, že v této jednoduché formě usnadní editaci rozsáhlejších textů.

3.10 Prostor pro další úpravy

V této fázi je již WYMeditor takřka plnohodnotnou součástí TiddlyWiki. Po důkladném otestování si nejsem vědom větších nedostatků. Každopádně se zde nachází prostor na další vylepšení a úpravy. Jednou z nich může být například vytvoření předdefinovaných stylů vzhledu, které by se pak daly použít při editaci textu podobně jako styly a formátování v Microsoft Word. To už by ale bylo nad rámec mé práce a proto to nechám na vývojářích, kteří budou na toto dílo navazovat.

4 Závěr

Prvním cílem této práce bylo vytvoření přehledové dokumentace Mozilla Application Frameworku v rozhraní TiddlyWiki. Domnívám se, že jsem zde vytvořil solidní základ pro vývojáře zabývající se tvorbou v tomto frameworku. Přehledný seznam interfaců, s funkčními odkazy na jejich dokumentaci, doufám, ušetří mnoho času při hledání informací, nehledě na možnost budoucího upravení aktualizačního skriptu a tím i zvětšení rozsahu extrahovaných dat z dokumentačních serverů.

Databázi jsem navrhl tak, aby bylo možné pohodlně dopisovat k jednotlivým interfacům další informace. Tímto způsobem lze tedy souhrn dokumentace manuálně rozšířit. Abych i toto zpříjemnil, rozhodl jsem se v druhé části této práce věnovat kompletnímu zaintegrování WYMeditoru do rozhraní TiddlyWiki. Tato úprava myslím potěší nejen programátory pracující s mou dokumentací, ale uplatnění najde především mezi širokým spektrem obyčejných uživatelů, kteří se nehodlají zabývat učením značek wikikódu.

V úplném závěru bych chtěl dodat, že jsem přesvědčený o přínosu mé studie pro vývojáře i pro řadové uživatele, kterým by měla usnadnit práci. Doufám, že moje náměty nepřijdou nazmar a budou základem pro další nástavby a rozšíření.

Literatura

[1] AJAX – Wikipedia [online]. Dostupné na WWW: <http://cs.wikipedia.org/wiki/AJAX>

[2] Dev:Core Code Overview - TiddlyWiki.org [online]. Dostupné na WWW: <http://tiddlywiki.org/wiki/Dev:Core_Code_Overview>.

[3] jQuery JavaScript Library [online]. Dostupné na WWW: <http://docs.jquery.com/Main_Page>.

[4] jQuery UI – Demos & Documentation [online]. Dostupné na WWW: <http://jqueryui.com/demos>.

[5] Mozilla Application Framework in Detail [online]. Dostupné na WWW: <https://developer.mozilla.org/en/Mozilla_Application_Framework_in_Detail>.

[6] Mozilla Cross-Reference [online]. Dostupné na WWW: <http://mxr.mozilla.org/>.

[7] Mozilla Developer Network - Mozilla Developer Center [online]. Dostupné na WWW: <https://developer.mozilla.org/En>.

[8] Mozilla Documentation [online]. Dostupné na WWW: <http://doxygen.db48x.net/mozilla/html/>.

[9] RichTextPlugin [online]. Dostupné na WWW: <http://tiddlywiki.ouvaton.org/Demo.html>.

[10] ScrollTo – jQuery Plugin [online]. Dostupné na WWW: <http://plugins.jquery.com/project/ScrollTo>

[11] TiddlyWiki [online]. Dostupné na WWW: <http://www.tiddlywiki.com/>.

[12] TiddlyWiki Markup [online]. Dostupné na WWW: <http://tiddlywiki.org/wiki/TiddlyWiki_Markup>

[13] Translating Global jQuery Event Coordinates To A Local Context [online]. Dostupné na WWW: <http://www.bennadel.com/blog/1871-Translating-Global-jQuery-Event-Coordinates-To-A-Local-Context.htm>

[14] WYMeditor [online]. Dostupné na WWW: <http://trac.wymeditor.org/trac>.

[15] WYMeditor: web-based XHTML editor [online]. Dostupné na WWW: <http://www.wymeditor.org/>

Příloha A

Seznam použitých zkratek

AJAX - Asynchronous JavaScript and XML

CSS - Cascading Style Sheets

CVS - Concurrent Version System

DATA URI - Data Uniform Resource Identifier

DOM - Document Object Model

W3C - World Wide Web Consortium

WYSIWYG - What You See Is What You Get

WYSIWYM - What You See Is What You Mean

(X)HTML - (Extensible) HyperText Markup Language

XML - Extensible Markup Language

XUL - XML User Interface Language

Příloha B

Přehled editovaných tiddlerů

B.1 Nově vytvořené systemConfig tiddlery

jQueryUIPlugin	kód javascriptové knihovny jQuery UI + plugin ScrollTo
WYMFormatter	vytvoření nového formatteru pro překlad z wiki do HTML
WYMeditor	tlačítka edit a save, inicializace WYMeditoru s počátečním nastavením, převod z HTML do wiki
WYMeditorDialog	inicializace funkce jQuery UI dialogového okna
WYMeditorPlugin	načtení vlastního kódu WYMeditoru nastavení jazyka a vzhledu
InterfaceGenerator	automatické vygenerování tidlerů s interfacy
InterfaceIndex	tvorba stromu odkazů na interfacy dle hierarchie
InterfaceSublist	makro vypisující odkazy na tiddlery dle společného tagu
InterfaceUpdate	funkce zajišťující vlastní aktualizaci tiddlerů
InterfaceUpdateStory	funkce zajišťující volání aktualizační funkce na právě otevřené tiddlery
InterfaceUpdateAll	funkce zajišťující volání aktualizační funkce na všechny tiddlery

B.2 Další vytvořené tiddlery

StyleSheetjQueryUI	styly vzhledu dialogových oken jQuery UI
StyleSheetWYMeditor	styly vzhledu WYMeditoru
StyleSheetWYMiframe	styly vzhledu editačního iframe
WYMTemplate	šablona editačního režimu WYMeditoru

B.3 Editované shadow tiddlery

StyleSheet	přidané definice nově začleněných stylů
ToolbarCommands	přidána definice nového toolbaru WYMToolbar
MainMenu	vložený zástupný kód pro zobrazení loading animace
SidebarOptions	přidána tlačítka pro aktualizaci a generování tiddlerů

Příloha C

Obsah přiloženého CD

```
.
|-- program
|   |-- empty.html – prázdná TiddlyWiki bez generovaných dat
|   |-- index.html – vlastní soubor aplikace TiddlyWiki
|   |-- jquery.wymeditor.pack.js – soubor aplikace WYMeditor
|   \-- logo.png – upravené logo Firefox použité v TiddlyWiki
|-- text
|   |-- vybihpav_2010bach.doc – text BP ve formátu DOC
|   \-- vybihpav_2010bach.pdf – text BP ve formátu PDF
\-- readme.txt – popis obsahu CD
```

Pro možnost ukládání změn v TiddlyWiki je nutné aplikaci přenést na pevný disk včetně doprovodných souborů!