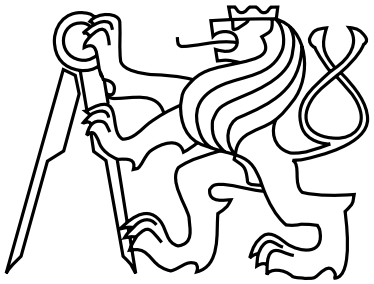
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačové grafiky a interakce



Bakalářská práce

Extrakce webu pomocí Mozilla Application Frameworku

Jiří Mašek

Vedoucí práce: Ing. Tomáš Novotný

Studijní program: Softwarové technologie a management, Bakalářský

Obor: Web a multimedia

28. května 2010

Poděkování

Za trpělivost a cenné rady děkuji vedoucímu své práce Ing. Tomáši Novotnému.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 28. 5. 2010 .....

Abstract

The aim of this thesis was to design and implement an interface that would enable to create a configuration for the web extractor, developed by supervisor of this project. Definition of content intended for the extraction is based on CSS3 selectors. They allow to describe non-structured documents with satisfactory universality. The result of this thesis is the extension for Mozilla Firefox web browser, which generates these selectors according to user selection in document and allows to modify them easily.

Abstrakt

Cílem bakalářské práce bylo navrhnout a implementovat rozhraní, které by pomocí vizuálních nástrojů umožňovalo vytvořit konfiguraci pro webový extraktor vyvinutý vedoucím tohoto projektu. Definice obsahu určeného k extrakci je přitom založena na CSS3 selektorech, které dovolují popsat nestrukturované dokumenty dostatečně obecně. Výsledkem práce je tak rozšíření webového prohlížeče Mozilla Firefox, který na základě uživatelského výběru v dokumentu tyto selektory generuje a následně umožňuje jejich jednoduchou editaci.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Objem sdílených informací v celosvětové síti Internet dnem ode dne roste, což jistě posiluje jeho pomyslnou roli vědomostní studnice lidstva. Jistý problém však lze vidět v nestrukturovanosti těchto dat. Jedním příkladem za všechny může být Wikipedie, otevřená encyklopedie, jejíž obsah tvoří sami uživatelé. Ta v současné době disponuje obrovským množstvím nejrůznějších informací, avšak získat z ní data v takové podobě, aby je bylo posléze možné použít například ke statistickému zpracování, je poměrně pracné a zdlouhavé. Proto je cílem této práce navrhnout a implementovat rozhraní, které by uživatelům umožnilo extrakci žádaných dat jednoduše automatizovat.

Toto rozhraní bude mít podobu rozšíření webového prohlížeče Mozilla Firefox, který s pomocí vizuálních nástrojů umožní i uživatelům bez hlubších znalostí problematiky vytvořit konfigurační soubor pro samotný webový extraktor. Hlavním nástrojem pro popisu extrahovaných dat zde přitom budou CSS3 selektory, neboť právě díky nim je možné zachytit strukturu dokumentu dostatečně obecně. Seznámit se nimi a s dalšími použitými technologiemi je možné v třetí kapitole této práce. Extraktor jako takový zde nicméně popsán není, neboť navrhované rozšíření bude využívat program vyvíjený vedoucím tohoto projektu.

Během návrhu a realizace tak bude kladen velký důraz na to, aby funkcionalita doplňku mohla být do budoucna co nejsnadněji modifikována, popřípadě dále rozvíjena, o čemž se lze podrobněji dočíst v čtvrté a páté kapitole. Výslednou podobu rozšíření webového prohlížeče je pak možné najít na přiloženém kompaktním disku.

Kapitola 2

Analýza

Po deklaraci záměrů této práce, je v průběhu analýzy nejprve shrnut současný stav na poli extrakce webů, načež je vyjádřena představa situace, která by měla po jejím dokončení následovat. Na základě toho je pak sestaven scénář, který nastiňuje, jak by práce s výsledným produktem měla ve výsledku vypadat. Toto neformální vyjádření požadavků na systém je posléze přeformulováno do podoby již konkrétních funkčních a nefunkčních požadavků.

2.1 Deklarace záměru

Hlavním záměrem práce je vytvořit rozšíření pro webový prohlížeč Mozilla Firefox, které by umožnilo uživatelům jednoduše automatizovat extrakci dat z nestrukturovaných webů, aniž by k tomu potřebovali nějaké hlubší znalosti této problematiky.

K dispozici jim proto bude vizuální nástroj, který jim dovolí vybrat elementy, z nich chtějí data extrahovat, a podle toto výběru automaticky vygeneruje CSS3 selektor, který v rámci dokumentu cestu k elementu popisuje.

Uživatel pak bude moci tyto selektory dále editovat s tím, že mu budou vždy nabídnuty možné změny a on si tak bude moci vše jednoduše „naklikat“. Během úprav uživatel samozřejmě ihned uvidí dopad jím provedených změn, neboť elementy vyhovující selektoru budou v dokumentu barevně odlišeny, přičemž při každé změně bude toto zvýraznění aktualizováno.

Tímto výběrem tedy bude uživatel konfiguraci postupně vytvářet, přičemž bude moci kdykoliv během tohoto procesu extraktor spustit, aby získal přehled, jaké výsledky jsou mu za aktuálního stavu vraceny.

Výsledek své snahy pak bude moci samozřejmě uložit do externího souboru. Z něj poté může být konfigurace kdykoliv znovu načtena a případně i dále editována.

2.2 Současný stav

V současné době, když chce uživatel z Wikipedie získat kupříkladu přehled o počtu obyvatel v jednotlivých metropolích Evropy, musí projít stránky všech těchto měst a informace z nich „ručně“ přepsat. To je jistě značně pracné a zdlouhavé. Chce-li však tato data, nemá jinou možnost.

2.3 Budoucí stav

S rozšířením, jenž bude výsledkem této práce, by se mu ale měla práce výrazně zjednodušit. Několika kliknutími totiž jen vybere seznam odkazů na stránky s informacemi o jednotlivých městech a následně pak na jedné z nich (opět pomocí několika kliknutí) vybere hodnotu, která má být ukládána. Poté už jen spustí extraktor a soupis požadovaných informací má během několika chvil hotový.

2.4 Požadavky

2.4.1 Scénář

1. Uživatel otevře stránku se seznamem odkazů na dokumenty, jenž obsahují informace o hlavních městech států Evropy.
2. Tuto stránku nastaví v konfiguraci jako výchozí.
3. Dále pak spustí nástroj pro selekci elementů a s jeho pomocí vybere jeden odkaz ze seznamu. Systém automaticky vygeneruje selektor a přidá ho do konfigurace.
4. Uživatel si ověří, zdali navrženému selektoru odpovídají všechny odkazy v seznamu, nebo případně jen ty, které ho zajímají.
5. Pokud mu selektor nevyhovuje, ručně ho pomocí editoru modifikuje. Během úprav jsou mu přitom nabízeny různé varianty změn, a tak uživatel může selektor editovat prakticky bez použití klávesnice.
6. Uživatel ale zároveň také v editoru nastaví, že stránka, na níž odkaz směřuje, má byt vždy načtena.
7. Následně pak sám přejde na jednu ze stránek, na než je v seznamu odkazováno, a prostřednictvím selekčního nástroje opět vybere některý z elementů. Tentokrát však již s hodnotou, kterou bude chtít extrahovat.
8. Pomocí editoru může selektor opět modifikovat, nicméně zcela jistě musí nastavit, aby se z vybraného elementu „přečetla“ jeho hodnota.
9. Poté může uživatel ještě přejít na některou z dalších stránek ze seznamu, aby si ověřil, že i zde odpovídá definovaný selektor elementu, z nějž mají být data načtena.
10. Pokud selektor vyhovuje, je již vše připraveno pro zahájení. Uživatel tedy spustí extraktor a sleduje, jak se data postupně načítají.
11. Do několika chvil má tak požadované hodnoty o všech hlavních městech států Evropy.

2.4.2 Funkční požadavky

- Systém umožní uživateli vizuálně vybrat libovolný element dokumentu.
- Systém automaticky vygeneruje podle uživatelského výběru selektor a přidá jej do konfigurace.
- Systém umožní uživateli editaci vygenerovaného selektoru.
- Systém nabídne nástroje umožňující editaci selektoru pomocí jednoho kliknutí myší.
- Systém zvýrazní elementy odpovídající editovanému selektoru.
- Systém zvýrazní kontext, vněmž se uživatel aktuálně nachází.
- Systém umožní uložení konfigurace do souboru.
- Systém umožní načtení konfigurace ze souboru.
- Systém umožní spuštění extrakce kdykoliv v průběhu tvorby konfigurace.
- Systém automaticky spustí extrakci při otevření konfigurace v podobě HTML dokumentu.

2.4.3 Nefunkční požadavky

- Systém musí podporovat webový prohlížeč Mozilla Firefox 3.5.
- Systém musí být funkční ve všech OS podporovaných prohlížečem Mozilla Firefox 3.5.
- Systém využívá extraktor vyvíjený vedoucím této práce.
- Systém využívá pro selekci elementů existujícího rozšíření Aardvark.
- Systém musí umožňovat snadnou rozšířitelnost editoru selektorů.

Kapitola 3

Technologické řešení

Jak je již ze zadání práce patrné, navrhovaný systém bude postaven na Mozilla Application Frameworku použitém ve webovém prohlížeči Mozilla Firefox. Tím pádem jsou ale i dány technologie, které budou při jeho tvorbě použity. Jsou jimi XUL, kaskádové styly a JavaScript. Extraktor, který bude v doplňku využíván, nás rovněž limituje, co se možného výběru použitých technologií týče. Výchozí konfiguraci totiž přijímá pouze v podobě objektu typu JSON. Navíc v ní při popisu elementů používá CSS3 selektory. Aby s nimi bylo možné pracovat i v tomto projektu, bude zde využito existujících knihoven, a to konkrétně Sizzle a jQuery. Nástroj použitý pro výběr elementů bude rovněž převzatý, neboť se jedná o rozšíření známé pod názvem Aardvark.

3.1 Mozilla Application Framework

Mozilla Application Framework [1] je kolekce multiplatformních softwarových komponent sloužící k vývoji aplikací běžících na řadě různých operačních systému. Záměrem tvůrců přitom bylo poskytnout sadu multiplatformních funkcí, která by byla vhodná pro vývoj webových prohlížečů, emailových klientů a jim podobných aplikací. Je tedy zjevné, že webový prohlížeč Mozilla Firefox je postaven právě na tomto balíku.

Jak již bylo řečeno, balík se skládá z řady součástí jakou je například i velmi výkonné vykreslovací jádro Gecko. Vzhledem k záměru této práce však naše pozornost bude směřovat pouze ke dvěma komponentám, kterými MAD disponuje, a to k XUL a XPInstall.

3.1.1 XUL

XML User Interface Language, neboli XUL [čti zůl], je jazyk vyvinutý organizací Mozilla, který slouží k snadné tvorbě multiplatformních grafických rozhraních a je proto používán v aplikacích jako např. Mozilla Firefox nebo Mozilla Thunderbird. Jak je již z jeho názvu patrné, jazyk vychází z XML. Stojí však na již existujících standardech a technologiích, jakými je například CSS, JavaScript nebo DOM, a tak pro lidi, kteří již tyto technologie ovládají, je velmi jednoduché se jej naučit [2].

Možností použití jazyka XUL je přitom hned několik [3]:

- **Rozšíření prohlížeče Firefox**
  Přidává další funkce do prohlížeče Firefox. Tohoto je dosaženo pomocí vlastnosti jazyka XUL zvané Overlays (překrytí).
- **Samostatná XULRunner aplikace**
  XULRunner je zabalená verze Mozilla platformy, která umožňuje vytvoření samostatné XUL aplikace. Ke spuštění není potřeba prohlížeč Firefox, nebot' aplikace má vlastní spouštěcí soubor.
- **XUL balíček**
  Aplikace podobná rozšíření prohlížeče Firefox. Spouští se v samostatném okně a chová se jako samostatná aplikace. Používá se, když nechceme k aplikaci přibalovat XULRunner. Ke spuštění vyžaduje prohlížeč Firefox.
- **Vzdálená XUL aplikace**
  Zdrojový soubor aplikace je možno umístit na webový server a aplikaci spouštět vzdáleně v internetovém prohlížeči Firefox podobně jako webovou stránku. Nevýhodou jsou určitá bezpečnostní omezení.

Nás nicméně bude zajímat pouze první z možností využití, a to rozšíření funkčnosti webového prohlížeče Mozilla Firefox.

3.1.1.1 Struktura

Jelikož XUL vychází z XML, musí každý dokument začínat XML hlavičkou. Aby však bylo zcela jasné, že jedná o XUL dokument, musí být patřičně nastaven jeho jmenný prostor. Jednoduchý dokument pak může vypadat třeba takto:

```
<?xml version="1.0"?>
<window id="main-window" title="Hlavní okno"
        xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
    <description>Ukázkový XUL dokument</description>
</window>
```

Kořenovým elementem XUL souboru je vždy jeden z top-level elementů (viz. 3.1.1.2) nebo ve zvláštním případě element `<overlay>` (viz. 3.1.1.5). Takový prvek se přitom může v dokumentu vyskytnou pouze jedenkrát. Jeho obsah pak tvoří výsledné uživatelské rozhraní.

3.1.1.2 Elementy

XUL definuje celou řadu elementů, které je možné rozdělit zhruba do těchto kategorií [4]:

- **top-level elementy**
  window, page, dialog, wizard, atd.

- **widgets**
  label, button, text box, list box, combo box, radio button, check box, tree, menu, toolbar, group box, tab box, colorpicker, spacer, splitter, atd.
- **box model**
  box, grid, stack, deck, atd.
- **events and scripts**
  script, command, key, broadcaster, observer, atd.
- **data source**
  template, rule, atd.
- **ostatní**
  overlay, iframe, browser, editor, atd.

Kromě toho je v XUL dokumentech možné použít elementy jiných jazyků založených rovněž na XML, jako například XHTML, SVG nebo MathML.

3.1.1.3 Aplikační logika

Aplikační logika XUL se implementuje v JavaScriptu [5]. Podobně jako u HTML disponují i XUL elementy atributy typu `oncommand`, jejichž obsahem je JavaScriptový kód, potažmo volání nějaké externí funkce. Ta je pak následně volán vždy, když se u elementu vyskytne událost, kterou atribut reprezentuje. Externí JavaScript je přitom možné do XUL vložit pomocí elementu `<script>`, tedy stejně jako v HTML.

Zde ovšem podobnost s HTML dokumenty nekončí. Stejně jako v jejich případě je totiž i XUL v paměti reprezentováno prostřednictvím objektu DOM, a tak je zjevné, že manipulovat s ním je možně naprosto stejným způsobem, jako je tomu u HTML.

3.1.1.4 Vzhled XUL aplikací

Jak již bylo řečeno, XUL je především multiplatformní technologie a ve webovém prohlížeči Mozilla Firefox je stejně jako HTML vázána na vykreslovací jádro Gecko [5]. Z toho může patrné, že vzhled XUL bude stejně jako u HTML definován pomocí kaskádových stylů. Ty však nejsou pro tvorbu GUI příliš vhodné, poněvadž nebyly pro tento účel navrhovány a navíc v tomto směru neposkytují dostatečnou funkcionalitu. Vývojáři Mozilly však tento problém vyřešili tím, že standardní implementaci CSS rozšířili tak, aby vyhovovala potřebám XUL, zároveň však ale nechali původní jádro netknuté tak, aby plně vyhovovalo standardu. Mozilla Firefox má tím pádem všechny prvky GUI definované právě prostřednictvím CSS.

Avšak narozdíl od HTML CSS v XUL neslouží k definici rozložení elementů. K tomu jsou zde tzv. „box model“ elementy (viz. 3.1.1.2). Díky nim je layout XUL oproti HTML výrazně stabilnější. Avšak tento fakt stále ještě neznamená, že pomocí CSS není přeci jen možné rozložení XUL definovat.

3.1.1.5 Overlays

XUL Overlays je technika, která kromě jiného umožňuje za běhu změnit stávající XUL dokument dle vlastních potřeb. Právě díky ní je možné vyvíjet doplňky pro Mozilla aplikace, potažmo pro Mozilla Firefox.

Kořenovým elementem souboru XUL Overlay je element `<overlay>` a jeho obsahem jsou fragmenty XUL kódu. Těmi bude na základě shody identifikátorů „překryta“ část kódu původního dokumentu. Pomocí toho je možné v původního UI prvky mazat, měnit, anebo k nim přidávat nové [6]. Tento princip si názorně objasníme na následujícím příkladu [7].

Fragment kódu, jenž představuje menu se čtyřmi položkami:

```
...
<menu id="file_menu">
    <menuitem name="New"/>
    <menuitem name="Open"/>
    <menuitem name="Save"/>
    <menuitem name="Close"/>
</menu>
...
```

My toto menu pomocí XUL Overlay rozšíříme o jednu položku. Nezbytností přitom je, abychom u elementu menu použili stejný identifikátor.

```
<?xml version="1.0"?>
<overlay id="singleItemEx"
         xmlns:html="http://www.w3.org/1999/xhtml"
         xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
    <menu id="file_menu">
        <menuitem name="Super Stream Player"/>
    </menu>
</overlay>
```

Ve výsledku, tedy po „překrytí“, bude menu fakticky vypadat takto:

```
...
<menu id="file_menu">
    <menuitem name="New"/>
    <menuitem name="Open"/>
    <menuitem name="Save"/>
    <menuitem name="Close"/>
    <menuitem name="Super Stream Player"/>
</menu>
...
```

Při vývoji doplňku je ale třeba mít na paměti, že jmenný prostor ID je společný pro všechna rozšíření. Abychom se tedy vyhnuli případným kolizí s jinými doplňky, měli bychom u všech našich identifikátorů používat unikátní prefix [7].

3.1.1.6 Adresářová struktura doplňku

Adresářová struktura rozšíření webového prohlížeč Mozilla Firefox vypadá typicky takto:

```
+- Doplnek
   -- chrome.manifest
   -- install.rdf
   +- content
      -- overlay.xul
      -- overlay.js
   +- locale
      +- cs-CZ
      +- en-US
   +- skin
      - overlay.css
```

V hlavní složce se povinně nacházejí soubory, a to `chrome.manifest` a `install.rdf`. Ty obsahují informace o daném rozšíření a o jeho dalším vnitřním uspořádání.

V složce `content` jsou pak umístěný XUL soubory s definicí rozhraní a skripty zajišťující jeho aplikační logiku. Složka `skin` dále obsahuje soubory s kaskádovými styly, jejichž prostřednictvím je definován vzhled doplňku. V poslední z uvedených složek, ve složce `locale`, je posléze možné najít lokalizační soubory.

install.rdf

První povinný soubor rozšíření představuje jakousi jeho hlavičku. Z něj má Mozilla Firefox možnost si přečíst informace o jeho názvu, autorovi a celou řadu dalších údajů. Pro ukázku je zde uveden příklad tohoto souboru:

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

  <Description about="urn:mozilla:install-manifest">
    <em:id>doplnek@jirimasek.cz</em:id>
    <em:version>1.0</em:version>
    <em:type>2</em:type>

    <em:targetApplication>
      <Description>
        <em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</em:id>
        <em:minVersion>1.5</em:minVersion>
        <em:maxVersion>3.5.*</em:maxVersion>
      </Description>
    </em:targetApplication>
```

```
    <em:name>Doplněk</em:name>
    <em:description>Doplněk pro Mozilla Firefox.</em:description>
    <em:creator>Jiří Mašek</em:creator>
    <em:homepageURL>http://www.example.net/</em:homepageURL>
  </Description>
</RDF>
```

Na první pohled je patrné, že se jedná o XLM soubor, k čemuž asi není třeba nic dodávat. Pro doplnění zde ale bude osvětlen význam některých jeho elementů.

- **id**
  Jednoznačný identifikátor doplňku, který by měl být unikátní z rámci všech ostatních rozšíření. Proto se zpravidla vytváří tím způsobem, že se spojí název rozšíření se jménem autora do tvaru podobného emailové adrese.
- **name**
  Název rozšíření, který bude zobrazen ve Správci doplňků.
- **version**
  Způsob zápisu verze doplňku není žádným způsobem limitován, nicméně je vhodné používat stejný formát, jaký používají aplikace Mozilla, tedy např. `3.5.9`.
- **description**
  Popis rozšíření, který bude zobrazen ve Správci doplňků.
- **creator**
  Jméno tvůrce doplňku.
- **targetApplication**
  Prostřednictvím tohoto elementu jsou definovány aplikace, pro něž je rozšíření určeno. Společně s tím je zde i uveden rozsahu verzí, v nichž bude doplněk podporován.

chrome.manifest

Jak bylo již v kapitole 3.1.1.5 zmíněno, „překrývání“ umožňuje rozšiřovat nebo jinak modifikovat původní rozhraní prohlížeče. K tomu je ale ještě potřeba říci, který soubor budeme překrývat a jakým. K tomu kromě jiného slouží právě tento soubor. Pro názornost je zde opět uveden příklad:

```
content doplnek  content/
skin    doplnek  classic/1.0    skin/
overlay chrome://browser/content/browser.xul          chrome://doplnek/content/overlay.xul
style   chrome://global/content/customizeToolbar.xul  chrome://doplnek/skin/overlay.css
```

První řádek určuje složku s rozhraním a obslužnými skripty. Dále následuje definice skinu. Řádek začínající klíčovým slovem `overlay` poté jasně definuje, který XUL soubor bude rozšiřovat tím naším. A na posledním řádku najdeme definic souboru s kaskádovými styly.

3.1.2 XPInstall

Cross-Platform Install, neboli XPInstall, je technologie, kterou webový prohlížeč Mozilla Firefox spolu s jiný aplikacemi založenými na XUL používá pro instalaci doplňků [8].

Instalační modul XPI [čti zippy] (také nazývaný „Bundles“) je přitom obyčejný ZIP soubor, který obsahuje instalační skript a samozřejmě soubory určené k instalaci [9]. Aplikace Mozilla při interakci s těmito moduly automaticky nabídnou jejich instalaci.

3.2 CSS3 selektory

Selektor představuje podobně jako regulární výraz strukturu, která může být použita buďto jako podmínka (např. v CSS jako pravidlo), která určuje elementy v stromové struktuře HTML, potažmo XML dokumentu, nebo jako obecný popis HTML či XML fragmentu, který této struktuře odpovídá [10]. Selektory přitom můžou odkazovat na elementy pouze na základě jejich typu, nebo lze s jejich pomocí vytvořit velmi specifický popis vycházející ze vzájemných vztahů prvků ve struktuře dokumentu.

Extraktor, na němž je tato práce postavena, využívá, ke specifikaci elementů právě tyto selektory, což ostatně bylo již několikrát zmíněno. Je však ale namístě zamyslet nad tím, zdali jsou zrovna selektory tím nejlepším možným řešením.

Mohly zde být například využity XPath, nebo XQuery. Aadrvark (viz. 3.4), který bude v rozšíření použit k výběru elementů, dokonce sám XPath výrazy přímo generuje a MAF navíc i implicitně podporuje jejich interpretaci [11]. Zápis cest pomocí těchto technologií je však poměrně zdlouhavý a velmi neintuitivní. Navíc je pro jejich editaci třeba hlubších znalostí této technologie. CSS3 selektory naproti tomu umožňují popsat elementy dostatečně obecně při zachování minimálně možné délky zápisu. Zápis je přitom velmi přehledný a na první pohled je patrné, jaký výsledek od něj může uživatel očekávat.

3.2.1 Syntaxe [10]

Selektor je řetězec složený z jedné či více **sekvencí jednoduchých selektorů** navzájem od sebe oddělených **kombinátory**. K poslední sekvenci přitom může být připojen jeden **pseudo-element**.

Sekvence jednoduchých selektorů je řetězec složený z **jednoduchých selektorů**, které nejsou navzájem odděleny **kombinátory**. Začíná vždy **selektorem typu** nebo **univerzálním selektorem**. Žádný další selektor typu či univerzální selektor se však již v sekvenci vyskytovat nesmí.

Jednoduchý selektor je buďto **selektor typu**, **univerzální selektor**, **selektor atributu**, **selektor třídy**, **selektor ID** nebo **pseudo-třída**.

Kombinátory jsou **bílé znaky**, **znaménko „větší než“**, **znaménko „plus“** a **„tilda“**. Bílé znaky se přitom mohou vyskytovat i mezi jednoduchým selektorem a jiným kombinátorem.

3.2.2 Jednoduché selektory

3.2.2.1 Selektor typu

Selektorem typu může být název libovolného elementu dokumentu. Selektor pak reprezentuje všechny instance daného elementu v dokumentu. Například selektor `h1` tak odpovídá všem nadpisům první úrovně.

3.2.2.2 Univerzální selektor

Univerzální selektor se značí symbolem `*` a zahrnuje všechny elementy v dokumentu.

V případě, že univerzální selektor není jedinou součástí sekvence jednoduchých selektorů, je možné jej vynechat, poněvadž jeho přítomnost se implicitně předpokládá.

Příklady použití:

- `*[hreflang|=en]` je ekvivalentní s `[hreflang|=en]`
- `*.warning` je ekvivalentní s `.warning`
- `*#myid` je ekvivalentní s `#myid`

3.2.2.3 Selektory atributů

Elementy mohou být též specifikovány na základě vlastností svých atributů.

Selektory atributů s ověřením existence či hodnoty
Již specifikace CSS2 obsahovala čtyři selektory atributů [12]:

- `[att]`
  Tento selektor popisuje pouze ty elementy, u nichž je uveden atribut `att`, přičemž na jeho hodnotě již nezáleží.
- `[att=val]`
  Naproti tomu elementy, které vyhovují tomuto selektoru, nejenže mají atribut `att` uvedený, ale jeho hodnota přímo odpovídá hodnotě `val`.
- `[att~=val]`
  Pokud hodnotou atributu `att` je seznam slov oddělených bílými znaky a je-li jedním z těch slov právě `val`, pak daný element odpovídá tomuto selektoru.
- `[att|=val]`
  V tomto případě selektor reprezentuje elementy, u nichž hodnota atributu `att` začíná řetězcem `att`, přičemž bezprostředně za ním následuje pomlčka. Tento selektor je přitom primárně navržen pro určování jazykových variant obsahu (podle RFC 1766), poněvadž ty jsou právě v tomto tvaru uváděny. Totiž zatímco `[lang=en]` odpovídá pouze elementům s atributem `lang="en"`, selektor `[lang`=en]|| vyhovuje všem variantám angličtiny, tedy například `lang="en-US"` nebo `lang="en-GB"`.

Příklady použití:

Tento selektor odpovídá nadpisu první úrovně, jenž má definovaný atribut `title` bez ohledu na jeho hodnotu.

```
h1[title]
```

V následujícím příkladu selektor reprezentuje element `span`, jehož atribut `class` přesně odpovídá hodnotě „example“.

```
span[class="example"]
```

Selektory atributů je přitom možné libovolně kombinovat. Tento selektor tak například představuje element `span`, jehož atribut `id` odpovídá hodnotě „sample“ a atribut `class` je roven „example“.

```
span[id="sample"][class="example"]
```

Následující CSS pravidlo ukazuje rozdíl mezi `=` a `~=`. Prvnímu selektoru vyhovuje element `a`, u nějž je hodnota atributu `rel` například „copyright copyleft copyeditor“. Naproti tomu druhý selektor odpovídá pouze těm elementům, u nichž je atribut `href` přesně roven hodnotě „http://www.w3.org/“.

```
a[rel~="copyright"] { ... }
a[href="http://www.w3.org/"] { ... }
```

Poslední selektor pak odpovídá všem elementům, u nichž hodnota atributu `hreflang` začíná řetězcem „en“, tedy například „en“, „en-US“ či „en-GB“.

```
a[hreflang|="en"]
```

Selektory atributů s vyhledáváním v řetězci
Specifikace CSS3 rozšířila selektory atributů tím způsobem, že je možné je koncipovat na základě podřetězců obsažených v jejich hodnotách.

- `[att^=val]`
  Tento selektor popisuje ty elementy, u nichž hodnota atributu `att` začíná řetězcem `val`. Je-li `val` prázdný řetězec, pak selektor neodpovídá žádnému elementu.
- `[att$=val]`
  Naproti tomu u elementů vyhovujících následujícímu selektoru hodnota atributu `att` řetězcem `val` končí. Je-li `val` prázdný řetězec, pak selektor opět neodpovídá žádnému elementu.

- `[att*=val]`
  Poslední ze selektorů atributů odpovídá těm elementům, jejichž hodnota atributu `att` alespoň jednou obsahuje podřetězcem `val`. Je-li `val` prázdný řetězec, pak i zde selektor neodpovídá žádnému elementu.

Příklad:

Selektor, který následuje, reprezentuje všechny objekty typu „image“.

```
object[type^="image/"]
```

Naproti tomu tento selektor odpovídá všem odkazům, u nichž hodnota atributu `href` končí řetězcem „.html“.

```
a[href$=".html"]
```

Poslední selektor pak představuje takový odstavec, u nějž hodnota atributu `title` obsahuje podřetězec „hello“.

```
p[title*="hello"]
```

3.2.2.4 Selektor třídy

Abychom si usnadnili práci, může v HTML při práci s třídami použít namísto selektoru atributu `[att~=val]` selektor třídy `.val`. V případě HTML dokumentů mají totiž selektory `div.value` a `div[class~=value]` stejný význam. Název třídy nicméně musí v případě tohoto selektoru bezprostředně následovat za tečkou.

Příklady použití v CSS:

Pomocí tohoto pravidla změníme barvu písma u všech elementů s `class~="pastoral"`:

```
*.pastoral { color: green }
```

Je nicméně možný i tento zápis:

```
.pastoral { color: green }
```

Následují pravidlo naproti tomu změní barvu písma pouze u nadpisů první úrovně s `class~="pastoral`:

```
h1.pastoral { color: green }
```

V důsledku toho si první nadpis zachová svoji původní barvu a druhý se změní na zelený:

```
<h1>Nadpis</h1>
<h1 class="pastoral">Zelený nadpis</h1>
```

Poslední pravidlo pak odpovídá odstavci, u nějž je hodnotou atributu `class` seznam bílými znaky oddělených slov, mezi nimiž se vyskytují řetězce „pastoral“ a „marine“:

```
p.pastoral.marine { color: green }
```

Toto pravidlo tak vyhovuje například elementu s `class="pastoral blue aqua marine"`, nikoliv však již `class="pastoral blue"`.

3.2.2.5 Selektor ID

Podobně jako je tomu u selektorů tříd, můžeme i v případě identifikátorů elementů použít namísto selektoru atributu `[id=val]` selektor ID `#val`. Název třídy zde přitom musí bezprostředně následovat za křížkem.

Selektory `div#value` a `div[class=value]` však nemají zcela stejný význam. Selektory ID jsou totiž z hlediska kaskádování mnohem specifičtější. Navíc vždy odpovídají pouze jednomu elementu, a to prvnímu, u něhož je daný identifikátor použit. Právě proto se pro něj též užívá označení „hard ID“. V některých zejména starších HTML dokumentech jsou však identifikátory používány opakovaně. Proto je v takových případech mnohem vhodnější použít selektor atributu `div[class=value]`, tedy tzv. „soft ID“.

Příklady použití:

Tento selektor odpovídá takovému nadpisu první úrovně, u nějž je identifikátor roven „kapitola1“:

```
h1#kapitola1
```

Naproti tomu tento selektor představuje jakýkoliv element s hodnotou identifikátoru rovnou „kapitola1“:

```
kapitola1
```

Poslední uvedený selektor opět reprezentuje libovolný element, avšak tentokrát s hodnotou identifikátoru rovnou „z98y“.

```
*#z98y
```

3.2.2.6 Pseudo-třídy

Hlavní myšlenkou pseudo-tříd je umožnit výběr elementů takovým způsobem, že by vycházel z informacích, které se nacházejí mimo strukturu dokumentu, anebo nemůže být vyjádřeny prostřednictví jiných jednoduchých selektorů.

Tento selektor sestává vždy z dvojtečky, která je pak bezprostředně následována názvem pseudo-třídy a případně ještě závorkami s nějakou hodnotou dále specifikující výběr.

Užití pseudo-tříd je možné ve všech sekvencích jednoduchých selektorů v selektoru, přičemž v rámci jedné sekvence může být umístěna kdekoliv za selektorem typu či za univerzálním selektorem. Pseudo-tříd přitom může být použito v sekvenci hned několik.

Zároveň mohou být dynamické, a to v tom smyslu, že elementy pseudo-třídou popisované se mohou objevovat a znovu mizet v závislosti na interakci uživatele s dokumentem.

Pseudo-třídy je přitom možné rozdělit hned do několika kategorií, avšak pro potřeby této práce stačí znalost pouze některých z nich.

Strukturální pseudotřídy

Pseudo-třídy z této kategorie umožňují popis elementů na základě struktury dokumentu. V jejich následujícím výčtu je vždy stručně popsána skupina elementů, kterou daná pseudo-třída reprezentuje:

- `:root`
  Kořenový element dokumentu. V případě HTML 4 je to vždy `html`.
- `:nth-child(n)`
  Element, který je n-tým potomkem svého rodiče.
- `:nth-last-child(n)`
  Element, který je n-tým potomkem svého rodiče při počítání odzadu.
- `:nth-of-type(n)`
  Element, který je n-tým potomkem svého typu.
- `:nth-last-of-type(n)`
  Element, který je n-tým potomkem svého typu při počítání odzadu.
- `:first-child`
  Element, který je prvním potomkem svého rodiče.
- `:last-child`
  Element, který je posledním potomkem svého rodiče.
- `:first-of-type`
  Element, který je prvním potomkem svého typu.
- `:last-of-type`
  Element, který je posledním potomkem svého typu.

- `:only-child`
  Element, který je jediným potomkem svého rodiče.
- `:only-of-type`
  Element, který je jediným potomkem svého typu.
- `:empty`
  Element, který nemá žádné potomky.

Co je však činní tyto pseudo-třídy opravdu mocnými, je možnost nahradit argument `n` namísto čísla polynomem `an + b`. Je zde ale také možné použít klíčová slova `odd` a `even`, přičemž `odd` odpovídá polynomu `2n + 1` a `even` je zas shodný s výrazem `2n`.

Příklady použití v CSS:

```
/* první řádek tabulky bude mít červené pozadí */
tr:first-child { background-color: red; }

/* každý sudý řádek tabulky bude mít červené pozadí */
tr:nth-child(even) { background-color: red; }

/* pátý řádek tabulky bude mít červené pozadí */
tr:nth-child(5) { background-color: red; }

/* každý pátý řádek tabulky bude mít červené pozadí */
tr:nth-child(5n) { background-color: red; }

/* každá prázdná buňka tabulky bude mít červené pozadí */
td:empty { background-color: red; }
```

Pseudo-třída Content

Pseudo-třída `:contains("foo")` představuje element, v jehož textovém obsahu se vyskytuje zadaný podřetězec.

Textový obsah elementu přitom zahrnuje veškerá `PCDATA` obsažená v něm i v jeho následnících.

Příklad použití:

```
p:contains("Markup")
```

Tento odstavec odpovídá výše uvedenému selektoru:

```
<p>
    <strong>H</strong>yper<strong>t</strong>ext
    <strong>M</strong><em>arkup</em>
    <strong>L</strong>anguage
</p>
```

Pseudo-třída Not

Pseudo-třída `:not(X)` představuje ty elementy, které neodpovídají selektoru uvedenému v jejím argumentu. Negace nicméně nemůže být vnořená. Zápis `:not(:not(...))` tedy není validní.

Příklady použití:

Následující selektor reprezentuje všechna tlačítka v dokumentu, která nejsou deaktivována:

```
button:not([DISABLED])
```

Zatímco tento selektor odpovídá všem elementům v dokumentu s výjimkou odstavců.

```
*:not(p)
```

3.2.3 Seskupování selektorů

Seznam selektorů oddělených čárkami představuje souhrn všech elementů odpovídajících každému jednotlivému z nich. Bílé znaky se přitom mohou vždy vyskytovat před a/nebo za čárkou.

Příklady použití v CSS:

Zde máme tři pravidla s naprosto shodnou deklarací:

```
h1 { font-family: sans-serif; }
h2 { font-family: sans-serif; }
h3 { font-family: sans-serif; }
```

Jejich zápis můžeme nicméně zjednodušit:

```
h1, h2, h3 { font-family: sans-serif }
```

3.2.4 Kombinování selektorů

Selektory je možné též vytvářet vzájemným kombinováním jednodušších selektorů a vytvářet tak specifičtější popis v závislosti na vzájemných vztazích prvků ve struktuře dokumentu [13].

3.2.4.1 Následník

Může nastat situace, kdy je potřeba specifikovat pouze ty prvky, které jsou následníky některého jiného elementu v rámci struktury dokumentu (např. element `em`, jenž je obsažen v nadpisu první úrovně). Tento vztah je možné popsat tak, že selektor nadřazeného prvku, v tomto případě `h1`, spojíme pomocí bílého znaku se selektorem požadovaného elementu, tedy `em`. Bílým znakem je přitom prakticky vždy mezera. Výsledný selektor `h1 em` pak bude reprezentovat všechny elementy `em`, které jsou libovolně vnořené do nadpisu první úrovně.

Příklad použití:

```
h1 em
```

Výše uvedený selektor popisuje element `em` v následujícím fragmentu kódu. Není přitom nezbytné, aby element `em` byl přímým potomkem nadpisu.

```
<h1>Tento <span class="myclass">nadpis je <em>velmi</em> důležitý</span></h1>
```

3.2.4.2 Potomek

Tento vztah je velmi podobný tomu předcházejícímu, avšak s tím rozdílem, že zde je popisovaný element přímým potomkem nadřazeného prvku. Kombinátorem je zde tedy namísto mezery symbol „větší než“.

Příklady použití:

Tomuto selektoru odpovídají všechny odstavce, které jsou přímými potomky elementu `body`:

```
body > p
```

Zatím co tento selektor kombinuje vztah následníka a potomka:

```
div ol>li p
```

Odpovídá tak odstavci, který je libovolně vnořen do položky seznamu. Tato položka však musí být přímým potomkem číselného seznamu. Ten však již může být opět libovolně vnořen do elementu `div`. Za povšimnutí též stojí fakt, že bílé znaky, které je možné umístit před i za symbol „větší než“, byly v tomto případě vynechány.

3.2.4.3 Sourozenec

Mezi sousedícími elementy je možné popsat dva druhy vzájemného postavení. V prvním případě se tyto dva prvky nalézají bezprostředně vedle sebe. V druhém pak není na potenciální existenci elementů nacházejících se mezi nimi brán zřetel. Každopádně ani v jednom případě není brán zřetel na uzly, které by se sice nacházely mezi sousedícími prvky, ale zároveň by nebyly elementy (např. text mezi prvky).

Bezprostředně následující sourzozenec

U tohoto vztahu odděluje dvě sekvence jednoduchých selektorů znaménko „plus“. Elementy reprezentované oběma z nich přitom mají v struktuře dokumentu společného rodiče a prvek popsaný první částí selektoru vždy bezprostředně předchází elementu vyjádřeného druhou sekvencí.

Příklady použití:

První selektor představuje odstavec, který bezprostředně následuje za nadpisem první úrovně:

```
h1 + p
```

Naproti tomu tento selektor,ačkoliv je konceptuálně shodný s předešlým, odpovídá pouze takovému odstavci, který bezprostředně následuje za nadpisem první úrovně, přičemž hodnota jeho atributu `class` je rovna „opener“.

```
h1.opener + p
```

Obecně následující sourozenec

Oproti předešlému vztahu je zde kombinátorem znak „tilda“. Elementy reprezentované oběma sekvencemi mají ve struktuře dokumentu opět společného rodiče, nicméně prvek popsaný první částí selektoru nemusí nutně bezprostředně předcházet elementu vyjádřenému druhou sekvencí.

Příklad použití:

```
h1 ~ pre
```

Výše uvedený selektor představuje v následujícím fragmentu kódu předformátovaný text následující za nadpisem první úrovně:

```
<h1>Definice funkce a</h1>
<p>Funkci a(x) je třeba použít na všechny hodnoty v tabulce.</p>
<pre>fce a(x) = 12x/13.5</pre>
```

3.2.5 Shrnutí

Souhrnný přehled všech výše zmíněných selektorů je možné posléze najít v tabulce B.1.

3.2.6 Sizzle

Sizzle je knihovna, jenž je napsaná čistě ve skriptovacím jazyce JavaScript. Díky tomu je nezávislá nejen na platformě, ale i na prohlížeči, v němž je používána. Představuje tzv. „selector engine“, což by se obecně dalo popsat jako program, který umí vyhodnocovat CSS selektory, na jejichž základě pak vrací elementy, které jim ve zpracovávaném dokumentu odpovídají.

Použití knihovny je přitom snadné. Pro získání elementů vyhovujících danému selektoru totiž stačí zavolat metodu `Sizzle(String selector, DOMElement|DOMDocument context)`. Jejími parametry jsou přitom vyhodnocovaný selektor a objekt zpracovávaného dokumentu, popř. reference na element, v jehož kontextu bude selektor vyhodnocován.

Jako nedostatek této knihovny by přitom mohl být vnímán fakt, že plně neimplementuje specifikaci CSS3. Nicméně v naprosté většině vyhodnocovaných selektorů se tato skutečnost nikterak neprojeví.

Naopak její velkou výhodou je, že umožňuje zjednodušení zápisu některých pseudo-tříd. Jejich výčet následuje v tabulce 3.1. Sizzle navíc specifikaci rozšiřuje o vlastní pseudo-třídy `:has(s)`, `:lt(n)` a `:gt(n)`. Pseudo-třída `:has(s)` přitom představuje ty elementy, které obsahují prvky odpovídající selektoru `s`. Naproti tomu `:lt(n)` a `:gt(n)` reprezentují prvky, jenž předcházejí či následují n-tého potomka svého rodiče.

Tabulka 3.1: Rozšíření specifikace CSS3 v knihovně Sizzle

CSS3	Sizzle
`:not([att=value])`	`[att!=value]`
`:first-child`	`:first`
`:last-child`	`:last`
`:nth-child(even)`	`:even`
`:nth-child(odd)`	`:odd`
`:nth-child(n)`	`:eq(n)`
`input[type=text]`	`:text`
`input[type=checkbox]`	`:checkbox`
`input[type=file]`	`:file`
`input[type=password]`	`:password`
`input[type=submit]`	`:submit`
`input[type=image]`	`:image`
`input[type=reset]`	`:reset`

3.2.7 jQuery

Knihovna jQuery, rovněž napsaná v JavaScriptu, přitom s knihovou Sizzle úzce souvisí. Právě ji totiž jQuery využívá jako svůj „selector engine“. Stručně by se tato knihovna dala charakterizovat tím způsobem, že podobně jako CSS odděluje „zobrazovací“ charakteristiku od struktury HMTL, tak jQuery od ní odděluje „chování“ [14].

Nicméně v tomto projektu se s jQuery setkáme jen velmi okrajově. Bude zde totiž pouze využito jeho schopnosti manipulovat s CSS, a to jen při té příležitosti, když bude potřeba zvýraznit elementy, jenž vyhovují aktuálně editovanému selektoru.

3.3 JavaScript Object Notation

JavaScript Object Notation, neboli JSON, je způsob zápisu dat nezávislý na počítačové platformě, určený pro přenos dat, která mohou být organizována v polích nebo agregována objektech. Vstupem je libovolná datová struktura (číslo, řetězec, boolean, objekt nebo z nich složené pole), výstupem je vždy řetězec. Složitost hierarchie vstupní proměnné není teoreticky nijak neomezena [15].

Tento způsob zápisu dat je zde přitom využíván při konfiguraci extraktoru. To je podrobněji rozebráno v kapitole 4.1.

3.4 Aardvark

Aardvark je rozšíření webového prohlížeče Mozilla Firefox, které je zároveň velmi efektivním a uživatelsky příjemným nástrojem pro výběr elementů v rámci HTML dokumentů (viz. obrázek 3.1). Na tomto nástroji je založena řada existujících doplňků jako například Adblock

Plus: Element Hiding Helper [16], což je nástroj slouží k blokování reklamních bannerů, nebo AutoPager, který prozměnu slouží k automatickému načítání stránek.

Obrázek 3.1: Editace selektoru

Práce s tímto nástrojem je velmi jednoduchá. Pro jeho spuštění totiž stačí zavolat metodu `aardvarkSelector.start(browser)`, jejímž parametrem je reference na objekt reprezentující prohlížeč. Pro upřesnění je ale třeba říci, že prohlížečem zde není myšleno celé okno webového prohlížeče ani proces, v němž je spuštěn, nýbrž konkrétní XUL element, v němž je HTML dokument vykreslován.

Odchytávání vybraných elementů pak není o nic složitější, neboť Aardvark sám při každém potvrzení výběru elementu, a je jedno jestli stisknutím tlačítka `s` nebo kliknutím myši, volá metodu `aardvarkSelector.select(elem)`, jejímž argumentem je právě vybraný element. Nám tak stačí pouze nahradit původní kód kódem vlastním. O své ukončení se Aardvark stará rovněž sám. Dojde k němu buďto po stisknutí tlačítka `q` nebo `Esc`.

Aardvark kromě toho disponuje ještě několika velice užitečnými metodami, které při implementaci rozšíření využijeme. Jsou jimi metody `aardvarkUtils.currentDocument()`, `aardvarkUtils.currentBrowser()` a `aardvarkUtils.currentWindow()`, přičemž každá z nich vždy vrací referenci na aktuálně užívaný dokument, prohlížeč, či okno.

Kapitola 4

Návrh řešení

V návrhu řešení je nejprve nutné rozebrat, jak vlastně probíhá konfigurace extraktoru, neboť právě od toho se celé řešení bude odvíjet. Extraktor se konfiguruje prostřednictvím objektu typu JSON, přičemž jeho ukázku je možné najít v dodatku C.

4.1 Konfigurace extraktoru

Jak již bylo řečeno, konfigurace extraktoru se provádí prostřednictvím objektu typu JSON. Ten má následující strukturu:

```
{
    delay: int,
    url: String,
    def: Object,
    select: Object
}
```

Význam prvních dvou hodnot je zcela prostý. Atribut `url` udává adresu, z níž bude extraktor vycházet, a hodnota `delay` naproti tomu představuje délku prodlení mezi jednotlivými operacemi.

4.1.1 Objekt Select

Složitější je však situace v případě atributu `select`, neboť právě podle něj je řízen celý proces extrakce. Z praktického hlediska se přitom jedná od podobjekt, jehož obsahem je libovolné množství těchto struktur, kdy každá fakticky představuje jeden příkaz:

```
[name]: { area: String,
          url: Object,
          read: String,
          use: String,
          attach: String,
          select: Object }
```

Hodnotou `[name]` je zde přitom textový řetězec, který příkaz pojmenovává a v důsledku tak zastává roli identifikátoru. Zejména proto tak musí být v rámci objektu vždy unikátní.

Vlastnosti `read`, `use` a `attach` následně určují, co přesně se bude dít s elementy, které popisuje selektor obsažený v atributu `area`.

V jejich kontextu budou rovněž prováděny podřízené příkazy obsažené v objektu `select`. To však platí pouze za předpokladu, že aktuální příkaz nemá definovaný atribut `url`. Ten totiž značí přechod na jinou stránku, z čehož je tudíž jasné, že vnořené příkazy tak pracují se zcela odlišným dokumentem.

Každopádně v důsledku toho se nám z objektu `select` fakticky vytvoří stromová struktura příkazů, kde `[name]` je rodič a atribut `select` obsahuje všechny jeho potomky. Pro větší názornost si toto ukážeme na praktickém příkladu. Zde máme potenciálně obsah atributu `select`:

```
mesto: {
    area: "",
    select: {
        nazev: {
            read: "text"
        },
        adresa: {
            read: "link"
        }
    }
}
```

Abychom v něm onu stromovou strukturu viděli jasněji, transformujeme ho do následující podoby:

```
name:"mesto", area:"td:eq(1) a"
|- name:"nazev", read:"text"
|- name:"adresa", read:"link"
```

V tomto stavu je již možné objekt `select` zobrazit prostřednictvím XUL elementu `tree`, který k zobrazování stromových struktur slouží.

4.1.2 Objekt Def

Objekt `def` funguje na naprosto stejném principu jako objekt `select`, akorát s tím rozdílem, že neřídí proces extrakce, nýbrž jen definuje opakující se sekvence příkazů. Pro názornost bude opět nejjednodušší si toto ukázat na praktickém příkladu:

```
def: {
    cesky_nazev: {
        url: {
            area: ".interwiki-cs a",
```

```
            read: "link"
        },
        area: "h1",
        read: "text"
    }
},
select: {
    cesky_nazev: {
      use: ["cesky_nazev"]
    }
}
```

V objektu `def` máme definovaný příkaz `cesky_nazev`, na který se pak z části `select` odkazujeme prostřednictvím atributu `use`. Při zpracování tak bude řídící sekvence příkazů fakticky vypadat takto:

```
select: {
    cesky_nazev: {
        url: {
            area: ".interwiki-cs a",
            read: "link"
        },
        area: "h1",
        read: "text"
    }
}
```

4.1.3 Objekt URL

O objektu `url`, který je atributem ve struktuře příkazu, jsme se doposud nezmínili. V jeho případě je totiž situace poněkud komplikovanější, neboť atribut sám je totiž příkazem. Pro lepší pochopení následuje příklad:

```
detail: {
    url: {
        read: "link"
    },
    select: {
        plny_nazev: {
            area: ".infobox.geography .country-name",
            read: "text"
        }
    },
    attach: "inline"
}
```

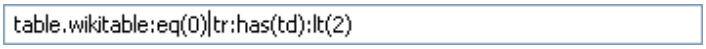
Z tohoto faktu přitom plyne jedna důležitá skutečnost, a to že každý atribut `url` může obsahovat stejnou stromovou strukturu příkazů jako objekty `def` a `select`.

4.2 Editace konfigurace

Z předchozího rozboru konfigurace extraktoru tedy vyplynulo, že se zde nacházejí tři různé stromové struktury příkazů. Díky transformaci, kterou jsou s nimi schopni provést (viz. 4.1.1), je tak můžeme snadno zobrazit prostřednictvím XUL elementu `tree`, a umožnit tak jejich editaci.

Jeden uzel stromu tak bude vždy odpovídat jednomu příkazu, přičemž pro každý z nich pak bude možné samostatně otevřít editor, který dovolí úpravu jeho vlastností, tj. `[name]`, `area`, `url`, `read`, `use` a `attach`.

Hlavní důraz zde nicméně bude kladen na atribut `area`. Editor tak uživateli nabídne řadu nástrojů, které mu jeho editaci co nejvíce zjednoduší. V centru pozornosti těchto nástrojů přitom vždy bude aktuálně editovaná sekvence jednoduchých selektorů, která bude rozpoznána podle pozice kurzoru v textboxu se selektorem. Například v textboxu zobrazeném na obrázku 4.1 se za editovanou bude pokládat sekvence `table.wikitable:eq(0)`.



Obrázek 4.1: Editace selektoru

Nástroje pak s ohledem na kontext příkazu a strukturu dokumentu nabídnou další jednoduché selektory, které bude moci jedním kliknutím myši k sekvenci přidat, odebrat, či v případě selektoru typu změnit. Rovněž bude umožněno pomocí jednoho tlačítka celou editovanou sekvenci ze selektoru odstranit, čímž se výrazně usnadní jejich zobecňování. Během implementace přitom bude kladen velký důraz na to, aby skupina těchto nástrojů mohla být kdykoliv a co nejjednodušeji rozšířena.

U atributů `read`, `attach` a `use` je pak pro změnu omezena množina možných hodnot, jenž jim mohou být přiřazeny. V případě prvních dvou tyto množiny vychází z možností extraktoru, a proto je vždy nutné z něj tyto informace při spuštění rozšíření získat. Hodnoty atributu `use` pak zase vycházejí z příkazů předdefinovaných v objektu `def`.

Při dokončení editace celé konfigurace, tj. při jejím ukládání, nebo při přímém spouštění extraktoru, se struktura z elementu `tree` transformuje zpět do podoby objektu JSON tak, aby mohl být rovnou předán extraktoru ke zpracování.

4.3 Generování sektorů

Jak již bylo v analýze řečeno, cílem práce je konfiguraci co nejvíce zjednodušit, a právě proto se dalším klíčovým nástrojem celého rozšíření stane Aardvark. Ten totiž umožní uživateli vizuálně vybrat element, který chce do editované struktury přidat, přičemž pro něj bude automaticky vygenerován selektor tak, aby již odpovídal kontextu, do kterého je vkládán.

Kapitola 5

Implementace

Rozšíření pro webový prohlížeč Mozilla Firefox bylo po dohodě s vedoucím práce vyvíjeno pod licencí MPL 1.1, a to z toho důvodu, aby mohlo být do budoucna modifikováno a vylepšováno kýmkoliv, kdo by projevil o problematiku extrakce webu prostřednictvím doplňku prohlížeče zájem.

Během samotné implementace bylo nicméně postupováno podle výše uvedeného návrhu řešení a až na drobné komplikace se jej podařilo celý úspěšně realizovat.

5.1 Grafické uživatelské rozhraní

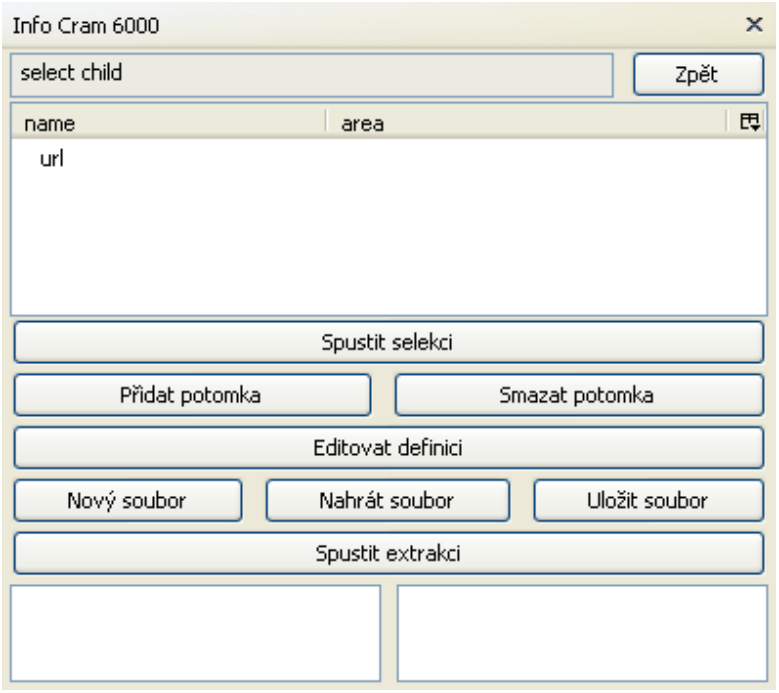
Grafické rozhraní implementovaného rozšíření se skládá ze dvou základních částí. První z nich je postranní lišta (viz. obrázek 5.1), která zobrazuje aktuálně editovanou příkazovou strukturu. Druhou je pak popup panel (viz. obrázek 5.2), který slouží jako editor jednotlivých příkazů.

5.1.1 Postranní lišta

Hlavním a nejdůležitějším prvkem postranní lišty (viz. obrázek 5.1) je XUL element `tree`. Pouze jeho prostřednictvím totiž může být zobrazena editovaná struktura příkazů. O jakou se přitom aktuálně jedná je vždy možné poznat z kořenového uzlu stromu. Je-li přitom editován objekt `url`, je nad elementem `tree` navíc zobrazena „cesta“, která k němu vede, a tlačítko umožňující návrat k nadřazenému objektu. V dolní části lišty jsou pak tlačítka, která dovolují například spustit nástroj umožňující selekci elementů, ručně přidat či odebrat příkaz ze struktury, ap.

5.1.2 Popup panel

Jak již bylo zmíněno, hlavním účelem popup panelu (viz. obrázek 5.2) je umožnit uživateli modifikaci jednotlivých příkazů. Zobrazí si jej přitom tak, že v postranním panelu klikne pravým tlačítkem na uzel stromu, který si přeje v danou chvíli editovat. Panel mu následně



Obrázek 5.1: Postranní lišta

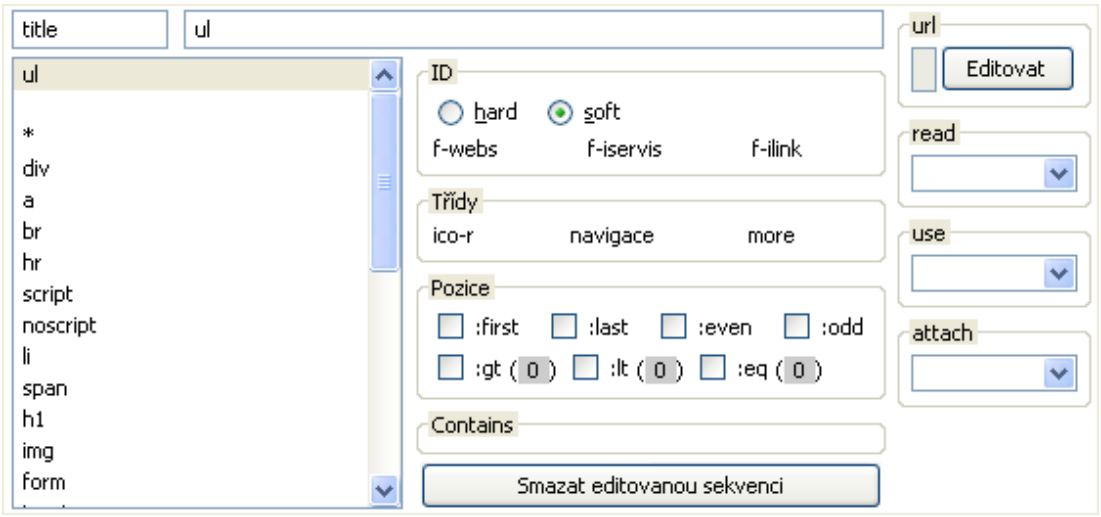
umožní změnit nastavit hodnotu všech atributů příkazu, přičemž navíc jsou zde pro něj připraveny nástroje, které se mu snaží editaci selektoru, potažmo atributu `area`, co nejvíce zjednodušit. Podrobnější informace o principu fungování těchto nástrojů a možnostech jejich rozšíření je možné nalézt v kapitole 5.2.0.4.

5.2 Struktura aplikační logiky

Veškerá aplikační logika rozšíření je obsažena v objektu `infocram`. Ten se přitom skládá z šesti dílčích podobjektů, kdy každý z nich obsluhuje pouze specifickou část požadavků systému. V následujících kapitolách tak bude osvětlen význam každého z nich.

5.2.0.1 Objekt GUI

První z objektů, objekt `gui`, má na starost všechny činnosti související s postranní lištou, zejména pak s jejím elementem `tree`. Umožňuje do něj přidat či z něj odebrat jednotlivé uzly, měnit jejich dílčí hodnoty a mnoho dalšího. V neposlední řadě se tento objekt též stará o zvýrazňování objektů vyhovujících právě editovanému selektoru a o vyznačování kontextu, jenž odpovídá aktuálně vybranému příkazu.



Obrázek 5.2: Popup panel

5.2.0.2 Objekt Tree

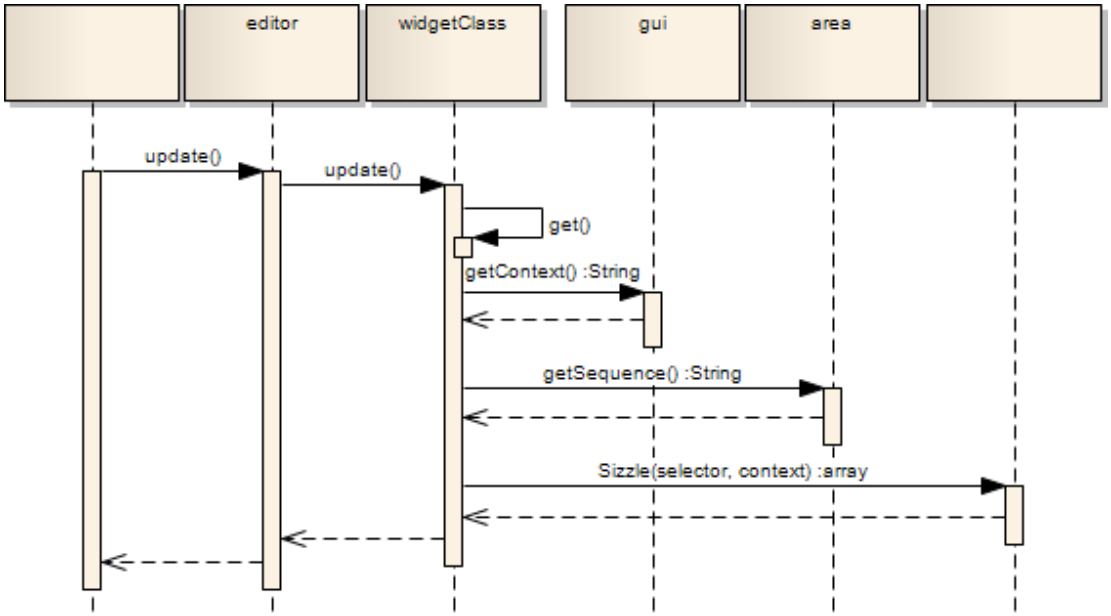
Hlavním a zároveň jediným posláním objektu `tree` je uchování těch částí konfigurace, které nejsou aktuálně zobrazovány elementem `tree`. Jediná metoda, kterou přitom objekt disponuje, tak vrací celou konfiguraci v podobě objektu JSON.

5.2.0.3 Objekt Aardvark

Z názvu toho objektu je již patrné, že jeho obsahem budou metody, které zajišťují úkony souvisejí s nástrojem Aardvark. Předně je to tedy jeho samotné spuštění a následně pak zpracovávání uživatelem vybraných elementů. Pro ně je objekt schopen vygenerovat takový selektor, který bude odpovídá kontextu aktuálně vybraného příkazu, a následně ho pak přidat do příkazové struktury jako jeho potomka.

5.2.0.4 Objekt Editor

Tento objekt se od ostatních podobjektů poněkud liší. Totiž navzdory skutečnosti, že je zodpovědný za veškeré dění související s editorem příkazů, je jeho funkcionalita zcela minimální. Místo něj ji totiž obstarávají jemu podřízené objekty. Každý z nich má přitom na starosti jen dílčí část editoru. Před každým jeho otevřením, potažmo při změně editované sekvence jednoduchých selektorů či přesunu na jinou, je pak volána metoda `infocram.editor.update()` (viz. obrázek 5.3). Ta ovšem neaktualizuje žádné konkrétní prvky editoru, nýbrž projde všechny prvky objektu `infocram.editor` a u těch, jejichž název vyhovuje regulárnímu výrazu `^widget.*$`, nejprve ověří existenci metody `update()` a tu případně pak zavolá.

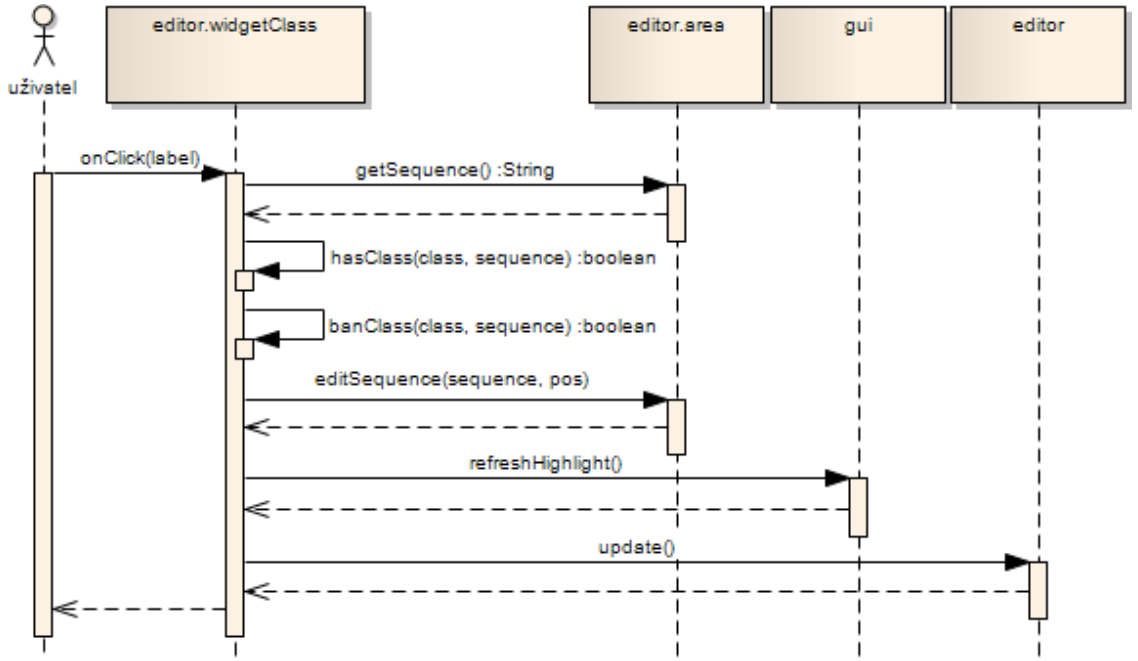


Obrázek 5.3: Aktualizace editoru příkazů

Bude-li tedy někdo chtít rozšířit editor o nějaký vlastní prvek, jehož obsah bude vázán právě na editovanou sekvenci jednoduchých selektorů, je možné to udělat například takto:

```
infocram.editor.widgetAttr = {
    update: function() {

        ⋮

    },
    onClick: function(elem) {

        ⋮

    }
}
```

Obslužná metoda události vyvolané tímto prvek pak probíhá přibližně tak, jak ilustruje diagram uvedený na obrázku 5.4. Tedy v hrubých tím způsobem, že získá aktuálně editovanou sekvenci jednoduchých selektorů, tu vyhodnotí a modifikuje, načež vyvolá aktualizaci zvýraznění elementů vyhovujících aktuálně editovanému selektoru a celkovou obnovu samotného editoru.



Obrázek 5.4: Obsloužení události u nástroje pro editaci selektoru

5.2.0.5 Objekt File

Funkce objektu `file` je více než zjevná. Poskytuje totiž uživateli metody, které mu dovolují editovanou konfiguraci uložit, opětovně načíst či ji začít tvořit znovu zcela od začátku.

5.2.0.6 Objekt Extractor

Smysl posledního z objektů je rovněž prostý, neboť náplní jeho jediné metody je vyžádat si od objektu `tree` aktuálně editovanou konfiguraci v podobě objektu JSON, a ten pak předat extraktoru ke zpracování.

S tím nicméně souvisí jeden z problémů, které se během implementace vyskytly. Jedním z požadavků na systém totiž bylo, aby se při načtení HTML dokumentu, jenž by obsahoval objekt s konfigurací, automaticky spustila extrakce právě podle tohoto zadání. Překážkou v tom se však stal patrný rozdíl v implementaci webového prohlížeče Mozilla Firefox v různých operačních systémech. Zatímco v Linuxu toto funguje bez větších obtíží, v OS Windows Firefox na takovýto HTML dokument nereaguje.

Kapitola 6

Závěr

Hlavního cíle této práce, jímž bylo vytvoření rozšíření pro webový prohlížeč Mozilla Firefox, bylo úspěšně dosaženo. V průběhu jeho návrhu a implementace jsem se přitom podrobně seznámil s Mozilla Application Frameworkem a s jeho možnostmi na poli extrakce webu.

Výsledkem praktické části je pak plně funkční doplněk webového prohlížeče, který umožňuje jednoduše vytvořit konfiguraci pro extraktor vyvíjený vedoucím této práce. Je u něj nicméně ponechán prostor pro dotvoření dalších nástrojů, které by ještě více usnadňovaly tvorbu a modifikaci selektorů, jenž zde použity pro popis dat určených k extrakci.

Nicméně i přesto, že toto rozšíření extrakci webu výrazně usnadňuje, není pravděpodobné, že se tato technika masově rozšíří. Konfigurace extraktoru je totiž stále poměrně dost znalostně náročná. Částečným řešením tohoto problému by přitom mohlo být maximální zjednodušení a zefektivnění uživatelského rozhraní. To je však již námět pro ty, kteří by chtěli v této práci pokračovat a rozvíjet tak dále možnosti extrakce webu prostřednictvím rozšíření webového prohlížeče.

Literatura

[1] Wikipedia. *Mozilla application framework — Wikipedia, The Free Encyclopedia* [online]. c2010 [citováno 16. 05. 2010].
Dostupný z WWW: <http://en.wikipedia.org/w/index.php?title=Mozilla_application_framework&oldid=327090624>

[2] Wikipedie. *XUL — Wikipedie: Otevřená encyklopedie* [online]. c2009 [citováno 19. 05. 2010].
Dostupný z WWW: <http://cs.wikipedia.org/w/index.php?title=XUL&oldid=4471904>

[3] KOVÁCS, Petr. *Analyzátor jazyka XUL pro systém DIG.* Brno, 2007. v, 27 s. Bakalářská práce. Masarykova univerzita, Fakulta informatiky.
Dostupné z WWW: <http://is.muni.cz/th/99085/fi_b/>

[4] Wikipedia. *XUL — Wikipedia, The Free Encyclopedia* [online]. c2010 [citováno 20. 05. 2010].
Dostupný z WWW: <http://en.wikipedia.org/w/index.php?title=XUL&oldid=362530715>

[5] GAGYI Matej. *.Rozšírte si Firefox a Thunderbird - sami! - II* [online]. c2005 [citováno 20. 05. 2010].
Dostupný z WWW: <http://www.abclinuxu.cz/clanky/programovani/rozsirte-si-firefox-a-thunderbird-sami-ii>

[6] Mozilla Foundation. *Building an Extension* [online]. c2010 [citováno 20. 05. 2010].
Dostupný z WWW: <https://developer.mozilla.org/en/Building_an_Extension>

[7] Mozilla Foundation. *XUL Overlays* [online]. c2010 [citováno 20. 05. 2010].
Dostupný z WWW: <https://developer.mozilla.org/en/XUL_Overlays>

[8] Mozilla Foundation. *XPInstall* [online]. c2010 [citováno 18. 05. 2010].
Dostupný z WWW: <https://developer.mozilla.org/en/XPInstall>

[9] Mozilla Foundation. *XPI* [online]. c2010 [citováno 21. 05. 2010].
Dostupný z WWW: <https://developer.mozilla.org/en/XPI>

[10] ÇELIK Tantek, ETEMAD Elika J., GLAZMAN Daniel, HICKSON Ian, LINSS Peter a WILLIAMS John. *Selectors Level 3* [online]. c2009 [citováno 17. 05. 2010].
Dostupný z WWW: <http://www.w3.org/TR/2009/PR-css3-selectors-20091215>

[11] Mozilla Foundation. *Mozilla Application Framework in Detail* [online]. c2007 [citováno 17. 05. 2010].
Dostupný z WWW: <https://developer.mozilla.org/en/mozilla_application_framework_in_detail>

[12] BOS, Bert, ÇELIK Tantek, HICKSON Ian a LIE Håkon Wium. *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification* [online]. c2009 [citováno 22. 05. 2010].
Dostupný z WWW: <http://www.w3.org/TR/2009/CR-CSS2-20090908/>

[13] STANÍČEK Petr. *Praktická příručka CSS, 3. díl - Selektory v CSS* [online]. c2001 [citováno 22. 05. 2010].
Dostupný z WWW: <http://www.webtip.cz/art/wt_titulka/wt_cssserial_003.html>

[14] Wikipedie. *JQuery — Wikipedie: Otevřená encyklopedie* [online]. c2010 [citováno 27. 05. 2010].
Dostupný z WWW: <http://cs.wikipedia.org/w/index.php?title=JQuery&oldid=5375979>

[15] Wikipedie. *JavaScript Object Notation — Wikipedie: Otevřená encyklopedie* [online]. c2010 [citováno 23. 05. 2010].
Dostupný z WWW: <http://cs.wikipedia.org/w/index.php?title=JavaScript_Object_Notation&oldid=5361221>

[16] *Adblock Plus: Element Hiding Helper* [online]. c2010 [citováno 18. 05. 2010].
Dostupný z WWW: <http://adblockplus.org/en/elemhidehelper>

Dodatek A

Seznam použitých zkratek

CSS Cascading Style Sheets

DOM Document Object Model

JSON JavaScript Object Notation

HTML HyperText Markup Language

MAF Mozilla Application Framework

OS Operační systém

UI User Interface

XML Extensible Markup Language

XPI Cross-Platform Install

XUL XML User Interface Language

Dodatek B

Souhrnný přehled CSS3 selektorů

Tabulka B.1: Souhrnný přehled CSS3 selektorů.

Selektor	**Význam**
`*`	všechny elementy
`E`	element `E`
`E[foo]`	element `E`, u nějž je uveden atribut `foo`
`E[foo="bar"]`	element `E`, kde je hodnota atributu `foo` rovna "bar"
`E[foo~="bar"]`	element `E`, kde je hodnotou atributu `foo` seznam slov oddělených bílým znaky a "bar" je zároveň právě jedním z nich
`E[foo^="bar"]`	element `E`, kde hodnota atributu `foo` začíná řetězcem "bar"
`E[foo$="bar"]`	element `E`, kde hodnota atributu `foo` končí řetězcem "bar"
`E[foo*="bar"]`	element `E`, kde hodnota atributu `foo` zahrnuje podřetězec "bar"
`E[foo|="en"]`	element `E`, kde hodnota atributu `foo` začíná řetězcem "en", přičemž bezprostředně za ním následuje pomlčka
`E:root`	element `E`, který je kořenovým elementem dokumentu
`E:nth-child(n)`	element `E`, který je n-tým potomkem svého rodiče
`E:nth-last-child(n)`	element `E`, který je n-tým potomkem svého rodiče odzadu
`E:nth-of-type(n)`	element `E`, který je n-tým potomkem svého typu
`E:nth-last-of-type(n)`	element `E`, který je n-tým potomkem svého typu odzadu
`E:first-child`	element `E`, který je prvním potomkem svého rodiče
`E:last-child`	element `E`, který je posledním potomkem svého rodiče
`E:first-of-type`	element `E`, který je prvním potomkem svého typu
`E:last-of-type`	element `E`, který je posledním potomkem svého typu
`E:only-child`	element `E`, který je jediným potomkem svého rodiče
`E:only-of-type`	element `E`, který je jediným potomkem svého typu
`E:empty`	element `E`, který nemá žádné potomky (včetně textových uzlů)
`E.warning`	element `E`, jehož jednou z tříd je "warning"
`E\#myid`	element `E`, jehož identifikátorem je "myid".
`E:not(s)`	element `E`, který neodpovídá selektoru `s`

Pokračování na další straně...

Tabulka B.1 – Pokračování

Selektor	**Význam**
`E F`	element `F`, který je následníkem elementu `E`
`E > F`	element `F`, který je přímým potomkem elementu `E`
`E + F`	element `F`, který bezprostředně následuje za elementem `E`
`E ~ F`	element `F`, který následuje za elementem `E`

Dodatek C

Ukázková konfigurace extraktoru

```
{
  delay: 1000,
  def: {
    cesky_nazev: {
      url: {
        area: ".interwiki-cs a",
        read: "link"
      },
      area: "h1",
      read: "text"
    }
  },
  url: "http://en.wikipedia.org/wiki/List_of_capitals_and_largest_cities_by_country",
  select: {
    staty: {
      area: "table.wikitable:eq(0) tr:has(td):lt(2)",
      select: {
        zeme: {
          area: "td:eq(0) a",
          select: {
            nazev: {
              read: "text"
            },
            adresa: {
              read: "link"
            },
            detail: {
              url: {
                read: "link"
              },
              select: {
                plny_nazev: {
```

```
                area: ".infobox.geography .country-name",
                read: "text"
              },
              cesky_nazev: {
                use: ["cesky_nazev"]
              }
            },
            attach: "inline"
          }
        },
        attach: "single"
      },
      mesto: {
        area: "td:eq(1) a",
        select: {
          nazev: {
            read: "text"
          },
          adresa: {
            read: "link"
          },
          detail: {
            url: {
              read: "link"
            },
            select: {
              mistni_nazev: {
                area: ".infobox.geography .nickname:eq(0)",
                read: "text"
              },
              cesky_nazev: {
                use: ["cesky_nazev"]
              }
            },
            attach: "inline"
          }
        },
        attach: "single"
      }
    }
  }
}
```

Dodatek D

Obsah přiloženého CD

```
/
|-- INSTALL.TXT                postup instalace rozšíření
|-- README.TXT                 soubor s popisem obsahu CD a návodem k instalaci
|-- data                       data související s bakalářskou prací
|   |-- ...
|-- sample                     adresář obsahující soubor se vzorovou konfigurací
|   `-- mesta.htm              soubor se vzorovou konfigurací extraktoru
|-- src                        adresář obsahující zdrojové soubory rozšíření
|   `-- ...
|-- text                       adresář obsahující vlastní text bakalářské práce
|   `-- bakalarska-prace.pdf   text bakalářské práce ve formátu PDF
`-- xpi                        adresář s instalačním balíčkem rozšíření
    `-- InfoCram_1.0.xpi       instalační balíček rozšíření
```