VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

# ANOTAČNÍ DOPLNĚK PRO FIREFOX

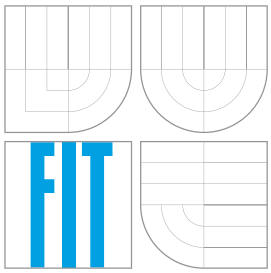
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JIŘÍ TRHLÍK

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

# ANOTAČNÍ DOPLNĚK PRO FIREFOX

ANNOTATION ADDON FOR FIREFOX

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JIŘÍ TRHLÍK
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAROSLAV DYTRYCH
SUPERVISOR

BRNO 2011

**Vysoké učení technické v Brně - Fakulta informačních technologií**

Ústav počítačové grafiky a multimédií Akademický rok 2010/2011

# Zadání bakalářské práce

Řešitel: **Trhlík Jiří**
Obor: Informační technologie
Téma: **Anotační doplněk pro Firefox**
**Annotation Addon for Firefox**
Kategorie: Web

Pokyny:

1. Seznamte se s možnostmi tvorby doplňků ve webovém prohlížeči Firefox.
2. Seznamte se technologiemi potřebnými pro tvorbu doplňků do webového prohlížeče.
3. Navrhněte doplněk pro Firefox, který umožní anotovat text webové stránky a zasílat fragmenty anotovaného textu s anotacemi na server. Doplněk musí umožnit také nastavení parametrů na serveru.
4. Implementujte navržené řešení.
5. Zhodnoťte dosažené výsledky a navrhněte další možná vylepšení do budoucna.

Literatura:

- podle dohody

Při obhajobě semestrální části projektu je požadováno:

- Body 1, 2 a 3.

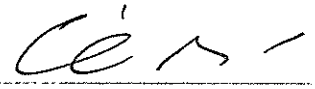
Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

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

Vedoucí: **Dytrych Jaroslav, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2010
Datum odevzdání: 18. května 2011

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



doc. Dr. Ing. Jan Černocký
*vedoucí ústavu*

# Abstrakt

Tato bakalářská práce analyzuje anotace a proces vytváření doplňku pro Mozzila Firefox. Tyto problémy stručně popisuje a zaměřuje se na návrh a implementaci doplňku, který dokáže vytvářet a ukládat strukturované anotace.



# Abstract

This bachelor's thesis aims to analyse annotations and process of creating a Mozzila Firefox addon. It provides a brief description of both problems and focuses on a design and an implementation of a Mozilla Firefox addon, which can create and store structured annotations.



# Klíčová slova

Mozilla Firefox, doplněk, anotace, strukturovaná anotace, uživatelské rozhraní, XUL

# Keywords

Mozilla Firefox, addon, annotation, structured annotation, user interface, XUL



# Citace

Jiří Trhlík: Anotační Doplněk pro Firefox, bakalářská práce, Brno, FIT VUT v Brně, 2011

# Anotační Doplněk pro Firefox

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jaroslava Dytrycha. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jiří Trhlík
18. května 2011

## Poděkování

Děkuji svému vedoucímu práce Ing. Jaroslavu Dytrychovi za odbornou pomoc a cenné rady při řešení této práce.

# Obsah

# Kapitola 1

# Úvod

Cílem této práce je navrhnout a implementovat anotační doplněk pro Firefox, který bude součástí implementace pro ověření konceptu systému 4A Framework. Doplněk se má přihlásit k anotačnímu serveru a umožnit vytváření a ukládání strukturovaných anotací na tento server.

V kapitole 2 jsou vysvětleny důležité pojmy, jako je Firefox a doplněk.

Kapitola 3 obsahuje analýzu procesu vytvoření doplňku. Je zde především popsaná struktura doplňku a jeho základní části. Dále jsou zde uvedeny některé principy, které se při vytváření doplňku používají. Nakonec jsou zmíněny potřebné nástroje pro vývoj doplňku.

Aby bylo možné implementovat doplněk pro vytváření anotací, je třeba také provést analýzu anotace. Stručný popis, co je anotace, k čemu slouží a její vlastnosti jsou uvedeny v kapitole 4. V této kapitole je také popsán systém 4A Framework, pro který má být doplněk určen, a je popsána komunikace v tomto systému.

Na základě provedených analýz byly zvoleny technologie, které budou využity k vývoji doplňku. Ty jsou uvedeny v kapitole 5. Jsou zde také stručné charakteristiky technologií a jejich aktuální stav.

Návrhu doplňku se věnuje kapitola 6. Nejdříve popisuje grafické uživatelské rozhraní, poté se věnuje samotné funkčnosti a procesu vytváření anotace. U konce kapitoly je popsána důležitá reprezentace dat pro stromy, které zobrazují typy nebo atributy anotací.

Kapitola 7 obsahuje základní popis implementace a souborů se zdrojovými kódy.

V kapitole 8 je popsán proces testování, na kterých systémech bylo testování provedeno a výsledky testů.

V 9. kapitole je zhodnocení práce a dosažené výsledky. Jsou zde také uvedena možná rozšíření do budoucna.

# Kapitola 2

# Analýza požadavků

V této kapitole jsou vysvětleny základní pojmy z požadavků, kterým jsou Firefox a doplněk.

## 2.1 Firefox

Přestože většině čtenářů bude pojem Firefox dobře známý, rád bych ho v této kapitole stručně popsal a vysvětlil, protože se jedná o zásadní část mé aplikace a v této práci také vystupuje jako framework, který poskytuje své knihovny a funkce.

Firefox (nebo také *Mozilla Firefox*) je multiplatformní webový prohlížeč, který vznikl jako projekt, jež založili Dave Hyatt a Blake Ross v roce 2002 (tehdy pod názvem *Phoenix*), za cílem vytvoření jednoduchého a uživatelsky přívětivého nástroje k prohlížení webu. Od roku 2002 bylo vydáno několik verzí a v současné době (květen 2011) je aktuální verzí Firefox 4.0. Více se o historii a vývojových verzích lze dozvědět z [58].

Vývojem Firefoxu se zabývá Mozilla Corporation, která je vlastněna Mozilla Foundation [30], a která jej šíří s otevřeným zdrojovým kódem ve více než osmdesáti lokalizovaných podobách.

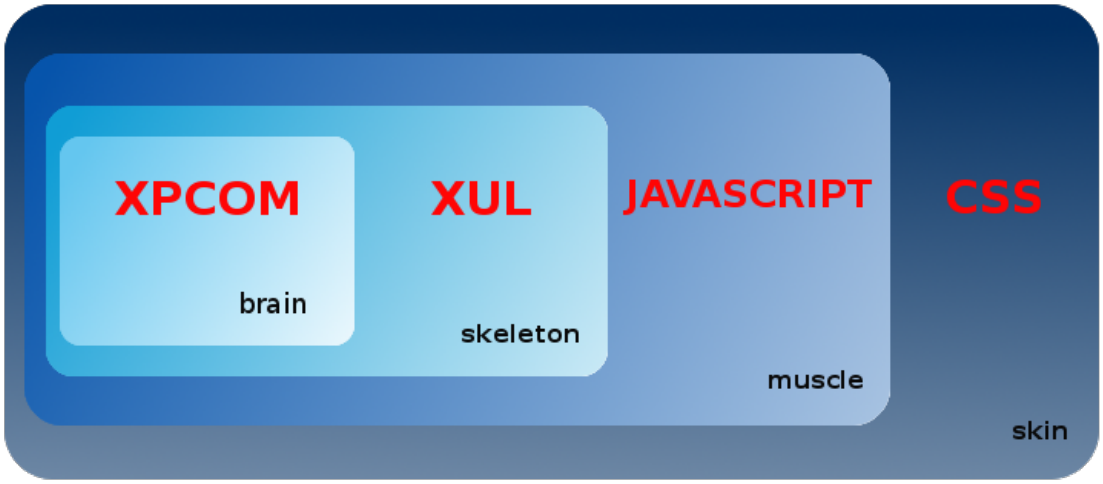
Firefox je z velké části vybudován na technologiích XUL, CSS, JavaScript a XPCOM [19] (o těchto technologiích více v kapitole 5), které jsou zobrazeny v obrázku 2.1. Firefox k vykreslování webových stránek (ale i samotného grafického rozhraní) používá renderovací jádro[1] Gecko (viz níže).

### 2.1.1 Gecko

Jak již bylo zmíněno v předchozí kapitole, je Gecko renderovací jádro a používá se ve webových prohlížečích, mimo Firefox, například v Camino [6]. Gecko podporuje otevřené internetové standardy, které jsou popsány v [27] včetně míry podpory.

Vývoj jádra Gecko začal v roce 1998 firmou Netscape, ale od roku 2003 se jeho vývojem zabývá Mozilla Foundation. Nyní je aktuální verzí jádra Gecko 2, na kterém je také založen Firefox 4.0. V době psaní tohoto textu je již naplánováno Gecko verze 5 [26].

[1] Renderovací jádro prohlížeče je aplikace, která převádí obsah a formátovací informace webové stránky do naformátované podoby a tu poté zobrazuje do okna prohlížeče




Obrázek 2.1: Technologie webového prohlížeče Firefox a jejich význam, převzato z [19]

## 2.2 Doplněk

V rámci této práce je doplněk chápán jako programové vybavení, které nepracuje samostatně, ale pouze jako část jiné aplikace a rozšiřuje původní funkčnost aplikace. Protože v mé práci budu rozšiřovat funkčnost aplikace Firefox, dále budu slovem doplněk označovat doplněk pro Firefox.

Doplněk může přidat jednoduchou funkcionalitu, v podobě například tlačítka, které je schopné zobrazit textovou hlášku, ale také komplexní vývojový nástroj, jakým je například Firebug[2]. Je třeba rozlišovat doplněk (anglicky *addon*) a zásuvný modul (anglicky *plugin*) [34], který do aplikace Firefox také přidává funkčnost, ale v podobě podpory pro zobrazení určitého obsahu, například videí.

[2] Jeden z nejpopulárnějších nástroju při vývoji webu [10]

# Kapitola 3

# Analýza vytváření doplňku

V této kapitole je stručně vysvětlen proces vytvoření doplňku. Při analýze tohoto procesu jsem vycházel z několika zdrojů, které jsou postupně uvedeny v dalších kapitolách. Nejdůležitějšími zdroji pro mě byly [51] a [9].

Základem uživatelského rozrahní v Mozilla Firefox je část, jež se nazývá Chrome. *Chrome je soubor prvků uživatelského rozhraní okna aplikace, které jsou mimo část okna pro zobrazení obsahu webu. Lišty nástrojů, lišty nabídek, lišty průběhu a lišty titulku okna jsou příklady prvků, které jsou typicky součástí chrome* [21]. *Chrome* má ale více významů a jejich popis lze nalézt v [20].

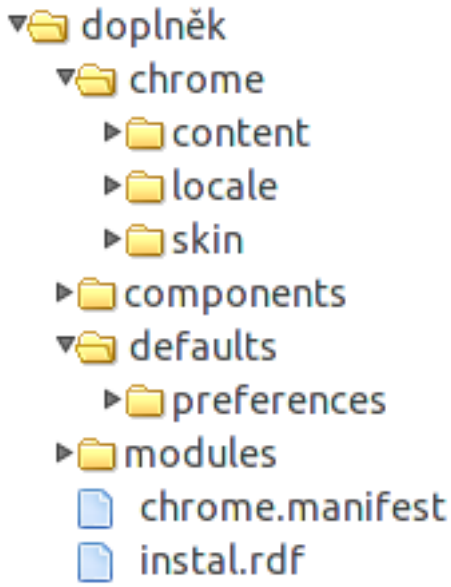
## 3.1 Adresářová struktura doplňku

Na obrázku 3.1 je zobrazena typická adresářová struktura, která se pro doplněk vytváří. Není nutné tuto strukturu dodržovat. Je však vhodné ji vytvářet, pro logické oddělení souborů pro chrome a dalších, a proto se této konvence budu také držet. Význam jednotlivých adresářů a částí bude vysvětlen v následujících kapitolách a více se o tomto problému lze také dočíst v [52].

## 3.2 Základní části doplňku

Základem doplňku pro Firefox (s grafickým uživatelským rozhraním) jsou *poskytovatelé chromu* (anglicky *chrome providers*), kteří se sdružují do jednoho *balíčku chromu* (anglicky *chrome package*). *Poskytovatelé chromu* se při vytváření doplňku většinou nacházejí ve složce *chrome* a základními *poskytovateli* jsou:

- *content*, obsahuje soubory pro popis uživatelského rozhraní doplňku (XUL soubory), ale také soubory, které definují chování uživatelského rozhraní (soubory se skripty v jazyce JavaScript),
- *locale*, obsahuje soubory s popisky uživatelského rozhraní (popisky tlačítek apod.), pro snadný překlad popisků uživatelského rozhraní do různých jazyků,
- *skin*, obsahuje soubory pro vzhled a popis vzhledu uživatelského rozhraní (CSS soubory, obrázky apod.).




Obrázek 3.1: Typická adresářová struktura doplňku pro Firefox

## 3.3 Integrace doplňku do Mozilla Firefox

Doplňek je "nainstalován" pomocí Gecko služby nazvané *registrace chromu* (anglicky *chrome registration*), která nalezne soubor `chrome.manifest`, kde jsou uvedeni *poskytovatelé chromu*. Soubor `chrome.manifest` se musí nacházet v kořenovém adresáři doplňku a *poskytovatel chromu*, kterého chceme do Firefoxu přidat, je v něm uveden na jednom řádku a poté ve Firefoxu přístupný pomocí speciální URI[4]. Více se o *registraci chromu* a `chrome.manifest` lze dozvědět z [21]. Při instalaci se také spouští XUL aplikace nazvaná *Install Manifest*, která ze souboru `install.rdf` přečte metadata o doplňku (jméno tvůrce doplňku, verze apod.), ale také pro jaké verze Firefoxu je doplněk určen, viz [29].

Doplňky můžou definovat nová okna, ve kterých mají popsané uživatelské rozhraní, nebo můžou rozšiřovat již existující okna ve Firefoxu. Nová okna jsou definována v XUL souborech pomocí elementu `<window>` a uživatelské rozhraní, které rozšiřuje existující část, je uvedeno v elementu `<overlay>`. Pokud má být uživatelské rozhraní nového doplňku přístupné, obsahuje alespoň jeden soubor s kořenovým elementem `<overlay>`, ve kterém je například definované tlačítko, jež otevře okno s doplňkem. Pro tento soubor je potřeba přidat do `chrome.manifest` řádek, ve kterém je údaj, do kterého souboru přidává nové prvky (soubory se identifikují pomocí URI, jak jsou přístupné pro Firefox). Ve většině případů se tak rozšiřuje hlavní okno prohlížeče, jehož URI je: `chrome://browser/content/browser.xul`.

Díky této praxi je možné přidat vlastní prvky uživatelského rozhraní do jakékoliv části Firefoxu, teměř bez omezení. Referenční příručku elementů Firefoxu lze nalézt v [50].

## 3.4 Sdílení kódu mezi okny Firefoxu

Pro každé otevřené okno Firefoxu se replikuje kód chromu (popsán v předchozích kapitolách), kód tedy není sdílen. To bývá často potřeba, proto pro sdílení kódu mezi okny existuje několik řešení (v dalších kapitolách jsou stručně popsána):

- Použití *JavaScript code module*,

[4] *Uniform Resource Identifier*, řetězec znaků sloužící k identifikaci zdroje

- použití XPCOM komponenty,
- použití FUEL objektu `Application`.

Každé uvedené řešení sdílení dat je "správné". Výhody a nevýhody každého z nich jsou uvedeny v [42].

### 3.4.1 Javascript code module

V adresářové struktuře doplňku se vytvářejí ve složce *modules* a ve Firefoxu se zpřístupní připsáním řádku `resource` do souboru `chrome.manifest`, viz [40]. *JavaScript code module* (uváděný též jako JSM), je JavaScriptový kód sdílený mezi okny Firefoxu, který se provede jen jednou při spuštění prvního okna. Více se o něm lze dozvědet z [40].

### 3.4.2 XPCOM

Technologie XPCOM je popsána v kapitole 5.6. XPCOM komponenty doplňků bývají napsané v JavaScriptu a ukládájí se do složky *components*. Každou takovou komponentu je třeba zvlášť zpřístupnit řádkem `component` v `chrome.manifest`. Další informace o vytváření XPCOM komponent lze nalézt v [47].

### 3.4.3 FUEL

FUEL je JavaScriptová knihovna pro jednoduché sdílení dat mezi okny Firefoxu. Obsahuje `Application` objekt, pomocí kterého lze ukládat a sdílet vlastní objekty. Tento objekt je přístupný v *chrome* kódu. Více se o FUEL knihovně a vlastnostech objektu `Application` lze dovědět z [25].

## 3.5 Uložení nastavení doplňku

Pro uložení perzistentního nastavení doplňku (např. zda se má okno doplňku při spuštění Firefoxu otevřít) slouží *Systém Nastavení* (anglicky *Preferences System*) [35]. Díky tomuto systému mají doplňky možnost zjistit nastavení Firefoxu a zároveň vytvářet nastavení vlastní.

Každé nastavení má v *Systému Nastavení* jméno a hodnotu. K hodnotě nastavení lze přistoupit pomocí jeho jména XPCOM komponentou, která je ve Firefoxu implementována. Tato komponenta je také implementována v knihovně FUEL (viz předchozí kapitola). A nastavení lze získat pomocí vlastnosti `prefs` objektu `Application`. Více se o tomto postupu a dalších možnostech nastavení lze dovědět z [28].

Doplněk může mít také výchozí nastavení. To se uvádí v JavaScriptovém souboru ve složce doplňku *preferences* (viz obrázek 3.1). Jak tento soubor vytvořit a ustálené praktiky při jeho vytváření lze opět nálezt v [28].

## 3.6 Přístup k systémovým zdrojům

Pokud doplněk potřebuje pracovat se systémovými zdroji (nebo zdroji Firefoxu), např. se soubory na disku, má možnost využít XPCOM komponentu, která implementuje rozhraní s požadovanými funkcemi (např. pro zmíněný příklad komponentu `nsLocalFile`). Seznam komponent, které poskytuje XPCOM knihovna, lze nalézt v referenční příručce API [45], kde jsou také uvedena rozhraní, jež komponenta implementuje (seznam rozhraní lze také nalézt v [46]). Komponenty (nejen poskytované XPCOM knihovnou) jsou reprezentovány vlastnostmi v objektu `Components.classes` [22], jenž je přístupný kdekoliv ve Firefoxu a referenci na danou komponentu získáme pomocí: `Component.classes[ContractID]`. *ContractID* jednoznačně identifikuje komponentu a lze ho nalézt v [45]. Více se o použití komponent lze dozvědet v [48] a [41].

## 3.7 Nástroje potřebné k vytvoření doplňku

K vytvoření doplňku nejsou potřeba žádná vývojová prostředí, ale může postačit jakýkoliv textový editor. Toto řešení ale neshledávám vhodným, protože práce by se stala nepřehlednou, a proto použiji Komodo Edit[1], který je doporučený v [38]. Komodo Edit je nástroj určený k editaci zdrojových kódů, které dokáže sdružovat do projektů, a pro mé použití je dostačující podpora pro kódy v jazycích Javascript, XML a CSS.

[1] http://www.activestate.com/komodo-edit

# Kapitola 4

# Analýza systému anotování

Tato práce vzniká jako součást většího systému 4A Framework, který je stále ve vývoji. Proto jsem při analýze systému anotování vycházel z dokumentu [7], jenž je zatím jeden z mála existujících dokumentů ke zmíněnému projektu, a v dalších kapitolách popisuji systém anotování 4A Framework. Z uvedeného dokumentu jsem také čerpal význam pojmu anotace, který je stručně popsán v následující kapitole.

## 4.1 Anotace

Slovo anotace může označovat samotný proces anotování nebo se jím označuje další informace přidaná k textu. Nejčastěji se anotace vyskytují jako jednoduché poznámky. Takto může anotace obohacovat či sdílet další myšlenky o anotovaném subjektu nebo přidávat k nestrukturovanému textu strukturované informace.

Anotace může být také využita k tagování[1]. Uživatelé tak můžou k obsahu přiřazovat jeho druh, či vlastnost, ale také jejich názor. Tyto tagy jsou poté využívány ke kategorizaci obsahu, navigaci a vyhledávání.

Anotace tedy kromě svého obsahu obsahuje typ, který je použit k tagování a dále může mít atributy, pomocí nichž lze k anotaci připojit další doplňující informace. Tyto atributy mají také svůj typ a může jím být nejen základní typ (číslo, řetězec, datum apod.), ale i vnořená anotace či odkaz na anotaci. Tímto způsobem tak vznikají strukturované anotace, kdy jedna anotace obsahuje další.

Doplněk se bude zabývat strukturovanými anotacemi (příklad strukturované anotace v systému 4A Framework je v příloze A) a ještě je třeba zdůraznit, že tyto anotace bude vytvářet pro webové stránky (které se zobrazí v prohlížeči Firefox) mající svá specifika oproti různým dokumetům, které je možné anotovat (viz [7]).

## 4.2 Proces anotování

Samotný proces anotace neobsahuje pouze úkon vytvoření poznámky k textu, ale lze do něj zahrnout i další akce, které s tímto procesem souvisí nebo mu musejí předcházet. V této kapitole popíši akce pouze obecně a vysvětlím, jak systém 4A Framework funguje.

Sytém pro anotovaní je navrhnut pro kolaborativní práci a uživatel má mít možnost anotovat kdekoliv a kdykoliv (odtud vychází i anglický název systému 4A – *Annotatinon Anywhere, Annotations Anytime*), pokud má přístup k Internetu. Proto se vytvořené anotace uživateli neukládají na lokální úložiště, ale jsou odeslány na server[2], kde jsou dále zpracovány. K tomuto účelu je nutné se k serveru nejdříve přihlásit pod vlastním uživatelským jménem a heslem, jenž jsou na serveru uloženy. Uživatel poté získá od serveru vlastní identifikátor (URI), kterým se označují i jeho anotace.

[1] Tag, krátka textová anotace

[2] Označení pro počítač nebo aplikaci, poskytující služby nebo data

Po přihlášení má uživatel možnost vytvořit anotaci, ale před jejím odesláním je nutná ještě tzv. *synchronizace dokumetu*. Při synchronizaci dokumentu klient[2] (v tomto případě můj doplněk) odešle aktuálně zobrazený dokument na server, který jej vyhodnotí a uloží si jeho kopii. Tento proces je velice důležitý, protože se při něm zjišťuje, zda k dokumentu na serveru již existuje jeho kopie a zda se od této kopie liší, protože by mohlo dojít k zneplatnění některé uložené anotace (v dokumentu již nelze nalézt anotovaný fragment uložené anotace). Pokud by uložením nové kopie dokumentu došlo k zásadním změnám či zneplatnění anotace, je zasláno upozornění uživateli, který se rozhodne, jak chce postupovat (uloží novou kopii nebo ponechá starou kopii a tu bude anotovat). Klientovi poté server zašle URI, kterým dokument v systému identifikuje (klient je povinnen tento URI uvádět v anotacích, čímž sděluje, ke kterému dokumentu se anotace vztahuje).

Vytvoření anotace většinou probíhá stylem: uživatel vybere fragment určený k anotaci, vytvoří k němu poznámku, vybere typ anotace a anotaci uloží (odešle na server). Pokud bude chtít přidat další informace, může k anotaci přidat atribut. Atributy jsou vázány na daný typ anotace a pokud se uživatel při přidání atributu rozhodne, že se stane výchozím atributem typu, bude se u tohoto typu vždy zobrazovat. U atributu uživatel také vyplňuje jeho typ a obsah. Obsahem může být například číslo, textový řetězec nebo anotace (viz další kapitola).

Anotace se po vytvoření musí uživateli také zobrazit. To ale není náplní této práce a je součástí jiného projektu. V této práci se zabývám popsanými částmi (s aplikací do doplňku prohlížeče Firefox), tedy:

- přihlášením uživatele,
- synchronizací dokumentu,
- přijímáním, zobrazováním a vytvářením typů a atributů anotace,
- vytvářením a uložením (odesíláním na server) anotací.

## 4.3 Typy anotací a atributů

Typy anotací, které uživatel může vybrat, jsou získány ze serveru. Typy jsou hierarchické a jsou tedy uspořádané do stromu. Klient si o typy žádá zvláštní zprávou a server mu zašle celý strom typů. Pokud potřebuje jen určitou část stromu, může do zprávy přidat filtr a je mu zaslána jen daná větev stromu typů.

Základními typy jsou běžné typy anotací (poznámka, popis apod.) a základní typy entit (věc, člověk apod.). Uživatel má ale možnost přidávat nové typy a podtypy.

Každý typ je identifikován linearizovaným názvem či URI. Linearizovaný název je cesta v hierarchii typů od kořenového typu a jednotlivé typy jsou odděleny znaky „->“.

Jak již bylo zmíněno, anotace může mít také atributy. Typem atributu může být typ anotace (v typu atributu je uveden linearizovaný název typu anotace nebo URI) nebo jednoduchý datový typ (v typu atributu je uveden název tohoto typu). Mezi jednoduché datové typy, které jsou v současné době podporovány, patří (více se o těchto typech lze dozvědět z [7]):

- `String`
- `URI`
- `DateTime`
- `Integer`
- `Decimal`
- `Date`
- `Time`
- `Boolean`

[2] Označení pro aplikaci, požadující vzdálené služby nebo data

- `Person`
- a `GeoPoint`.

## 4.4 Nastavení uživatele

Uživatel má po přihlášení také možnost nastavovat na serveru vlastní parametry. Tyto parametry jsou mu zaslány při přihlášení a slouží k nastavení různých vlastností, jako je například jazyk chybových zpráv. Více o nastavení parametrů na serveru se lze dozvědět z [7].

## 4.5 Protokol pro komunikaci

Komunikace mezi klientem a serverem probíhá pomocí XML zpráv s požadavky od klienta (např. přihlášení) či odpověďmi od serveru (např. informace o úspěšném přihlášení). Protokol pro komunikaci s anotačním serverem je podrobně popsán v příloze B, kterou jsem přebral z [7]. Z protokolu jsou pro mou práci významné zprávy (tak, jak jsou uvedeny v příloze) pro:

- Správu sezení,
- synchronizaci dokumentu,
- přenos typů anotací,
- přenos anotací,
- přenos nastavení,
- chyby a varování
- a potvrzení.

Tyto zprávy využiji pro komunikaci a přenos dat. Další zprávy, které protokol poskytuje, v této práci nevyužiji, protože se nezabývám problémy, které s nimi souvisejí.

# Kapitola 5

# Použité nástroje a technologie

Jak již bylo zmíněno v kapitole 2.1, je Firefox z velké části vybudován na technologiích XUL, CSS, JavaScript a XPCOM (více se o těchto technologiích zmíním v následujících odstavcích), proto se pro vytváření doplňků používají také tyto technologie a jejich použití se v mé práci jeví jako nutné. Doplněk je tedy především napsán v jazyce XUL (grafické uživatelské rozhraní) a o funkčnost se stará JavaScript.

## 5.1 XML

XML (*Extensible Markup Language*) je univerzální značkovací jazyk, navržený pro přenos dat. Jazyk byl vyvinut a standardizován konsorciem W3C v roce 1998. Často se používá k vytváření konkrétních značkovacíh jazyků (např. XHTML, viz další odstavce), které se nazývají aplikace, nebo k serializaci dat.

Aktuální verzí je verze 1.0, pátá edice, jejíž specifikaci lze nalézt v [1].

## 5.2 XUL

XUL (*XML User Interface Language*), česky jazyk k popisu uživatelského rozhraní, je aplikací jazyka XML. Aplikace napsané pomocí XUL lze snadno přizbůsobit místním požadavkům uživatelů - lokalizovat, změnit grafiku. XUL je nezávislý na běhovém prostředí a používán v aplikacích Mozzila. Více se lze dozvědět v [49].

XUL je založený na již existujících standardech (CSS, DOM, JavaScript - viz další kapitoly) [31] a jeho implementaci poskytuje renderovací jádro Gecko popsané v kapitole 2.1.1.

Více informací o jazyce XUL se lze dovědět z knihy [9].

## 5.3 CSS

CSS (*Cascading Style Sheets*) je jazyk pro definici vzhledu dokumentu napsaného ve značkovacím jazyce (z pohledu této práce definuje vzhled dokumentu XUL). V současné době je standardizována CSS verze 2.1, jejíž specifikaci lze najít na webu W3C [5] a stále se pracuje na CSS verzi 3 [4], již současná verze Firefoxu - 4 z velké části podporuje [32].

## 5.4 JavaScript

JavaScript je objektově orientovaný skriptovací jazyk a jeho auterem je Brendan Eich[1]. Tento jazyk je slabě typovaný a dědičnosti je dosaženo prototypováním. Jako konstruktory objektů slouží funkce,

[1] http://brendaneich.com

které mají přidružený prototyp (zvláštní objekt) a objekty sdílí vlastnosti tohoto prototypu. Více informací se lze dovědět z [61].

JavaScript byl standardizován v roce 1997 organizací ECMA[2] a pojmenován ECMAScript. Specifikaci ECMAScript (aktuální je pátá edice [8]) odpovídá JavaScript verze 1.8.5 (více o korelaci ECMAScriptu a Javascriptu lze nalézt v [59]).

Přestože značku JavaScript nyní vlastní Oracle Corporation [53], je především vyvíjen organizací Mozilla Foundation [30], pod kterou spadá i Firefox. Aktuální verze Firefoxu - 4 implementuje Javascript verze 1.8.5, jak je uvedeno v [16].

## 5.5 AJAX

AJAX (*Asynchronous JavaScript and XML*) není ani technologie, ani standard, ale spíše několik technologií dohromady. Pojem AJAX poprvé zveřejnil ve svém článku Jesse James Garrett v roce 2005, viz [11]. Základem aplikací využívajících AJAX jsou technologie: XHTML, CSS, DOM, JavaScript, XML, XSLT a XMLHttpRequest, více informací o těchto tegnologiích lze nalézt ve zmíněném článku [11].

AJAX se zabývá přístupem ke komunikaci mezi klientem a serverem. Komunikace probíhá „na pozadí“ a server zasílá pouze ty části dat, které se u klienta mají změnit. Více se lze dočíst v [60].

## 5.6 XPCOM

XPCOM (*Cross Platform Component Object Model*) je systém objektů používaný Mozzilou a poskytuje nástroje ke zpřístupnění funkcí napsaných v různých jazycích dalším aplikacím, či funkcím využívajícím XPCOM.

XPCOM definuje proces vytváření a rušení komponent. Každá komponenta implementuje vlastní rozhraní (definice sady funkcí, které jsou komponentou implementovány), ale také několik předepsaných XPCOM rozhraní.

XPCOM také poskytuje několik základních komponent a tříd pro správu souborů a paměti, vláken a další. Více o XPCOM, základních komponentách a rozhraní lze najít v [44].

## 5.7 RDF

RDF (*Resource Description Framework*) - systém popisu zdrojů je rodina specifikací a standard organizace W3C[3] [2] z roku 2004.

RDF je metadatový model, který každý zdroj popisuje výrazem podmět, vlastnost, předmět - *trojicí*. Podmět označuje zdroj a vlastnost popisuje vztah mezi podmětem a předmětem. Zdrojem dat RDF bývá většinou XML dokument. Více lze najít v [36].

## 5.8 DOM

DOM (*Document Object Model*) definuje rozhraní, pomocí kterého jsou skriptu (v rámci Firefoxu JavaScript) přístupné objekty HTML[4] a XML dokumentu. Více lze nalézt v [54].

DOM je standardizován W3C a nejnovější verze DOM level 3 je z roku 2004 [12]. Firefox má dobrou podporu do DOM level 2 a DOM level 3 podporuje z větší části, viz [24].

[2] http://www.ecma-international.org/

[3] World Wide Web Consortium, http://www.w3.org/

[4] HyperText Markup Language, více lze nalézt v [55]

## 5.9 XHTML

XHTML (*EXtensible HyperText Markup Language*) je jazyk, který upravuje jazyk HTML tak, aby odpovídal podmínkám tvorby XML dokumentů a slouží k vytváření hypertextových dokumentů[5]. Stal se W3C standardem v roce 2000 a měl zastoupit jazyk HTML. Více o XHTML lze nalézt v [56].

V současné době je aktuální standard XHTML 1.1 [15] z roku 2001. Ve vývoji je XHTML 5, které má být součástí standartu HTML 5.

## 5.10 XPATH

XPATH (*XML Path Language*) je jazyk, který se používá k navigaci v XML dokumentech. Umožňuje vyjádřit relativní cestu od určitého XML uzlu k jinému elementu nebo atributu. K popisu cesty používá XPath výraz, který se skládá z několika přechodů mezi uzly, jež se spojují lomítky. Více se o XPATH lze dozvědět z [57].

XPATH se stal W3C standardem v roce 1999 a jeho aktuální verzi z roku 2010 (verze 2.0, druhé edice) lze nalézt v [3]. V současné době se již pracuje na XPATH 3.0.

## 5.11 HTTP

HTTP (*Hypertext Transfer Protocol*) je jeden z nejpoužívanějších protokolů[6] v Internetu, který funguje na principu dotaz-odpověď. Klient zasílá dotazy na dokumenty na server, který zpět odešle odpověď s požadovanými daty.

První verze tohoto protokolu se objevila v roce 1991, kterou popisovalo W3C. V roce 1996 byl popsán ve standardu RFC 2068, který později nahradil dnes aktuální standard RFC 2616[7], který popisuje HTTP verze 1.1.

## 5.12 XMLHttpRequest

XMLHttpRequest je objekt umožňující v JavaScriptu komunikaci pomocí protokolu HTTP (ale podporuje i další protokoly, viz [43]). Tento objekt poprvé uvedl Microsoft a byl adoptován Mozzila Foundation, která ho v roce 2000 implementovala jako součást jádra Gecko. V současné době se připravuje jeho standardizace [13]. Více o XMLhttpRequest se lze dovědět z [43].

[5] Dokumenty obsahující odkazy (na jiné dokumenty apod.)
[6] Standard, podle kterého probíhá komunikace mezi dvěma body, nejčastěji počítači
[7] http://www.ietf.org/rfc/rfc2616.txt

# Kapitola 6

# Návrh řešení

V této kapitole popisuji návrh řešení. Nejprve se zabývám návrhem grafického uživatelského rozhraní, které považuji za důležité a je mu věnována největší část. Dále jsou popsány návrhy výběru typu anotace a práce s atributy. Poté je popsáno přihlášení uživatele a komunikace se serverem. U konce návrhu řešení jsou popsány možnosti vytvoření a uložení anotace. Jako poslední je uveden návrh datových zdrojů.

## 6.1 Umístění uživatelského rozhraní

Při volbě umístění uživatelského rozhraní pro anotování textu jsem měl několik možností:

- Přidat vlastní prvky do bočního panelu, který je k tomuto účelu vytvořen, jak je zmíněno v [17]
- vytvořit nové okno s vlastním uživatelským rozhraním,
- vložit nové prvky do té části okna Firefoxu, která je určena k prohlížení webových stránek, tak jako to implementuje například doplněk Firebug [10].

První návrh není pro můj účel příliš vhodný, jak bude vidět v kapitole 6.2, protože panel je určen spíše k vertikálnímu uspořádání prvků a mé uživatelské rozhraní by se tak stalo nepřehledným. Panel lze sice do určité míry roztáhnout, ale to není dostačující.

Pokud bych vytvářel nové okno pro psaní anotací, bude uživatel nucen „překlikávat“ mezi oknem webové stránky a anotačního doplňku. To by mohlo vést k negativnímu ovlivnění uživatele, který by anotace nevytvářel v takovém množství, jako je potřeba.

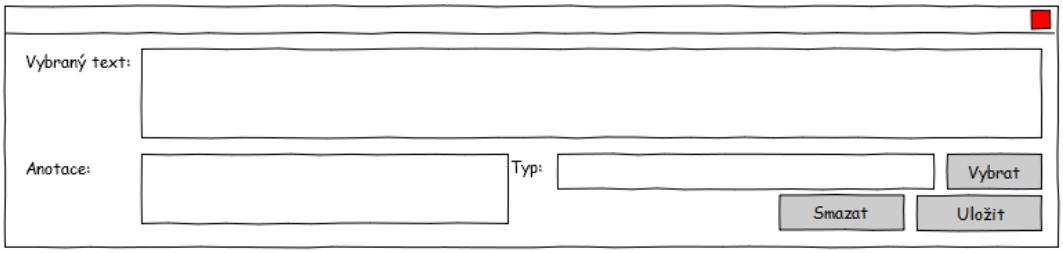
Poslední možností, kterou jsem také zvolil, je vložení uživatelského rozhraní do části okna s obsahem webové stránky. Oproti možnosti vytvoření nového okna má výhodu, že je blízko anotovanému textu a je zabudován „přímo“. Navíc nejsem omezen ve velikosti šířky rozhraní, jako je tomu u bočního panelu, ale můžu využít celou šírku okna Firefoxu. Přestože se výška prohlíženého dokumentu opticky zmenší, není tento problém při anotování nijak zásadní.

## 6.2 Návrh uživatelského rozhranní

Jednou z důležitých částí tohoto projektu je grafické uživatelské rozhraní. Všechny části uživatelského rozhraní musí být přehledné a přívětivé tak, aby se uživateli s doplňkem dobře pracovalo. V mém návrhu jsem vycházel z návrhu v [7].

Základními a nezbytnými částmi uživatelského rozhraní při vytváření anotací jsou pole pro vytvoření obsahu anotace (`anotace`), výběru typu a tlačítko uložení (`uložit`). Pro lepší přehlednost bude rozhraní doplněno o textové pole (`vybraný text`), které bude zobrazovat aktuální výběr textu k anotovaní. Díky tomuto poli si uživatel také může uvědomit, zda je jeho výběr správný a zamýšlený.

Prvky budou uspořádány dle předpokládaného postupu uživatele. Uživatel nejprve zvolí text k anotoci (ten se mu zobrazí v poli), vybere typ, dále vyplní obsah anotace a nakonec anotaci odešle na server. Na obrázku 6.1 je zobrazen návrh grafického uživatelského rozhraní se zmíněnými prvky vytvořený pomocí nástroje Pencil[1].




Obrázek 6.1: Návrh základního grafického uživatelského rozhraní doplňku

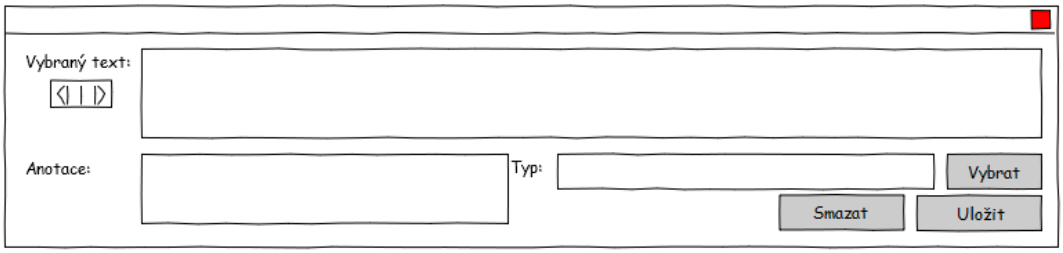
Pole pro výběr typu musí být dostatečně dlouhé, protože bude zobrazovat linearizovaný název od kořenového typu ve stromu typů. V reálném nasazení lze předpokládat alespoň tři úrovně zanoření, je tedy potřeba vedle sebe zobrazit alespoň čtyři názvy typů, které by se měly do pole vejít, aby uživatel pohodlně přečetl celý název typu. Proto bude pole zabírat téměř polovinu šířky okna.

Typ bude možné zadat nejen přímo do textového pole, ale uživatel může typ vybrat i ze stromu typů (jeho návrh bude popsán později). Pro zobrazení stromu typů je vedle pole pro výběr typu tlačítko (`vybrat`), kterým se otevře nové okno pro výběr typu.

Protože uživatelské rozhraní bude umístěno do části s webovou stránkou, musí být možnost, jak toto rozhraní skrýt. Pro tento účel vytvořím v horní části uživatelského rozhraní lištu, ve které bude tlačítko ke skrytí doplňku, a navíc se do lišty můžou přidat další potřebná tlačítka. Horní lišta je také zobrazena na obrázku 6.1. Pro zpětné zobrazení okna bude do lišty doplňků Firefoxu přidána ikona mého doplňku.

### 6.2.1 Vyběr více fragmentů textu

Firefox umožňuje vybrání více fragmentů textu. Při vybrání více fragmentů se uživateli zobrazí ikony šipek, pomocí kterých si může vybrat k anotaci určitý fragment nebo může anotovat všechny. U tohoto řešení se navíc projeví další výhoda zobrazení vybraného textu v poli doplňku, protože toto pole může zobrazit vybraný fragment pomocí šipek. Návrh uživatelského rozhraní s možností výběru fragmentu je na obrázku 6.2.




Obrázek 6.2: Grafické uživatelské rozhraní doplňku s šipkami pro výběr fragmentu

[1] http://pencil.evolus.vn/en-US/Home.aspx

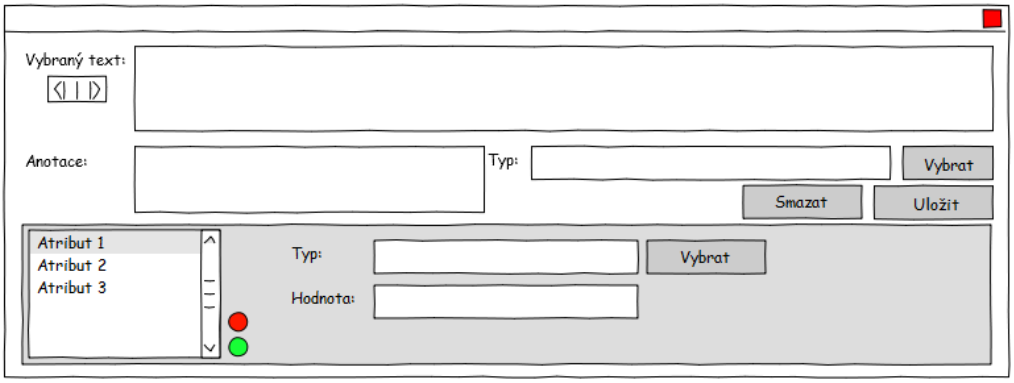
### 6.2.2 Zadávání atributů

Doplněk také musí zobrazovat atributy anotace. Ty se budou zobrazovat ve spodní části uživatelského rozhraní a budou pomocí jiné barvy pozadí opticky oddělené od hlavní části okna. Atributy se budou zobrazovat ve stromu tak, aby bylo možné zobrazit atributy vnořené anotace.

Pro přidání atributu bude vedle zmíněného stromu tlačítko (bude se zobrazovat či skrývat při vybrání či zrušení výběru typu anotace), které vyvolá okno pro vytvoření či přidání atributu (toto okno bude popsáno později). Protože typem atributu může být i typ anotace, lze k tomuto atributu přidat další atribut. Pro rozlišení přidání atributu k vybranému typu anotace či přidání atributu k atributu bude vedle stromu další tlačítko.

Uživatelské rozhraní pro zadávání atributu se bude zobrazovat v závislosti na vybrém atributu napravo od stromu atributů. Výběr typu atributu bude podobný jako výběr typu anotace.

Vzhled doplňku i s popsaným uživatelským rozhraním pro zadávání atributu je na obrázku 6.3.




Obrázek 6.3: Grafické uživatelské rozhraní doplňku s atributy

### 6.2.3 Výběr typu anotace ze stromu

Jak již bylo popsáno, uživatel může vybrat typ anotace ze stromu typů. Tento strom se zobrazí v novém okně po kliknutí na tlačítko `vybrat` typ. Uživatelské rozhraní výběru typu je na obrázku 6.4. Pod stromem typů bude navíc tlačítko pro znovu-načtení typů ze serveru a pole na přídání nového typu.

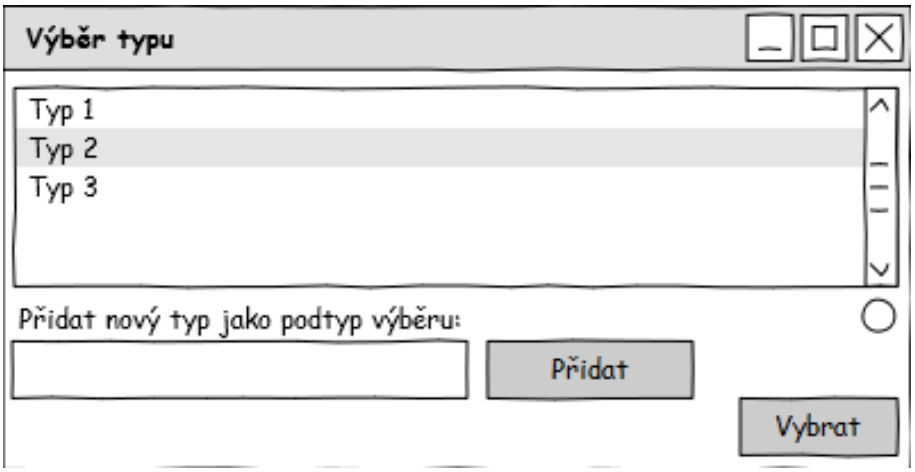
### 6.2.4 Přidání nového atributu a výběr typu atributu

Popsaná tlačítka na přidání atributu v kapitole 6.2.2 vyvolají okno, jenž je na obrázku 6.5. Při přidání atributu je třeba vyplnit jeho jméno, vybrat typ a určit, zda má být výchozí (zobrazovat se všem uživatelům) a povinný.

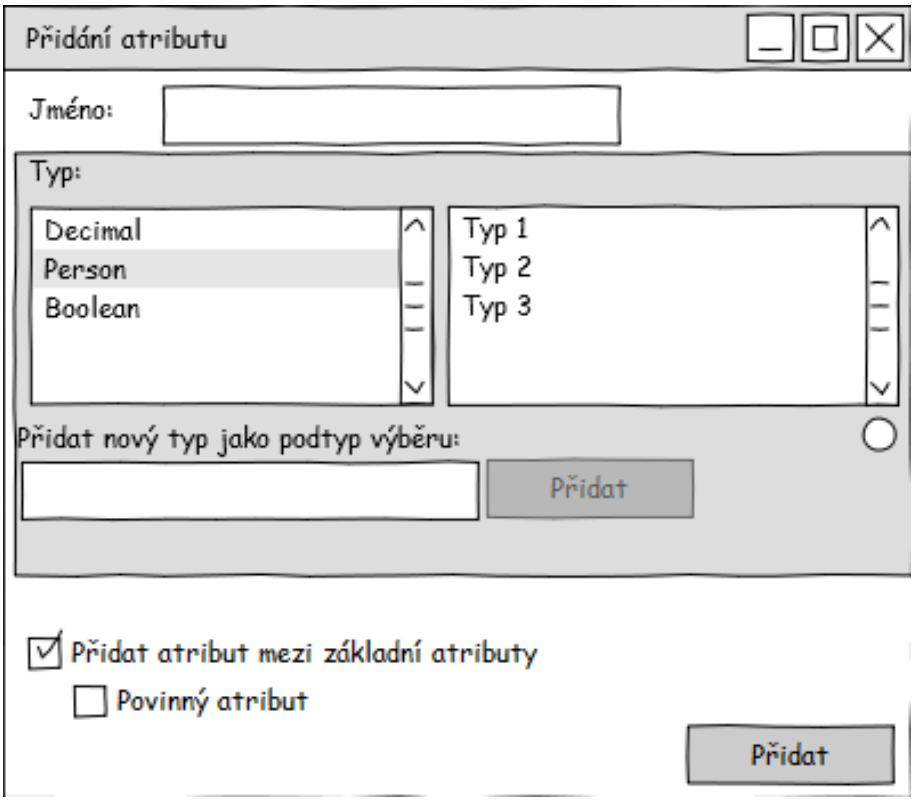
Typ atributu se bude vybírat opět ze stromu. Protože typ atributu ale může být jednoduchý typ nebo typ anotace (viz kapitola 4.3), jsou zobrazeny 2 stromy, které toto rozlišují.

Při přidávání atributu je možné vytvořit nový typ jako podtyp výběru ve stromu (což není možné pro jednoduchý typ, proto se tlačítko `přidat` typ při výběru ve stromu jednoduchých typů zablokuje).

Při výběru typu atributu pomocí tlačítka `vybrat` se zobrazí podobné okno jako při výběru typu anotace (pouze bude obsahovat dva stromy jako na obrázku 6.5 a tlačítko na přidání nového typu se zablokuje podle pravidel popsaných výše).




Obrázek 6.4: Výběr typu anotace ze stromu



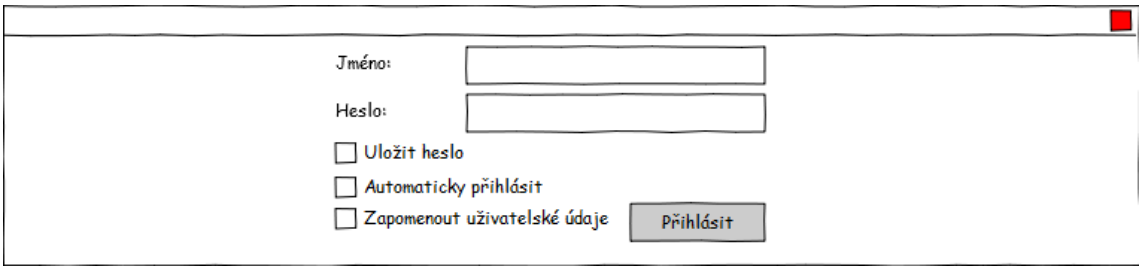

Obrázek 6.5: Přidání nového atributu

### 6.2.5 Přihlašovací okno

Návrh přihlašovacího okna je na obrázku 6.6. Při návrhu jsem se inspiroval z existující aplikace ICQ[1].

Přihlašovací okno bude zobrazeno ve stejné části jako okno určené k anotaci, přičemž zobrazené může být pouze jedno z nich. Pro toto řešení zvolím umístění obou oken do elementu `<deck>`, který je k této potřebě vhodný.

Uživatel bude mít při přihlašování možnost, zda chce pro své uživatelské jméno uložit heslo (jméno se uloží implicitně). Bude mít také na výběr možnost automatického přihlášení, při které se doplněk přihlásí při spuštění Firefoxu. Protože se přihlašovací jméno bude ukládat implicitně, bude mít na výběr jeho smazání po odhlášení.




Obrázek 6.6: Grafické uživatelské rozhraní přihlašovacího okna

## 6.3 Lokalizace

Jak bylo popsáno v kapitole 3, u doplňku se vytváří složka `locale`, která obsahuje texty a popisky doplňku. Můžou se zde vyskytovat soubory s popisky v různých jazycích. Já vytvořím lokalizaci pro český a anglický jazyk. Přestože tato praxe není nutná, je vhodná. Texty se takto dají snadno udržovat a v budoucnu se lze lehce přidat další lokalizace.

## 6.4 Nastavení doplňku

Doplněk by měl obsahovat nastavení a to nejen vlastní, ale také nastavení parametrů na serveru. Proto v horní liště doplňku (popsána v 6.2) bude navíc tlačítko pro nastavení. Využiji již popsaný *Systém Nastavení*, který pro zobrazení nastavení využívá okno `<prefwindow/>`. V tomto okně budou záložky pro nastavení doplňku, serveru a parametrů serveru. Protože parametry serveru jsou získány až po přihlášení uživatele a nejsou předem známé, budou se jejich názvy a hodnoty zobrazovat v tabulce tak, jak budou získány od serveru.

## 6.5 Systém upozornění

Doplněk potřebuje upozorňovat uživatele o chybách či varováních, které při anotování můžou vzniknout (především při komunikaci se serverem). Proto vytvořím třídu, která se bude starat o zobrazení zpráv. Tato třída bude implementovat metodu `alert()` s jediným parametrem pro hlšení, které má metoda zobrazit. Pro samotné zobrazení použiji rozhraní, které implementuje Firefox – `nsIAlertsService`. Firefox poskytuje další možnosti pro zobrazení upozornění (viz [18]), ale tento způsob se mi zdá nejvhodnější, protože nevyžaduje od uživatele zbytečnou interakci, ale pouze zobrazí informaci. V budoucnu bude ale možné třídu rozšířit a implementovat vlastní systém upozornění.

[1] http://www.icq.com/

## 6.6 Objekt `annotationWindow`

Tento objekt v chrome kódu bude sloužit jako centrální objekt a bude poskytovat funkčnost grafickému uživatelskému rozhraní doplňku popsaného v 6.2. Bude implementovat důležité metody `init` a `destroy`, které se budou volat při přihlášení a odhlášení uživatele a provádět tak nutné operace při těchto událostech. Jeho význam bude zmíněn v dalších kapitolách.

## 6.7 Výběr a zobrazení textu k anotaci

Protože návrh uživatelského rozhraní obsahuje také pole pro zobrazení vybraného textu, musím zajistit, aby se výběr textu ihned reflektoval do okna doplňku. Firefox neposkytuje způsob, jak upozornit na výběr textu, proto budu sledovat událost uvolnění tlačítka myši v oblasti webové stránky (element `content`) pomocí metody `addEventListener()`. Po přijetí této události zjistím, zda je vybrán nějaký text a ten případně zobrazím do okna vybraného textu.

Jak bylo popsáno v kapitole 6.2.1, Firefox umožňuje vybrání více fragmentů. Proto, pokud bude vybrán více než jeden fragment, zobrazí se šipky na výběr fragmentu, který se má do pole vybraného textu zobrazit.

Protože je zbytečné, aby se událost na uvolnění tlačítka sledovala od spuštění doplňku (Firefoxu), metoda `addEventListener()` se zavolá až po přihlášení uživatele a tedy po zobrazení pole na vybraný text. Po odhlášení uživatele se sledování události zruší.

## 6.8 Výběr typu anotace ze stromu

Při výběru typu anotace ze stromu se otevře nové okno, které je popsáno v kapitole 7.7, a typy se zobrazí z RDF zdroje typů (RDF zdroj dat je popsán v kapitole 6.16). RDF zdroj typů bude u uživatele lokálně uložen (ve složce doplňku). Typy se tedy nebudou požadovat ze serveru při každém otevření okna. Pro obnovení typů ze serveru je v uživatelském rozhraní navrhnuto tlačítko, které vyvolá metodu modulu pro komunikaci se serverem (klient je popsán v kapitole 6.12) pro obnovení typů.

Protože okno s typy má vlastní chrome kód a při zavření tohoto okna se jeho kód ztratí, před samotným zavřením se případný vybraný typ uloží do objektu `annotationWindow` chrome kódu okna, které okno s typy otevřelo. K tomuto účelu je ve Firefoxu objekt `opener`, který má přístup do chrome kódu rodiče. Po vybrání typu se provede výběr atributů (viz kapitola 6.10), pokud typ nějaké má, a zobrazení typu do pole pro volbu typu. Pokud byl již nějaký typ vybrán, vybrání nového typu způsobí smazání anotačního okna a především smazání atributů předchozího typu.

Pokud uživatel v okně s typy přidá nový typ, ten bude ihned zaslán na server.

## 6.9 Výběr typu pomocí *autocomplete textbox*

V kapitole 6.2 je zmíněno, že uživatel může typ zadat přímo do textového pole typu. Pokud by takto uživatelé typy zadávali, rychle by na serveru rostl jejich počet, třeba s malými odchylkami v názvu. Tomuto je třeba se vyhnout, proto i při zadání typu do textového pole přímo, bude uživateli zobrazena nabídka. Toho bude docíleno pomocí *autocomplete textbox*, kdy při psaní typu uživatelem se bude pod polem, ve kterém jej zadává, zobrazovat nabídka typů, jejichž linearizovaný název obsahuje zadaný text. Tento text bude uživatel postupně upřesňovat a může vybrat z již existujících typů. Tímto způsobem se značně sníží diverzita typů, protože uživatel si před vytvořením nového typu může najít podobný typ (nebo stejný typ s odchylkou v názvu), který vystihne požadovaný význam, a ten vybere.

Pokud uživatel vybere typ z nabídky, zobrazí se mu pro daný typ existující základní atributy. Pokud uživatel typ nevybere z nabídky, ale pouze ho napíše a ten existuje, atributy se zobrazí až po potvrzení klávesou `enter`.

Pokud si uživatel nezvolí existující typ, nový typ bude zaslán na server s jeho případnými atributy až s anotací.

Pro tento *autocomplete textbox* je potřeba implementovat komponentu s rozhraním `nsIAutoCompleteSearch`, viz [39]. Protože komponenty nemají přístup k chrome kódu, ve kterém se bude nacházet klient, bude potřeba mu zajistit přístup pomocí `nsIWindowMediator`, implementovaného Firefoxem.

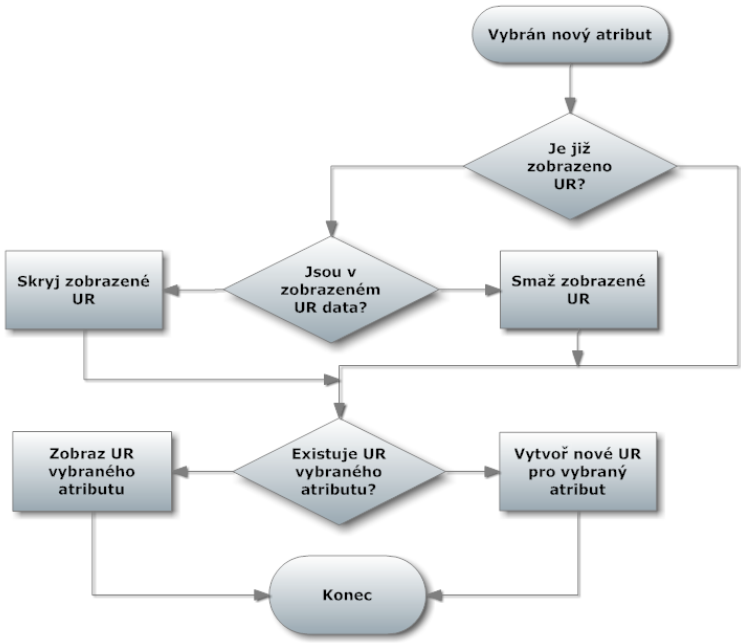
## 6.10 Atributy

Atributy se budou zobrazovat při vybrání typu do stromu navrženého v kapitole 6.2.2. Bude vytvořen RDF zdroj dat, který bude pro jednotlivé typy přijaté od serveru ukládat jejich atributy. Při výběru typu se v tomto RDF zdroji nalezne vybraný typ a jeho atributy. Tyto atributy se zkopírují do RDF zdroje zobrazených atributů.

Toto řešení jsem zvolil, protože jsem při návrhu narazil na problém, pokud je ve vybraném typu anotace atribut stejného typu jako je je vybraný typ anotace. Uživateli se při výběru typu vždy zobrazí první úroveň atributů a pokud je nějaký atribut typu anotace, až po jeho vybrání se načtou a zobrazí jeho atributy z RDF zdroje typů s atributy. Takto se uživateli zobrazí atributy do libovolné úrovně.

Uživatelské rozhraní pro zadávání atributů z obrázku 6.3 se zobrazí po jeho vybrání. Pro každý jednoduchý typ atributu budou vytvořeny do části DOM dokumentu uživatelského rozhraní doplňku prvky, které jsou atributem vyžadovány. Pro typy anotace jsou uživatelská rozhraní stejná.

Z uživatelského rozhraní pro zadávání atributů se také budou přímo číst vyplněná data při vytváření anotace (viz kapitola 6.13) tak, že se projdou všechna vytvořená uživatelská rozhraní. Pokud atribut nebude mít vyplněná žádná data, nebude pro něj ve stromu DOM existovat uživatelské rozhraní a k anotaci se nepřipojí. Proces vytváření a mazání uživatelského rozhraní pro atribut je pro názornost na obrázku 6.7.




Obrázek 6.7: Vytvoření a smazání uživatelského rozhraní atributu, šipka vlevo znamená kladnou odpověď, šipka vpravo zápornou, UR : uživatelské rozhraní

Uživatelská rozhraní se budou do DOM dokumentu připojovat na jediné místo a budou identifikovatelná (tedy dohledatelná) pomocí URI atributu, které bude v RDF zdroji zobrazených atributů, protože to musí být pro každý atribut jedinečné. Vznikne tak vazba mezi atributem ve stromu a jeho uživatelským rozhraním.

Při přidání nového atributu je třeba rozlišovat, zda se přidává k vybranému typu anotace či se přidává k atributu (tedy do vnořené anotace). Proto jsou v návrhu v kapitole 6.2.2 dvě tlačítka a oknu se podle zmáčknutého tlačítka předá parametr, ke kterému typu se má atribut přidat (při první možnosti se zjistí z RDF zdroje typů, při druhé možnosti z vybraných atributů). Atribut se přidá do RDF zdroje typů s atributy a poté se přídá do RDF zdroje zobrazených atributů k atributům, ke kterým patří.

Při výběru typu atributu je potřeba zajistit změnu zobrazených atributů:

- Pokud je původní typ typu anotace, musí se smazat jeho atributy,
- pokud je nový typ typu anotace, musí se nahrát jeho atributy.

Musí se změnit jeho uživatelské rozhraní a nový typ reflektovat do RDF zdroje zobrazených atributů a typů s atributy.

Atributy, které jsou typu anotace, budou mít stejné uživatelské rozhraní. Aby se pro tento typ atributu dal vybrat text k anotavání, bude v uživatelském rozhraní tlačítko, které umožní vybraný fragment zobrazit do pole vybraného textu vnořené anotace a určit, že se vybírá vnořená anotace. Po ukončení výběru (opětovném zmáčkutí tlačítka nebo vybrání jiného atributu) se vnořená anotace uloží (toto je popsáno v kapitole 6.14).

## 6.11 Přihlášení uživatele

Přihlášení uživatele bude probíhat pomocí modulu pro komunikaci se serverem, který je popsán v kapitole 6.12. Při návrhu přihlašování jsem zjistil, že je potřeba zajistit několik důležitých kroků, které dále popíši.

Protože uživatel může mít otevřených několik oken zároveň, je důležité aby byl ve všech oknech stejně přihlášen či odhlášen. Navíc synchronizaci dokumetu lze provést pouze jednu, pro jedno sezení.

Proto jsem navrhl následující řešení: zda je uživatel přihlášen či odhlášen, jeho uživatelské jméno a heslo bude udržováno v kódu JavaScript modulu (mimo chrome), který je sdílen všemi okny Firefoxu. Při spuštění okna Firefoxu se zjístí stav uživatele a podle toho se doplněk pokusí pro dané okno přihlásit, v každém okně tedy bude udržováno jiné sezení pro stejného uživatele.

Objekt uživatele bude využívat návrhový vzor observer a v kódu chrome bude implementovat metodu `observe()`, pomocí které bude přijímat zprávy — od dalších objektů uživatele v ostatních oknech — o přihlášení či odhlášení. K zasílání těchto zpráv slouží rozhraní `nsIObserverService`, které implementuje Firefox, viz [33], a které může kromě zprávy přenést i další data.

Díky tomuto řešení může mít uživatel synchronizován každý otevřený dokument a nemusí se přihlašovat či odhlašovat pro každé okno zvlášť.

Pro uložení hesla použiji správce hesel Firefoxu (`nsILoginManager`), který hesla dokáže bezpečně uložit.

## 6.12 Návrh modulu pro komunikaci se serverem

Modul bude implementovat zprávy protokolu, který byl popsán v 4.5. Každou zprávu bude reprezentovat jedna metoda modulu. V metodě odchozí zprávy se také určí metoda, která zpracuje odpověď od serveru. Pokud se v odpovědi vyskytnou chyby, využije modul systém pro upozornění uživatele, popsáný v kapitole 6.5.

Komunikace bude probíhat pomocí objektu `XMLHttpRequest`.

Přestože jsou zprávy ve formátu XML a nabízí se řešení je vytvářet jako DOM strom, který objekt `XMLHttpRequest` dokáže zpracovat, budu zprávy vytvářet textově (do proměnné se vždy připojí požadovaná část textu zprávy). Vytváření zpráv pomocí DOM rozhraní shledávám příliš komplikovaným, zvyšujícím složitost zpracování a zhoršujícím odezvu doplňku.

Protože komunikace bude probíhat asynchronně, je potřeba zajistit upozornění pro část kódu, která bude modul využívat. Využiji rozhraní `nsIObserverService`, které je popsáno v předchozí kapitole, a kód, který využije služby klienta, bude očekávat informaci o přijetí patřičné odpovědi ze serveru pomocí metody `observe`.

V odpovědi serveru, ve které se budou vyskytovat typy anotací, tyto typy modul nejprve uloží do globálního pole a teprve poté upozorní na přijetí odpovědi. Typ bude reprezentován třídou `type`, jejíž vlastnosti budou identické s atributy typu v XML zprávě. Další postup zpracování typů je zmíněn v kapitole 6.17. Podobného principu lze využít i u předání dalších dat.

## 6.13 Vytvoření a zaslání anotace

Před zasláním anotace na server proběhne synchronizace dokumentu. Dokument se bude ale synchronizovat jen v případě, pokud ještě nebyl synchronizován. Po synchronizaci dokumentu se projdou všechna vytvořená uživatelská rozhraní atributů (popsány v kapitole 6.10) a z nich se vytvoří zpráva, která se odešle na server pomocí příslušného modulu. Díky označení uživatelských rozhraní pomocí identifikátoru z RDF zdroje zobrazených atributů lze určit jejich hierarchii a připojit hodnoty uživatelského rozhraní a tedy atribut do správné vnořené či „hlavní“ anotace.

## 6.14 Uložení anotace

Uložení anotace zahrnuje proces výpočtu cesty anotovaného fragmentu (případně více fragmentů) a její uchování. Pro uložení anotace jsem navrhl třídu `annotation`, která bude obsahovat vybrané fragmenty (viz kapitola 6.15). Každá instance třídy bude mít navíc identifikátor, který bude stejný jako uživatelské rozhraní atributu (popsáno v kapitole 6.10). Takto bude jednoznačně určeno, kterému atributu daná instance třídy `annotation` náleží. Tyto vytvořené anotace budou ukládány do globálního pole, aby byly snadno přístupné pro zpracování a odeslání na server.

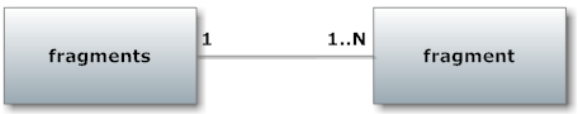
## 6.15 Cesta k anotovanému fragmentu

Anotovaný fragment se určí pomocí XPath, jak je popsáno v [7], protože je to jedno z nejlepších možných řešení pro určení cesty k textu v dokumentu webové stránky.

Fragment se musí rozdělit na více fragmentů, pokud je vybrán přes více uzlů DOM dokumentu, tak, aby v něm nebyly obsaženy značky jazyka dokumentu, které snižují odolnost anotací vůči změnám v dokumentu. Pro tento problém využiji funkce objektu `treeWalker` (viz [23]), který dokáže projít elementy stromu od zadaného elementu a vybrat dle vlastní funkce určité uzly. Uzly označeného textu, které `treeWalker` musí projít, jsou uloženy v objektu `Selection` Firefoxu [37].

Po určení všech uzlů, které obsahují označený text, se pro každý tento uzel vypočte XPath a přesná pozice vybraného textu pomocí offsetu a délky (offset pro uzly uprostřed fragmentu je nulový a délka odpovídá délce textu v uzlu, pro offset a délku textu prvního a posledního uzlu využiji funkce z [37]).

Pro popsaný problém jsem navrhl třídy `fragment` a `fragments`. Třída `fragment` reprezentuje vybraný text v jednom uzlu a třída `fragments` bude obsahovat funkce pro určení fragmentů – s využitím objektu `treeWalker` — a reprezentuje výběr. Vztah těchto tříd je na obrázku 6.8.




Obrázek 6.8: Vztah mezi třídami `fragment` a `fragments`

## 6.16 Poskytnutí dat stromu

V předchozích kapitolách bylo popsáno zobrazení dat do stromů (například typů anotací). Je potřeba navrhnout způsob poskytování dat, která se mají v těchto stromech zobrazit. Ve Firefoxu existuje několik způsobů, jak data stromu poskytnout (detailní popis těchto způsobů lze nalézt v [51]):

- Vytvoření tzv. *content tree*,
- vytvoření vlastního pohledu pro strom (anglicky *Custom tree view*),
- vytvoření tzv. *RDF tree*.

První způsob přidává ke stromu (elementu `<tree/>`) DOM elementy prvků – `<treeitem/>`, které jsou také uspořádány ve stylu DOM dokumentu. Prvky jsou přístupné pomocí DOM rozhraní a přes něj se s nimi také manipuluje. Při větším počtu prvků a zanoření ve stromu se tak strom stává náročným na zorientování, ale také na manipulování a na zdroje. Proto je určen k zobrazení malého množství prvků.

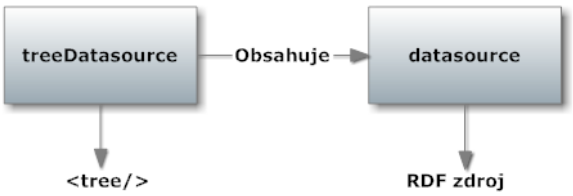
U druhého způsobu je nutné implementovat rozhraní, které se stará o získání a zobrazení dat do stromu. Objekt implementující toto rozhraní se pak připojí ke stromu. Přestože je tento způsob vhodný pro vetší množství dat, je u něj problém ve složitosti implementace, která roste se složitostí stromu (počtem zanoření prvků ve stromu).

Posledním způsobem je vytvoření stromu z RDF zdroje (RDF je popsáno v kapitole 5.7), který se ke stromu připojí. Data jsou stromu poskytována přímo z RDF zdroje, který se podle něj vybuduje. Pro manipulaci s prvky stromu se tak využivá manipulace s RDF zdrojem. Tento způsob je vhodný pro velké množství dat zobrazených ve stromu, ale také vhodný pro implementaci složitějších stromů. Protože v reálném prostředí lze očekávat poměrně velké množství typů k zobrazení, zvolil jsem toto řešení pro implementaci, kterou použiji i u dalších stromů. Tohoto řešení jsem navíc využil i pro jiné účely, viz kapitola 6.10.

## 6.17 RDF zdroje dat

Přestože strom, kterému poskytuje data RDF zdroj dat, se vybuduje sám, je potřeba vytvořit funkce pro práci s RDF zdrojem, protože samotné funkce poskytnuté Firefoxem pro práci s RDF nejsou dostačující. RDF zdroj dat navíc neslouží pouze pro poskytnutí dat stromu, ale k obecnému uložení dat (jako je tomu při uložení atributů typů).

Proto jsem navrhl třídy `datasource` a `treeDatasource`, které lze využít pro jakýkoliv strom. Třída `datasource` je vázaná k RDF zdroji dat, jejž vytváří, a zároveň poskytuje komplexní metody pro práci s ním. Třída `treeDatasource` je vázaná ke stromu a obsahuje třídu `datasource`, jejíž metody využívá a připojuje RDF zdroj dat třídy `datasource` ke stromu. Vztah mezi třídami je na obrázku 6.9.




Obrázek 6.9: Vztah mezi třídami `treeDatasource` a `datasource`

Mezi důležité metody třídy `datasource`, které je zde potřeba zmínit, patří přidání nové položky (objektu) do RDF, změna vlastnosti položky v RDF a smazání položky v RDF. Díky těmto metodám půjde uložit do RDF zdroje dat objekty, které klient uloží jako objekty do globálního pole (například typy anotací).

# Kapitola 7

# Implementace

Při implementaci doplňku jsem vycházel z popsaného návrhu. V této kapitole jsou pouze základní principy implementace důležitých částí.

Respektoval jsem rozdělení složek doplňku popsaného v kapitole 3. V kořenovém adresáři je tedy soubor `chrome.manifest`, ve kterém jsou pro Firefox popsána umístění složek s kódy, a soubor `install.rdf`, ve kterém je jméno doplňku, jeho popis a autor.

Složka `chrome` obsahuje chrome kód. Ve složce `chrome/content` jsou uloženy soubory s popisem uživatelského rozhraní (XUL) a JavaScriptové soubory, `chrome/locale` obsahuje soubory s lokalizovanými texty a `chrome/skin` obsahuje soubory pro vzhled uživatelského rozhraní – CSS soubory a ikony. Ve složce `defaults/prefs` se nachází JavaScriptový soubor s výchozím nastavením doplňku: velikost okna, adresa a port serveru apod. Jediná komponenta pro *autocomplete textbox* se nachází ve složce `components` a složka `modules` obsahuje JSM soubory.

## 7.1 Jmenný prostor

Pro doplněk jsou vytvořeny dva jmenné prostory. Pro chrome kód je to `annotationExtensionChrome`, který je implementován v souboru `chrome/content/namespace.js` a pro JSM jmenný prostor `annotationExtension`. Všechny objekty doplňku budou v těchto jmenných prostorech, aby nedocházelo ke kolizím s ostatními doplňky, případně Firefoxem. V následujícím textu budu pouze pro přehlednost a úsporu místa uvádět `annotationExtensionChrome` jako `aEC` a `annotationExtension` jako `aE`.

## 7.2 Uživatelské rozhraní

Uživatelská rozhraní popsaná v návrhu jsou rozdělena do XUL douborů následovně: hlavní okno doplňku je v souboru `annotationWindow.xul`. Zde je vytvořeno grafické uživatelské rozhraní pro vytváření anotací a zobrazení atributů. Okno pro výběr typu se nachází v `typesWindow.xul`. Pro přidání atributu a výběr typu atributu jsem vytvořil uživatelské rozhraní v souborech `addAttributeWindow.xul` a `attrTypesWindow.xul`. Okno nastavení je definováno v souboru `config.xul`. Přihlašovací formulář doplňku je definován v souboru `loginPrompt.xul`.

Ke každému uživatelskému rozhraní přísluší `.js` soubor stejného názvu jako soubor `.xul`. V tomto souboru jsou pak implementovány metody, které s daným uživatelským rozhraním souvisí.

Pro uživatelské rozhraní je přípojným bodem do Firefoxu soubor `browserOverlay.xul`. Tento soubor definuje `<overlay/>` pro hlavní okno Firefoxu. Přidá tak položku anotačního doplňku do menu `Zobrazení` a vytvoří tlačítko, které je možné přidat do lišty doplňků Firefoxu a zobrazit tak uživatelské rozhraní. Nejdůležitější částí je přídání elementu `<deck/>` do oblasti webové stránky, jak bylo popsáno v návrhu, bude tento element zobrazovat přihlašovací obrazovku nebo uživatelské rozhraní ze souboru `annotationWindow.xul` podle stavu přihlášení uživatele.

Pomocí tohoto souboru jsou také zpřístupněné a nahrané všechny skripty chromu pro jednodušší údržbu.

## 7.3 Inicializace

K souboru `browserOverlay.xul` je vytvořen soubor `browserOverlay.js`, který implementuje objekt `browserOverlay`. Tento objekt se inicializuje při spuštění Firefoxu (metodou `init()`) a podle nastavení (ze *Systému nastavení*) zobrazí okno anotačního doplňku a případně určí jeho výšku apod. Slouží také jako výchozí bod pro inicializaci dalších objektů, např. uživatele (viz další podkapitola).

V souboru `browserOverlay.js` jsou také zpřístupněny JSM skripty, protože je doplněk většinou potřebuje po celou dobu své existence.

## 7.4 Přihlášení uživatele

Uživatel je reprezentován v souborech `user.js` a `user.jsm`. Objekt `aE.user` se inicializuje pomocí metody `init()` při načtení doplňku a získá nastavení uživatele (pokud existuje), které se ukládá pomocí *Systému nastavení*, tedy: jméno uživatele a zda se má automaticky přihlásit. Vždy je takto uložen naposledy přihlášený uživatel. Případné heslo je uloženo pomocí správce hesel a jeho získání je popsáno dále. Tyto vlastnosti jsou načtené dříve, než se provede kód inicializace objektu `aEC.user`, který si přečte hodnoty z `aE.user` a případně se pokusí přihlásit na server.

Při inicializaci objektu `aEC.user` se také přidá `observer` na zprávy s `aTopic` rovné hodnotě `annotationextension-user-topic`. Takto je zajištěna synchronizace přihlášení uživatele mezi okny.

Při přihlášení uživatele pomocí tlačítka přihlašování se vyvolá metoda `aEC.user.login()`, která uloží data z formuláře (viz další odstavec) a vyvolá upozornění pro `observery`. Tlačítko pro odhlášení pak vyvolá metodu `aEC.user.logout()`, které zašle podobné upozornění.

Metody pro uložení a získání uživatelského jména a jeho hesla ze správce hesel (s kterým pracuje rozhraní `nsILoginManager`) jsou implementovány v souboru `users.jsm`. Pro uložení hesla uživatele slouží metoda `aE.users.addUser()`. Metoda `aE.users.getUserPassword()` implementuje získání hesla pro uživatele.

## 7.5 Modul pro komunikaci se serverem

Implementace modulu je podle návrhu a pro každou zprávu protokolu je vytvořena jedna metoda s reprezentativním názvem, ve které se určí, která metoda modulu má zpracovat odpověď. Metody na zasílání zprávy serveru jsou volány různými částmi doplňku. Modul je implementován v souboru `client.js` a přístupný jako objekt `aEC.client`.

Při přijetí odpovědi od serveru se tato odpověď nejdříve prozkoumá, zda obsahuje element s chybou. Pokud tak server odpověděl, modul použije metodu `aEC.alerts.alert()` pro informování uživatele o chybě. Toto upozornění modul použije také v případě, kdy dojde k selhání komunikace nebo je odpověď poškozena.

Pokud je odpověď v pořádku, modul o tom informuje pomocí metody `notifyObservers()` s určitým tématem a kód, který očekává odpověď od serveru, implementuje metodu `observe()`, která na `notifyObservers()` s patřičným tématem reaguje.

Při přijetí zprávy s typy se každý typ modulem zpracuje a uloží se jako objekt třídy `type` (viz 7.12) do `aEC.types` (tento objekt je popsán v 7.13). A až poté se zašle zpráva o přijetí odpovědi od serveru. Takto můžou být typy dále zpracovány. Podobně se ukládají a předávají skupiny uživatelů.

## 7.6 Hlavní objekt, `annotationWindow.js`

Po přihlášení uživatele dojde k inicializaci hlavního objektu `aEC.annotationWindow` (implementován v `annotationWindow.js`).

Inicializace přidá sledování události puštění tlačítka myši. Při této události se zavolá metoda `selectedText()`, která zobrazí případný vybraný text do pole okna a také se stará o zobrazení tlačítek na výběr fragmentu. Pro výběr fragmentu slouží metoda `showRange()`, která se vyvolá po kliknutí na tlačítko výběru.

Další důležitou věcí při inicializaci je vytvoření instancí třídy `aEC.datasource` pro uložení typů a uložení atributů typů. Jsou to objekty `aEC.typesDatasource` a `aEC.typeAttrDatasource`. Tyto objekty se starají o uložení typů a jejich atributů, které přijme modul, do RDF zdroje dat. K uložení prvků do těchto zdrojů dojde při upozornění modulu (viz předchozí kapitola). Typy jsou v tu chvíli v objektu `aEC.types` a v metodě `processTypes()` se tyto typy přenesou do RDF (metody třídy `datasource`, které toto umožňují, jsou popsány později). `aEC.typesDatasource` je tak připraven poskytnout data pro strom typů. Dále se vytvoří objekt `aEC.attrDatasource`, který poskytuje data pro strom atributů.

V objektu `aEC.annotationWindow` jsou také proměnné ukládájící vybraný typ (jeho jméno a URI). Tyto proměnné jsou sledovány a pokud jsou změněny (například z okna pro výběr typu anotace), zobrazí se patřičná hodnota do pole vybraného typu anotace a také se zobrazí případné atributy.

Do tohoto objektu jsem také zařadil funkce pro práci se stromem atributů.

## 7.7 Výběr typu ze stromu

Okno pro výběr typu používá kód v souboru `typesWindow.js`. Zde jsou metody pro práci se stromem a metoda `addNewType` pro přidání nového typu anotace, která k tomu využívá modul pro komunikaci se serverem. Při inicializaci se vytvoří objekt typu `aEC.treeDatasource` a ke stromu se tak připojí zdroj RDF z `aEC.typesDatasource`.

Výběr typu poté vyvolá funkci `selectType()`, která nastaví nové hodnoty vybraného typu v `aEC.annotationWindow`.

## 7.8 Autocomplete

Komponenta pro *autocomplete textbox* je implementována v souboru `autocomplete.js`. Implementace se řídí předpisem, který tato komponenta musí splňovat, aby ji `<textbox/>` mohl využít. Při vyplňování textu je poté v komponentě volána metoda `aESearch()`, která odesílá pomocí modulu zprávy pro získání typů. Filtrování tak zařídí server. Pokud je navíc využita polem pro výběr typu atributu, přidává dle zadaného textu i jednoduché typy pro zobrazení uživateli. Rozlišení pole pro výběr typu anotace a výběr typu atributu je pomocí atributu `autocompletesearchparam`, který element `<textbox/>` předává v parametru metodě `aESearch()`.

*ContractID* komponenty je `@mozilla.org/autocomplete/search;1?name=aeautocomplete`, pomocí které je přístupná pro `<textbox/>`.

## 7.9 Atributy

Funkce pro práci s atributy a uživatelské rozhraní atributů jsou v `annotationWindow.js`. K načtení a zobrazení atributů dojde při výběru typu. Je při něm spuštěna metoda `selectAttributes()`, která pro vybraný typ ze zdroje atributů typů, nahraje atributy do zdroje zobrazených typů. Tato metoda se také volá, pokud se mají zobrazit atributy vnořené anotace.

Při výběru atributu ve stromu se volá metoda `attrSelected()`, která se postará o zobrazení uživatelského rozhraní atributu apod. tak, jak je popsáno v návrhu v kapitole 6.10.

Při otevření okna na přidání atributů se zdroj dat pro strom typů anotací vytvoří podobně jako v kapitole 7.7 ve skriptu `addAttributeWindow.js`. Strom jednoduchých typů je definován staticky v `.xul` souboru, protože ten se nemění.

V souboru `addAttributeWindow.js` jsou také definovány funkce na přidání atributů. Především pak funkce `addAttribute()`, která se zavolá po kliknutí na tlačítko přidat atribut a přidá tak atribut do RDF zdrojů zobrazených atributů a atributů typů.

V souboru `attrTypesWindow.js` jsou funkce pro výběr typu atributu. Kód stromů je podobný jako pro okno s přidáním atributu. Po kliknutí na tlačítko vybrání typu atributu se zavolá metoda `setTypeToAttribute()` hlavního objektu, která atributu nastaví nový typ a zařídí případnou změnu uživatelského rozhraní či změnu typu atributu v RDF zdrojích dat.

## 7.10 Vytvoření anotace

Pro zadání vnořené anotace slouží tlačítko u uživatelského rozhraní atributu, jak bylo popsáno v kapitole 6.10. Při tomto zmáčknutí se vyvolá metoda hlavního objektu `selectNestedAnnotation()`, která uloží základní anotaci. Základní anotace se ukládá do *hlavního objektu* v jeho vlastnosti `annotation`.

Po ukončení výběru vnořené anotace se vytvoří objekt typu `aEC.annotation` (třída je definována v souboru `annotation.js`), který reprezentuje vnořenou anotaci, a ten se uloží do `aEC.annotations` (viz kapitola 7.13). Třída `aEC.annotation` je implementována dle návrhu v kapitole 6.14.

Po zmáčknutí tlačítka na odeslání anotace se zavolá metoda `saveAnnotations()` hlavního objektu. V této metodě se synchronizuje dokument pomocí metod objektu `aEC.document` (nachází se v souboru `document.js`) a jak je popsáno v kapitole 6.13, projdou se všechna uživatelská rozhraní atributů a vytvoří se z nich anotace.

Proto se volá metoda `aEC.annotationProcessor.makeAnnotations()`, která se postará o vytvoření zprávy na přidání anotace a tu předá v návratové hodnotě.

Objekt `aEC.annotationProcessor` je popsán v souboru `annotationProcessor.js` a obsahuje metody pro zpracování uživatelských rozhraní atributů (pokud je atribut vnořená anotace nalezne pro ni uložené fragmenty v `aEC.annotations`).

## 7.11 Třída `datasource` a treeDatasource

Třída `aEC.datasource` byla navržena pro uložení typů a jejich atributů do RDF, ale prakticky je možné ji využít k uložení jakéhokoliv objektu (viz dále). Každý objekt, který se má do RDF zdroje dat uložit pomocí této třídy, musí obsahovat položky `name` a `uri`. Pokud má třída do RDF ukládat další vlastnosti objektu, jsou sděleny v konstruktoru `aEC.datasource`. Objekty se pak ukládají do RDF pomocí metody `addNewResource()`. Pokud objekt navíc obsahuje vlastnost `ancestor`, přidá se do RDF kontejneru, jehož `uri` je totožná hodnotě `ancestor`. Takto lze například typy anotací do RDF hierarchicky ukládat a poskytnout stromu pro zobrazení.

Atributy v objektu typu se poté uloží do vyhrazeného zdroje RDF pomocí metody `addNewResource2()`, která do RDF uloží typ a všechny jeho atributy (viz 7.12).

Při vytvoření instance třídy `aEC.treeDatasource` dojde k přidání RDF zdroje dat ke stromu, jehož identifikátor (z DOM dokumentu) je obsažen v konstruktoru. Zdroj dat RDF je vzat z objektu typu `aEC.datasource`, který je také předán v konstruktoru. Pokud tento objekt není předán, vytvoří se nový.

## 7.12 Třída pro typ anotace

Tato třída reprezentuje typ anotace a objekty tohoto typu lze ukládat do RDF zdroje dat pomocí `aEC.datasource`. Pro typ je třeba uložit `uri`, `name`, `ancestor` a `group`. Tyto položky jsou v objektu typu uloženy ve stejnojmenných vlastnostech. Atributy typu se ukládájí do pole `attributes[]`. Každý prvek `attributes[]` je pole o třech prvcích: jména atributu, typu atributu a údaji, zda je atribut povinný.

Tato třída je popsána v `type.js`.

## 7.13 Pole pro uložení a předání objektů

Pro předání objektů a jejich uložení je vytvořena univerzální třída `aeArray` v souboru `aeArray.js`. Tato třída obsahuje pole pro uložení objektů a manipuluje s nimi pomocí metod `addNew()` (přidání nového objektu) a `shift()` (vrátí první objekt pole a smaže ho). V doplňku jsou vytvořeny instance této třídy pro typy anotací (`aEC.types`), skupiny uživatelů (`aEC.groups`) a anotace (`aEC.annotations`). Význam těchto objektů byl vysvětlen v kapitole 7.5.

## 7.14 XPath

Pro výpočet XPath slouží objekt `aEC.xpath`, který je implementován v souboru `xpath.js`. Jeho nejdůležitější metodou je `getXPathFromNode()`, která pro zadaný uzel vrátí jeho XPath jako textový řetězec. V návrhu jsem nepočítal se zobrazením anotací pomocí jiného doplňku, který bude k tomuto účelu do DOM dokumentu webové stránky přidávat další elementy. XPath poté nemusí odpovídat skutečnému XPath bez těchto elementů. V budoucnu se tedy musí funkce tohoto objektu mírně upravit.

# Kapitola 8

# Testování a ladění

Po implementaci jsem prováděl testování, během kterého bylo odhaleno několik chyb, které byly opraveny.

Doplněk byl testován na systému Ubuntu 10.4 a 11.04. Vývoj doplňku jsem zaměřil na Mozilla Firefox 4, na kterém jsem také prováděl testování.

Při testech jsem využil anotační server poskytnutý panem Ing. Jaroslavem Dytrychem. Přestože implementace tohoto serveru nebyla dokončená, hlavní a stěžejní části funkčnosti potřebné mým doplňkem bylo možné využít. Toto testování bylo provedeno na lokálním síťovém rozhraní.

Grafické uživatelské rozhraní nelze testovat pomocí automatických skriptů či testů. Proto jsem navrhl několik případových testů, které jsem provedl:

- přihlášení uživatele ve více oknech,
- zobrazení typů ve stromu, přidání typu a vytvoření jednoduché anotace,
- vytvoření strukturované anotace s atributy jednoduchých typů,
- vytvoření strukturované anotace s vnořenými anotacemi, které obsahují další atributy,
- přidání nového atributu a odeslání anotace a
- změnu typu atributu vnořené anotace na jiný typ vnořené anotace.

Pro testování uživatelského rozhraní jsem také použil doplněk DOM inspector[1], který umožňuje zobrazit DOM dokument. Pomocí tohoto nástroje bylo testováno především dynamické vytváření a rušení uživatelského rozhraní atributu.

Při testování jsem také využil program Wireshark[2], pomocí kterého lze sledovat komunikaci na síťovém rozhraní. Sledoval jsem s ním zprávy, které si vyměňoval doplněk se serverem. Přestože lze zprávy zobrazit do konzole nebo pomocí jiné metody, tento nástroj mi pomohl, především v počátcích.

Dále jsem využil doplněk Firebug[3], který poskytuje nástroje pro správu zdrojových kódů v jazyce JavaScript.

[1] https://addons.mozilla.org/en-US/firefox/addon/dom-inspector-6622/
[2] http://www.wireshark.org/
[3] http://getfirebug.com/

# Kapitola 9

# Závěr

V rámci semestrálního projektu jsem se zaměřil na analýzu systému 4A Framework, pro který je doplněk určen. Nastudoval jsem strukturovanou anotaci a protokol pro komunikaci se serverem. Dále jsem provedl analýzu vytváření doplňků pro Firefox. Jako první jsem se zaměřil na pochopení adresářové struktury a významu jednotlivých souborů a poté jsem studoval funkce, které Firefox poskytuje a které budu potřebovat pro implementaci. Po provedené analýze jsem navrhl řešení, které splňuje požadavky na vytvářený doplněk.

V rámci bakalářské práce jsem doplněk implementoval podle návrhu a podařilo se mi naplnit všechny body zadání mé práce. Doplněk jsem také otestoval a ověřil jeho funkčnost.

V budoucnu lze doplněk rozšířit o několik funkcí, které nyní nebyly požadovány. Předně je to vytvoření rozhraní, pomocí kterého bude možné editovat již vytvořené anotace. V současné době není pro vnořenou anotaci podporováno vybrání více fragmentů najednou. Proto by se doplněk mohl rozšířit také o tuto funkci.

Projekt se má stát součástí implementace pro ověření konceptu systému 4A Framework. Dle dostupných informací je vytvářen doplněk pro Firefox, který se stará o zobrazení anotací. V budoucnu tedy bude potřeba spojit implementaci obou doplňků.

# Literatura


[1] Extensible Markup Language (XML) 1.0 (Fifth Edition). [online], [Online; navštíveno 10.5.2011].
URL http://www.w3.org/TR/REC-xml/

[2] Beckett, D.: RDF/XML Syntax Specification (Revised). W3C, c2004, [Online; navštíveno 10.5.2011].
URL http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/

[3] Berglund, A.; Boag, S.; Chamberlin, D.; aj.: XML Path Language (XPath) 2.0 (Second Edition). W3C, c2010, [Online; navštíveno 10.5.2011].
URL http://www.w3.org/TR/2010/REC-xpath20-20101214/

[4] Bos, B.: CSS current work. W3C, c2011, [Online; navštíveno 10.5.2011].
URL http://www.w3.org/Style/CSS/current-work

[5] Bos, B.; Çelik, T.; Hickson, I.; aj.: Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. W3C, c2011, [Online; navštíveno 10.5.2011].
URL http://www.w3.org/TR/CSS2/

[6] Camino. Camino, c2011, [Online; navštíveno 10.5.2011].
URL http://caminobrowser.org/

[7] Dytrych, J.: Webové služby a komponentní technologie, 2011, unpublished.

[8] ecmascript: ECMAScript Documentation. ecmascript, [Online; navštíveno 10.5.2011].
URL http://www.ecmascript.org/

[9] Feldt, K. C.: *Programming Firefox*. O'Reilly Media, 2007, 512 s. ISBN 978-0-596-10243-2.

[10] Firebug. Firebug, c2010, [Online; navštíveno 10.5.2011].
URL http://getfirebug.com/

[11] Garrett, J. J.: Ajax: A New Approach to Web Applications. Adaptive Path Inc., Únor 2005, [Online; navštíveno 10.5.2010].
URL http://www.adaptivepath.com/ideas/e000385

[12] Hors, A. L.; Hégaret, P. L.; Wood, L.; aj.: Document Object Model (DOM) Level 3 Core Specification. W3C, c2004, [Online; navštíveno 10.5.2011].
URL http://www.w3.org/TR/DOM-Level-3-Core/

[13] van Kesteren, A.: XMLHttpRequest. W3C, c1999, [Online; navštíveno 10.5.2011].
URL http://www.w3.org/TR/XMLHttpRequest/

[14] ISO 639-2 Language Code List - Codes for the representation of names of languages. Library of Congress, Říjen 2010, [Online; navštíveno 2.1.2011].
URL http://www.loc.gov/standards/iso639-2/php/code_list.php

[15] McCarron, S.; Ishikawa, M.: XHTML™ 1.1 - Module-based XHTML - Second Edition. W3C, c2010, [Online; navštíveno 10.5.2011].
URL http://www.w3.org/TR/xhtml11/

[16] About this Reference. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/JavaScript/Reference/About#JavaScript

[17] Adding sidebars. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/XUL_School/Adding_sidebars

[18] Alerts and Notifications. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/Code_snippets/Alerts_and_Notifications

[19] Chapter 2: Technologies used in developing extensions. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/En/Firefox_addons_developer_guide/Technologies_used_in_developing_extensions

[20] Chrome. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/chrome

[21] Chrome registration. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/Chrome_Registration

[22] Components.classes. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/Components.classes

[23] document.createTreeWalker. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/dom/document.createtreewalker

[24] DOM Levels. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/En/DOM_Levels

[25] FUEL. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/FUEL

[26] Gecko. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/Gecko

[27] Gecko FAQ. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/Gecko_FAQ

[28] Handling Preferences. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/XUL_School/Handling_Preferences

[29] Install Manifests. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/install_manifests

[30] JavaScript. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/JavaScript

[31] The Joy of XUL. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/The_Joy_of_XUL

[32] Mozilla CSS support chart. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/Mozilla_CSS_support_chart

[33] nsIObserverService. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/NsIObserverService

[34] Plugins. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/Plugins

[35] Preferences. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/Code_snippets/Preferences#Resources

[36] RDF. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/rdf

[37] Selection. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/DOM/Selection

[38] Setting Up a Development Environment. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https:
//developer.mozilla.org/en/XUL_School/Setting_Up_a_Development_Environment

[39] textbox (Toolkit autocomplete). Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/En/XUL/Textbox_%28Toolkit_autocomplete%29

[40] Using JavaScript code modules. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/JavaScript_code_modules/Using

[41] Using XPCOM Components. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https:
//developer.mozilla.org/en/Creating_XPCOM_Components/Using_XPCOM_Components

[42] Working with windows in chrome code. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/Working_with_windows_in_chrome_code

[43] XMLHttpRequest. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/XMLHttpRequest

[44] XPCOM. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/XPCOM

[45] XPCOM API Reference. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/XPCOM_API_Reference

[46] XPCOM Interface Reference. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/XPCOM_Interface_Reference

[47] XPCOM Objects. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/XUL_School/XPCOM_Objects

[48] XPCOM Objects. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/XUL_School/XPCOM_Objects

[49] XUL. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/XUL

[50] XUL Reference. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/XUL_Reference

[51] XUL School Tutorial. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/XUL_School

[52] XUL Structure. Mozilla Developer Network, c2011, [Online; navštíveno 10.5.2011].
URL https://developer.mozilla.org/en/XUL_Tutorial/XUL_Structure

[53] Oracle. Oracle, 2011, [Online; navštíveno 10.5.2011].
URL http://www.oracle.com/index.html

[54] HTML DOM Tutorial. w3schools.com, 2011, [Online; navštíveno 10.5.2011].
URL http://www.w3schools.com/htmldom/default.asp

[55] HTML Tutotrial. w3schools.com, 2011, [Online; navštíveno 10.5.2011].
URL http://www.w3schools.com/html/default.asp

[56] Introduction to XHTML. w3schools.com, c2011, [Online; navštíveno 10.5.2011].
URL http://www.w3schools.com/xhtml/xhtml_intro.asp

[57] XPath Introduction. w3schools.com, c2011, [Online; navštíveno 10.5.2011].
URL http://www.w3schools.com/xpath/xpath_intro.asp

[58] History of Firefox. In *Wikipedia – The Free Encyclopedia*, nadace WIKIMEDIA, 2011, [Online; navštíveno 10.5.2011].
URL http://en.wikipedia.org/wiki/History_of_Firefox

[59] JavaScript. In *Wikipedia – The Free Encyclopedia*, nadace WIKIMEDIA, 2011, [Online; navštíveno 10.5.2011].
URL http://en.wikipedia.org/wiki/JavaScript

[60] Zakas, N. C.; McPeak, J.; Fawcett, J.: *Ajax - Profesionálně*. Brno: Zoner Press, 2007, 672 s. ISBN 978-80-86815-77-0.

[61] Zakas, N. Z.: *JavaScript pro webové vývojáře*. Computer Press, 2001, 832 s. ISBN 978-80-251-2509-0.

# Přílohy

# Seznam příloh

# Příloha A

# Ukázka strukturované anotace

```
<rdf:Description rdf:about="http://example.com/annotations/123456">
  <rdf:type rdf:resource="http://example.com/types/g01/annotation/task"/>
  <a:dateTime rdf:value="2011-01-01T:20:00:00Z" />
  <a:author id="http://example.com/authors/123456" name="Jaroslav Dytrych"
    address="idytrych@fit.vutbr.cz"/>
  <a:source rdf:resource="http://example.com/documents/getDoc?id=123456"/>
  <a:fragment>
    <a:path>
 /html/body/div[@id='container']/div[@id='main']/div[@id='post1']/DIV[2]/p[1]
    </a:path>
    <a:offset>22</a:offset>
    <a:length>32</a:length>
    <a:annotatedText>Fakulta informačních technologií</a:annotatedText>
  </a:fragment>
  <a:content>
    <![CDATA[
        ...
      ]]>
  </a:content>
  <a:attribute name="place" type="geoPoint">
    <geo:Point>
        <geo:lat>55.701</geo:lat>
        <geo:long>12.552</geo:long>
    </geo:Point>
  </a:attribute>
  <a:attribute name="date" type="nestedAnnotation">
    <rdf:Description rdf:about="http://example.com/annotations/123457">
      <rdf:type
      rdf:resource="http://example.com/types/g01/annotation/description"/>
      <a:dateTime rdf:value="2011-01-01T:20:00:00Z" />
      <a:author id="http://example.com/authors/123456"
                name="Jaroslav Dytrych"
                address="idytrych@fit.vutbr.cz"/>
      <a:source
      rdf:resource="http://example.com/documents/getDoc?id=123456"/>
      <a:fragment>
        <a:path>
/html/body/div[@id='container']/div[@id='main']/div[@id='post1']/p[1]
        </a:path>
```

```
        <a:offset>92</a:offset>
        <a:length>14</a:length>
        <a:annotatedText>14. ledna 2011</a:annotatedText>
      </a:fragment>
      <a:content>
      <![CDATA[
            ...
          ]]>
      </a:content>
      <a:attribute name="date" type="DateTime"
          rdf:value="2011-01-14T:00:00:00Z"/>
    </rdf:Description>
  </a:attribute>
  <a:attribute
    name="reason"
    type="annotationLink"
    uri="http://example.com/annotations/1234567"/>
</rdf:Description>
```

# Příloha B

# Protokol pro přenos anotací mezi klientem a serverem

## B.1 Správa sezení

Správa sezení zahrnuje dohodu na verzi protokolu, přihlášení a odhlášení uživatele.

Nejprve klient zahájí spojení tak, že zašle na server zprávu `connect`, kde v atributu uvede nejvyšší verzi protokolu, kterou může využít:

```
<connect protocolVersion="1.0"/>
```

Server odpoví chybou nebo následující zprávou:

```
<connected protocolVersion="1.0" sessionID=""/>
```

V atributu `protocolVersion` server uvede verzi protokolu, kterou bude komunikovat. Server by měl komunikovat stejnou verzí jako klient, nebo nejnižší verzí, která je zpětně kompatibilní s verzí klienta. Pokud klient nabídne novější verzi než server, server použije nejnovější verzi, kterou zná. Pokud klient zjistí, že jeho verze není zpětně kompatibilní, musí přepnout na verzi serveru (případně jinou s ní kompatibilní), nebo se odpojit. Pokud verze protokolu není podporována a server vrátí chybovou zprávu, klient se může pokusit o spojení se starší verzí protokolu.

Vzhledem k tomu, že novější verze protokolu může být zpětně kompatibilní, klient a server mohou implementovat různé verze protokolu. Pokud server či klient umožňuje využít novou funkcionalitu, protistrana se starší verzí protokolu ji nevyužije a zaslané elementy či atributy navíc bude ignorovat. Pokud nová verze protokolu nebude zpětně kompatibilní, server či klient, který ji implementuje, musí spojení s danou kombinací verzí protokolu odmítnout.
Ukončení spojení je signalizováno následující zprávou:

```
<disconnect/>
```

Ve zprávě `connected` server zašle také id sezení (atribut `sessionID`), které bude klient zasílat se všemi následujícími zprávami v atributu `id` elementu session:

```
<session id=""/>
```

Přihlášení je realizováno následující zprávou:

```
<login user="" password=""/>,
```

kde `user` je uživatelské jméno nebo e-mail uživatele a `password` je heslo uživatele. V případě úspěšného přihlášení server na tuto zprávu odpoví seznamem parametrů nastavení a následující zprávou:

```
<logged id="" name=""/>,
```

kde `id` je identifikátor uživatele a `name` jeho zobrazované jméno. V případě neúspěšného přihlášení server odpoví chybovou zprávou.
Odhlášení bude prováděno zprávou `<logout/>`.

Přihlášení lze provést současně se zahájením spojení a dohodou na verzi protokolu, odhlášení společně s jeho ukončením.

## B.2 Uživatelé a skupiny

Pro získání informací o profilech uživatelů zašle klient zprávu:

```
<queryPersons filter="" withGroups=""/>
```

Ve filtru lze pro výběr pole využít klíčové slovo `id`, `e-mail` či `name` následované dvojtečkou. Při filtrování dle více polí budou filtry odděleny středníkem. Pokud bude uveden i volitelný atribut `withGroups` s hodnotou `true`, budou v informacích o profilech zahrnuty i informace o členství ve skupinách. Server na tuto zprávu odpoví zprávou:

```
<persons>
  <person id="" login="" name="" email="" photoURI=""/>
</persons>
```

Atribut name obsahuje celé jméno uživatele, `photoURI` slouží k získání URI fotografie, která bude u uživatele zobrazena. Pokud byly požadovány informace o skupinách, bude každá značka `person` obsahovat i značku `userGroups` (viz níže). Značka `person` by mohla obsahovat i další značky a textový obsah s informacemi o profilu uživatele.

Pro získání informací o skupinách uživatelů zašle klient zprávu:

```
<queryUserGroups filter="" withPersons=""/>
```

Ve filtru lze využít URI skupiny nebo její název. Pokud bude uveden i volitelný atribut `withPersons` s hodnotou `true`, budou v informacích o skupinách zahrnuty i informace o jejich členech. V názvu lze využít zástupné symboly „*“ (libovolný počet libovolných znaků). Server na tuto zprávu odpoví:

```
<userGroups>
  <group uri="" name=""/>
</userGroups>
```

Pokud byly požadovány informace o členech skupin, v každé značce `group` bude obsažena i značka `persons` (viz výše).
Pro přihlášení ke skupině uživatelů klient zašle následující zprávu:

```
<join group=""/>
```

kde atribut group obsahuje URI skupiny. K odhlášení uživatele ze skupiny slouží zpráva:

```
<leave group=""/>
```

## B.3 Řízení odběru anotací

Klient může přijímat pouze anotace zvolených typů ze zvolených zdrojů. Zdrojem může být jiný uživatel nebo URI, který identifikuje anotační server, skupinu uživatelů či jiný obecný zdroj.
Klient se k odběru anotací přihlašuje následující zprávou:

```
<subscribe>
  <source type="" user=""/>
  <source type="" uri=""/>
  <source type=""/>
  <source user=""/>
  <source uri=""/>
</subscribe>
```

Elementů `source` může být libovolné nenulové množství a mohou mít kombinace parametrů, které jsou uvedeny výše. Parametr `user` identifikuje uživatele, `uri` obecný zdroj anotací. Parametr `type` udává typ anotací, přičemž s typem jsou automaticky vybrány i všechny podtypy. V typu může být využit i zástupný symbol „*“, který nahrazuje libovolný počet libovolných znaků.

Pokud není uveden typ, budou přijímány všechny typy anotací (dle skupin, ve kterých se uživatel nachází) z daného zdroje. Pokud není uveden zdroj, budou přijímány všechny anotace daného typu. K odhlášení může klient využít zprávu `unsubscribe`:

```
<unsubscribe>
  <source type="" user=""/>
  <source type="" uri=""/>
  <source type=""/>
  <source user=""/>
  <source uri=""/>
</unsubscribe>
```

Pro odhlášení platí stejná pravidla jako pro přihlášení (libovolný počet elementů `source`, atd.). Klient automaticky odebírá všechny anotace od svého přihlášeného uživatele, pokud se od jejich odběru explicitně neodhlásí.

## B.4 Synchronizace dokumentu

Synchronizace dokumentu je proces, při kterém server získá kopii aktuální verze anotovaného dokumentu. Pokud server tento dokument získá poprvé, uloží si jej a vrátí klientovi adresu uložené verze, kterou bude klient využívat v anotacích. Pokud má server dokument uložený, porovná novou verzi s uloženou verzí, a pokud se shodují, zašle klientovi adresu uložené verze. Pokud se dokumenty shodují částečně, server vyhodnotí změny. Pokud změny neovlivní žádné anotace, dokument se transparentně aktualizuje (pro klienta stejné jako shoda dokumentů). Pokud by změny ovlivnily některé anotace, server zašle klientovi varování, že některé anotace musely být aktualizovány, a synchronizaci dokončí. V případě zásadnějších změn či zneplatnění anotací dojde k chybě synchronizace a uživatel se musí rozhodnout, zda synchronizaci dokončí a zneplatní tak některé či všechny anotace (vymaže či přesune na úroveň celého dokumentu), nebo nedokončí a bude anotovat uloženou (starší) verzi dokumentu či jiný dokument.

Protože klient může být i jednoduchý textový editor, který nepracuje se strukturovaným textem, server musí podporovat i linearizaci dokumentu. V tomto případě klient dokument linearizuje na prostý text a zašle jej serveru v linearizované podobě. Pokud má server strukturovanou podobu, linearizuje ji a porovná se zaslanou. Pokud se linearizované verze neshodují, dojde k chybě synchronizace. Pokud se shodují, synchronizace bude dokončena a server bude každou následně zaslanou anotaci upravovat pro strukturovanou verzi dokumentu.
Syntaxe:

```
<synchronize
  resource="http://example.com/documents/doc1.txt"
  linearize="false" overwrite="false">
  <content>
    <![CDATA[
      ...
```

```
    ]]>
  </content>
</synchronize>
```

Klient pošle serveru kopii anotovaného dokumentu a adresu, ze které pochází. Parametr `resource` udává umístění zdroje (např. URI anotované webové stránky). Element content obsahuje obsah daného dokumentu.

Nepovinný parametr `linearized` udává, zda klient pracuje s linearizovanou verzí dokumentu. Výchozí hodnota je false.

Nepovinný parametr `overwrite` umožňuje vynutit synchronizaci v případě, kdy je dokument s daným URI na serveru uložen, ale jeho obsah se neshoduje. Server v tomto případě musí nahradit uložený dokument a upravit (přesunout na úroveň celého dokumentu a doplnit textový obsah o informaci o změně) či vymazat všechny anotace. Klient by tento atribut neměl využít při prvním pokusu o synchronizaci. Jeho použití musí být schváleno uživatelem. Pokud se obsah textu shoduje, server atribut ignoruje. Výchozí hodnota je false.

Při úspěšné synchronizaci server odpoví zprávou:

```
<synchronized resource=""/>
```

Atribut `resource` obsahuje URI kopie anotovaného dokumentu, která je uložena na serveru. Tento URI musí klient využívat v anotacích.

Pokud v průběhu práce dojde k situaci, kdy se obsah anotovaného fragmentu neshoduje s obsahem, který server nalezne na dané pozici v dokumentu, dojde k tzv. rozsynchronizování. V tomto případě server zašle chybovou zprávu a zprávu `<resynchronize/>` (element bez obsahu a atributů), čímž požádá o resynchronizaci.
Klient provádí resynchronizaci zasláním obsahu v následující zprávě:

```
<resynchronize>
  <content>
    <![CDATA[
      ...
    ]]>
  </content>
</resynchronize>
```

Po resynchronizaci je vždy nutné znovu načíst všechny anotace. Server je tedy automaticky zašle v odpovědi.
Pokud klient provádí modifikace dokumentu, musí každou změnu zaslat na server:

```
<textModification path="" offset="" length="">
  <![CDATA[
    Nový obsah vybraného fragmentu ...
  ]]>
</textModification>
```

Atribut `path` udává XPath uzlu DOM dokumentu, ve kterém byla změna provedena. Atributy `offset` a `length` udávají offset a délku změněného fragmentu. V těle elementu `<textModification/>` je potom uveden nový obsah fragmentu. Pokud je vložen nový fragment, délka původního fragmentu je nulová. Pokud je fragment vymazán, element je prázdný. Server následně tuto zprávu rozešle všem klientům pracujícím se stejným dokumentem.

## B.5 Přenos typů anotací

Přenos typů anotací probíhá obousměrně. Pokud je přidán, upraven či vymazán typ, klient tuto změnu okamžitě zašle serveru a ten ji rozešle všem ostatním klientům, kterých se týká. Klient však nemusí udržovat kompletní strom typů, ale může mít načtenou pouze jeho část. V tomto případě

může server buď zasílat všechny změny, nebo může udržovat informaci o tom, které části stromu má klient načtené, a zasílat pouze informace o změnách v těchto částech. Server musí vždy udržovat kompletní strom typů dané skupiny uživatelů (všechny změny jsou mu zasílány).
Klient o část stromu typů žádá zprávou:

```
<queryTypes filter=""/>
```

Atribut `filter` umožňuje získání určitého podstromu typů. Jedná se o linearizovaný název či URI typu se zástupnými symboly „*“ (libovolný počet libovolných znaků). Filtr tedy umožňuje vybrat podstrom nebo množinu typů, jejichž název či URI obsahuje daný text.

Pokud klient zažádá o strom typů, server na to odpoví zprávou s přidáním typů (viz níže). Pokud filtru nevyhovuje žádný typ, server zašle prázdný seznam typů.
Přenos typů anotací je prováděn následující zprávou:

```
<types>
  <add>
    <type name="" ancestor="" uri="" group="">
      <attribute name="" type="" required=""/>
    </type>
  </add>
  <change/>
  <remove/>
</types>
```

Element `types` obsahuje:

- element `add`, pokud byl přidán typ,
- element `change`, pokud byl upraven typ,
- element `remove`, pokud byl vymazán typ.

V každém ze tří výše uvedených elementů může být obsažen libovolný počet elementů `type`. Atributy tohoto elementu jsou `name` (název typu), `ancestor` (URI rodičovského typu), `uri` (URI typu) a `group` (URI skupiny uživatelů, které typ patří). Pokud je URI rodičovského typu prázdný, jedná se o základní typ. URI jednoznačně identifikuje typy. Pokud není uvedena skupina, určí se z URI typu, podle rodičovského typu nebo podle výchozí skupiny uživatele.

U každého typu mohou být definovány i výchozí atributy. Tyto jsou potom obsaženy v elementech `attribute`. Každý atribut má název (`name`) a typ (`type`). Pokud se jedná o jednoduchý datový typ, je uveden název tohoto typu. Pokud se jedná o vnořenou anotaci či odkaz na anotaci, je uveden očekávaný typ této anotace (informace o vnoření či odkazování zde není potřebná, protože obě varianty jsou zde významově ekvivalentní). Pokud je atribut povinný, má element `attribute` i atribut `required` s hodnotou `true`.

Název typu nelze upravit jinak, než smazáním starého typu a přidáním nového.

## B.6 Přenos anotací

Přenos anotací je prováděn obdobně jako přenos typů:

```
<annotations>
  <add>
    <annotation/>
  </add>
  <change/>
  <remove/>
</annotations>
```

Každé přidání, úprava či vymazání anotace jsou ihned zaslány na server. V elementu `annotations` jsou dle provedené operace obsaženy elementy `add` (přidané) `change` (upravené) a `remove` (vymazané). V každém z těchto elementů může být obsažen libovolný počet elementů `annotation`.

Server každou změnu ihned zašle klientům, kterých se týká (u kterých dotčená anotace patří mezi odebírané). Pokud je přidána anotace, je tato anotace vždy zaslána i klientovi, který ji přidal, aby získal přidělený identifikátor anotace.

Po synchronizaci či resynchronizaci dokumentu jsou klientovi jako přidané automaticky zaslány všechny anotace, které patří k tomuto dokumentu a vyhovují klientem definovaným požadavkům na odběr anotací (zdroje a typy).

Pokud klient potřebuje znovu načíst některou anotaci (např. po neúspěšném pokusu o editaci) či všechny anotace, může serveru zaslat jednu z následujících dvou variant zprávy:

```
<reload uri="http://example.com/annotations/123456"/>
<reload all="true"/>
```

Atribut `uri` u první varianty zprávy udává URI požadované anotace, atribut `all` u druhé varianty udává, že mají být znovu zaslány všechny anotace.

## B.7 Nabízení anotací

Server může klientovi nabídnout automaticky vygenerované anotace k danému dokumentu či jeho části. Klient o nabídku anotací zažádá následovně:

```
<suggestAnnotations path="" offset="" length="" type=""/>
```

Atributy `path`, `offset` a `length` udávají cestu (XPath), offset a délku fragmentu dokumentu, ke kterému mají být nabídnuty anotace. Pokud je uvedena pouze cesta, budou nabídnuty anotace k celému uzlu DOM dokumentu. Pokud není uveden žádný z těchto atributů, budou nabídnuty anotace k celému dokumentu. Volitelný atribut type udává požadovaný typ nabízených anotací. S tímto typem budou současně nabízeny i všechny jeho podtypy.
Server odpoví nabídkou anotací:

```
<suggestions>
  <annotation tmpId="" confidence=""/>
</suggestions>
```

V elementu `suggestions` může být libovolný počet anotací (elementů `annotation`). Atribut `confidence` udává odhadnutou míru jistoty anotace v procentech. Hodnota může být využita klientem při automatickém přijímání a odmítání anotací. Anotace v nabídce nemají trvalý identifikátor, ale pouze dočasný (`tmpId`). Dočasný identifikátor slouží k informování serveru o manipulaci s nabídkami a vyřazení anotace z nabídky při její aktualizaci.

Pokud klient chce anotaci potvrdit (ať už akcí uživatele či automaticky na základě uživatelského nastavení), vrátí ji serveru mezi přidávanými anotacemi (ve zprávě `annotations`), přičemž dané anotaci (elementu `annotation`) přidá atributy `confirmed` a `tmpId`. Atribut `tmpId` udává dočasný identifikátor potvrzené anotace. Atribut `confirmed` udává způsob potvrzení a může nabývat následujících hodnot:

- `manually` - uživatelem potvrzená anotace,
- `manuallyEdited` - uživatel anotaci editoval a uložil,
- `automatically` - automaticky potvrzená anotace (dle nastavení doplňku),
- `automaticallyEdited` - automaticky potvrzená anotace s provedením automatických úprav.

Pokud dojde ke změně dokumentu, může být potřeba upravit nabídku anotací. V tomto případě server okamžitě zašle aktualizaci nabídky anotací. Pro jednodušší implementaci klienta není podporována úprava nabídnutých anotací. Pokud se některá anotace změní, je odstraněna a server nabídne novou verzi. V elementu `suggestions` tedy může být i libovolný počet elementů `delete`:

```
<suggestions>
  <annotation tmpId="" confidence=""/>
  <delete tmpId=""/>
</suggestions>
```

Pokud klient nemá anotaci s daným dočasným identifikátorem, element `delete` ignoruje. Pokud uživatel (či klient, dle nastavení) některou nabídku odmítne, klient zašle na server zprávu:

```
<refusedSuggestions>
  <suggestion tmpId="" method=""/>
</refusedSuggestions>
```

Atribut `method` udává způsob odmítnutí a může mít jednu z následujících hodnot:

- `manually` - odmítnutí uživatelem,
- `automatically` - automatické odmítnutí na základě nastavení.

Pokud klient nechce přijímat další aktualizace nabídky anotací, zažádá server o nabídky anotací k fragmentu dokumentu s nulovou délkou.

## B.8 Přenos nastavení

Nastavení je seznam položek, které mají název a řetězcovou hodnotu. Nastavení lze rozdělit na nastavení serveru a nastavení klienta s tím, že nastavení klienta budou mít prefix „`Client`“ (např.: „`ClientAnnotationTypeColor:Animal->People->Employee`“ s hodnotou „`green`“). Při zobrazení uživateli se potom některá (známá) nastavení zpracují a zobrazí ve formulářích a ostatní se vypíšou v tabulce pro ostatní nastavení, kde je uživatel může měnit.

Konkrétní položky nastavení závisí na implementaci serveru a klienta. Aby nedocházelo k problémům při využití více různých klientů jedním uživatelem, měly by být názvy položek nastavení prefigovány i typem a názvem klienta (např. „`ClientFFExtAnnotFox`“ bude prefix pro rozšíření Firefoxu nazvané AnnotFox) nebo by měly být takové, aby byl jejich význam zcela zřejmý (např. „`ClientDefaultAnnotationType`“ pro výchozí typ anotace).

Vzhledem k tomu, že je předpokládán malý počet položek a malá frekvence přenášení, vždy je přenášen kompletní seznam položek nastavení. Neuvedení položky tedy povede k jejímu odstranění (je-li možné). Pokud položku není možné odstranit (např. nastavení serveru), bude nastavena na výchozí hodnotu.
Nastavení se přenáší zprávou:

```
<settings>
  <param name="" value=""/>
</settings>
```

Elementů `param`, které tvoří jednotlivé položky, může být libovolný (i nulový) počet. Atribut `name` obsahuje název položky, atribut `value` řetězcovou hodnotu položky.

## B.9 Chyby a varování

Chybové zprávy slouží k informování klienta o chybě. Chybová zpráva obsahuje číslo chyby (`number`) a její textový obsah (v elementu `message`). Může obsahovat i doplňující informace, které se týkají konkrétní chyby. Při chybě oprávnění při přístupu k anotacím bude obsažena informace o tom, ke kterým zdrojům byl odepřen přístup. Při nezdařené operaci s existující anotací (úprava či mazání) musí chybová zpráva obsahovat informaci o tom, které anotace se týká (kterou anotaci je třeba znovu načíst). Při problému s atributy musí obsahovat i informaci o tom, kterých atributů se týká.

Textový obsah chybových zpráv bude lokalizován do jazyka nastaveného parametrem „`ServerLanguage`“, který bude mít hodnoty dle ISO 639-2 [14] (varianty pro bibliografické účely). Pokud tento parametr nebude nastaven, zprávy budou v angličtině.
Syntaxe:

```
<error number="1">
  <message>
    <![CDATA[
      Neplatné přihlašovací jméno nebo heslo.
    ]]>
  </message>
</error>

<error number="2">
  <accessDenied user=""/>
  <accessDenied uri=""/>
  <accessDenied type=""/>
  <accessDenied type="" user=""/>
  <accessDenied type="" uri=""/>
  <message>
    <![CDATA[
      Nemáte oprávnění k˜prohlížení zvolených anotací.
    ]]>
  </message>
</error>
<error number="3">
  <message>
    <![CDATA[
      Nemáte oprávnění přidávat anotace.
    ]]>
  </message>
</error>
<error number="4">
  <reload uri="http://example.com/annotations/123456"/>
  <message>
    <![CDATA[
      Editace dané anotace není povolena.
    ]]>
  </message>
</error>
<error number="5">
  <reload uri="http://example.com/annotations/123456"/>
  <message>
    <![CDATA[
      Vymazání dané anotace není povoleno.
    ]]>
  </message>
</error>
<error number="6">
  <reload uri="http://example.com/annotations/123456"/>
  <message>
    <![CDATA[
      Anotace tohoto typu musí obsahovat následující atributy: ...
    ]]>
  </message>
  <attribute name="" type=""/>
  <attribute name="" type=""/>
</error>
<error number="7">
  <reload uri="http://example.com/annotations/123456"/>
```

```
  <message>
    <![CDATA[
      Hodnota atributu ... musí být v˜rozsahu ...
    ]]>
  </message>
  <attribute name="" type=""/>
</error>
```

Čísla chyb jsou:

**0** Nepodporovaná verze protokolu.
**1** Chybné přihlašovací údaje.
**2** Nedostatečná oprávnění k požadovaným anotacím.
**3** Nelze přidávat anotace - přístup je pouze pro čtení.
**4** Editace zvolené anotace není povolena.
**5** Mazání anotace není povoleno.
**6** Chybí povinné atributy.
**7** Nedovolená hodnota atributu.
**8** Chybné určení textu pro návrhy anotací.
**9** Chyba synchronizace (s daným URI je asociován rozdílný obsah).
**10** Nucená synchronizace není povolena.
**11** Rozsynchronizování (neshoda anotovaného textu s textem na dané pozici).
**12** Editace daného typu není povolena.
**13** Mazání daného typu není povoleno.
**14** Nelze přidávat typy anotací.
**15** Neznámý typ atributu.
**16** Chyba v přidávaném typu atributu.
**17** Atributy přidávaného typu jsou chybné. Tyto atributy byly vynechány.
**18** Chyba v upravovaném typu anotace - změny nelze uložit.
**19** Neznámý typ anotace
**20** Změna názvu typu či nadtypu není možná.
**21** Chyba v nastavení. Změny nelze uložit.
**22** Pokus o synchronizaci bez adresy dokumentu.
**23** Pokus o synchronizaci bez obsahu dokumentu.
**24** Chybná adresa zdroje anotovaného fragmentu.
**25** Chybný anotovaný fragment. Anotace bude uložena bez tohoto fragmentu.
**26** Chybný atribut anotace. Atribut nelze uložit.
**27** Chybná hodnota v atributu rozšířeného typu.
**28** Chyba v informacích o způsobu potvrzení nabídnuté anotace.
**29** Upravovaná anotace nebyla nalezena. Změny nelze uložit.
**30** Mazaná anotace nebyla nalezena. Anotaci nelze vymazat.
**31** Vaše přihlášení vypršelo.
**32** Chybná zpráva. Pravděpodobně chyba klienta nebo nekompatibilní protokol.
**33** Chyba v modulu serveru.
**34** Požadovaná anotace nebyla nalezena.
**35** Chybná cesta v anotovaném fragmentu. Fragment nebude uložen.
**36** Chyba v anotovaném dokumentu.
**37** Chybný offset nebo délka v anotovaném fragmentu. Fragment nebude uložen.
**38** V upravené anotaci jsou chyby. Změny nelze uložit.
**39** Modifikaci textu není možné aplikovat na dokument.
**40** Neznámý typ anotací. Nabízení anotací není možné.
**41** Chybný formát data v anotaci. Datum bylo upraveno na aktuální.
**42** Chybný formát data v atributu. Atribut byl vynechán.
**43** Bez synchronizace dokumentu nelze manipulovat s anotacemi.
**44** Chyba v informacích o autorovi anotace.
**45** Neznámá skupina uživatelů.
**46** Neznámá skupina uživatelů v typu anotace. Skupina bude nastavena dle definovaných pravidel.
**47** Mazání využitých typů anotací není povoleno. Nejprve je nutno vymazat anotace tohoto typu.
**48** Interní chyba serveru způsobila neúspěch při ukládání dat. Žádná data nebyla uložena.

**49** Pokus o vytvoření duplicitního typu anotace (shoda URI).
**50** Chybný popis modifikace textu.
**100** Neznámá chyba.

Pokud dojde k zneplatnění anotace či k jejímu přesunu na globální úroveň, aniž by došlo k chybě, může být potřeba varovat uživatele. V tomto případě server zašle zprávu s varováním. Klient by měl varování zobrazit uživateli a to buď přímo u dotčené anotace, nebo pomocí nějakého postranního panelu či dialogového okna.

Zpráva s varováním má následující syntaxi:

```
<warning number="1" annotation="http://example.com/annotations/123456">
  <![CDATA[
    Anotace byla zneplatněna editací textu.
  ]]>
</warning>
```

Každé varování má číslo (atribut `number`) a textový obsah pro zobrazení uživateli. Pokud se varování týká konkrétní anotace, element `warning` má i atribut `annotation`, který obsahuje URI dané anotace.
Čísla varování jsou:

**1** Anotace zneplatněna.
**2** Anotace přesunuta na globální úroveň.
**3** Anotace automaticky aktualizována.
**4** Některé anotované fragmenty v anotaci byly zneplatněny.
**100** Jiné varování.

## B.10 Potvrzení bez doplňujících dat

Pokud je zaslána zpráva, která vyžaduje provedení nějaké operace, je třeba na ni odpovědět, aby druhá komunikující strana měla potvrzeno přijetí zprávy, případně úspěšné provedení operace, kdy nejsou vrácena data. Pokud není zaslána chybová zpráva, předpokládá se úspěšné provedení operace. Odpověď tedy může obsahovat pouze informace, které jsou důsledkem provedené operace, či jiné užitečné informace. V některých případech však nejsou k dispozici žádné užitečné informace k zaslání a je potřeba, aby server či klient potvrdil úspěšnost operace. V tomto případě pošle následující zprávu:

```
<ok/>
```

## B.11 Zjednodušený příklad komunikace mezi klientem a serverem

Klient (zahájení komunikace, uvedení verze protokolu, autentizace):

```
<connect/>
<login/>
```

Server (potvrzení, uvedení verze protokolu, nastavení parametrů):

```
<connected/>
<logged/>
<settings/>
```

Klient (volba zdrojů anotací a anotovaného textu, požadavek na seznam typů):

```
<session/>
<subscribe/>
<synchronize/>
<queryTypes/>
```

Server (adresa zasynchronizovaného zdroje (kopie na serveru), požadované anotace a typy):

```
<synchronized/>
<annotations/>
<types><add/></types>
```

Klient (požadavek na přidání typu anotace):

```
<session/>
<types><add/></types>
```

Server:

```
<ok/>
```

Klient (zašle vloženou anotaci):

```
<session/>
<annotations/>
```

Server (zašle zpět vloženou anotaci s přiděleným identifikátorem):

```
<annotations/>
```

Klient (zašle upravenou anotaci):

```
<session/>
<annotations/>
```

Server (vrátí chybu):

```
<errors/>
```

Klient (požádá o původní obsah anotace):

```
<session/>
<reload/>
```

Server (vrátí požadovanou anotaci):

```
<annotations/>
```

Klient (přidání anotace):

```
<session/>
<annotations/>
```

Server (rozsynchronizování):

```
<errors/>
<resynchronize/>
```

Klient (opakování synchronizace):

```
<session/>
<resynchronize/>
```

Server (všechny anotace pro daný text ze zvolených zdrojů):

```
<annotations/>
```

Klient (požadavek na změnu nastavení):

```
<session/>
<settings/>
```

Server (aktuální nastavení):

```
<settings/>
```

Klient (požadavek na získání nabízených anotací):

```
<session/>
<suggestAnnotations/>
```

Server (nabídka anotací):

```
<suggestions/>
```

Klient (přechod na jiný anotovaný text):

```
<session/>
<synchronize/>
```

Server:

```
<synchronized/>
<annotations/>
```

Klient (odhlášení, ukončení spojení):

```
<session/>
<logout/>
<disconnect/>
```

Server:

```
<ok/>
```