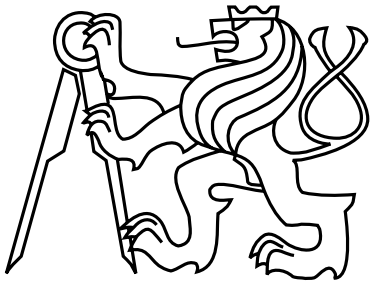
České vysoké učení technické
Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

# Vylepšení editoru v Mozilla Thunderbirdu

*Jakub Podlaha*

Vedoucí práce: Ing. Tomáš Novotný

Studijní program: Softwarové technologie a management, Bakalářský

Studijní obor: Softwarové inženýrství

7. ledna 2011

# Poděkování

Rád bych poděkoval rodičům, sestře a přátelům za podporu při psaní mé bakalářské práce. Dále chci poděkovat vedoucímu své práce Ing. Tomáši Novotnému za trpělivost a vřelý přístup.

# 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 7. 1. 2011 .....

# Abstract

The aim of my thesis is to document current implementation of e-mail editor in Mozilla Thunerbird and create an extension, on the bases of the observations, which grants access to editor's functionality to other developers of ExtBrain project. The thesis also involves an implementation of styling toolbar enhancing formating capabilities in e-mail comunication.

# Abstrakt

Cílem mé bakalářské práce je zdokumentovat stávající implementaci editoru e-mailových zpráv v Mozilla Thunderbirdu a na základě získaných poznatků vytovřit rozšíření poskytující metody pro práci s editorem dalším vývojářům projektu ExtBrain. Součástí práce je i implementace stylovacího panelu rozšiřujícího možnosti formátování textu v e-mailové komunikaci.

# Obsah

# Seznam obrázků

# Seznam tabulek

# Kapitola 1

# Úvod

V době sociálních sítí už není Internet jen nekonečným zdrojem informací. Jedná se také o vysoce univerzální komunikační prostředek, jehož dostupnost a obliba den ode dne roste. Jedním z nejstarších a v současné době nejjedodušších internetových komunikačních prostředků po dlouhá léta zůstává e-mail. Díky záplavě bezplatných emailových schránek může mít e-mail i ten nejméně zkušený uživatel celosvětové sítě. Tyto takzvané freemailové schránky svým uživatelům poskytují webové rozhraní pro přístup k jejich e-mailové korespondenci. Webové rozhraní má vždy podobnou strukturu, pro různé poskytovatele schránek se však liší vzhledem i poskytovanou funkcionalitou. Pokročilejší uživatelé, lidé užívající více emailových schránek, nebo například ti, kteří vyřizují větší množství korespondence pak jistě rádi vymění webové rozhraní e-mailových schránek za lokální aplikaci, takzvaného e-mailového klienta, jakým je například Mozilla Thunderbird.

Uživatelé e-mailového klienta mohou od elektronické korespondence očekávat více, než jen výměnu informací v podobě prostého textu. K plnohodnotnému využití možností elektronické korespondence patří mimo jiné i různé formátování textu, které mu přidává na přehlednosti, možnostech vyjádření a zvyšuje jeho osobitost. Proto je má bakalářská práce zaměřena na analýzu a rozšíření možností formátování textu v editoru Mozilla Thunerbirdu a zahrnuje i implemetaci panelu formátovacích nástrojů pro rozšíření jeho možností.

Tento panel poskytuje uživatelům jednoduchou podporu práce s formátováním textu přesně podle jejich potřeb. Dovolí jim ukládat jejich vlastní textové styly, či zobrazit seznam stylů použitých v právě editované e-mailové zprávě a tyto dále využívat. Tímto rozšířením se snažím zjednodušit uživatelům práci se stylováním textu, obohatit možnosti e-mailové komunikace a zvýšit tak její oblibu.

Implementace tohoto rozšíření slouží i jako příklad vývojářům projektu ExtBrain pro tvorbu dalších funkcionalit ExtBrain Editoru a dále pro jeho napojení na stávající editační komponenty. Při návrhu a vývoji rozšíření je myšleno zejména na jednoduchost, přehlenost a rozšiřitelnost. Při návrhu stylovacího panelu se pak soustředím hlavně na uživatele a důraz kladu na přehlednost, přístupnost a snadné použití.

V následující kapitole se pokouším přesněji vystihnout, čeho se v této práci snažím dosáhnout. Definuji zde konkrétnější cíle práce a shrnu je zde do podoby formálních požadavků.

Ve třetí kapitole se věnuji prostředí na kterém svou práci stavím. Od obecnějších informací o aplikaci a platformě se dostávám až k popisu jednotlivých použitých technologií. Již v této

kapitole se snažím prokládat obecný text konkrétními příklady a přibližovat tak praktický pohled na vývoj na platformě Mozilla.

Čtvrtá kapitola je rozdělena na dvě části. V první z nich se věnuji architektuře editoru, popisuji jeho stavbu a propojení jeho částí. V druhé pak sleduji průběh volání příkazu na editoru. Tato část je oproti předchozím velice technicky zaměřená a má sloužit hlavně navazujícím vývojářům.

Pátá kapitola je věnována implementovanému rozšíření pro Mozilla Thunderbird. Kromě struktury samotného rozšíření v ní popisuji již jen užší části vývoje, které mohou obohatit vývojáře Mozilla aplikací.

# Kapitola 2

# Analýza požadavků

V této kapitole jsou upřesněny konkrétní cíle, které si tato práce klade. Po deklaraci záměru popisuji situaci, z jaké ve své práci vycházím, a výsledek, jakého bych rád dosáhl. Závěry jsou pak shrnuty do konkrétní podoby funkčních a nefunkčních požadavků zaměřených na implementační část práce.

## 2.1 Deklarace záměru

Hlavním cílem práce je zdokumentovat editační komponentu Mozilla Thunerbirdu, získat co největší přehled o její funkcionalitě a poskytnout užitečné informace dalším vývojářům. Tento úsek by měl zahrnout i části editoru, které, například pro složitost, není vhodné pro další vývoj použít, ale u kterých je velká možnost, že se s nimi vývojáři pracující s editační komponentou dříve, či později setkají. Speciální důraz je pak kladen na podporu stylování, cesty jakými lze se styly v editoru pracovat a hledání nejefektivnější z nich. V této části je popsán i proces samotné analýzy.

Dalším záměrem práce je vytvořit funkční rozšíření pro Mozilla Thunerbird. Toto rozšíření poskytne uživateli panel s tlačítky, představující seznam stylů. Stylem se rozumí konkrétní formátování části textu, ať už slova, věty, nebo libovolného úseku, do jiné podoby než je prostý neformátovaný text. Příkladem stylu může být například podtržený, červený a zároveň zvětšený text. Uživatel bude mít možnost všechny tyto formátovací úpravy provést stiskem jediného tlačítka. Krom toho mu formátovací panel zobrazí všechny styly použité v aktuálně editovaném dokumentu a dovolí mu jeho oblíbené styly uložit, pojmenovat a v budoucnosti kdykoli použít.

Mezi podporované formátování textu patří i styly odstavců jako jsou nadpisy různých velikostí, prostý odstavec, či takzvaný preformát, který zobrazí všechna písmena i mezery stejně široké jako na psacím stroji.

V závislosti na zpětné vazbě od testerů a uživatelů bude rozšíření připraveno pro implementaci nových funkcionalit. Již během vývoje vyplynula na povrch potřeba nových funkcionalit například zvýrazění aktuálního stylu, či podpora dalších způsobů stylování.

Vývoj tohoto rozšíření pak bude veden tak, aby se panel stal jediným stylovacím prvkem, který uživatel bude potřebovat při běžné práci. Tímto by se mělo docílit zjednodušení grafického prostředí i usnadnění práce s editorem.

## 2.2 Současný stav

Aktuální podoba editoru dává uživatelům k dispozici pestrou nabídku formátování, jakým mohou ozdobit text svých zpráv, či v ní zdůraznit podstatné informace. Na stávajícím formátovacím panelu jsou možnosti formátování zobrazeny v následujícím pořadí:

- nastavení typu odstavce,
- výběr použitého fontu písma,
- volba barvy textu a pozadí dokumentu,
- změna velikosti písma,
- trojice bold, italic, underline, neboli tlustý text, kurzíva a podtržený text.

Dále je tu možnost použití seznamu s odrážkami, nastavení odsazení, či zarovnání. Nabídka je, jak vidno, opravdu pestrá a to nezmiňuji tlačítka z menu Formát, které doplňují řadu dalších specifických možností. Pro jednoduché formátování textu je taková podoba editoru jistě dostačující, pokud chce ale uživatel měnit opakovaně více formátovacích vlastností najednou, musí každou z nich při každém použití poctivě „naklikat“.

## 2.3 Budoucí stav

Rozšíření doplní k ovládacímu panelu ještě seznam stylů. Ten umožní uživateli nastavit či zrušit všechny vlastnosti formátování jedním kliknutím. Dovolí mu styly uložit, a tím je uchovat pro další použití i po vypnutí aplikace. Zároveň budou automaticky vyhledány všechny styly aplikované v editovaném dokumentu a nabídnuty uživateli k dalšímu využití.

## 2.4 Požadavky

### 2.4.1 Scénář

1. Uživatel otevře okno s editorem e-mailové zprávy.
2. Vedle editoru se mu zobrazí seznam jeho dříve uložených stylů a pokud není editovaná zpráva prázdná, zobrazí se mu i seznam stylů použitých v textu zprávy.
3. Během psaní zprávy použije uživatel některé z textových či odstavcových stylů, například vytvoří nadpis, obarví část textu a zároveň tento text podtrhne.
4. Po každé úpravě textu se obnoví seznam aktuálních stylů a přidají se nově použité.
5. Uživatel odmaže některý z dříve uložených stylů, který již nechce uchovávat.
6. Namísto něj uloží některý ze stylů nalezených v aktuálním dokumentu. Tento styl si pro přehlednost pojmenuje.

7. Ve zbytku dokumentu pak tento styl aplikuje kliknutím na tlačítko v panelu. Pro jeho ukončení klikne na systémový styl „none“ který styl ukončí a dále pokračuje prostý text.

8. Uživatel může schovat formátovací panel, neboť veškeré formátování obstarává přes seznam stylů. Získává tím na monitoru o kousek více prostoru pro editaci a zjednodušší se tak grafické prostředí, které ho tím méně odvádí od obsahu sdělení.

9. Po dopsání zprávy uživatel zavře okno odesláním, či zrušením zprávy, uložené styly se však uchovají pro další použití.

### 2.4.2 Funkční požadavky

- Rozšíření vytvoří seznam textových a odstavcových stylů.
- Rozšíření umožní uživateli tyto styly vytvářet, upravovat, ukládat, pojmenovat a mazat.
- Rozšíření implementuje metodu pro aplikování vypočítaných stylů.
- Rozšíření zvýrazní aktuálně použité styly v seznamu.
- Rozšíření zobrazí u textových stylů náhled, jak bude text po úpravách vypadat.
- Rozšíření bude implementovat API pro ostatní komponenty ExtBrainu pro zpřístupnění práce s editorem.

### 2.4.3 Nefunkční požadavky

- Rozšíření bude fungovat v Mozilla Thunderbirdu 3.
- Rozšíření bude fungovat na všech operačních systémech, které podporuje Mozilla Thunderbird.
- Rozšíření bude fungovat s editorem i v jiných oknech než je editor e-mailů.
- Rozšíření bude psáno s důrazem na modularitu a snadný vývoj.

# Kapitola 3

# Prostředí a nástroje

Ze zadání je zřejmé, že implementovaná práce má být založena na technologiích Mozilla Application Frameworku, neboť na tomto frameworku je postaven i Mozilla Thunderbird. Využitými technologiemi jsou zejména XUL, JavaScript a kaskádové styly.

Vzhledem k tomu, že vytvářím grafickou komponentu, používám navíc také jazyk XBL pro definování vlastních XUL komponent. Pro zjednodušení a zlepšení čitelnosti kódu využívám v některých místech metody JavaScriptové knihovny jQuery. Pro ukládání uživatelsky definovaných stylů ve formátu JSON se bude hodit také rozšíření této knihovny jQuery.JSON.js.

## 3.1 Platforma Mozilla

Projekt Mozilla zahrnuje několik technologií, které dohromady tvoří Mozilla Platformu [1]. Vznikl roku 1998 s cílem vytvořit následovníka pro webový prohlížeč Netscape Communicator 4.x a jemu přidružené technologie.

S nárůstem možností internetových technologií se projekt rozrostl z webového prohlížeče do sady komponent umožňujících stavbu aplikací pro různé operační systémy i použití. Dnešní Mozilla vytváří rozličné aplikace jak pro lokální instalaci a užívání, tak pro použití vzdáleně přes internet.

## 3.2 Mozilla Thunderbird

Mozilla Thunderbird [25] je e-mailový klient vyvýjený programátory Mozilla Fundation od roku 2003. Kromě e-mailových zpráv lze Thunderbird použít i pro správu RSS a news vláken. Podobně jako další produkty sdružení Mozilla je i Thunderbird postaven na Mozilla Application Frameworku. Je licencován pod trojlicencí MPL/GPL/LGPL jako open-source software. Již od roku 2004 lze do Thunderbirdu instalovat add-on Lightning, který rozšiřuje jeho funkcionalitu o kalendář s podporou událostí a upomínek. Tím se z něj stává plnohodnotný organizér a také rovnocenný oponent konkurenčním produktům jakým je například Microsoft Outlook.

Thunderbird je mimo jiné také aplikací s velkým prostorem pro seberealizaci. Díky jednoduchosti vývoje a instalace add-onů na Mozilla Application Frameworku může každý programátor jednoduše dotvářet nové funkce a technologie, jako by mohla být komunikace v sociálních sítích, chat přes vestavěného IM klienta, tedy posílání textových zpráv pomocí služeb Jabber, MSN, GoogleTalk, icq a dalších, či pokročilejší správa kontaktů.

V současné době je vývojový potenciál na Mozilla Platformě navíc podpořen skutečností, že vycházejí nové verze programů jako je Firefox 4, Thunderbird 3.3, nebo SeaMonkey 2.1 založené na nové verzi vykreslovacího enginu Gecko 2.0. V takové chvíli je třeba velice zapracovat na podpoře starších aplikací a rozšíření a s využitím nových technologií zaujmout co největší množství uživatelů.

## 3.3 Mozilla Application Framework

The Mozilla Application Framework (MAF) je soubor multiplatformních softwarových komponent. Cílem frameworku je poskytnout multiplatformní jádro pro stavbu aplikací, podporující softwarové standardy. Aplikace postavené na MAF se tak nemusí starat o podporu standardů a běží na všech operačních systémech, které MAF podporuje. Takovými aplikacemi je například webový prohlížeč Firefox, či e-mailový klient Thunderbird.

Mozila Application Framework poskytuje pro vývoj aplikací řadu komponent jako například XUL, XBL, Gecko engine, XPCOM, XPInstall a další. S jejich pomocí lze napsat plnohodnotnou aplikaci postavenou na MAF. Projekt Mozilla je však zaměřen předně na internetové technologie, a proto jsou tyto nástroje nejvhodnější právě na vývoj aplikací pro práci s weby a pro internetovou komunikaci.

Při psaní rozšíření editoru využívám funkcionality většiny těchto komponent, proto se v následujících odstavcích blíže zmíním o některých z nich.

## 3.4 Chrome

Než popíšu technologie, které při práci na Mozilla Application Frameworku využívám, rád bych se zmínil o důležitém pojmu, který není názvem pro konkrétní technologii, ale který budu v dalším textu potřebovat. Tímto pojmem je `chrome`.

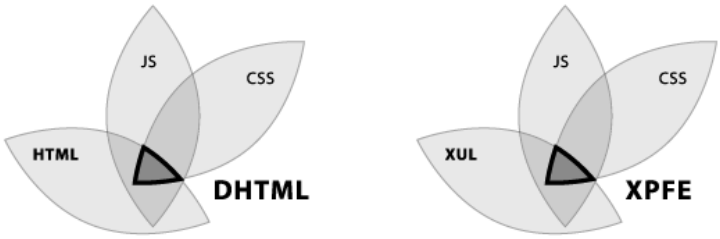
Vrstva programu tvořící grafické prostředí se v aplikacích postavených na Mozilla Application Frameworku nazývá `chrome` [22]. Nejedná se jen o uživatelské rozhraní, ale o celou programovou část, která grafické prostředí utváří. Do `chrome` paří například menu řádky a položky, panely nástrojů, kontextové nabídky, stavový řádek, a to s veškerou jejich funkcionalitou i vzhledem. Části `chrome` jsou definovány pomocí zdrojových kódů v jazycích XUL, XBL, JavaScript a v DTD a CSS souborech.

Zdrojové soubory mohou být odkazovány pomocí klasických HTTP URL adres podobně jako webové soubory. V balíčkách instalovaných do chrome aplikace jsou tyto soubory registrovány do speciálních `chrome://` URL adres. Balíčky, které jsou součástí aplikace jsou za běhu aplikace vždy již v chrome registrovány, je však možné do něj přidat nové v podobě rozšíření (add-onu).

Registrace těchto souborů do chrome probíhá při startu aplikace a je popsána v kořenovém adresáři jednotlivých add-onů v souboru `chrome.manifest`. Pomocí takovýchto souborů jsou do chrome registrovány i zdrojové kódy samotné aplikace [16].

## 3.5 XUL

XUL [čti: zůl] neboli XML User Interface Language [21] je značkovací jazyk z dílny Mozilla odvozený od jazyka XML. Pomocí tohoto jazyka se dá velmi snadno vytvářet grafické prostředí aplikace. Jeho XML struktura umožňuje jednoduše měnit texty, grafiku i vzhled aplikace pomocí kaskádových stylů a DTD entit, takže se snadno přizpůsobí cílovým uživatelům například lokalizací. Práce s ním je obdobná jako s dynamickými HTML dokumenty, pro řadu vývojářů je z tohoto důvdou velmi snadné se ho rychle naučit. Tuto podobnost naznačuje obrázek 3.1. Grafickou podobu okna, která se zobrazí uživateli, definovaného pomocí XULu a dalších nástrojů, společně označovaných XPToolkit [1], vytváří renderovací engine Gecko (viz 3.7).




Obrázek 3.1: Srovnání jazyka XUL a dynamického HTML

XUL tagy značí jednotlivé elementy grafického prostředí, které se skládají v XUL dokumentech a definují tak okna i jiné části aplikací. Elementy XUL mohou být:

- kořenové elementy mezi něž patří `window`, `dialog`, `wizard`, či `overlay` a které specifikují další použití XUL dokumentu,
- konkrétní viditelné ovládací a grafické prvky, jako například `button`, `label`, `separator`, `tree`,
- XUL elementy pro definování stavby a struktury grafického prostředí, mezi které patří `hbox`, `vbox`, `grid` či `arrowscrollbox`,
- složitější komponenty jako `browser` či `editor` pro zobrazení XML dokumentů.

[1] Zkratka XPFE na obrázku sloužila dříve pro označení nástrojů pro tvorbu aplikací na MAF. Dnes se tyto nástroje někdy označují jako XPToolkit.

Kromě viditelných komponent grafického prostředí obsahuje XUL také elementy pro definování funkcionality a chování jako například `command`, `key`, či `observes`. Pomocí jmenného prostoru lze také použít elementy z ostatních jazyků založených na XML a podporovaných Gecko enginem, jako HTML, SVG nebo MathML.

### 3.5.1 Vztahy jazyka XUL v aplikacích MAF

Základní strukturu a vlastnosti XUL elementů definuje XBL binding (viz 3.6). Tento binding je mapován na XUL tag pomocí speciálního CSS pravidla `-moz-binding` umístěného kdekoliv v aplikovaných kaskádových stylech. Element specifikovaný tímto tagem pak nabývá vlastností z namapovaného bingingu.

Textový obsah a lokalizace[2] je obsažena v entitách DTD souborů. Tyto soubory jsou uloženy ve speciálních adresářích, jejichž název odpovídá konkrétní jazykové mutaci. Pro změnu jazyka grafického prostředí pak stačí změnit část `chrome://` adresy těchto DTD souborů podle aktuálního nastavení.

Stylování vzhledu elementů určuje CSS styl. Není nezajímavé, že libovolně pojmenovaný dříve nedefinovaný XUL tag, lze jen za pomoci kaskádového stylu stylovat jako například HTML element `div`, protože tento dříve nedefinovaný element se v XULu chová jako prázdný kontejner. Pro vytvoření primitivního grafického elementu stačí tedy jen použít v XUL souboru tag s jeho jménem. Bez jakýchkoli dalších operací ho pak můžeme rovnou začít stylovat, plnit obsahem a chovat se k němu, jako k již existujícímu XUL elementu.

Chování grafického prostředí je definováno JavaScriptem. Interakce uživatele pomocí klávesnice a myši vyvolává události (eventy) na grafických komonentách. Pokud je taková událost odchycena, zavolá JavaScriptovou metodu, která je jí přiřazena.

Některé metody jdou společné pro více uživatelských vstupů. Například příkaz pro kopírování označeného textu do clipboardu lze typicky vyvolat klávesovou zkratkou `Ctrl+C`, pravým tlačítkem myši z kontextového menu na textové komponentě nebo z menu Edit. Metoda pro kopírování do schránky se proto přiřadí pouze XUL elementu `command` nesocímu identifikátor `id="commandID"` a ostatní tlačítka i klávesová zkratka se na něj napojí atributem `command="commandId"`. Funkce, kterou má uživatelský vstup spustit, se pak přiřadí pouze elementu `command` a volá se při kterékoli akci uživatele, ať už při stisku tlačítka či kávesové zkratky. Command element potom navíc slouží jako broadcaster, což znamená, že atributy na něm nastavené, se dále distribuují navázaným elementům. Takto lze například zakázat všechny ovládací prvky pro kopírování pouze přiřazením atributu `disalbed="true"` elementu `command`.

Jednouduchou grafickou interakci lze implementovat pomocí pokročilých css selectorů jako je `:hover`, `:not([disabled])` a podobně. Ve většině případů toto řešení ušetří velké množství kódu a značně přidá programu na přehlednosti.

S jádrem aplikace a externími XPCOM (viz. 3.8) komponentami, které popíši dále v textu, je XUL dokument spojen v JavaScriptu přes XPConnect rozhraní. Jednotlivé komponenty se pak získávají ze speciálního objektu `Components` následujícím způsobem:

[2] Lokalizace se v kódu a komentářích často označuje numeronymem l10n neboli „localization“, případně se lze setkat i s pojmem „internationalization“ v podobě i18n.

```
var ioService = Components.classes['@mozilla.org/network/io-service;1']
                    .getService(Components.interfaces.nsIIOService);
```

Objekt classes mapuje XPCOM komponentu na její identifikátor. Tento identifikátor je obvykle ve formátu `@doména/jméno_modulu/jméno_komponenty;verze`. Metoda getService pak dostane jako parametr interface, který má vrácený objekt implementovat. Na objektu `ioService` z příkladu můžeme tedy následně volat všechny metody definované interfacem nsIIOService.

Úpravy mezi jednotlivými XUL soubory umožňuje `overlay` system. Overlay je název pro XUL soubor, který je zobrazen „přes“ XUL dokument na který je aplikován. Overlay může přidávat, či ubírat XUL elementy, upravovat jejich atributy i funkcionalitu. Pomocí overlaye přidávají například jednotlivá rozšíření své prvky do GUI aplikace. Overlay system také umožňuje oddělit jednotlivé části okna do samostatných XUL zdrojových kódů pro přehlednost a modularitu.

K zaměřování upravovaných XUL elementů se používá jejich atribut `id`. Jmenný prostor, ve kterém se tyto identifikátory nacházejí, je společný pro všechny rozšíření aplikace i pro samotnou aplikaci. Proto je třeba zvolit vždy unikátní a výstižné označení konkrétního elementu. V aplikaci jsou tyto identifikátory jednodušší a konkrétnější, například `menu_file` pro menu Soubor. U samostatných rozšíření je pak nejlépe dát identifikátorům unikátní prefix, pro zaručení jednoznačnosti. Pro ExtBrain Editor, používám zatím jako prefix identifikátorů `extb`. Po sjednocení se zbytkem projektu ExtBrain se upraví podle konvencí.

XUL dokument lze otevřít i v prohlížeči vzdáleně přes web, tedy nikoli jen z chrome aplikace. Stránka, ve které je zobrazen, pak nemá přístup do lokálních metod, ale zobrazí se, a funguje stejně jako běžná webová stránka, jen místo HTML komponent se zobrazí komponenty XUL.

## 3.6 XBL

XBL je jazyk založený také na XML [21]. Definuje vazby (bindings) mezi XUL tagem a jeho vlastnostmi. Vázaný XUL element, který je tímto tagem definován, neboli „bound element“, jež XBL definice popisuje, vazbami získává na funkcionalitě, které lze využít kdekoli, kde se element vyskytuje v kontextu.

Binding je na element navázán pomocí CSS vlastnosti `-moz-binding` [12]. Do této se přiřadí adresa XBL souboru a za znak mřížky název bindingu z tohoto souboru. Lze tak snadno přepsat binding nějakého XUL tagu nebo použít existující binding pro nový tag.

V následujících kapitolách popisuji, jaké všechny vlastnosti a funkcionalitu [14] lze pomocí bindingu definovat na XUL elementu.

### 3.6.1 Konstruktor a destruktor

JavaScriptový kód konstruktoru elementu se v bindingu zapisuje do tagu `constructor`, destruktoru do tagu `destructor`. Konstruktor je volán kdykoli je binding přiřazen XUL elementu. Toto nastává pokaždé, když je zobrazován obsah XUL dokumentu v okně aplikace. Volání konstruktorů všech XUL elementů okna nazýváme také načítání, neboli „loading“

okna. Event `load` na okně, ve kterém je dokument zobrazován, je vyvolán až poté, co doběhnou konstruktory všech obsažených elementů. Pořadí volání konstruktorů není předem dané a nemělo by se na něj spoléhat. Konstruktor je volán také v případě, že je XUL tagu změněna css property `-moz-binding`. V takovém případě je zavolán destruktor z původního bindingu a následně konstruktor z bindingu nově přiřazeného.

### 3.6.2 Anonymní obsah

Anonymním obsahem XUL elementu se rozumí vnořené elementy fyzicky obsažené uvnitř něj, avšak odstíněné, a tedy nedostupné, z DOM stromu dokumentu. Anonymní obsah se umisťuje do tagu `content` XBL bindingu a je tvořen libovolnými XML elementy. Jejich tagům musí být správně přiřazen jmenný prostor, aby nekolidoval se jmenným prostorem bindingu. Tagy se tedy zadávají například s prefixem `xul:`, následovaným názvem tagu.

Anonymní obsah je možné zpřístupnit pomocí metody `getAnonymousNodes` z bindingu:

```
document.getAnonymousNodes(this);
```

případně prímo z XUL dokumentu:

```
document.getAnonymousNodes(document.getElementById("id_elementu"));
```

### 3.6.3 Metody

XBL binding může definovat i metody pro daný element. Metoda je v bindingu definovaná následujícím kódem:

```
<implementation>
  <method name="method_name">
    <parameter name="parameter1"/>
    <parameter name="parameter2"/>
    .
    .
    .
    <body>
      // obsah metody
    </body>
  </method>
</implementation>
```

Po navázání elementu na binding se metoda chová stejně jako na JavaScriptovém objektu. Metoda je přístupná na elementu například takto:

```
document.getElementById("id_elementu").method_name(parameter1, parmeter2);
```

### 3.6.4 Field a property

XBL field [15] slouží jako nosič hodnot. Lze do něj přiřadit libovolný JavaScriptový objekt, text či hodnotu. Field může však obsahovat i JavaScriptový kód a vracet dynamicky vypočítanou hodnotu. Naproti tomu, property neuchovává žádnou hodnotu, lze na ní však definovat metody setter a getter, které se volají při dotazu na property a při jejím nastavení. Pro uchování dat nastavených do property může sloužit například atribut XUL elementu (pro textové, číselné či booleovské hodnoty), případně specielně pro to vyhrazený field.

Příklady použití defaultní a vypočítané hodnoty fieldu:

```
<field name="number">
  25
</field>

<field name="currentTime">
  new Date().getTime();
</field>
```

Příklad užití property s hodnotou uloženou do atrubutu:

```
<property name="number">
  <getter><![CDATA[
    return this.getAttribute('number');
  ]]></getter>
  <setter><![CDATA[
    var v = parseInt(val,10);
    if (!isNaN(v)) return this.setAttribute('number',''+v);
    else return this.getAttribute('number');"
  ]]></setter>
</property>
```

### 3.6.5 Event handlery

Přímo v XBL bindingu lze elementům přiřadit i event handlery [13]. Kód obsažený v těchto handlerech je pak zavolán pokaždé, kdy objekt event zachytí. Handler se tedy chová stejně, jako by byl k elementu přiřazen pomocí atributu či metodou `node.addEventListener`.

Základní kód handleru vypadá takto:

```
<handlers>
  <handler event="event-name" action="script"/>
</handlers>
```

Užitečné jsou i handlery, určené speciálně pro uživatelské vstupy:

```
<handler event="click" button="0" action="alert('Left button pressed');"/>
```

```
<handler event="keypress" key="c" modifiers="control">
  <![CDATA[
    this.clipboard = document.getAnonymousNodes(this)[0].value;
  ]]>
</handler>
```

Takto definovaný handler je lépe čitelný, než handlery přidělené z JavaScriptu a hlavně se automaticky se přidělí všem XUL elementům ke kterým je binding navázán.

### 3.6.6 Dědičnost

Dědičnost XBL bindingů je zajištěna atributem `extends` na elementu `binding`, do kterého se zapisuje adresa bindingu, jehož vlastnosti má náš binding získat.

Například pokud vytváříme textbox, kterému chceme přiřadit nové vlastnosti, stačí oddědit původní binding textboxu a implementovat jen novou funkcionalitu:

```
<binding id="mytextbox"
         extends="chrome://global/content/bindings/textbox.xml#textbox">
```

Většina grafických komponent XUL je alespoň částečně implementována pomocí XBL bindingů. Z existujících komponent v jazycích XUL, HTML, SVG je tak pomocí XBL možné vytvářet skládáním (anonymní obsah) a oddděděním nové grafické komponenty a používat je v Mozilla aplikacích.

## 3.7 Gecko

Gecko [17] [24] je název vykreslovacího enginu, vyvýjeného v rámci projektu Mozilla. Jeho původní název byl NGLayout a byl Mozillou přejat z projektu Netscape Communicator.

Gecko engine se využívá ke generování webových dokumentů ze zdrojového kódu. Struktura a vlastnosti těchto dokumentů jsou definovány v jazycích HTML, CSS, XML, JavaScript, a dalších. Z těchto zdrojových kódů Gecko renderuje výslednou podobu dokumentu k zobrazení uživateli nebo k tisku. V aplikacích založených na jazyce XUL vykresluje Gecko engine také samotné uživatelské prostředí aplikace. Gecko se snaží o implementaci posledních internetových standardů u jazyků, které podporuje: HTML 4 a částečně 5, CSS 1/2, W3C DOM, ECMAScript 3.

Gecko[3] engine využívá řada aplikací, včetně webových prohlížečů, jako například Firefox, Thunderbird, Camino a další. Produkty užívající stejnou verzi Gecko enginu disponují identickou podporou užitých standardů.

[3]Pro zajímavost se jedná o další příklad zaměření The Mozilla Project na ochranu zvířat. Podobně jako u Firefoxu je zde použito jméno ohroženého druhu pro název komponenty. Již delší dobu navíc Projekt Mozilla používá pro kódová označení verzí Firefoxu názvy světových národních parků, které také podporuje.

## 3.8 XPCOM

XPCOM [19] je multiplatformní Component Object Model. Slouží k propojení komponent, zde ve smyslu částí programu, psaných v různých programovacích jazycích. Jazyky pro které je v XPCOM modelu vytvořen binding (neplést XPCOM binding s XBL bindingem) mohou mezi sebou sdílet objekty a volat na nich metody. Umožňuje implementovat a použít objekty XPCOM komponent v jazycích JavaScript, Java, Pythonu či dokonce v C++. Interfacy XPCOM komponent jsou definovány v XPIDL [20] souborech (s koncovkou .idl) s univerzální IDL syntaxí.

Samotný XPCOM poskytuje sadu základních komponent, například pro práci se soubory, či pamětí, pro užití vláken, nebo pro základní datové struktury (řetězce, pole, proměnné). Většina XPCOM komponent však není přímo součástí jádra XPCOM, ale jsou poskytovány jinými částmi platformy (např. Gecko a Necko), nebo samotnou aplikací, či dokonce rozšířeními aplikace.

Jako příklad užití XPCOM komponenty uvádím Preference manager, což je objekt nesoucí metody pro práci s hodnotami proměnných z preferences systému. Tyto hodnoty slouží jako konfigurační proměnné aplikace a jejích rozšíření a jsou uchovávány i po vypnutí aplikace. V následujícím příkladu uvádím způsob jakým se získávají XPCOM komponenty pomocí XPConnect systému. Pomocí identifikátoru je vybrána konkrétní komponenta ze seznamu registrovaných komponent, a metodou `getService` se z ní „vytáhne“ implementace interfacu z parametru. Dále se na objektu volají metody interfacu, jako kdyby byl implementován přímo v JavaScriptu. V příkladu jsou to metody `getCharPref`, a `setCharPref`.

```
var prefManager = Components.classes["@mozilla.org/preferences-service;1"]
                      .getService(Components.interfaces.nsIPrefBranch);

var lang = prefManager.getCharPref('extensions.extbrain-editor.lang');
prefManager.setCharPref('extensions.extbrain-editor.lang', 'cs');
```

## 3.9 System rozšíření

Pro aplikace Mozilla Application Frameworku je možné vytvářet rozšíření, neboli add-ony. Pomocí add-onů je možné rozšiřovat již existující aplikace o novou funkcionalitu a upravovat grafické prostředí pomocí XUL overlayí. Při startu aplikace se všechny instalované a povolené add-ony registrují do jejího chrome a tím se stanou součástí aplikace.

Rozšíření jsou distribuována jako XPI (čti „zippy“) balíčky, tedy zkomprimovaný ZIP archiv s koncovkou .xpi a konkrétní strukturou souborů v něm.

Základní struktura XPI balíku:

```
extension.xpi:
     /install.rdf
     /defaults/
     /defaults/preferences/*.js
     /chrome.manifest
     /chrome/
```

```
/chrome/content/
/chrome/locale/*/*.dtd
/chrome/skin/*.css
```

### 3.9.1 install.rdf

Soubor install.rdf v kořenu XPI balíku obsahuje obecné informace o add-onu. Jde hlavně o identifikátor a verzi add-onu. Dále identifikátory a verze aplikací pro které je add-on určen. Z dalších informací lze v souboru uvést například název a popis add-onu, přidat seznam jeho autorů, či specifikovat chrome URI okna předvoleb add-onu nebo URL na které se nachází jeho případné updaty.

Jako příklad uvádím soubor install.rdf přímo z vyvýjeného rozšíření ExtBrain Editor pro Mozilla Thunderbird.

```
<?xml version="1.0"?>
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:em="http://www.mozilla.org/2004/em-rdf#">

  <Description about="urn:mozilla:install-manifest">
    <em:id>extbrain-editor@kub1x.org</em:id>
    <em:name>ExtBrain Editor</em:name>
    <em:version>0.2</em:version>
    <em:description></em:description>
    <em:creator>Jakub Podlaha</em:creator>
    <em:homepageURL>http://extbrain.felk.cvut.cz</em:homepageURL>

    <!-- Thunderbird -->
    <em:targetApplication>
      <Description>
        <em:id>{3550f703-e582-4d05-9a08-453d09bdfdc6}</em:id>
        <em:minVersion>3.0</em:minVersion>
        <em:maxVersion>3.3</em:maxVersion>
      </Description>
    </em:targetApplication>
  </Description>
</RDF>
```

### 3.9.2 Adresář pro předvolby

V JavaScriptových souborech v adresáři `defaults/preferences/` jsou typicky specifikovány hodnoty proměnných pro preferences systém vztahující se k rozšíření. Těmito scripty se nastavují jejich výchozí hodnoty. Uživatelsky nastavené hodnoty těchto proměnných nejsou těmito soubory přepsány.

Pro potřeby vývoje existuje pro Thunderbird a Firefox nastavení aplikace, které dává vývojářům co možná největší výstup. Toto nastavení uvádím jako vzorový, preferences soubor. Neměl by samozřejmě být součástí odladěného add-onu, nýbrž jen vývojové verze.

```
/* debugging prefs */
pref("browser.dom.window.dump.enabled",     true);
pref("dom.report_all_js_exceptions",        true);
pref("extensions.logging.enabled",          true);
pref("javascript.options.showInConsole",    true);
pref("javascript.options.strict",           true);
pref("nglayout.debug.disable_xul_cache",    true);
pref("nglayout.debug.disable_xul_fastload", true);
```

### 3.9.3 chrome.manifest

Při zavádění rozšíření do aplikace se registrují jeho části do chrome této aplikace. Registrace, jak již bylo popsáno v kapitole 3.4, probíhá čtením souboru chrome.manifest v kořenovém adresáři add-onu. Tento soubor může vypadat následovně.

```
content  extbrain-editor  chrome/content/

## Overlay pro hlavni okno messengeru
overlay  chrome://messenger/content/mailWindowOverlay.xul\
             chrome://extbrain-editor/content/overlay.xul

## Overlay pro okno mesagecomposeru
overlay  chrome://messenger/content/messengercompose/messengercompose.xul\
             chrome://extbrain-editor/content/messengercomposeOverlay.xul

overlay  chrome://messenger/content/messengercompose/messengercompose.xul\
             chrome://extbrain-editor/content/overlay.xul

locale  extbrain-editor  en-GB  chrome/locale/en-GB/
locale  extbrain-editor  en-US  chrome/locale/en-GB/
locale  extbrain-editor  cs-CZ  chrome/locale/cs-CZ/

skin   extbrain-editor  classic/1.0  chrome/skin/
```

Na prvním řádku se vytváří URI `chrome://extbrain-editor/content/` a mapuje na podadresář `chrome/content/` XPI balíku. V tomto adresáři se nacházejí zdrojové soubory rozšíření. Od toho momentu je možné na veškerý obsah rozšíření odkazovat pomocí této chrome adresy.

Soubory které obsahují overlay se na dalších řádcích manifestu aplikují příkazem:

```
overlay přepisovaný_XUL_soubor overlay_XUL_soubor
```

a tím modifikují existující implementaci v aplikaci. V příkladu je vidět že soubor `overlay.xul` se aplikuje na dva různé XUL dokumenty. Tento soubor obsahuje rozšíření nabídky menu, která je XUL oknům definovaným cílovými soubory společná. Proto nám nic nebrání jej aplikovat na oba soubory a rozšířit si položky menu u obou. Jinou možností by bylo aplikovat

`overlay.xul` nikoli na konkrétní okno, nýbrž na nějakou overlay z aplikace již použitou pro všechna okna, která chceme upravit.

`locale` příkazy mapují kódy jazyků na adresáře obsahující jazykové mutace rozšíření, ve formě DTD souborů. V podobě entit v těchto souborech je uchováván textový obsah grafického prostředí, jako jsou popisky tlačítek či položek menu. V každém mapovaném podadresáři typicky `chrome/locale/` jsou tyto entity přeloženy do odpovídajícího jazyka.

Příkaz `skin` zde mapuje na chrome URI adresář, sloužící pro ukládání kaskádových stylů použitých v add-onu.

### 3.9.4 Adresář obsahu

Adresář `chrome/content/` XPI balíčku obsahuje zdrojové kódy rozšížení, zejména XUL a JavaScriptové soubory. Může zde však být umístěn jakýkoli další obsah. Ve většině add-onů je podadresář `content` ještě znovu zkomprimován do podoby tak zvaného jarballu, tedy archivu zkomprimovaného metodou ZIP s koncovkou `.jar`. Pro tyto archivy je pak v souboru `chrome.manifest` použita speciální syntaxe. Použití těchto balíků napomáhá aplikaci ke snadnější práci se soubory a umožňuje efektivnější cachování v paměti.

Příklad užití jarballu v souboru `chrome.manifest`, který namapuje obsah podadresáře jarballu na URI `chrome://global-platform/content/`:

```
content global-platform jar:toolkit.jar!/toolkit/content/global-platform/
```

## 3.10 Nástroje a prostředí pro vývoj v Mozilla Thunderbirdu

Následující odstavce obsahují doporučené nastavení a nástroje, usnadňující práci při vývoji rozšíření. Kromě zde vypsaných je na webu k dispozici ještě řada rozšíření, které se mohou hodit při specifických úlohách. Uvádím pouze ty, které jsem při vývoji opravdu potřeboval.

### 3.10.1 Debugovací předvolby Mozilla Thunderbirdu

Pomocí předvoleb (preferences), jejichž nastavení je popsáno v části 3.9.2, získáme z aplikace maximální množství informací o chybách a varováních v konzoli. Usnadníme si testování vypnutím XUL cachování a povolíme logování zpráv.

### 3.10.2 DOMInspector

Nepostradatelným pomocníkem při vývoji nejen rozšíření pro chrome aplikace založené na MAF, je add-on DOMInspector. Toto rozšíření dokáže ve stromové podobě zobrazit strukturu XUL dokumentu libovolného zobrazeného okna. Tato struktura je opravdu aktuální, tedy po aplikování všech overlay i scriptů. Poskytuje informace o elementech DOM stromu, jejich atributech, o grafické reprezentaci těchto elementů, XBL bindingu, a v neposlední řadě dává k dispozici seznam atributů a metod JavaScriptového objektu dostupného z těchto elementů. Jednou z klíčových vlastností je také možnost zaměřit v DOMInspectoru konkrétní element prostým kliknutím na něj v aktuálné zobrazeném okně.

### 3.10.3 Venkman JavaScript debugger

K ladění JavaScriptového kódu za běhu je pro aplikace MAF k dispozici debugger zvaný Venkman. Tento debugger je stejně jako DOMInspector instalován přímo do aplikace ve formě add-onu. Venkman je velice mocný nástroj poskytující funkcionalitu plnohodnotného debuggeru. Podporuje breakpointy, krokování kódu, výpisy paměti a mnoho dalšího. Vzhledem k tomu, že sám běží v aplikaci, kterou se snažíme odladit, ovlivňují ho bugy této aplikace, a například při chybě v rozšíření, dochází ke kolizi mezi Venkmanem a samotným Thunderbirdem, při které aplikace může až zamrznout. Při ladění složitějsího kódu je však práce s ním velkým usnadněním.

### 3.10.4 Rozšíření Restart

Jednoduchým, však velmi užitečným rozšířením pro Thunderbird je Restart add-on. Přidává do hlavního okna a několika dalších oken tlačítko pro restartování aplikace. Při opravách chyb a testování je toto rozšíření nezbytností.

### 3.10.5 Online nástroje

Esenciální pro vývoj na Mozilla platformě jsou také online nástroje a dokumentace. Vyjma hlavního zdroje přehledné dokumentace, kterým je Mozilla Developer Network `<https://developer.mozilla.org/>`, je třeba zmínit ještě alespoň dva zásadní weby určené vývojářům.

- MXR - Mozilla Cross Reference. Je k nalezení na adrese `<http://mxr.mozilla.org>` a poskytuje databázi zdrojových kódů s pokročilým vyhledáváním. Tato databáze dává vývoji nový rozměr. Ze zdrojových kódů se tímto stává jeden z hlavních zdrojů informací pro programátory a vývojáři na programech Mozilla jsou tímto nuceni psát přehledný a čitelný kód. Čtení kódu namísto nudné dokumentace navíc naučí budoucí vývojáře mnohem více.
- AMO - Add-ons Mozilla.org. Web obsahující databázi rozšíření pro programy Mozilla. Jeho adresa je `<https://addons.mozilla.org>` a v částech pro vývojáře obsahuje řadu doporučení pro psaní rozšíření, časté chyby a programátorské konvence. Poskytuje také prostor pro uchování hotových add-onů.

### 3.10.6 Vlastní nástroje v rámci rozšíření

Podle vlastního uvážení a zkušeností, je možné si vlastní debugovací nástroje naimplementovat přímo v rámci rozšíření aplikace. Nejjednodušším pomocníkem v tomto směru je příkaz `alert()`, který otevře prostý dialog s tlačítkem OK. A s textem, který je mu předán jako parametr. V ExtBrain Editoru tohoto využívám v soubory overlay.xul, ve kterém vytvářím novou položku menu „ExtBrain“ do které umísťuji tlačítka se zrovna potřebnými akcemi pro testování.

Stálicí mezi těmito tlačítky je otevření aktuálního okna v rozšíření DOMInspector. Je pořeba tak často, že jsem si přístup k němu ještě více zjednodušil pomocí kláveskové zkratky

`Ctrl+Shift+I`. Dalším příkladem, kdy lze využít přímo položku menu, může být například rychlý přístup do seznamu předvoleb.

Vše je vidět v následujícím XUL příkladu souboru `overlay.xul` z vyvýjeného rozšíření:

```
<?xml version="1.0"?>
<?xml-stylesheet
     href="chrome://extbrain-editor/skin/overlay.css" type="text/css"?>
<!DOCTYPE overlay SYSTEM "chrome://extbrain-editor/locale/overlay.dtd">
<overlay id="extbrain-editor-overlay"
     xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

<script type="application/x-javascript"
        src="chrome://inspector/content/hooks.js" />

<commandset id="composeCommands">
   <command id="extbCmd_inspect"
            oncommand="inspectDOMDocument(window.document);" />
</commandset>

<keyset id="baseMenuKeyset">
   <key id="extbKey_inspect"
         command="extbCmd_inspect"
         modifiers="accel,shift" key="I" />
</keyset>

<menubar id="mail-menubar">
   <menu id="extbMenu" position="9" label="ExtBrain" accesskey="E">
      <menupopup id="extbMenuPopup">
         <menuitem id="extbMenuInspectThis"
                   label="Inspect this"
                   command="extbCmd_inspect" key="extbKey_inspect" />
         <menuitem id="extbMenuAboutConfig"
                   label="about:config"
                   oncommand="window.open('about:config', '', 'chrome');" />
      </menupopup>
   </menu>
</menubar>
</overlay>
```

# Kapitola 4

# Technologická analýza

V následující kapitole velmi podrobně proberu funkcionalitu editoru. Pokusím se zaměřit na podstatné technické detaily a popsat funkcionalitu dostatečně podrobně. Tato kapiotla má sloužit zejména navazujícím programátorům, ujasnit jim, jak editor funguje, aby se nemuseli dále zabývat analýzou a mohli soustředit své síly na vývoj.

Začnu obecným popisem architektury editoru a následně se zaměřím na jeho jednotlivé části. Při popisu těchto částí budu postupovat směrem od uživatelského prostředí k jádru editoru. Začnu u analýzy kódu editoru přímo v grafickém prostředí, tedy u částí editoru psaných v XULu a JavaScriptu. Dále se přes XBL binding elementu editor přesunu k jeho vnitřní funkcionalitě a XPCOM komponentám.

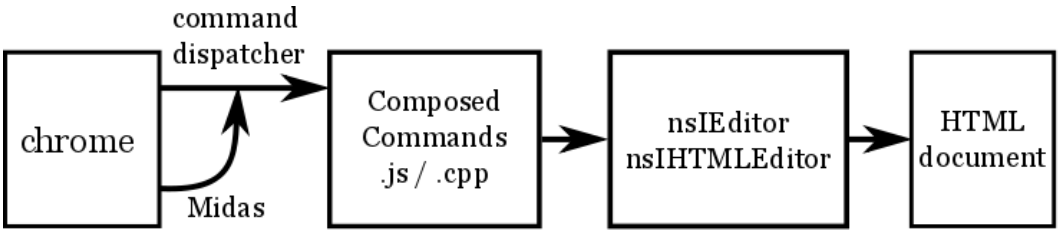
Po tomto popisu částí editační komponenty následuje analýza samotné editace. Zde simuluji průběh formátování od uživatelského vstupu až po samotné provedení příkazu a zaznamenávám důležité části.

Následně se zaměřím na editaci přímo na dokumentu, tedy na takzvaný Midas editor. I přestože analýza Midas editoru zabrala relativně hodně času, jsou výsledky spíše informativní a navazují na již zdokumentované poznatky o editoru.

Nakonec se pokusím zhodnotit získané informace pro možnosti dalšího vývoje.

## 4.1 Architektura Komponenty Editor

Editor je v Mozilla Application Frameworku tvořen několika částmi. Rám editoru, který uživatel vidí na obrazovce je ve zdrojovém kódu okna reprezentován XUL elementem `editor`. V XULu jsou také vytvářeny všechny grafické ovládací prvky editoru, stylovací panel a podobně. Jádro editoru je psáno v programovacím jazyce C++ a aplikaci poskytuje interfacy v podobě XPCOM komponent, se kterými je možné libovolně pracovat z chrome aplikace. Mezi těmito dvěma vrstvami, jádrem aplikace a uživatelským prostředím (chrome), se nachází XBL binding, který mapuje funkcionalitu komponent jádra editoru na XUL element a poskytuje tak základní metody editoru k volání JavaScriptem.




Obrázek 4.1: Model částí editoru

### 4.1.1 XUL Element editor

Element `editor` se na první pohled chová stejně jako XUL element `browser`, použitý například pro zobrazení obsahu v prohlížeči Firefox, či známější HTML element `iframe`, dříve často používaný ve webových stránkách. Jde o prostý rám, zobrazující HTML dokument renderovaný z HTML kódu. Vnitřně editor skutečně takovýto rám obsahuje. Vytváří však nad ním vrstvu pro práci s dokumentem, který je tímto rámem vykreslován.

WYSIWYG editací zobrazeného dokumentu, tedy editací kdy uživatel píše přímo do výsledné podoby dokumentu, ne do zdrojového HTML kódu, se pomocí editoru upravuje nejprve právě zdrojový kód dokumentu a tím, v důsledku, i samotná zobrazovaná podoba dokumentu. Přestože se tedy uživateli zdá, že upravuje přímo dokument, vnitřně editor upravuje pouze HTML kód. Tyto úpravy jsou navíc prováděny v podobě jednosměrných, „DOM savy“ úprav. Tím je zajištěno, že výstupem bude vždy správně zobrazitelný a konzistentní HTML dokument. [2]

Element editor samotný neobsahuje žádné ovládací prvky. Vývojář má tak možnost si naimplementovat vlastní grafické rozhraní editoru, tlačítka, klávesové zkratky, nabídky menu apod. [1]

Atributy elementu `editor` jsou:

- editortype - nastavuje se na „html“ pro plnohodnotnou HTML editaci, nebo „text“ pro editaci prostého textu
- src - URI adresa editovaného dokumentu
- type - obvykle se nastavuje na hodnotu „content-primary“ čímž se obsah editoru označí jako hlavní obsah okna a jeho reference se uloží do proměnné `window.content`

Základní okno e-mailového editoru v Mozilla Thunderbirdu, se v jeho chrome nachází na URI `chrome://messenger/content/messengercompose/messengercompose.xul`. Podle této adresy ho budu nadále označovat jako okno `messengercompose`.

Samotný XUL element `editor` nedisponuje žádnými metodami pro editaci. Slouží pouze jako nosič několika XPCOM komponent a rámu pro zobrazení obsahu editovaného dokumentu.

[1] V rámci editace jsou však přímo v zobrazovaném dokumentu k dispozici elementární grafické prvky, kterými jsou vodítka pro změnu velikosti obrázků a tlačítka pro přidávání či rušení řádků a sloupečků tabulek.

### 4.1.2 Povolení editovatelnosti obsahu

Pro povolení „rich text editing“, neboli plnohodnotného upravování formátovaného textu je třeba elementu `editor` nastavit atribut `type`. Pro potřeby stylovaného textu musíme tento atribut nastavit na hodnotu `html`. Druhá jeho možná hodnota je `text` a slouží pro editování prostého textu.

V případě dodatečného nastavení typu editoru (atribut `type`) je třeba manuálně zavolat jeho metodu `makeEditable()`, jelikož není zavolána automaticky z konstruktoru (viz 4.1.5).

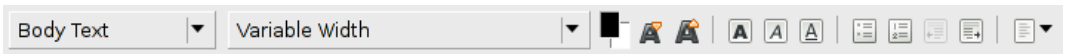
Jak už bylo řečeno, pro definování editovaného dokumentu slouží atribut `src`. Pokud je tento atribut nastaven přímo na `editor` tagu ve zdrojovém XUL souboru, nestane se automaticky editovatelným. Je li však src atribut nastaven dodatečně po načtení okna, v němž je editor zobrazen, například z metody `onload()`, je editovatelnost zapnuta automaticky.

Posledním způsobem, jakým lze povolit změny obsahu je nastavit `designMode` na objektu `document` obsaženém v editoru na `'on'`.

Pro editaci nového dokumentu lze nastavit atribut `src` na hodnotu `about:blank`, což je defaultní adresa pro prázdý dokument podobně jako u webového prohlížeče.

### 4.1.3 XUL Ovládací prvky editoru

V okně `messengercompose` se krom elementu editor v XULu nachází i jeho ovládací prvky. Nejnápadnější je panel s formátovacími tlačítky a nabídkami hned nad editorem, který je vidět na obrázku . Dále k editoru patří položky nabídek „Vložit“ a „Formát“ z menu okna. Všechna tato tlačítka jsou navázána na elementy `command`, které obstarávají volání příslušných funkcí při jejich stisku. Jednotlivé funkce, které tato tlačítka vyvolávají, popíšu dále během analýzy konkrétních formátovacích příkazů.




Obrázek 4.2: Formátovací panel editoru

### 4.1.4 Zdrojové soubory v chrome

Krom metod poskytnutých XPCOM komponentami na XUL elementu `editor` skrz XBL binding, mohou pro použití ve standardním okně e-mailového editoru v Thunderbirdu posloužit také již hotové scripty `editor.js` a `editorUtilities.js` a overlaye pro XUL okna `editor.xul` a `editorOverlay.xul`. V případě potřeby lze, za určitých podmínek, tyto overlaye aplikovat i pro editor v jiném okně než `messengercompose`.

Soubor `editor.js` [4] (dostupný přes URI `chrome://editor/content/editor.js`) obsahuje funkce a objekty specifické pro okno e-mailového editoru v Thunderbirdu. Jednotlivé funkce slouží například k inicializaci položek menu, pro přizpůsobování těchto menu aktuálnímu stavu editoru, inicializaci a zavírání editoru a obsluhu příkazů, stavových příznaků na těchto příkazech a event listenerů pro editor. Většina funkcí v tomto souboru se překvapivě vztahuje ke konkrétní podobě grafického prostředí takové, jaká je použita v okně

`messengercompose`, aby nad nimi mohla být aplikována overlay `editor.xul`. Nejsou psány univerzálně pro libovolné použití editoru třeba v jiných oknech, pokud by takové okno ovšem neobsahovalo naprosto stejnou strukturu xul kontejnerů a dalších tagů se stejnými identifikátory, jaké jsou ve scriptu `editor.js` a overlayi `editor.xul` na pevno uloženy.

Některé funkce souboru `editor.js` však jsou využitelné pro univerzální použití. Dle jejich podoby a funkcionality by možná měly patřit spíše do souboru `editorUtilities.js`, který obsahuje právě funkce vyvolávající konkrétní stylovací příkazy (viz dále).

Uvádím seznam funkcí použitelných pro stylování:

- EditorSetFontSize(size) - nastavuje velikost písma na velikost vyjádřenou textem (například: „xx-small“)
- EditorSelectColor(colorType, mouseEvent) - otevře collor picker a následně nastaví barvu textu
- GetAlignmentString() - vrátí zarovnání aktuálního odstavce

Soubor `editorUtilities.js` [6] naproti předchozímu implemetuje funkce pro práci s konkrétní instancí editoru. Nachází se na `chrome://editor/content/editorUtilities.js`. Zjednodušuje přístup k funkcím pro práci se samotným textem v místech, kde není tato funkcionalita implementována v některém ze scriptů z části `messengercompose`. Nutno dodat, že funkce v tomto souboru pracují za pomoci editačních XPCOM komponent přímo s HTML elementy editovaného dokumentu a jedná se tedy o velmi nízkoúrovňové úpravy textu.

Podobně jako v předchozím případě uvádím seznam funkcí užitečných pro stylovací komponentu.

- GetCurrentEditor() - Vrátí objekt implementující rozhraní nsIEditor, případně nsIHTMLEditor. Tato metoda vrací objekt pouze pro první nalezený editor. Současná implementace nepodporuje více než jeden editor na XUL okno.
- IsHTMLEditor() - V případě, že vrátí false, jedná se o plaintext editor a stylování není podporováno.
- EditorGetTextProperty(property, attribute, value, firstHas, anyHas, allHas) - Poslední tři parametry této metody jsou prázdné objekty. Do atributu `value` těchto oběktů se uloží boolean návratové hodnoty. Pokud mají první, respektive jakýkoli, či všechny tagy v selekci atribut danné hodnoty na danné property, nastaví se příznak na příslušném objektu.
- EditorSetTextProperty(property, attribute, value) - Nastaví či přepíše hodnotu atributu pro aktuální selekci či okolo kurzoru, v případě, že není označen žádný text.
- EditorRemoveTextProperty(property, attribute) - Zruší výskyt danného tagu s danným atributem pro aktuálně označený text, nebo okolo kurzoru v případě, že není označen, žádný text.
- GetHTMLOrCSSStyleValue(element, attrName, cssPropertyName) - Slouží k získávání informace o konkrétní vlastnosti textu.

Script `editorUtilities.js` také není vložen přímo do okna `messengercompose.xul`, nýbrž do jeho rozšiřujícího souboru `editorOverlay.xul` [5], který obsahuje definice příkazů pro samotnou editaci a vytváří konkrétní editační ovládací prvky, zmiňované v kapitolách 2.2 a 4.1.3.

### 4.1.5 XBL binding tagu editor

Tento binding je na XUL tag editor navázán v hlavním globálním kaskádovém stylu `xul.css` [11] následujícím kódem:

```
editor {
  -moz-binding: url("chrome://global/content/bindings/editor.xml#editor");
}
```

Ve chvíli kdy je binding [7] aplikován na element editor, zavolá se jeho konstruktor. Pokud je elementu editor nastaven atribut `editortype` je dokument obsažený v editoru automaticky nastaven jako editovatelný. V opačném případě je potřeba nastavit editovatelnost manuálně (viz. 4.1.2).

XBL binding dále definuje na elementu editor metody pro získávání XPCOM komponent `getEditor(containingWindow)` a `getHTMLEditor(containingWindow)`, kterým se jako parametr předkládá DOM objekt okna obsaženého v editoru.

Krom těchto metod specifikuje binding ještě několik property spojených s editovaným dokumentem. Pro naše účely jsou podstatné property `contentDocument` a `contentWindow`. Obě jsou property pouze pro čtení. Označují DOM objekty `document` a `window` editovaného HTML dokumentu. Při dotazu na kterýkoli z nich je DOM objekt vrácen zabalen v JavaScriptovém wrapperu. Jedná se o bezpečnostní opatření. Editovaný dokument je potenciálně neznámého původu, a proto je možné, aby některé jeho metody byly úmyslně přepsány a tak zneužity. Wrapper sám poskytuje přeimplementované metody, které je pak bezpečné použít, neboť zabalený (wrapped) objekt k nim nemá přístup. Bohužel pro práci s vnitřním dokumentem ve stylovací komponentě je třeba přistupovat k editovanému dokumentu blíže než wrapper umožňuje. Proto se skrz atribut wrappedJSObject přistupuje rovnou k nezabaleným objektům. Z bezpečnostních důvodů je třeba zde tento objekt zabalit do nového, vlastního wrapperu, který implementuje všechny potřebné metody. Jde navíc o jedno z kritérií pro možnost umístit rozšíření do AMO databáze add-onů.

### 4.1.6 Metody poskytované XPCOM komponentami editoru

Pomocí metody `getEditor()` respektive `getHTMLEditor()` lze z XUL elementu `editor` získat objekty XPCOM komponent implementující rozhraní definované v IDL souborech `nsIEditor.idl` a `nsIHTMLEditor.idl`. Tyto objekty obsluhují opravdovou editaci dokumentu.

#### 4.1.6.1 rozhraní nsIEditor

V rozhraní nsIEditor [18] jsou obsaženy metody pro obecnou editaci a práci s běžným textem. Je zde obsažena základní práce se selekcí, práce s transakcemi editace včetně undo/redo

příkazů pro průchod historií editace. Dále tento interface zpřístupňuje spellChecker a kontrolu hláskování slov [2]. Z dalších zde lze nalézt metody pro práci s clipboardem, manipulaci s textem myší drag/drop a v neposlední řadě nízkoúrovňovou práci s DOM Elementy editovaného HTML dokumentu. Konkrétní výčet metod je dostupný nejlépe na MXR přímo u zdrojových kódů definice interface. [8]

#### 4.1.6.2 rozhraní nsIHTMLEditor

V rozhraní nsIHTMLEditor [9] implemetuje editor metody pro nastavování takzvaných „inline property“. To znamená, že metody obalí aktuální selekci nebo kurzor nějakým HTML tagem. Pokud jsou k volání přidány i atributy, nastaví se tomuto HTML tagu také. K dispozici jsou také metody pro odnastavení atributy, nebo property na označeném textu.

Dále jsou v tomto rozhraní poskytovány metody pro zvětšení a zmenšení velikosti textu, pokročilou práci se selekcí obsahu s DOM strukturou, nastavování a získávání informací o odstavcích i inline vlastnostech, práci s odsazením, zarovnáním a dále také metoda pro vkládání a odmazávání odkazů. Pro implementaci vlastního editoru jsou právě zde k dispozici metody pro vkládání a přepisování HTML obsahu dokumentu.

Krom metod jsou zde definovány i některé důležité parametry pro chování editoru, například `returnInParagraphCreatesNewParagraph`, který rozhoduje, zda se po stisku klávesy enter jen zalomí řádek, nebo jestli se vytvoří další odstavec.

## 4.2 Analýza formátování textu v okně messengercompose

V této části budu takzvaně „tracovat“ formátovací příkazy, které uživatel zadává. Budu pronásledovat volání metod z pohledu samotného Thunderbirdu od uživatelského vstupu až po samotné provedení editace. Záměrem této úlohy je správně pochopit jak editor doopravdy funguje, kde vznikají chyby a kde je nejlépe na něj při implementaci stylovacího panelu napojit.

Pro nalezení optimálního řešení pro stylovací panel je třeba se podrobně zaměřit na funkcionalitu kolem formátovací části komponenty. Formátovací příkaz je vždy vyvolán uživatelským vstupem. Analýzu tedy začneme na XUL tlačítkách, menu Formát a klávesových zkratkách.

Pro tuto analýzu je nejlépe používat nástroj DOMInspector. Ten nám poskytuje náhled pod povrch grafického prostředí, a ukazuje všechny vlastnosti XUL elementů v něm. Snadno proto dohledáme vazby mezi jednotlivými XUL elementy a funkčním kódem. Navíc je v DOMInspectoru vidět i stuktura editovaného dokumentu. Lze proto přesně zkoumat výsledný efekt formátovacího příkazu.

### 4.2.1 Uživatelský vstupu

První příkaz který podrobíme zkoumání je příkaz „Bold“ pro aplikování stylu „tlustý text“. Tento příkaz lze vyvolat buďto stiskem tlačítka na formátovacím panelu, tlačítkem z menu Formát / Styl textu, nebo klávesovou zkratkou `Ctrl+B`.

[2] Poslední verze MAF aplikací podporují i myspell slovníky.

Z DOMInspectoru získáme aktuální XUL podobu těchto tří ovládacích prvků:

```
<toolbarbutton type="checkbox"
               autoCheck="false"
               observes="cmd_bold"
               tooltiptext="Bold"
               oncommand="doStyleUICommand('cmd_bold')"
               state="false"
               checkState="0"
               disabled="true"
               id="boldButton">

<observes element="cmd_bold"
          type="checkbox"
          attribute="state"
          onbroadcast="onButtonUpdate(this.parentNode, 'cmd_bold')" />

</toolbarbutton>

<menuitem id="menu_styleBold"
          label="Bold"
          accesskey="B"
          observes="cmd_bold"
          type="checkbox"
          key="boldkb"
          oncommand="doStyleUICommand('cmd_bold')"
          state="false"
          checked="false"
          acceltext="Ctrl+B"
          disabled="true" />

<key id="boldkb"
     key="B"
     observes="cmd_bold"
     modifiers="accel"
     oncommand="doStyleUICommand('cmd_bold')"
     state="false"
     disabled="true" />
```

Z výpisu kódu je jasně vidět, že všechny tři elementy mají společně atribut `observes` nastavený na stejnou hodnotu `cmd_bold`. Společné mají i atributy `oncommand` a `disabled`. To je však už pouze důsledek společného atributu `observes`. Hodnota atributu `observes` vždy obsahuje identifikátor nějakého XUL elementu, který funguje jako broadcaster. To znamená, že atributy, které jsou nastaveny na tomto broadcaster elementu se zároveň nastaví i na jeho „pozorovatelích“ (observers).

Najdeme si tedy, pomocí DOMInspektoru, XUL element s atributem `id` nastaveným na `cmd_bold`.

```
<command id="cmd_bold"
         state="false"
         oncommand="doStyleUICommand('cmd_bold')"
         disabled="true" />
```

Vidíme že se jedná o element `command`. Tento element nese informaci o funkci, která se provede při volání příkazu kterýmkoli ze tří zmíněných uživatelských vstupů. Zároveň se se dá říct, že chová jako broadcaster. Jeho pozorovatelé od něj získávají atributy `oncommand` a případně `disabled`. Shoda těchto atributů je také zřejmá z uvedených částí kódu.

Dále je z kódu možné zjistit, že tlačítko formátovacího panelu od elementu `command` získává také ještě atribut `state` pomocí elementu `observes`. O aktualizování tohoto atributu na elementu `command` se stará samotný editor. Jeho booleovská hodnota ukazuje, zda je aktuální text pod kurzorem v editoru formátován jako „tlustý text“.

Po bližším ohledání příkazu na databázi zdrojových kódů MXR [10] je zřejmé, že příkazy se do okna obsahujícího editor přidávají pomocí XUL overlaye souborem `editorOverlay.xul`. Tato overlay je zabudována přímo ve zdrojovém kódu okna `messengercompose` na řádku:

```
<?xul-overlay href="chrome://editor/content/editorOverlay.xul"?>
```

Při stisku jednoho z tlačítek, či klávesové zkratky se aplikuje formátování „tlustý text“. Při pohledu do DOMInspectoru se toto na editovaném dokumentu projeví vložením tagu `b`, tedy HTML tagu pro tlustý text, okolo kurzoru, nebo selekce.

Obdobným způsobem pokračujeme pro všechny ovládací prvky, které formátují text v okně editoru. Tabulka B.1 obsahuje seznam všech nalezených stylů, identifikátor jejich příkazu, HTML značku, která reprezentuje daný styl a funkci, která se pro aplikování stylu zavolá. Úmyslně zde neuvádím příkazy pro vkládání odkazů, obrázků a tabulek, jelikož se nejedná o stylovací příkazy. Jejich volání však probíhá do jisté míry obdobně.

### 4.2.2 Strom volání formátovacího příkazu

Z tabulky B.1 můžeme vyčíst, že většina formátování pro své aplikování používá stejné funkce. Pomocí MXR se tedy vydáme po stopách volaných funkcí. Jako příklad nám opět poslouží formátování tlustého textu.

1. Funkce `doStyleUICommand` se nachází ve scriptu `ComposerCommands.js`, který je součástí balíku kódů pro uživatelské prostředí editoru, tato funkce přidá pouze prázdný objekt parametrů příkazu (objekt CommandParams), a zavolá funkci `goDoCommandParams`. Ta jako parametry dostane příkaz (v našem případě `cmd_bold`) a právě vytvořený objekt CommandParams.

   Před dalším pokračováním v analýze bych rád zmínil, že funkce `doStatefulCommand`, kterou pro stylování používají odstavcové styly, dělá téměř totéž co právě analyzovaná funkce `doStyleUICommand` s tím rozdílem, že objektu CommandParams nastaví hodnotu `state_attribute` na text, který vyjadřuje typ nastavovaného odstavce. Tento text obsahuje název HTML tagu odstavce, tedy „p“, „h1“, „pre“ a další. Pro zrušení odstavce a ponechání textu volně v dokumentu se jako parametr posílá prázdný řetězec.

   Na tomto místě se však také volá funkce `goDoCommandParams` a příkazy jsou tedy nadále zpracovávány stejně.

2. Funkce `goDoCommandParams` pouze pomocí command dispatcheru získá objekt controller pro prováděný formátovací příkaz (stále `cmd_bold`) a na tomto controlleru zavolá metodu `doCommandWithParams`, které jen předá své parametry.

   Pro další „tracování“, musíme zjistit, jak commandDispatcher vyhodnotí formátovací příkaz a jaký controller vrátí. Jednou z možností, jak k tomuto dospět, je najít kód, kde je do command dispatcheru registrován příslušný příkaz a jeho controller, tedy kde je vytvořena vazba mezi objektem controlleru a textovým identifikátorem příkazu, podle kterého je vyhledáván.

   Na tomto místě se však projevuje jedna z klíčových vlastností vývoje na Mozilla Platformě a tou je udržení přehledného kódu a srozumitelné jmenné konvence. Místo zdlouhavého hledání stačí projít seznam cotrollerů implementujících volanou metodu `doCommandWithParams`, jako nejpravěpodobnější z nich se, pro formátování tlustého textu, brzy ukáže `nsBaseCommandController`, který je opravdu tím správným controllerem pro tento stylovací příkaz.

3. Nacházíme se v metodě:

   ```
   nsBaseCommandController::DoCommandWithParams(const char *aCommand,
                                                nsICommandParams *aParams)
   ```

   Jak už hlavička napovídá dostali jsme se command dispatcherem do kódu psaném v jazyce C++ a tedy do jádra editoru. Z toho plyne, že volání editačních příkazů je v okně e-mailového editoru ošetřeno právě cestou registrovaných příkazů, nikoli přímého volání funkcí. Toto řešení poskytuje vývojářům svobodu registrovat na danný příkaz jinou, například vývojovou verzi, controlleru, bez nutnosti změny jakékoli části grafického prostředí.

   Místo na command dispatcher si teď před dalším postupem budeme hrát na hashTable. Metoda `DoCommandWithParams` totiž znovu pouze vyhledá objekt v tabulce pojmenované `mCommandTable` a jeho metodě `DoCommandParams` jen předá své parametry, ke kterým navíc přidá objekt kontextu.

4. A to už se nacházíme ve zdrojovém souboru `nsComposerCommands.cpp`, který obsahuje soubor všech příkazů HTML editoru. z metody `DoCommandParams` se jen rovnou zavolá metoda `DoCommand`. V té se do kontextu přiřadí instance editoru, a konečně se zavolá metoda `ToggleState`.

5. Teprve v metodě `ToggleState` probíhá opravdová editace. Zjistí se aktuální stav požadovaného stylu, a podle něj se styl buď zruší nebo aplikuje, kde je třeba. Je zde navíc také obsloužena správa transakcí voláním metod editoru `beginTransaction` a `endTransaction`. Tím je zajištěno zachování historie editace, která se dá pak příkazy undo a redo, vracet a znovu aplikovat, jak jsme u editace zvyklí.

   Důležité na této metodě ale je, že veškerá editace probíhá nad instancí editoru voláním metod již několikrát zmíněných XPCOM komponent `nsEditor` a `nsHTMLEditor`. Na tomto místě, v C++ kódu není potřeba využívat XPConnect rozhraní pro získání objektu XPCOM komponent editoru, neboť ty jsou psány také v C++ a přístup k nim je zde nativní. Jedná se zkrátka o další C++ objekt.

Implementace v C++ nejspíše napomáhá performanci a urychlí vyhodnocování stylovacích příkazů. Stejného efektu, jakého je dosaženo funkcionalitou metody `ToggleState` lze totiž dosáhnout obdobným způsobem z JavaScriptu, kde je také možný přístup ke XPCOM komponentám editoru.

Pro úplnost uvedu, že ostatní příkazy mají v souboru `nsComposerCommands.cpp` také implementace svých handlerů. Lze zde najít metodu

```
nsListItemCommand::ToggleState
```

která vytváří či ruší položku číslovaného nebo prostého seznamu. Pro odstavce se pak použije metoda

```
nsParagraphStateCommand::SetState
```

která opět volá jen metodu nsHTMLEditoru `SetParagraphFormat(newState);` a lze tedy triviálně nahradit z JavaScriptu.

Příkazy pro úpravu velikosti písma a odsazení lze také najít v `nsComposerCommands.cpp`. Volání na jejich command coltrolleru však probíhá standardním způsobem bez parametrů a proto je namísto funkcí editoru použita globální funkce `goDoCommand`.

### 4.2.3 Midas editace

Nejen v editoru, ale obecně na všech HTML dokumentech zobrazených v některém z XUL rámových elementů, lze nastavit vlastnost `contentEditable` na `true` a tím zapnout editovatelnost jeho obsahu. Dokument je pak možné běžným psaním editovat a navíc je možné používat jeho metodu `execCommand` pro sofistikovanější úpravy[3]. Tato metoda očekává tři povinné parametry:

1. název Midas příkazu
2. příznak pro zobrazení UI
3. string s hodnotou pro příkaz

První parametr je řetězec obsahující příkaz, který se má provést. Druhý parametr je typu boolean a je povinně nastaven na false. V opačném případě skončí metoda s chybou. Třetí parametr je také řetězec reprezentující hodnotu příkazu. Některé příkazy, jako například nastavení velikosti textu, zde očekávají hodnotu, na kterou se má velikost nastavit.

V tabulce C.1 je seznam příkazů pro metodu `execCommand`, které nesou stylovací funkce. Záměrně neuvádím příkazy pro práci s historií editace, s clipboardem, či příkazy pro vkládání odkazů, ačkoli jsou touto metodou podporovány.

Vystopovat dále příkazy v MXR je časově poměrně náročná činnost. Příkaz `execComand` se nachází v kódu `nsHTMLDocument.cpp`. Podle kódu volá na každý příkaz také command dispatcher. Řetězce, kterými jsou příkazy identifikovány, jsou však před voláním přeformátovány podle tabulky z kódu v příloze D.

Jak je zde vidět, při použití Midas editace jsou specifické příkazy pouze mapovány na klasické příkazy editoru. Dále je postaráno o parametry příkazu a pak už jen pomocí command dispatcheru nalezen handler, stejně jako u ostatních způsobů editace.

### 4.2.4 Zhodnocení analýzy

Po urputném pronásledování tedy víme, že nezáleží na tom, jakým směrem se vydáme po stopách příkazů editoru. Při stylování můžeme využít buď metody samotných XPCOM komponent jádra editoru, to v případě, že budeme chtít průběh stylování naprosto kontrolovat. Těchto metod nakonec využívá každá implementace, a jsou nejspodnějším stavebním prvkem pro práci s editovaným dokumentem.

Nebo využijeme command controllerů registrovaných na command dispatcheru pod jednotlivými příkazy editoru. Toto jsou už hotové metody, které k formátování využívá Thundebird. Neumožňují však stylování v jednom kroku a uživatel pak může být zmaten při pohybu v historii editace, kdy se jeden styl aplikuje v několika krocích. K této funkcionalitě je třeba přistupovat buď přes Midas příkazy, nebo napřímo s využitím command dispatcheru, či s využitím již hotových scriptů okna `messengercompose`.

Podstatným zjištěním ze seznamu Midas příkazů je navíc fakt, že editor podporuje i mód, ve kterém místo HTML tagů formátuje text pomocí kaskádových stylů. Toto totiž není defaultní chování pro editor v okně `messengercompose`, který v základu styluje právě pomocí HTML tagů. Naopak Midas editace automaticky zapíná příznak `styleWithCSS`, a tedy po nastavení editovaného dokumentu jako `contentEditable` se zároveň změní i chování formátovacích příkazů a použijí se kaskádové styly.

# Kapitola 5

# Implementace

Tato kapitola se zabývá vyvýjeným rozšířením ExtBrain Editor. Nejprve popíšu jeho XUL strukturu v okně `messengercompose`. Dále se zastavím u rozhodování o skutečně implementovaných způsobech formátování textu. Nadále nastíním detailněji hlubší strukturu rozšíření a zaměřím se na zajímavé programátorské obraty a na poznatky o aplikaci získané během programování. Nakonec bych rád uvedl, jakým směrem by měl vývoj dále směřovat, dle mého názoru a podle zkušeností, které jsem na práci získal.

## 5.1 ExtBrain Editor stylovací panel

Stylovací panel obsahuje seznam všech textových stylů použitých v dokumentu. Navíc se v něm zobrazují i uživatelem uložené styly, umožňuje styly editovat aplikovat, rušit, a to jak inline tak odstavcové styly.

### 5.1.1 XUL struktura

XUL struktura stylovacího panelu počítá s tím, že element `editor` je obsažen v XUL kontejneru typu box. Tomuto kontejneru je při inicializaci ExtBrain Editoru nastaveno horizontální uspořádání pro zobrazení komponent vedle sebe. Při pokusech tento box-container vytvořit kolem editoru přímo ze scriptu a neomezovat tak více vývojáře XUL okna, které bude ExtBrain Editor užívat, jsem narazil na problém, že manipulace s editorem během inicializace způsobuje nedefinovatelné chování. Během pokusů se mi nepodařilo element editor přesouvat po okně tak, abych nenarušil proces načítání vnitřního dokumentu. Naštěstí je dobrou praxí přávě pro takovéto potřeby všechny prvky grafického prostředí umísťovat do kontejnerů. Snad to tedy nebude pro další vývojáře přílišná zátěž.

Samotný panel je tvořen XUL elementem `arrowscrollbox`, který se používá převážně u vysouvacích menu, jako je kontextové menu, nebo nabídky hlavního menu okna. V případě, že obsah scrollboxu přesáhne jeho výšku, zobrazí se na horním a dolním okraji scrollovací šipky a pomocí plynulého scrollování, je možně procházet celým obsahem panelu. Krom tohoto chování se jedná o prostý kontejnerový element a lze do něj libovolně vkládat tlačítka stylů i oddělovače.

Mezi panelem a oknem editoru se nachází `splitter`, který umožňuje měnit velikost panelu podle potřeb uživatele, například v případě, že popisek stylu přesáhne až za okraj zobrazitelné oblasti okna, nebo když je potřeba více místa pro psaní. Navíc lze pomocí tohoto splitteru panel dočasně úplně schovat.

XUL zdrojový kód prázdného seznamu stylů pak vypadá takto:

```
<vbox orient='horizontal'>
  <editor ... />
  <splitter collapse="after"><grippy /></splitter>
  <arrowscrollbox id="extbStyleList" orient="vertical" flex="1" />
</vbox>
```

Původní implementace „natvrdo“ do XUL overlaye okna s editorem, v tomto případě do okna `messengercompose`, vyžadovala tento kód v každém XUL dokumentu, kde by byl panel použit. Namísto toho, teď pro panel vznikla samotná inicializační metoda ve scriptech ExtBrain Editoru a vytváří se dynamicky pomocí JavaScriptu. Stačí tedy do cílového XUL dokumentu vložit referenci na scripty rozšířeni a závislých knihoven a zavolat jedinou metodu `jQuery.extbraineditor(id)` s parametrem `id` cílového tagu `editor`. Tento princip byl inspirován architekturou JavaScriptového editoru WYMEDITOR.

Závislosti na jQuery, jQuery.JSON a případně scriptech editor.js a editorUtilities.js by se daly taktéž řešit přímo z JavaScriptu přidáním `script` tagu do dokumentu. Muselo by se ale čekat na jejich načtení, před zavoláním inicializačních metod, vyřešit duplicity, či opakované inicializace těchto scriptů. Vývojářům by to ale možná přeci jen ušetřilo trochu starostí. Zde je zejména prostor pro diskusi a nalezení kompromisu, ve chvíli kdy se ExtBrain Editor stane součástí celého projektu.

## 5.2 Výběr způsobu formátování

Jak již bylo několikrát zmíněno pomocí editoru v Mozilla Application Frameworku lze aplikovat formátování buď v podobě HTML tagů, či v pohodě kaskádových stylů. Výběr metody pak ovlivní způsob. Jakým budeme se styly pracovat, jak je budeme uchovávat i jakým způsobem je budeme aplikovat. Popíšu teď blíže chování jednotlivých způsobů.

### 5.2.1 HTML formátování

Při tomto způsobu formátování se setkáváme s několika skupinami HTML tagů.

Základní formátovací tagy jako `b`, `u`, `i`, které se různě vnořují a nastavují tak vlastnost textu, který je uvnitř nich uzavřen.

Tagy které se vyznačují svým speciálním chováním, například tagy `sub` a `sup` pro spodní a dolní index, které se navzájem vylučují, či tagy `small` a `big` pro zmenšení a zvětšení písma, které se sčítají do výsledné velikosti textu. Dále také ostatní „exotické“ tagy `strong`, `acronym` a další, které mají ve formátování speciální význam.

Velmi používané jsou odstavcové tagy: `p`, `h1 - h6`, `pre` a `address`, ze kterých je vždy nastaven nejvýše jeden. Tyto tagy jsou vždy aplikovány minimálně na celý řádek textu,

nehledě na to, jak veklá část textu je označena. Pokud text není uzavřen v žádné z těchto značek, je označen jako „body text“ tedy text přímo v těle dokumentu, neuzavřený v odstavci.

Dále se setkáváme se seznamy `ul`, `ol` a v nich se značkami `li` pro každou položku seznamu. Tyto tagy se také vylučují s předchozými, odstavcovými tagy a v případě kolize s odstavcem, či nadpisem nastaví formát odstavce na „body text“ a tím zruší odstavcové styly okolo seznamu. Zde dochází k dost podivnému chování při kombinaci odstavců, odsazení a zarovnání.

Krom samotných odstavců se setkáváme i s tagy pro jejich umistění na stránce. Odsazení se provádí přidáváním a ubíráním tagu blockquote kolem aktuálního odstavce, či řádku.

U zarovnání je situace trochu komplikovanější. Zarovnání se provádí atributem `align` na odstavcovém tagu. Pokud se však kurzor nachází v těle dokumentu, tedy nemáme odstavcový styl pro nastavení tohoto atributu, je kolem řádku textu vložen tag `div` a zarovnání je nastaveno na něm.

Jedním ze specifičtějších tagů je pak tag `font`. Tento zprostředkovává stylování hned třemi atributy: size, pro nastavení velikosti textu, color, pro nastavení barvy textu a face, pro nastavení použitého fontu. Tento tag nepodporuje nastavení barvy pozadí textu, v HTML stylování proto lze nastavit pouze barvu pozadí celého dokumentu. Tento tag je navíc označen jako deprecated [23] a neměl by tak být používán.

Jak je vidět jde v případě HTML formátování o směsici HTML značek a jejich atributů. Krom toho, že některé z nich jsou zastaralé, je v řadě případů také nemožné předem určit chování stylovacích příkazů. Například pokud chceme zároveň nastavit font i barvu označeného textu, nastaví se oboje na jednom tagu `font`. Při změně označení či jakémukoli překryvu tagů se však vkládají samostatné tagy `font` po celé délce označeného textu.

K podobným nepředvídatelným projevům dochází i při kombinaci formátování odstavec, seznam, zarovnání. Pokud zarovnáme odstavec a převedeme ho na seznam, zruší se jeho zarovnání. Pokud však zarovnáme seznam na střed a přepneme ho na odstavec, nejen, že se seznam nezruší, odstavec navíc zůstane uzravřen v zarovnávacím `div` tagu.

Za zmínku stojí také fakt, že tagy `big` a `small` se při opakovaném přenastavování velikosti textu neustále vrší na sebe a mimo jiné tak tvoří velmi hluboké větve v DOM stromu a zpomaluje práci nad ním. Navíc kvůli workaroundům v metodách pro aplikování stylů dochází při kombinaci stylování pomocí font tagu a zároveň těchto dvou značek k chybnému chování a velikost textu se nemění podle očekávání.

### 5.2.2 CSS formátování

U formátování pomocí kaskádových stylů je situace v mnohém obdobná. Stále zde zůstávají odstavcové styly, tagy se speciálním významem a chováním a seznamy ve formě jejich HTML značek.

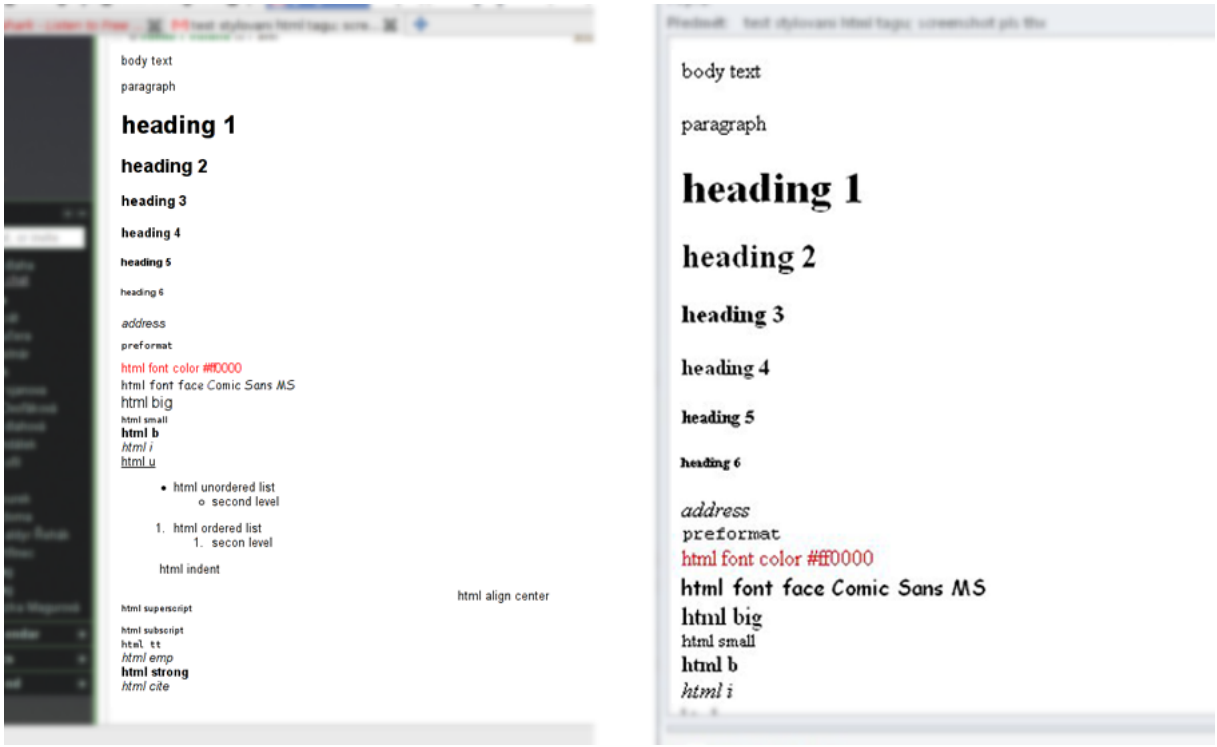
Základní formátovací tagy a tag `font` jsou všechny nahrazeny tagem `span` nesoucím jediný atribut `style` obsahující kaskádový styl který odpovídá nastavenému formátovnání textu.

Zarovnání a odsazení se namísto atributu `align` a tagu `blockquote` odehrává také přes atribut `style`. V tomto případě je atribut style nastavován stejně jako původně atribut `align` přímo na odstavcových značkách, případně na značce `div`.

Bohužel zde rozdíly ve způsobech formátování končí. Chyby při formátování a nedefinované chování zůstávají i zde stejné, jen se změní jména tagů a atributů. U tohoto formátování však lze navíc snadno za pomoci CSS vykreslit náhled výsledného stylu na tlačítkách stylovacího panelu. Výsledkem CSS formátování je navíc validní HTML dokument.

## 5.3 Podpora stylování v předních e-mailových klientech

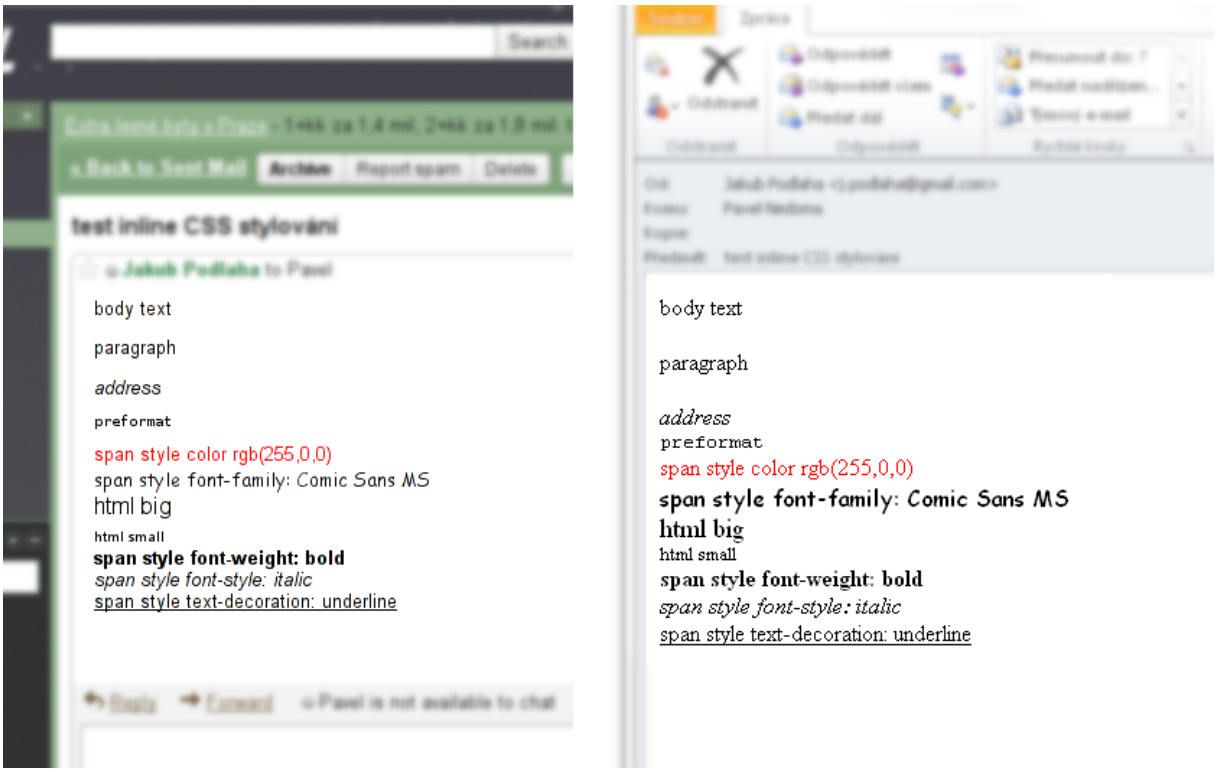
O podpoře stylování v emailových klientech se nedá nalézt příliž mnoho informací. Některé zdroje se rozcházejí v tom, jak je podporováno CSS stylování z tagu `style` v hlavičce dokumentu. Rozhodl jsem se proto alespoň inline stylování a podporu HTML formátovacích tagů otestovat na, v mém okolí nejpoužívanějších, e-mailových klientech.




Obrázek 5.1: Porovnání MS Outlooku a Gmailu pro HTML stylování

- Microsoft Outlook - největší konkurent Thunerbirdu. Jedná se o proprietární software s obdobnou funkcionalitou jakoku má Thunderbird.
- Gmail.com - coby zástupce freemailových webových klientů. Jedná se o velmi populární emailovou schránku od společnosti google.

Na obrázcích 5.1 a 5.2 je k nahlédnutí srovnání jak vypadají e-maily stylované pomocí HTML i CSS, v těchto dvou emailových klientech. Jak je vidět až na rozdíly ve velikostech,




Obrázek 5.2: Porovnání MS Outlooku a Gmailu pro CSS stylování

které mohou být částečně zkreslené rozdílnými grafickými parametry při pořizování screenshotů, jsou obě podoby velmi podobné, a hlavně jsou oba způsoby stylování plně podporovány.

V případě Gmailu lze namítnout, že o vykreslení zprávy se zde stará prohlížeč, zde Mozilla Firefox. Vnitřně jde tedy stejně jako u Thuderbirdu o praci Gecko enginu. O zpracování e-mailové zprávy se však stará server na straně Gmailu a prohlížeči je tak posílána jen výsledná webová podoba e-mailového klienta i se zprávou. Tímto testem bylo zjištěno, že pro e-mailové zprávy není v těchto prohlížečích problém použít a správně vykreslit veškeré podporované formátování ve obou podobách.

Zde bych rád dodal, že při testech použití tagu `style` v hlavičce emailové zprávy, došlo k řadě komplikací, které znemožnily test úspěšně provést. Nepodařilo se mi pracovat s uloženou emailovou zprávou. Práci s hlavičkou editované e-mailové zprávy jem byl také nucen po několika neúspěšných pokusech, kdy na ni editovaný dokument nereagoval, odložit. S novými poznatky by bylo možné na tyto testy znovu navázat. V další práci se však primárně zaměřím na dostupné způsoby formátování pomocí HTML a CSS v těle dokumentu.

## 5.4 Uchování stylů a práce s nimi

Způsob jakým ve stylovacím rozšíření zaznamenám jednotlivé styly ovlviní celou následující implementaci. Odvíjí se od toho metoda pro vyhledávání stylů v dokumentu, i pro samotné aplikování stylu při stisku tlačítka. Dále se toto projeví na implementaci tlačítek stylovacího panelu, které podle stylu zobrazují svůj popisek.

Volba způsobu formátování textu, jak byla popsána v předchozích odstavcích, ovlivní pouze text, který vytváří uživatel ve svých zprávách. Nepodaří se však ovlivnit zprávy, které uživateli posílají ostatní členové korespondence. Tito však typicky použijí kterýkoli ze způsobů stylování a proto je třeba postihnout oba.

Aby byly podporovány oboa způsoby stylování jsou styly textu uchovávány v JavaScriptovém objektu, který zaznamenává jak jednotlivé HTML tagy a jejich atributy, které ohraničují formátovaný text, tak součet kaskádových stylů, které utváří inline styl textu při CSS stylování.

Metody pro porovnání a aplikování stylů jsou pak přizpůsobeny použití těchto objektů.

## 5.5 stylelistbutton

Hlavním ovládacím prvkem stylovacího panelu jsou tlačítka jednotlivých stylů (náhled na obrázku ). Tato tlačítka jsou na panelu dvojího druhu. Uživatelské styly, tedy styly, které si uživatel uložil a zůstávají stále viditelné. Mezi těmito styly je zobrazen také prázdný styl, který slouží pro zrušení formátování. Příčnou čarou jsou pak odděleny vypočítané styly, tedy ty, které nejsou už součástí uživatelských stylů, ale jsou obsažené v editovaném dokumentu.


Obrázek 5.3: Náhled na stylovací panel

### XBL definice

Tlačítka stylovacího panelu jsou definována v XBL bindingu v rámci rozšíření. Tento binding definuje mimo jiné property `extbStyle` do které se ukládá objekt nesoucí styl přiřazený tomuto tlačítku. Z toho to stylu se následně vytváří náhled a text tlačítka, při stisku se pak předá stylovací metodě.

### styl tlačítka v grafickém prostředí

Vzhled tlačítka je definován v chrome adresáři skin rozšíření. Pomocí pokročilých CSS selektorů je zde postižen pohyb myší po tlačítkách, zvýraznění tlačítka při stisku a „zmrazení“ tlačítka, jako reakce na atribut `disabled`. Díky umístění na okraji tlačítka vytvářejí při pohybu myši po nich efekt vysouvání, díky jemnému gradientu a inverznímu obarvení okrajů pak snad potěší decentní 3D efekt. Tyto efekty jsou hlavně ukázkou možného vzhledu. Další úpravy velmi záleží na názorech uživatelů. Dosavadní ohlasy byly víceméně neutrální, což je dobrý výsledek. Stylovací panel nesmí uživatele ani zaujmout ani odpuzovat, aby nerušil od samotného psaní.

## 5.6 Vytváření tlačítek stylovacího panelu

Celý seznam stylů se vytváří dynamicky. Spolu s uživatelskými tlačítky se vytváří při inicializaci a pak při každé změně uživatelských stylů. Přepočítávání uživatelských stylů je namapováno jako handler na událost `commandupdate`, která je vyvolána vždy, když jsou změněny atributy command tagů, tedy zejména při samotném psaní do editovaného dokumentu.

Bohužel událost `commandupdate` není běžně vyvolána při formátovacích změnách na kurozru (nikoli na selekci) do té chvíle, než je na místě formátovaného kurzoru opravdu vložen nějaký text. Ve skutečnosti se do doby než uživatel po stisku formátovacího tlačítka napíše nějaký text, ani nevytvoří HTML tag reprezentující aplikovaný styl. Editor s vyvoláním události a vložením HTML tagů jednoduše čeká na uživatelský vstup proto, aby v dokumentu nezůstaly prázdné formátovací HTML tagy ve chvíli, kdy z nich uživatel beze vstupu překlikne.

Tuto situaci řeší Thunderbird pro svůj formátovací panel manuálním nastavením stavu příslušného formátovacího XUL příkazu při stisku tlačítka na formátovacím panelu. Například při stisku tlačítka pro tlustý text se manuálně nastaví stav jeho XUL příkazu.

```
<command id="cmd_bold"
         state="true"
         ...
 />
```

Tento stav je obnovem dalším uživatelským vstupem ať už pohybem kurzoru, kdy se příznak pro tlustý text zruší, či napsáním nějakého textu, který se zobrazí tlustě formátovaný a příznak pak zůstává nastaven. Pro potřeby stylovacího panelu je však přirozenější, aby odrážel skutečnou situaci v dokumentu. Infomrace o stavech příkazů tak, jak je vidí zbytek XULu však může být pro další vývoj užitečná a snadno se získá průchodem state atributu na XUL příkazech s `id` z tabulky B.1.

Procházení DOM stromu dokumentu a hledání dosud nezaznamenaných stylů při každé změně editovaného dokumentu se, zvláště při delších konverzacích stává výpočetně velmi náročnou úlohou. I při snaze ušetřit řadu operací bylo pro rozsáhlejší soubory zásadní zajistit, aby blokující přepočítávání nezdržovalo od uživatelských vstupů. Následující vzor nastiňuje, jak se provádí přepočítávání asynchronně a zároveň se zamezí jeho opakovanému volání.

```
function startThread() {
    var tmpvar;
    window.lockXY = false;
    var processInterval = window.setInterval( function(){
        window.lockXY = true;
        tmpvar = /*countSomething()*/;

        if ( /*end condition*/ )
            window.clearInterval(processInterval);

        window.lockXY = false;
        return;
    }, 100);
}
```

Pro použití této metody a další zrychlení výpočtu jsem přepsal původní rekurzivní procházení DOM stromu na zásobníkové. Toto je v JavaScriptu triviální úkol, jelikož všechny objekty typu Array se chovají zároveň pomocí metod `shift()`, `unshift()`, `push()` a `pop()` i jako fronta a zásobník.

Další méně známou technikou tentokrát pro urychlení práce s DOM stromem XUL dokumentu je použití objektu `documentFragment`. Tento objekt umožňuje aplikovat více úprav na DOM stromu najednou tak, že mezi jenotlivými úpravami nedochází k přerenderování měněného dokumentu, jako je tomu u přímých úprav. Při použití `documentFragment` pro rychlejší inicializaci stylovacího panelu se však nevolají metody setteru a getteru na property `extbStyle` tlačítek `stylelistbutton`.

Po delším testování jsem zjistil, že problém není v samotném `documentFragmentu`, ale až v mapování bindingu. Důvodem je, že CSS vlastnost pro binding se na element aplikuje až po jeho zařazení do DOM stromu dokumentu. V aktuální implementaci se tedy nejdříve přidá upravované tlačítko do XUL dokumentu a až následně se pracuje s jeho property. Jiným řešením by bylo manuální aplikování stylu `-moz-binding`, ještě před přidáním tlačítka do DOM struktury XUL dokumentu přes property `element.style.mozBinding`. V případě stylovacích tlačítek je však třeba šetřit spíše na zbytečných cyklech při hledání stylu, než při řádově méně častých úpravách GUI.

## 5.7 Práce se styly

V současné implementaci obsahuje ExtBrain Editor metody pro:

- ukládání a načítání seznamu uživatelských stylů,
- hledání použitých stylů v aktuálním dokumentu, jejich porovnání a případné sázení do seznamu v GUI,
- aplikování inline a odstavcových stylů.

Při ukládání rozšíření zapisuje uživatelské styly do svého adresáře ve formě JSON, tedy textové reprezentaci JavaScriptového objektu. Důležitým momentem je pak načítání těchto objektů zpět při startu editoru, kdy jsou načtené objekty přetypovány do jejich původní podoby přepsáním jejich prototype objektu, zejména kvůli toString() metodě, která se využívá pro porovnávání a tím i srovnávání stylů v poli.

Ukázka uložených stylů ve formě JSON souboru:

```
[
 { "name"  : "Monospace",
   "para"  : [ {"tag" : "pre", "atts" : []} ],
   "inline": [],
   "css"   : "" },
 {
  "name"  : "Bold",
  "para"  : [],
  "inline": [],
  "css"   : "font-weight: bold;" },
]
```

Při hledání použitých stylů, se prochází celý editovaný dokument a na každém samostatném textovém uzlu (jedná se o listy DOM stromu) se z jeho předků vypočítá jeho styl. Tento styl je pak poslán metodě, která, v případě, že už se v seznamu nenachází, přidá jeho tlačítko do GUI. Při vytváření stylu narážíme na kritickou část, kdy je třeba různé pořadí a formy stylů zaznamenat jednotým způsobem, aby se v seznamu neopakovaly. Navíc je třeba oddělit odstavcové styly od inline stylů. V této části se také nachází workaroundy na některé chyby Thnunderbirdu jakými je opakování tagů při změnách velikosti písma. V neposlední řadě je třeba se zde postarat o ignorování odsazovacích značek před starými zprávami při dlouhé konverzaci. Bohužel zde narážíme na odlišné zpracování odsazení starých zpráv různými e-mailovými klienty a nelze postihnout všechny jejich kombinace. Snahou této části je poskytnout uživateli co nepoužitelnější seznam stylů v co nejkratším čase.

Aplikování stylů je záležitostí pro neustálý vývoj. Aktuální verze používají příkazů Midas editace, neboť jejich funkčnost není vázána na dalších scriptech v jiných částech Thunerbirdu a tím se minimalizují programové závislosti. Metoda pro aplikování stylů je závislá na způsobu jakým zaznamenáváme styly. Jinak je od ostatních částí aplikace oddělena a využívá jen svých metod, reference na editor a stylu, který je jí předán jako parametr. Tuto metodu zatím volají jen při kliknutí jednotlivá tlačítka panelu.

## 5.8 Budoucí vývoj

Na základě analýzy a podle potřeb projektu ExtBrain se nyní bude odvíjet další vývoj tohoto rozšíření pro práci s editorem v Mozilla Thunderbirdu. Z vlastního pohledu bych navrhl dvě cesty, kterými je možné další vývoj vést.

Je možné se držet stávající implementace editoru a pokračovat ve vývoji rozšíření, jako doposud. Tato strategie vytváří pro potřeby ExtBrain projektu jakousi vrstvu, mapující

funkcionalitu editoru z Mozilla Application Frameworku na API dostupné všem rozšířením, které s ním potřebují pracovat.

Druhá cesta vývoje by mohla směřovat do hloubky. Mám na mysli „odstřihnout“ stávající implementaci editoru již na úrovni editačních příkazů a napojit se až na nízkoúrovňové metody editoru, či dokonce na úpravy samotného HTML zdrojového kódu. Tato cesta by zahrnovala implemetaci vlastních stylovacích metod a hlavně opravu všech stávajících výskytů editoru, nejspíš pomocí overlay.

Zatímco první cesta je vhodnější pro práci na editoru v okně `messengercompose`, tedy v e-mailovém editoru Mozilla Thunderbirdu, druhá se hodí spíše pro nové výskyty elementu editor v rozšířeních ExtBrainu, jakými mohou být editor poznámek, či chatovací IM klient.

Pomocí druhé cesty, implementace nových metod, vzniknou funkční opravy stávajících metod na chrome vrstvě aplikace. Ty by mohly napomoci inspiraci vývoje jádra editoru a tím obě cesty v důsledku sjednotit. Tato část však zahrnuje stoprocentní zapojení do vývoje Mozilla Thunderbirdu a zejména pro časovou náročnost ji nelze vměstnat do jedné práce.

Za cíl dalšího vývoje považuji, aby editor rozuměl jakkoli validně naformátovanému kódu zprávy, aby vytvářel jednoduchý, jasný HTML kód, aby veškeré vlastnosti dával k dispozici jednotným způsobem. Dále aby se znatelně zjednodušilo grafické prostředí editoru a zjednodušil přístup vývojářů k jeho funkcionalitě a definovalo jednotné API pro další práci na editoru použitelné na všech místech kde se vyskytuje.

# Kapitola 6

# Závěr

Cílem práce bylo vytvořit add-on pro Mozilla Thunderbird, který rozšíří možnosti editační komponenty o stylovací panel, a zároveň poskytne tuto funkcionalitu k zakomponování do projektu ExtBrain. Velký důraz jsem při vývoji kladl na hlubokou analýzu editoru a na zdokumentování jeho funkcionality, abych tím usnadnil práci následujícím vývojářům.

Výsledkem je funkční rozšíření v podobě panelu pro stylování v e-mailovém editoru Mozilla Thunderbirdu. Toto rozšíření usnadňuje uživatelům práci se složitějším formátováním textu. Může však úplně nahradit formátovací pruh a stát se tak jediným ovládacím prvkem pro formátování textu.

Při vývoji byly analyzovány možnosti stylování editační komponenty z Mozilla Application Frameworku a jejich funkcionalita zdokumentována a poskytnuta v podobě API pro další použití v projektu ExtBrain. Přehlednosti a srozumitelnosti zdrojových kódů jsem se snažil dosáhnout využitím pokročilých technologií Mozilla Application Frameworku a věřím, že jsem dle sých možností maximálně usnadnil navazující vývoj na ExtBrain Editoru.

Ačkoli bylo třeba, na základě stále nových poznatků analýzy, práci často upravovat, byla pro mě velým přínosem a napomohla mi získat řadu zkušenností, zejména možností podílet se na velkém open-source projektu. Je mi ctí, že prostřednictvím této práce mohu tyto zkušennosti poskytnout dalším vývojářům.

# Literatura


[1] David Boswell, Brian King, Ian Oescher, and Pete Collins. *Creating Applications with Mozilla*, volume 1. O'Reilly Media, 2002.

[2] Editor team Mozilla. Mozilla Editor — mozilla.org, 2001. <http://www.mozilla.org/editor>, stav z 15. 2. 2001.

[3] Editor team Mozilla. Midas Specification — mozilla.org, 2005. <http://www.mozilla.org/editor/midas-spec.html>, stav z 23. 8. 2005.

[4] Mozilla. editor.js — Mozilla Cross Reference, 2011. <http://mxr.mozilla.org/comm-central/source/editor/ui/composer/content/editor.js>, stav z 5. 1. 2011.

[5] Mozilla. editoroverlay.xul — Mozilla Cross Reference, 2011. <http://mxr.mozilla.org/comm-central/source/mail/components/compose/content/editorOverlay.xul>, stav z 5. 1. 2011.

[6] Mozilla. editorUtilities.js — Mozilla Cross Reference, 2011. <http://mxr.mozilla.org/comm-central/source/editor/ui/composer/content/editorUtilities.js>, stav z 5. 1. 2011.

[7] Mozilla. editor.xml — Mozilla Cross Reference, 2011. <http://mxr.mozilla.org/comm-central/source/mozilla/toolkit/content/widgets/editor.xml>, stav z 5. 1. 2011.

[8] Mozilla. nsIEditor.idl — Mozilla Cross Reference, 2011. <http://mxr.mozilla.org/comm-central/source/mozilla/editor/idl/nsIEditor.idl>, stav z 5. 1. 2011.

[9] Mozilla. nsIHTMLEditor.idl — Mozilla Cross Reference, 2011. <http://mxr.mozilla.org/comm-central/source/mozilla/editor/idl/nsIHTMLEditor.idl>, stav z 5. 1. 2011.

[10] Mozilla. vyhledávání: cmd_bold — Mozilla Cross Reference, 2011. <http://mxr.mozilla.org/comm-central/search?string=cmd_bold>, stav z 5. 1. 2011.

[11] Mozilla. xul.css — Mozilla Cross Reference, 2011. <http://mxr.mozilla.org/comm-central/source/mozilla/toolkit/content/xul.css>, stav z 5. 1. 2011.

[12] Mozilla Developer Network. -moz-binding — MDC Doc Center, 2010. <https://developer.mozilla.org/en/CSS/-moz-binding>, stav z 10. 8. 2010.

[13] Mozilla Developer Network. Adding Event Handlers to XBL-defined Elements — MDC Doc Center, 2010. <https://developer.mozilla.org/en/XUL_Tutorial/Adding_Event_Handlers_to_XBL-defined_Elements>, stav z 10. 8. 2010.

[14] Mozilla Developer Network. Adding Methods to XBL-defined Elements — MDC Doc Center, 2010. <https://developer.mozilla.org/en/XUL_Tutorial/Adding_Methods_to_XBL-defined_Elements>, stav z 10. 8. 2010.

[15] Mozilla Developer Network. Adding Properties to XBL-defined Elements — MDC Doc Center, 2010. <https://developer.mozilla.org/en/XUL_Tutorial/Adding_Properties_to_XBL-defined_Elements>, stav z 10. 8. 2010.

[16] Mozilla Developer Network. Chrome Registration — MDC Doc Center, 2010. <https://developer.mozilla.org/en/chrome_registration>, stav z 10. 8. 2010.

[17] Mozilla Developer Network. Gecko — MDC Doc Center, 2010. <https://developer.mozilla.org/en/Gecko>, stav z 21. 9. 2010.

[18] Mozilla Developer Network. nsIEditor — MDC Doc Center, 2010. <https://developer.mozilla.org/en/nsIEditor>, stav z 10. 8. 2010.

[19] Mozilla Developer Network. XPCOM — MDC Doc Center, 2010. <https://developer.mozilla.org/en/XPCOM>, stav z 21. 9. 2010.

[20] Mozilla Developer Network. XPIDL — MDC Doc Center, 2010. <https://developer.mozilla.org/en/XPIDL>, stav z 21. 9. 2010.

[21] Mozilla Developer Network. XUL — MDC Doc Center, 2010. <https://developer.mozilla.org/en/XUL>, stav z 10. 8. 2010.

[22] Mozilla Developer Network. The Chrome URL — MDC Doc Center, 2011. <https://developer.mozilla.org/en/XUL_Tutorial/The_Chrome_URL>, stav z 4. 1. 2011.

[23] W3 Schools. HTML fonts — w3 schools, 2010. <http://www.w3schools.com/html/html_fonts.asp>, stav z 23. 8. 2010.

[24] Wikipedie. Wikipedia — Gecko (layout engine), 2010. <http://en.wikipedia.org/wiki/Gecko_\%28layout_engine\%29>, stav z 10. 8. 2010.

[25] Wikipedie. Wikipedia — Mozilla Thunderbird, 2010. <http://en.wikipedia.org/wiki/Mozilla_Thunderbird>, stav z 9. 12. 2010.

# Příloha A

# Seznam použitých zkratek

**AMO** Add-ons Mozilla.org

**CSS** Casscading Style Sheet

**DTD** Document Type Definition

**GPL** General Public Licence

**HTML** HyperText Markup Language

**IM** Instant Messanger

**JSON** JavaScript Object Notation

**LGPL** Lesser General Public License

**MAF** Mozilla Application Framework

**MPL** Mozilla Public Licence

**RSS** Really Simple Syndication

**XBL** eXtensible Binding Language

**XML** eXtensible Markup Language

**XPCOM** Cross Platform Component Object Model

**XUL** XML User Interface Language

**WYSIWYG** What You See Is What You Get

# Příloha B

# Tabulka formátovacích příkazů

| **styl** | **command** | **html** | **metoda composeru** |
|---|---|---|---|
| tlustý text | cmd_bold | <b> | doStyleUICommand |
| kurzíva | cmd_italic | <i> | doStyleUICommand |
| podtržený text | cmd_underline | <u> | doStyleUICommand |
| přeškrtnutý text | cmd_strikethrough | <strike> | doStyleUICommand |
| spodní index | cmd_subscript | <sub> | doStyleUICommand |
| horní index | cmd_superscript | <sup> | doStyleUICommand |
| teletype | cmd_tt | <tt> | doStyleUICommand |
| zdůrazněný text | cmd_em |  | doStyleUICommand |
| zvětšený text | cmd_strong |  | doStyleUICommand |
| citace | cmd_cite | <cite> | doStyleUICommand |
| zkratka | cmd_abbr | <abbr> | doStyleUICommand |
| akronym | cmd_acronym | <acronym> | doStyleUICommand |
| počítačový kód | cmd_code | <code> | doStyleUICommand |
| ukázkový kód | cmd_samp | <samp> | doStyleUICommand |
| proměnná | cmd_var | <var> | doStyleUICommand |
| nečíslovaný seznam | cmd_ul | <ul> <li> | doStyleUICommand |
| číslovaný seznam | cmd_ol | <ol> <li> | doStyleUICommand |
| název výrazu | cmd_dd | <dd> | doStyleUICommand |
| definice výrazu | cmd_dt | <dt> | doStyleUICommand |
| odstavec | cmd_paragraphState | <p> | doStatefulCommand |
| preformat | cmd_paragraphState | <pre> | doStatefulCommand |
| nadpis 1.-6. úrovně | cmd_paragraphState | <h1> - <h6> | doStatefulCommand |
| zarovnání | cmd_align | align="`value`" | doStatefulCommand |
| smajlík | cmd_smile | `:-)` | doStatefulCommand |
| zvětšit písmo | cmd_increaseFont | <big> | goDoCommand |
| zmenšit písmo | cmd_decreaseFont | <small> | goDoCommand |
| indent | cmd_indent | <blockquote> | goDoCommand |
| outdent | cmd_outdent | „odmaže bq“ | goDoCommand |

Tabulka B.1: Tabulka příkazů pro html editor

# Příloha C

# Tabulka Midas příkazů

| command | hodnota | popis chování |
|---|---|---|
| backcolor | hexadecimální hodnota barvy | Tento příkaz nastaví barvu pozadí dokumentu. |
| hilitecolor | hexadecimální hodnota barvy | Pouze při zapnutém CSS formátování, nastavuje pozadí textu. |
| bold | — | Pokud není označen žádný text, aplikuje se „tlustý“ formát na následně napsané znaky. Pokud je označen text a některá jeho část není v tlustém formátu, aplikuje se na tuto část. Pokud je označen pouze tlustý text, zruší se tlustý formát pro celou označenou část. |
| italic | — | Stejně jako příkaz bold, jen nastaví text na kurzívu. |
| underline | — | Stejně jako příkaz bold, jen nastaví text na podtržený. |
| strikethrough | — | Stejně jako předchozí, nastaví text jako přeškrtnutý. |
| superscript | — | Jako předchozí, jen nastaví text jako horní index. |
| subscript | — | Jako předchozí, jen nastaví text jako spodní index. |
| increasefontsize | — | Vloží tag `big` kolem selekce, nebo kurzoru. |
| decreasefontsize | — | Vloží tag `small` kolem selekce, nebo kurzoru. |
| fontname | jméno fontu | Nastaví druh fontu selekci nebo na pozici kurzoru. |
| fontsize | velikost | Nastaví velikost písma na hodnotu v argumentu pro selekci, nebo na pozici kurzoru. |
| forecolor | hexa barva | Nastaví barvu textu pro selekci, nebo na pozici kurzoru. |
| formatblock | `h1 - h6` `address p` `pre` ... | Nastaví všem řádkům v selekci odstavec podle parametru. Jako parametr value lze použít i tvar bez závorek. |
| heading | `h1 - h6` | Totéž co předchozí, ale jen pro odstavce. |
| indent | — | Zvětší odsazení odstavce na pozici kurzoru. |
| outdent | — | Zmenší odsazení odstavce na pozici kurzoru. |
| insertorderedlist | — | Vloží číslovaný seznam. |
| insertunorderedlist | — | Vloží nečíslovaný seznam. |
| justifycenter | — | Zarovná text na střed. |
| justifyfull | — | Zarovná text do bloku. |
| justifyleft | — | Zarovná text do leva. |
| justifyright | — | Zarovná text do prava. |
| removeformat | — | Odstraní veškeré inline formátování. |
| styleWithCSS | true false | Defaultně true. Určuje zda bude použito CSS formátování, nebo formátování přes HTML tagy. |

Tabulka C.1: Tabulka Midas příkazů

# Příloha D

# Mapování midas příkazů pro command dispatcher

```
static const struct MidasCommand gMidasCommandTable[] = {
   { "bold",          "cmd_bold",            "", PR_TRUE,  PR_FALSE },
   { "italic",        "cmd_italic",          "", PR_TRUE,  PR_FALSE },
   { "underline",     "cmd_underline",       "", PR_TRUE,  PR_FALSE },
   { "strikethrough", "cmd_strikethrough",   "", PR_TRUE,  PR_FALSE },
   { "subscript",     "cmd_subscript",       "", PR_TRUE,  PR_FALSE },
   { "superscript",   "cmd_superscript",     "", PR_TRUE,  PR_FALSE },
   { "cut",           "cmd_cut",             "", PR_TRUE,  PR_FALSE },
   { "copy",          "cmd_copy",            "", PR_TRUE,  PR_FALSE },
   { "paste",         "cmd_paste",           "", PR_TRUE,  PR_FALSE },
   { "delete",        "cmd_delete",          "", PR_TRUE,  PR_FALSE },
   { "selectall",     "cmd_selectAll",       "", PR_TRUE,  PR_FALSE },
   { "undo",          "cmd_undo",            "", PR_TRUE,  PR_FALSE },
   { "redo",          "cmd_redo",            "", PR_TRUE,  PR_FALSE },
   { "indent",        "cmd_indent",          "", PR_TRUE,  PR_FALSE },
   { "outdent",       "cmd_outdent",         "", PR_TRUE,  PR_FALSE },
   { "backcolor",     "cmd_backgroundColor", "", PR_FALSE, PR_FALSE },
   { "forecolor",     "cmd_fontColor",       "", PR_FALSE, PR_FALSE },
   { "hilitecolor",   "cmd_highlight",       "", PR_FALSE, PR_FALSE },
   { "fontname",      "cmd_fontFace",        "", PR_FALSE, PR_FALSE },
   { "fontsize",      "cmd_fontSize",        "", PR_FALSE, PR_FALSE },
   { "increasefontsize", "cmd_increaseFont", "", PR_FALSE, PR_FALSE },
   { "decreasefontsize", "cmd_decreaseFont", "", PR_FALSE, PR_FALSE },
   { "inserthorizontalrule", "cmd_insertHR", "", PR_TRUE,  PR_FALSE },
   { "createlink",    "cmd_insertLinkNoUI",  "", PR_FALSE, PR_FALSE },
   { "insertimage",   "cmd_insertImageNoUI", "", PR_FALSE, PR_FALSE },
   { "inserthtml",    "cmd_insertHTML",      "", PR_FALSE, PR_FALSE },
   { "gethtml",       "cmd_getContents",     "", PR_FALSE, PR_FALSE },
   { "justifyleft",   "cmd_align",       "left", PR_TRUE,  PR_FALSE },
   { "justifyright",  "cmd_align",      "right", PR_TRUE,  PR_FALSE },
```

```
{ "justifycenter", "cmd_align",     "center", PR_TRUE,  PR_FALSE },
{ "justifyfull",   "cmd_align",    "justify", PR_TRUE,  PR_FALSE },
{ "removeformat",  "cmd_removeStyles",    "", PR_TRUE,  PR_FALSE },
{ "unlink",        "cmd_removeLinks",     "", PR_TRUE,  PR_FALSE },
{ "insertorderedlist",   "cmd_ol",        "", PR_TRUE,  PR_FALSE },
{ "insertunorderedlist", "cmd_ul",        "", PR_TRUE,  PR_FALSE },
{ "insertparagraph", "cmd_paragraphState", "p", PR_TRUE, PR_FALSE },
{ "formatblock",   "cmd_paragraphState",  "", PR_FALSE, PR_FALSE },
{ "heading",       "cmd_paragraphState",  "", PR_FALSE, PR_FALSE },
{ "styleWithCSS",  "cmd_setDocumentUseCSS", "", PR_FALSE, PR_TRUE },
{ "contentReadOnly", "cmd_setDocumentReadOnly", "", PR_FALSE, PR_TRUE },
{ "insertBrOnReturn", "cmd_insertBrOnReturn", "", PR_FALSE, PR_TRUE },
{ "enableObjectResizing", "cmd_enableObjectResizing", "", PR_FALSE, PR_TRUE },
{ "enableInlineTableEditing", "cmd_enableInlineTableEditing", "", PR_FALSE, PR_TRUE },
```

# Příloha E

# Instalační a uživatelská příručka

Rozšíření v podobě souboru XPI se do e-mailového klienta Mozilla Firefox instaluje přes menu "Nástroje / Správce doplňků". V okně Správce doplňků klikěte v levém dolním rohu na tlačítko "Instalovat..."a vyberte xpi balíček s rozšířením.

Rozšíření přidá stylovací panel do editačního okna a zobrazí v něm výchozí styly. Po stisku pravým tlačítkem lze z kontextové nabídky s panelem dále pracovat, například ukládat, odmazávat, či přejmenovávat styly.

# Příloha F

# Obsah přiloženého CD

```
/
|-- install.txt                postup instalace rozšíření
|-- README.txt                 soubor s popisem obsahu CD
|-- data                       adresář s datovým obsahem CD
|   `-- ...
|-- src                        adresář se zdrojovými kódy rozšíření
|   `-- ...
|-- text                       adresář obsahující vlastní text bakalářské práce
|   `-- bakalarska-prace.pdf   text bakalářské práce ve formátu PDF
`-- build                      adresář s instalačním balíčkem rozšíření
    `-- extbrain-editor-0.2.xpi instalační balíček rozšíření
```