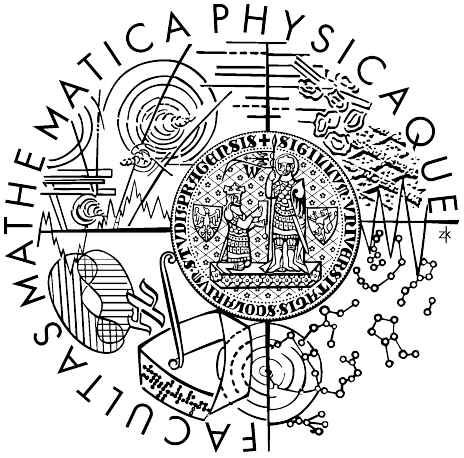
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Michal Sedlák

Webové rozhraní pro platformu Treex

Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: Mgr. Martin Popel

Studijní program: Informatika

Studijní obor: Obecná informatika

Praha 2013

Poděkování

Na tomto místě bych chtěl poděkovat svému vedoucímu Martinu Popelovi za jeho ochotu, pomoc a trpělivost s přípravou této bakalářské práce.

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Praze dne 1. srpna 2013

Abstrakt

Název práce: Webové rozhraní pro platformu Treex

Autor: Michal Sedlák

Katedra: Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: Mgr. Martin Popel

Abstrakt: Tato práce pojednává o webové aplikaci `Treex::Web`, která slouží jako webové rozhraní k platformě pro zpracování úloh přirozeného jazyka Treex. Práce poukazuje na několik nedostatků Treexu (především absenci grafického uživatelského rozhraní a komplikovanou instalaci) a nabízí `Treex::Web` jako jejich možné řešení. V úvodu práce je nejprve představena samotná platforma Treex. Následuje kapitola 3 popisující uživatelské rozhraní `Treex::Web`u a kapitola 4 zabývající se vlastní implementací webové aplikace. V závěru práce je provedeno srovnání s podobnými platformami jako Treex a jejich webovými rozhraními.

Klíčová slova: Treex, Treex::Web, NLP framework, Perl, Catalyst, REST, webové služby

---

Title: Web Interface for the Treex Framework

Author: Michal Sedlák

Department: Institute of Formal and Applied Linguistics

Supervisor: Mgr. Martin Popel

Abstract: This work deals with a web application called `Treex::Web` which serves as a web interface for NLP framework Treex. The work addresses several Treex issues (e.g. absence of graphical user interface and complicated installation) and offers `Treex::Web` as a possible solution. At the beginning of this work we introduce the Treex framework itself. The following chapters describe `Treex::Web`'s user interface (chapter 3) and the implementation of the whole web application (chapter 4). Conclusion of this work includes a comparison of NLP frameworks similar to Treex and their web interfaces.

Keywords: Treex, Treex::Web, NLP framework, Perl, Catalyst, REST, web services

Obsah

1

Úvod

Treex je modulární platforma určená k řešení různých úloh NLP (Natural Language Processing – zpracování přirozeného jazyka), vyvíjená na Ústavu formální a aplikované lingvistiky (ÚFAL) od roku 2005.[1] Treex vychází z předpokladu, že každou úlohu lze rozdělit na posloupnost menších kroků, které jsou implementovány jako jednotlivé moduly (nazývané *bloky*). Aplikace řešící danou úlohu představuje posloupnost takových bloků a nazývá se *scénář*. Výhoda této architektury je především v znovupoužitelnosti, tj. schopnosti pro nové úlohy jednoduše vytvářet nové scénáře z již existujících bloků.

Použití Treexu je ovšem pro běžné uživatele vcelku složité. Hlavní bariéry představují:

Závislost na platformě Velká část Treexu je k dispozici pouze pro operační systémy typu Linux.

Hardwarová náročnost Některé scénáře mají vysoké paměťové nároky, např. výchozí model pro závislostní parsing vyžaduje 1,5 GiB operační paměti.[2]

Komplikovaná instalace Návod na instalaci[3] má cca 3 strany textu a kompletní instalace včetně závislostí vyžaduje administrátorská práva a může trvat více než 30 minut.

Absence GUI Nutnost ovládání z příkazové řádky.

[1] `http://ufal.mff.cuni.cz/treex/`

[2] Existuje i menší model `pdt20_train_autTag_golden_latin2_pruned_0.10.model` vyžadující přibližně 0,5 GiB operační paměti, který má ovšem horší výsledky.

[3] `http://ufal.mff.cuni.cz/treex/install.html`

Plochá učicí křivka Vytváření vlastních scénářů vyžaduje dobrou znalost celého frameworku (především jednotlivých bloků).

Distribuce Část Treexu je dostupná přes CPAN,[1] ale většina jen přes oficiální SVN, kde je nestabilní vývojová verze.

Tyto nedostatky řeší webová aplikace[2] zastřešující Treex, která vznikla jako součást této práce. Aplikace se nazývá `Treex::Web` a je veřejně k dispozici na adrese:

`http://quest.ms.mff.cuni.cz/treex-web/`

Jediné, co uživatel v případě webové aplikace potřebuje, je prohlížeč a internetové připojení. Veškerá data a výpočty obsluhuje výkonný server, díky čemuž není třeba Treex instalovat.

Navíc `Treex::Web` je přístupný odkudkoliv a lze jeho prostřednictvím používat Treex i na zařízeních a operačních systémech, kde by to z důvodu nekompatibility nebo hardwarové náročnosti nebylo možné. Například lze spustit i paměťově náročné scénáře (viz závislostní parsing výše) z tabletů a dalších mobilních zařízení, které tolik paměti nemají a často používají s Treexem jinak nekompatibilní operační systém.

Celá aplikace stojí na jednoduchém a přehledném GUI, ve kterém lze jednoduše vybrat často používané scénáře. Zpracované výsledky scénářů je možné pohodlně procházet a automaticky vizualizovat přímo v prohlížeči.

Mnoho nástrojů vyvíjených na ÚFALu potřebuje webové rozhraní (demo verzi) a některé z nich (např. strojový překlad [1], rozpoznávač koreferencí [2]) jsou už zapojeny do platformy Treex, takže `Treex::Web` poslouží jako webové rozhraní pro tyto nástroje.

Ne ve všech případech je použití webového rozhraní pro Treex vhodné. S `Treex::Web`em je uživatel jako s každou jinou webovou aplikací 100% závislý na internetovém připojení a stabilitě serveru. Také zpracování velkého množství dat je momentálně komplikované, jelikož maximální délka vstupu i doba běhu scénáře jsou omezené.[3] Hlavně pokročilí uživatelé mohou za nedostatek považovat nemožnost upravovat stávající bloky nebo si přidávat vlastní.

[1] `https://metacpan.org/module/Treex`

[2] Webová aplikace není příliš ustálený pojem. V tomto kontextu se myslí aplikace přístupná ve webovém prohlížeči přes Internet nebo intranet. Nejde ovšem o běžnou webovou stránku, ale o web s charakteristikami desktopové aplikace.

[3] Vstup je omezený na 5 MB a běh scénáře na 5 minut

S možnostmi, které nabízí nová generace prohlížečů a webových technologií, bude velice snadné přiblížit se funkčnosti desktopových aplikací. Ovšem v případě Treexu nejspíše webový interface z výše zmíněných důvodů nikdy plně nenahradí lokální instalaci. Může ale dobře posloužit pro demonstrační účely a pro výuku.

V kapitole 2 si nejprve stručně představíme samotný Treex jako platformu. Potom si ukážeme (kapitola 3) uživatelskou část webového rozhraní a jeho funkce.

Kapitola 4 se zaměřuje přímo na jednotlivé části, ze kterých se webové rozhraní skládá, a na to, jaká rozhodnutí vedla k jejich současné podobě. Slouží také jako doplnění k vygenerované dokumentaci, kterou lze nalézt na DVD přiloženém k této práci.

V kapitole 5 provedeme krátké srovnání s ostatními NLP platformami a případně i s jejich webovými rozhraními, zhodnotíme přínos této práce a zmíníme některé plány do budoucna.

Součástí této práce je také DVD, které kromě zdrojových kódů a vygenerované dokumentace obsahuje také virtuální stoj s plně funkční instalací Treexu i webového rozhraní `Treex::Web`.

2

Platforma Treex

> Velká část této kapitoly je založena na článku TectoMT: Modular NLP Framework [3] z roku 2010, ale reflektuje současnou verzi Treexu.

Většina netriviálních úloh NLP vyžaduje několik druhů nástrojů (např. tokenizer, tagger, parser...) a často je třeba tyto nástroje nějakým způsobem spolu integrovat do výsledné aplikace. Samotné úpravy a *slepení* příslušných nástrojů dohromady můžou zabrat hodně času. Navíc ve většině případů je třeba tuto *zbytečnou* práci dělat znovu a znovu pro další nástroje nebo jiné aplikace. V takových a dalších případech může pomoci Treex jako NLP framework.

Treex se vyvinul z původní platformy TectoMT zaměřené především na strojový překlad (MT – Machine Translation) z angličtiny do češtiny přes tektogramatickou rovinu popisu jazyka. V roce 2010 byla provedena kompletní reimplementace velké části této platformy, přibylo mnoho dalších funkcí a platforma dostala současný název Treex.

Treex nyní kromě strojového překladu řeší i další NLP úlohy jako například part-of-speech (POS) tagging, rozpoznávání pojmenovaných entit (named entity recognition), přiřazování sémantických rolí (semantic role labeling) nebo harmonizaci treebanků (HamleDT[4]).

2.1 Architektura

Jak jsme již zmínili v úvodní kapitole, Treex klade důraz na modularitu a znovupoužitelnost. Jednotlivým modulům se v rámci Treexu říká *bloky*. Každý *blok* má přesně

definovaný vstup a výstup, jednotlivé bloky dokonce můžou mít své parametry (např. název modelu, který se použije pro POS tagging).

Aplikace řešící danou úlohu je posloupnost jednotlivých bloků, taková posloupnost se v Treexu nazývá *scénář*. Bloky v posloupnosti lze jednoduše zaměnit za jiné a vytvořit tak například alternativní řešení stejné úlohy s použitím jiných metod.

Příkladem takové úlohy může být morfologická a syntaktická analýza anglického textu (výpis kódu 2.1 a 2.2).

```
Read::Text from=input.txt
W2A::EN::Segment
W2A::EN::Tokenize
W2A::EN::TagMorce
W2A::EN::Lemmatize
W2A::EN::ParseMST
Write::Treex to=result.treex
```

Výpis kódu 2.1: Scénář A

```
Read::Sentences from=input.txt

W2A::EN::Tokenize
W2A::EN::TagFeaturama
W2A::EN::Lemmatize
W2A::EN::ParseMalt
Write::CoNLLX to=result.conll
```

Výpis kódu 2.2: Scénář B

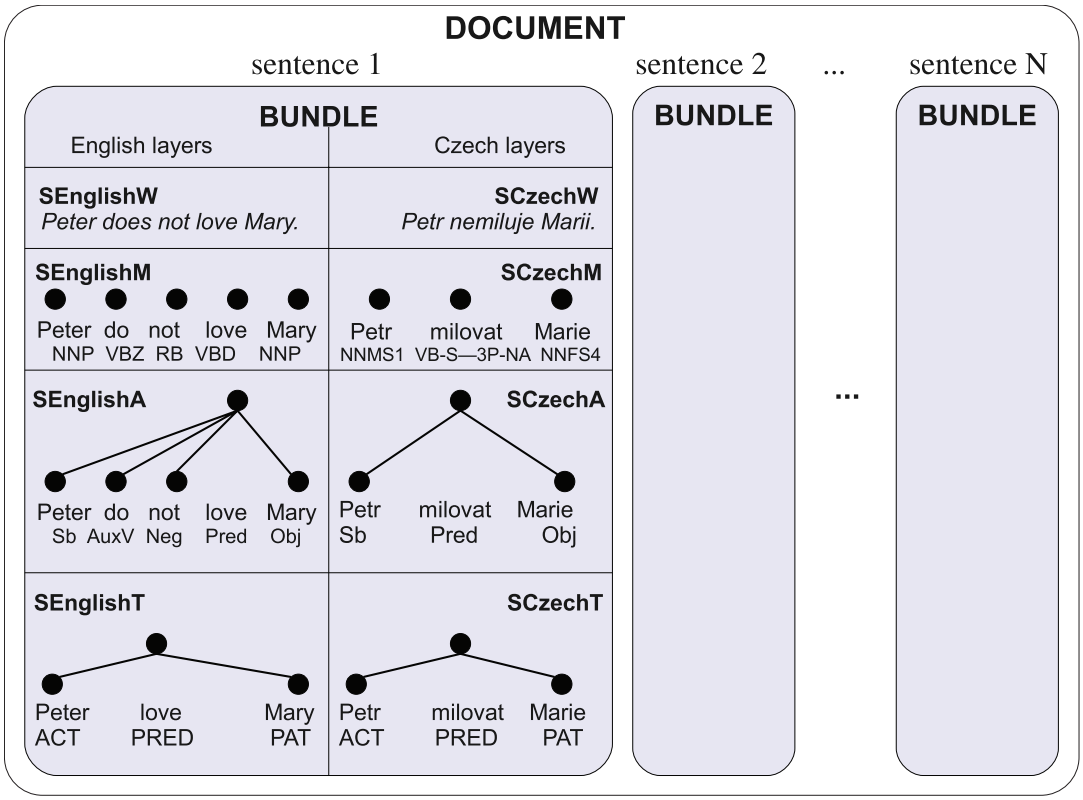
V scénáři A načítáme čistý text a zapisujeme soubor ve formátu Treex. V scénáři B je vstup už segmentovaný (každá věta na novém řádku), takže ze souboru přečteme celé věty. Výstup zapíšeme v conll formátu. Hlavní rozdíl je ale v použitém taggeru a parseru. Scénář A používá tagger Morče [5] a závislostní parser MST [6]. Scénář B používá tagger Featurama a závislostní parser Malt [7]. Takto lze například jednoduše porovnat oba taggery a parsery na stejném textu.

2.2 Struktura dat

Treex vychází z teorie Funkčního generativního popisu jazyka [8] a její implementace v podobě Pražského závislostního korpusu [9], který definuje čtyři roviny (*layers*) popisu: vstupní text (w-layer), morfologická rovina (m-layer), analytická rovina (a-layer), tektogramatická rovina (t-layer). Analytická a tektogramatická rovina jsou reprezentovány jako závislostní stromy (a-stromy a t-stromy). Treex tuto koncepci přebírá, ale z technických důvodů nezavádí samostatnou m-rovinu. Atributy slov příslušejících k m-rovině (slovní forma, lemma a POS tag) ukládá do a-stromů. Naopak přidává p-rovinu pro reprezentaci složkových stromů a n-rovinu pro reprezentaci pojmenovaných entit.

Treex pracuje s daty ve formě jednotlivých dokumentů (jeden dokument typicky odpovídá jednomu souboru na disku). Každý dokument je rozdělen na věty a každá věta je reprezentována jako tzv. *bundle*. Bundle je dále rozdělen do *zón* odpovídajících různým jazykům.[1] Zóna obsahuje už stromy pro jednotlivé roviny (a-strom, t-strom, p-strom, n-strom).

V nejjednodušším případě analýzy jednoho jazyka na a-rovinu obsahuje každý bundle jen jednu zónu a ta jen jeden strom (a-strom). Obrázek 2.1 ukazuje složitější případ analýzy paralelních česko-anglických vět až na t-rovinu.



Obrázek 2.1: Struktura treexového dokumentu – anglicko-český text anotovaný na třech rovinách, každá věta v jednom *Bundle*. Převzato z článku [3], který používá ještě starší značení a samostatnou m-rovinu.

[1] Zóny jsou označovány kódem příslušného jazyka (např. en=angličtina, cs=čeština). Někdy je potřeba uložit v jednom dokumentu dvě analýzy téže věty, například výstup dvou různých taggerů. Každá analýza se uloží do jedné zóny a tyto zóny se rozliší tzv. selektorem (např. cs-morce, cs-featurama).

2.3 Implementace Treexu

Jádro Treexu a jednotlivé *bloky* jsou napsané v Perlu, je ovšem běžné, že některé bloky jsou jen obal nad existující aplikací například v Javě nebo v C/C++. Celý Treex je postavený na knihovně Moose[10] a využívá tedy moderní Perlové techniky objektově orientovaného programování.

Kromě Perlového jádra a bloků obsahuje Treex ještě tzv. sdílenou (shared) datovou část, tedy hlavně jazykové modely, nástroje třetích stran a další pomocný software. Tato část se s Treexem běžně nedistribuuje a framework si umí potřebná data stáhnout z webu ve chvíli, kdy je potřebuje.

Formáty souborů

Interní datový formát Treexu (`.treex`) je XML (Extensible Markup Language) spolu se schématem definovaným v PML (Prague Markup Language)[11]. Pro dočasné soubory lze použít tzv. Storable Treex (`.streex`), což je serializovaná Perlová struktura, jejíž výhodou je rychlé načítání, ovšem není zaručena kompatibilita mezi verzemi Perlu.

Pro načítání jiných datových formátů lze použít širokou škálu Read bloků, které Treex obsahuje. Za zmínku stojí hlavně formáty Penn Treebank[12] a CoNLL[13].

Vizualizace dat

Samotný Treex data přímo vizualizovat neumí. Obsahuje ale modul pro Tree Editor TrEd,[14] díky kterému lze v data z Treexu v TrEdu zobrazit.

3

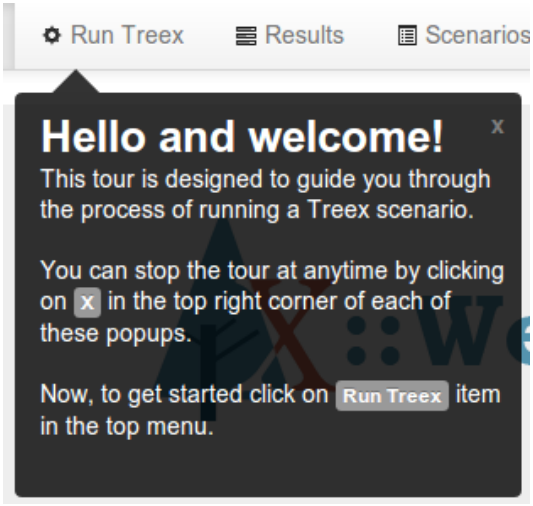
Uživatelská příručka

3.1 Začínáme

`Treex::Web` je veřejně dostupný na adrese:

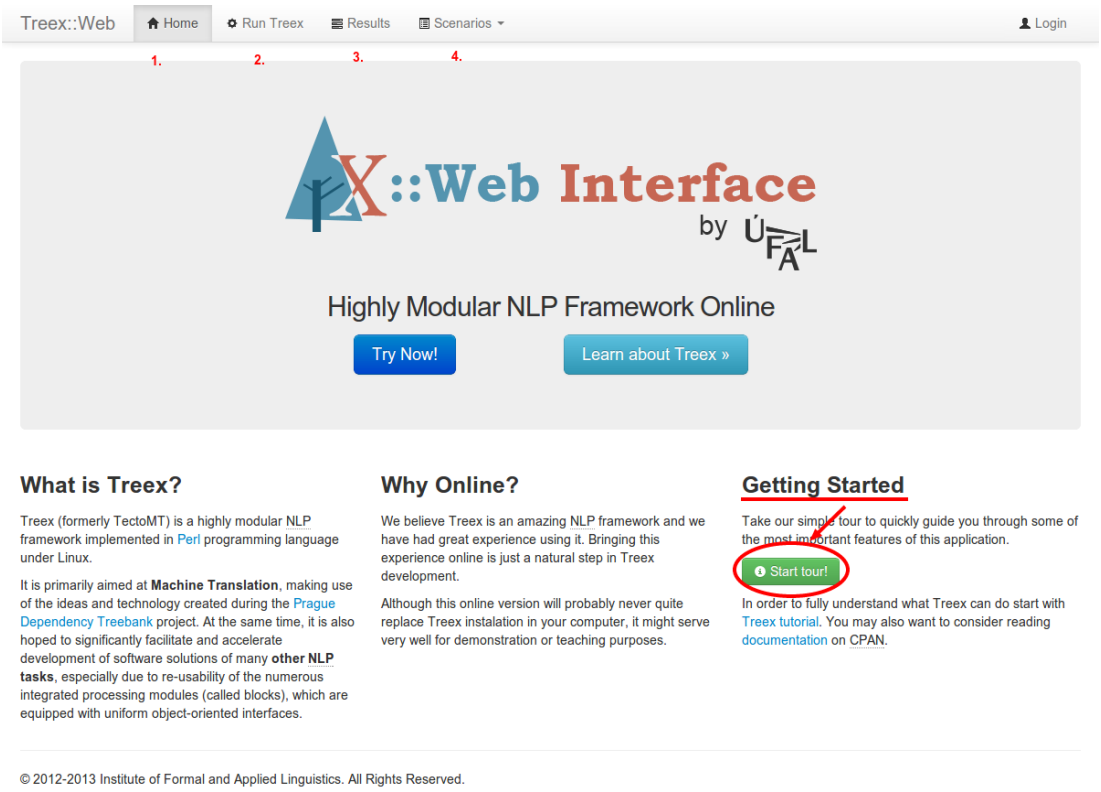
`http://quest.ms.mff.cuni.cz/treex-web/`

Úvodní stránka (obrázek 3.1) nabízí uživateli stručné informace o Treexu a především možnost interaktivní prohlídky webu (obrázek 3.2), která nového uživatele provede prvním spuštěním scénáře.



Obrázek 3.2: Interaktivní nápověda - Formou popisků provede nového uživatele procesem spuštění Treexového scénáře.

3. UŽIVATELSKÁ PŘÍRUČKA



Obrázek 3.1: Úvodní stránka - `http://quest.ms.mff.cuni.cz/treex-web/`

Uživatelské rozhraní (GUI) je navržené s maximálním ohledem na jednoduchost – všechny funkce jsou dostupné přes horní menu (viz horní část obrázku 3.1):

1. Úvodní stránka
2. Spuštění Treexového scénáře
3. Přehled výsledků
4. Nabídka pro práci se scénáři, přehled všech scénářů a přidání nového scénáře
5. Přihlašovací formulář

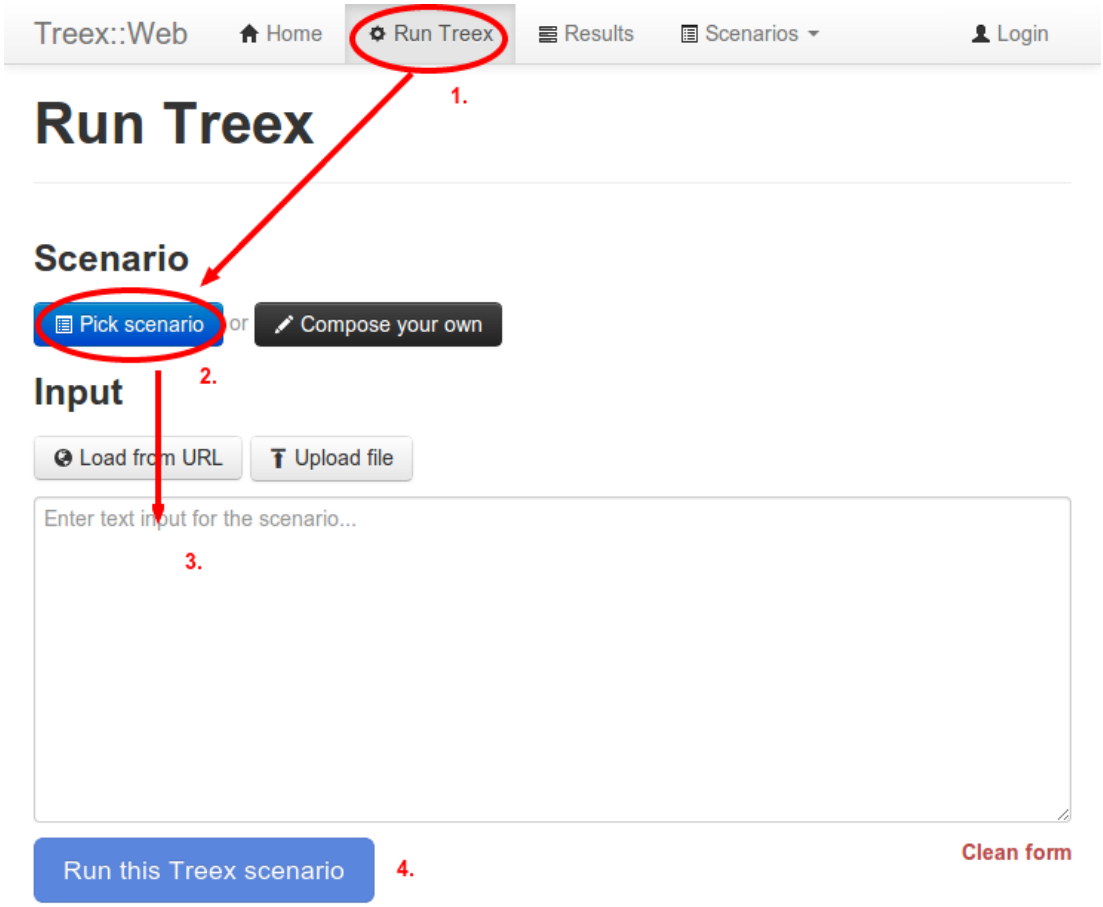
3.2 Spuštění scénáře

Pro spuštění scénáře stačí 3 jednoduché kroky (viz obrázek 3.3).

1. Vybrat v horním menu `Run Treex`
2. Kliknutím na `Pick scenario` vybrat scénář z nabídky nejčastěji používaných scénářů (který je možno později upravit) nebo napsat vlastní scénář (`Compose your own`)
3. Nakonec zbývá definovat vstup. K dispozici jsou až 4 možnosti:
   (a) Napsat ručně text nebo jej například kopírovat ze schránky přímo do textového pole.
   (b) Vložit text z webové stránky (`Load from URL`). V takovém případě se aplikace pokusí rozpoznat hlavní obsah stránky a ten použije jako vstup.
   (c) Nahrát soubor ze svého počítače. Podporované formáty jsou prostý text, treex a conll[13].
   (d) Většina scénářů obsahuje nějaký vzorek textu, na kterém lze scénář rychle vyzkoušet. V takovém případě přibude tlačítko `Insert scenario sample` (viz obrázek 3.4).

Vstup a výstup definují `Read` a `Write` bloky na začátku a na konci každého scénáře. `Treex::Web` se snaží definici vstupu pro nové uživatele zjednodušit, aby vůbec scénář upravovat nemuseli. Například v případě uploadu vlastního souboru (3c) se provede změna `Read` bloku podle formátu automaticky.

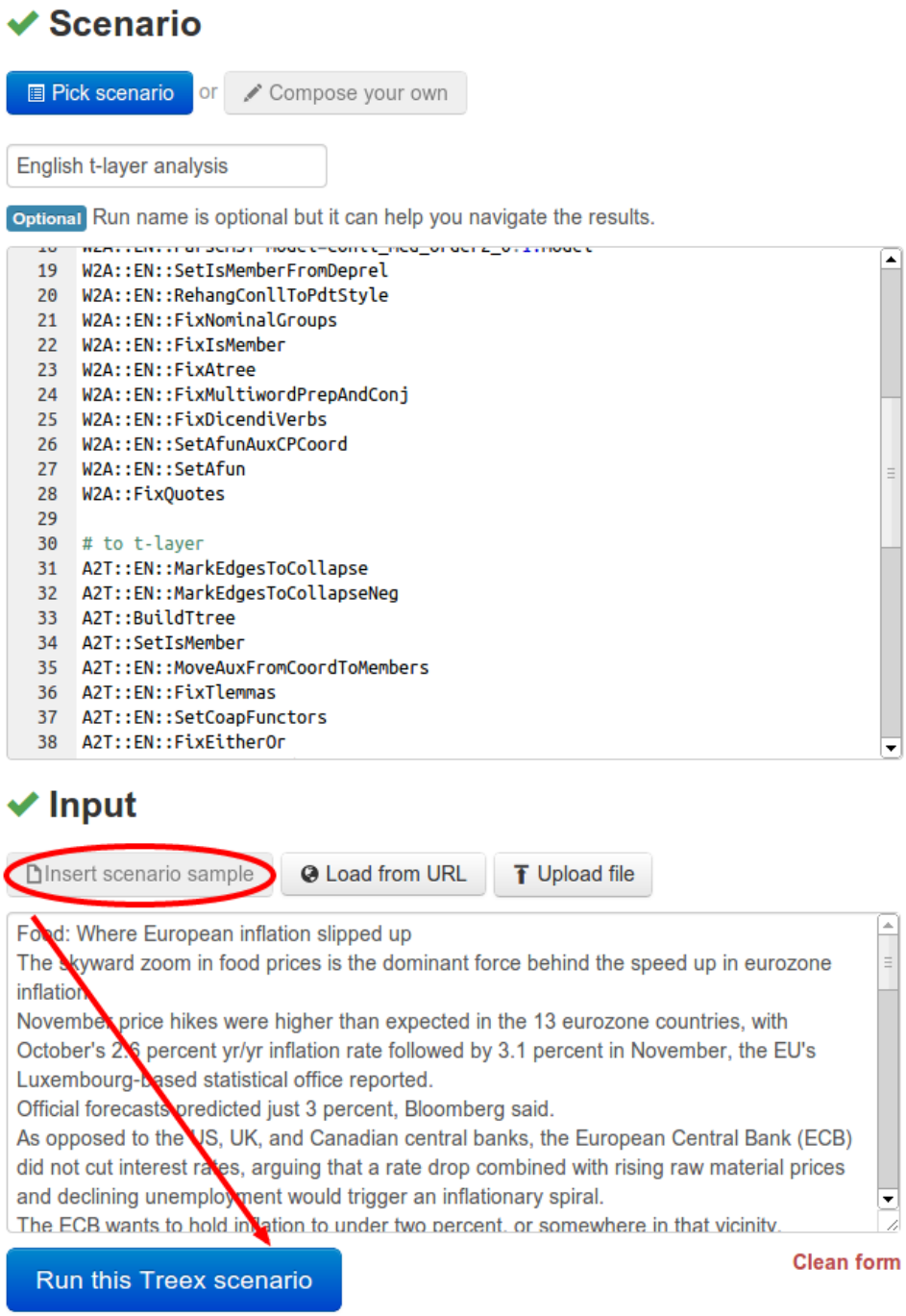
To ale neznamená, že si pokročilí uživatelé nemohou měnit `Read` a `Write` bloky podle potřeby. K dispozici jsou bloky: `(Read|Write)::Text`, `Sentences`, `Treex` a `CoNLLX`.



Obrázek 3.3: Spuštění scénáře 2

Z bezpečnostních důvodů je nutné použít bloky bez dalších parametrů.[1] Doplnění vstupních a výstupních souborů do parametrů `from`, resp. `to` proběhne automaticky.

[1] Aby nebylo možné načíst systémové nebo jiné soubory. Například `Read::Text from=/etc/passwd`



Obrázek 3.4: Spuštění scénáře 1

3.3 Práce s výsledky

Výsledky jsou Treexem zpracované scénáře. K přehledu výsledků se lze dostat přes položku `Results` v horním menu stránky (3 v obrázku 3.1).

Status	Name	Date
Queued	English t-layer analysis	less than a minute ago
Working	Czech analysis	less than a minute ago
Failed	English t-layer analysis	about a minute ago
Completed	Run 3	6 minutes ago

Obrázek 3.5: Přehled výsledků

U každého výsledku je zobrazeno několik základních údajů jako stav zpracování, název a okamžik zadání (viz obrázek 3.5). Stavy jsou následující:

Queued Výsledek zatím čeká ve frontě na zpracování

Working Treex právě výsledek zpracovává, výstup bude brzy k dispozici

Completed Zpracování je kompletní, výsledek je možné si prohlédnout

Failed Během zpracování došlo k chybě:

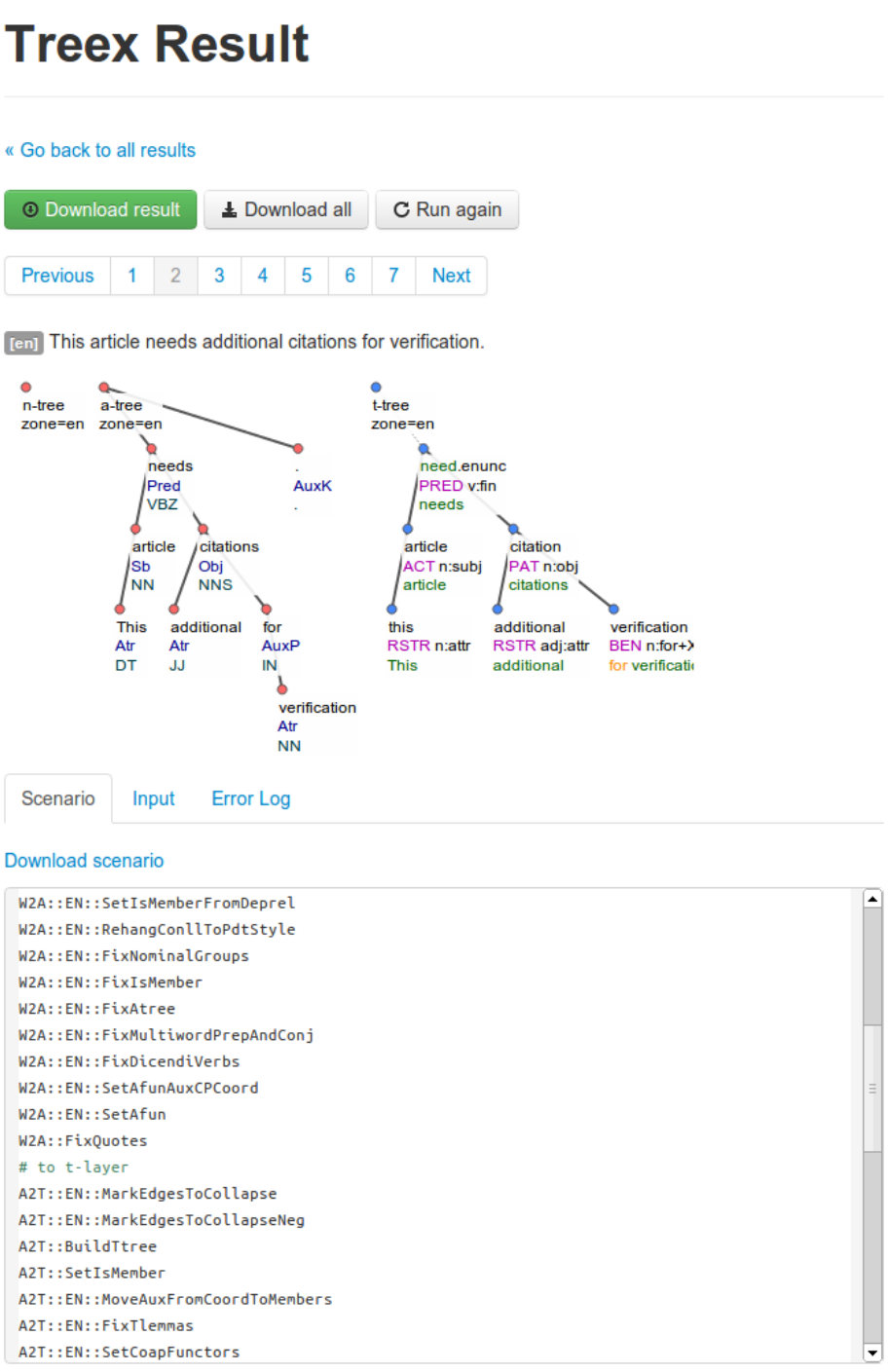
1. Treex nemohl zpracovat scénář
2. Vypršel časový limit na zpracování
3. Došlo k chybě na serveru

Celý přehled výsledků se automaticky aktualizuje daty ze serveru, takže uživatel vždy vidí aktuální informace. Stejně tak místo nezpracovaného výsledku se zobrazí vyčkávací animace a po dokončení se automaticky přepne do zobrazení výsledku (viz obrázek 3.6).

Pokud byl výstupem scénáře soubor v Treexovém formátu, nabídne se vizualizace jednotlivých vět. Jinak je možné stáhnout si výsledný soubor do počítače přes tlačítko `Download result`.

U každého zpracovaného výsledku si lze navíc prohlédnout:

1. Scénář
2. Zadaný vstup, ať už čistý text nebo soubor



Obrázek 3.6: Detail výsledku

3. Chybový výstup Treexu, který obsahuje dodatečné informace o průběhu zpracování scénáře a slouží dobře pro případnou diagnostiku chyb

Přes tlačítko `Run again` lze provést nové spuštění s možností upravit původní scénář i vstup. Tlačítko `Download all` nabízí možnost stáhnout si kompletní[1] výsledek do počítače.

Všechny výsledky vytvořené nepřihlášenými uživateli mají jen omezenou životnost a přibližně do 42 hodin (v závislosti na konfiguraci) jsou ze serveru smazány. Pro trvalé uložení výsledků je nutné se přihlásit.

3.4 Přihlášení

Přihlášení je možné prostřednictvím jednoduchého dialogového okno. Přihlášený uživatel si může ukládat vlastní scénáře a výsledky jím zpracovaných scénářů nejsou automaticky mazány. Má možnost se tak ke své práci kdykoliv vrátit bez obav, že by o ni přišel.

Nové uživatelské účty může vytvářet jen administrátor pomocí scriptu na serveru. Volná registrace je v aplikaci implementovaná, ale do budoucna se s ní nepočítá a přihlašování do celého systému bude nejspíše vytvořeno přes Shibboleth[15] nebo jiný centralizovaný systém.

Pro potřeby této práce je v každé instalaci aplikace (včetně veřejně dostupné verze) vytvořen uživatelský účet s přihlašovacími údaji, které lze nalézt v tabulce 3.1.

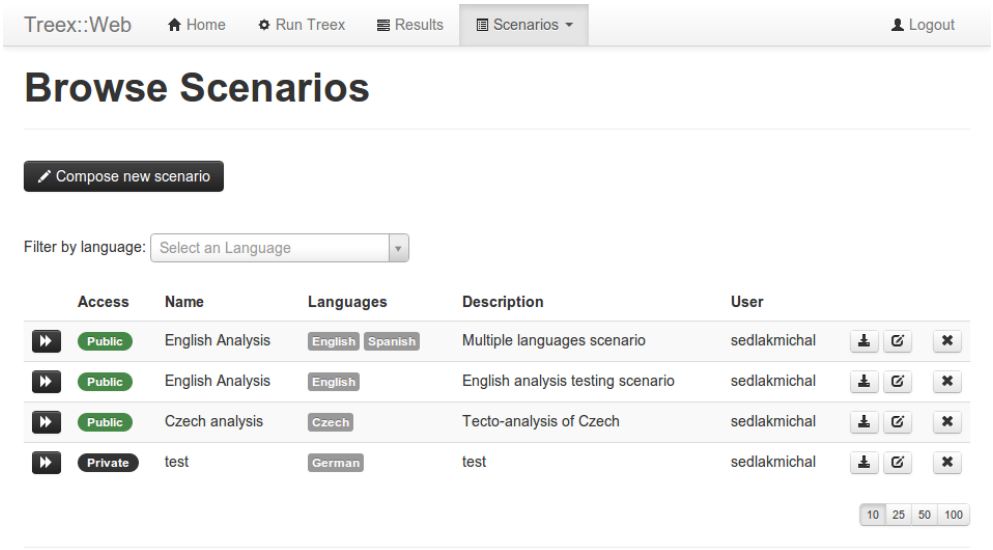
Email:	`treex@ufal.mff.cuni.cz`
Heslo:	`LetMeIn`

Tabulka 3.1: Přihlašovací údaje pro testování

3.5 Vlastní scénáře

Každý uživatel si může spustit vlastní scénář nebo upravit stávající (viz obrázek 3.3 – tlačítka `Pick scenario` a `Compose your own`). Ovšem jen přihlášený uživatel může ukládat vlastní scénáře.

[1] Archiv ve formátu `zip` s výstupním souborem, vstupním souborem, scénářem a chybovým výpisem



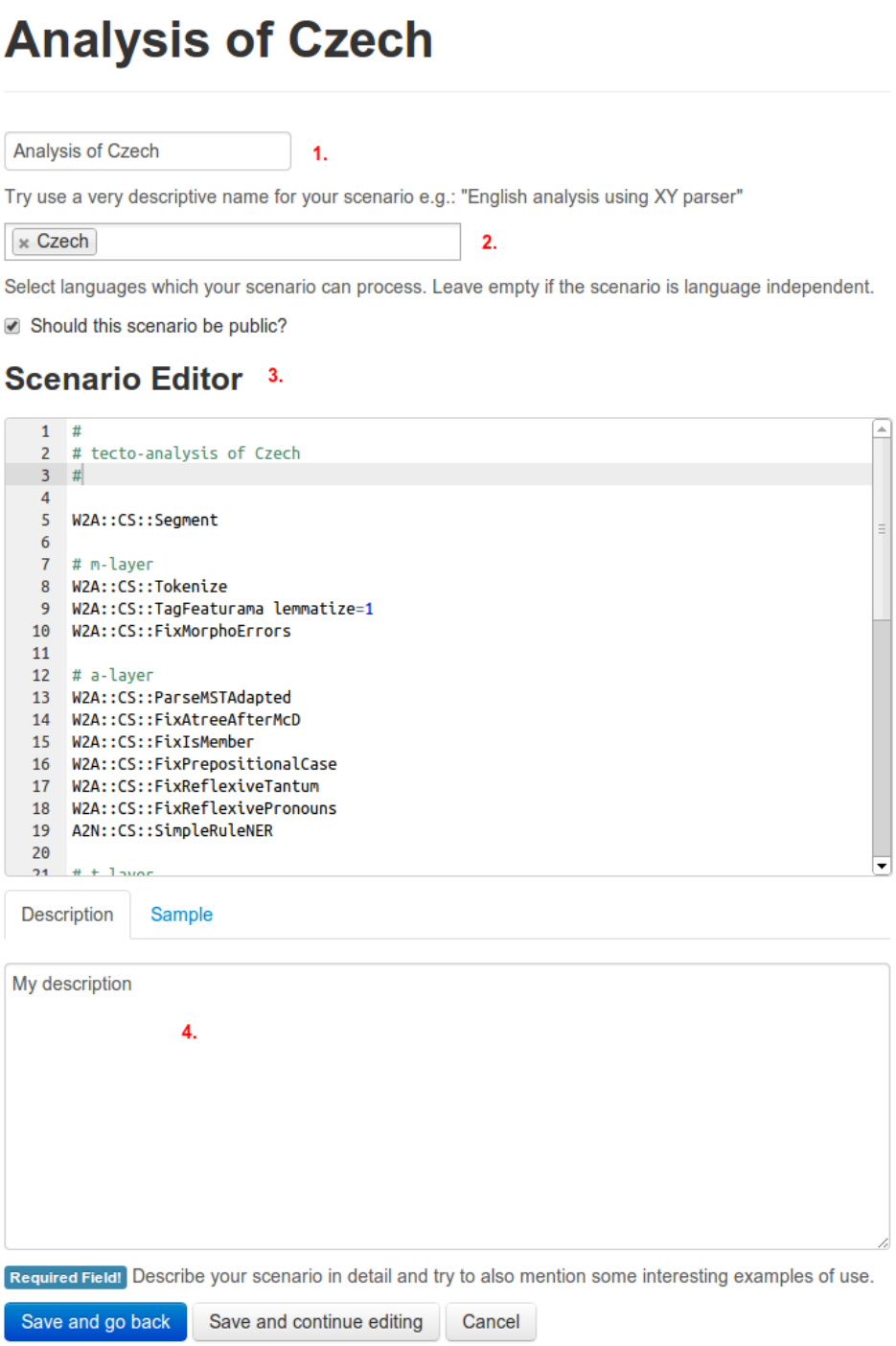
Obrázek 3.7: Přehled scénářů

K formuláři pro vložení vlastního scénáře se lze dostat skrze nabídku `Scenarios` v hlavním menu (4 v obrázku 3.1) přes položku `New Scenario` nebo v přehledu všech scénářů (obrázek 3.7) přes tlačítko `Compose new scenario`. Definice scénáře se skládá z celkem šesti položek (obrázek 3.8). Povinné jsou:

1. Název scénáře
2. Podporované jazyky
3. Samotný scénář
4. Popis scénáře - krátký popis

Nepovinné položky jsou:

- Vzorek textu, na kterém lze scénář jednoduše vyzkoušet.
- Označení, zda je scénář veřejně dostupný (Should this scenario be public?). Lze totiž definovat scénáře, které nevidí ostatní uživatelé – ty se potom v přehledu scénářů zobrazí jako `private`.



Obrázek 3.8: Přidávání nového scénáře

4

Implementace

Jelikož tato kapitola je velice technická, pokusím se v první části stručně vysvětlit principy webových aplikací a objasnit některé pojmy, které jsou použity dále.

4.1 Typická webová aplikace

Webová stránka stažená ze serveru, jak ji vidíme v prohlížeči, obvykle kombinuje tři technologie: HTML (HyperText Markup Language), CSS (Cascading Style Sheets) a Javascript. HTML zajišťuje především obsah a sémantiku, CSS vzhled a Javascript často potřebnou další logiku.

Je potřeba rozlišit mezi logikou na straně prohlížeče (klienta) a na straně serveru. Dnešní web je totiž z velké části dynamicky generovaný (zobrazuje nějaká dynamická, v čase se měnící data). O generování se musí starat aplikace běžící na straně serveru.

Velké množství takových aplikací používá návrhový vzor MVC (Model View Controller), který se skládá ze tří částí. Model reprezentuje data (např. databáze s daty uživatelů), View je typicky šablona nějakého HTML dokumentu a Controller je vlastní logika, která tyto části spojuje.

Cílem tohoto návrhového vzoru je rozdělit aplikaci na části, které spolu komunikují podle daných pravidel, ale jinak jsou na sobě nezávislé. Výhody z toho plynoucí jsou především snadnější údržba a možnost jednotlivé části jednodušeji testovat.

4.2 Poněkud netypická webová aplikace

Už na začátku vývoje bylo jasné, že `Treex::Web` nebude jen obyčejná webová aplikace. Jelikož Treexový scénář může běžet až několik minut, je nutné spuštět Treex tak, aby neblokoval hlavní proces aplikace.
K tomu můžeme využít některý z následujících způsobů:

- Spustit Treex ve zvláštním vlákně (vyžaduje použití synchronizačních primitiv a management vláken)
- Použít asynchronní neblokující event-driven programování.[16]
- Využít tradiční unixový fork.[17] *Forkující* proces vytvoří kopii sebe sama jako svůj vlastní nový podproces. Dostaneme tak dva paralelně bežící procesy. Toto řešení by ovšem znamenalo velké plýtvání pamětí, jelikož by se kopírovala celá aplikace
- Pro zpracování scénáře spouštět samostatný proces odděleně od hlavního procesu aplikace.

Nejpraktičtější je poslední možnost – spouštět scénáře jako samostatný proces odděleně od hlavního procesu aplikace. Získáme tím hned několik výhod najednou, mezi které patří:

Vyšší stabilita
Treexový scénář může skončit chybou nebo jinak selhat např. díky nedostatečně odladěnému bloku. Toto by nemělo mít v žádném případě vliv na chod hlavní aplikace.

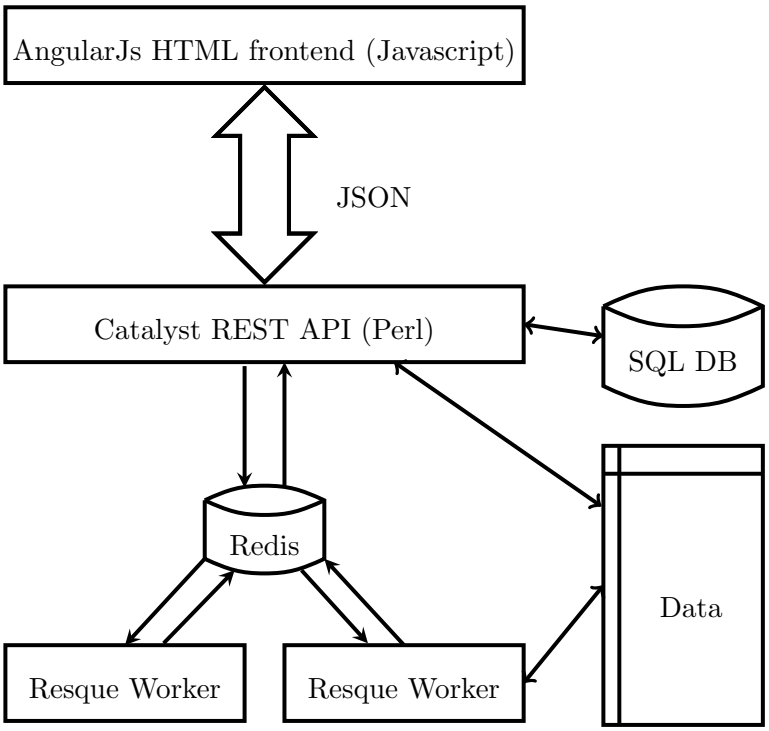
Škálovatelnost
Procesů spouštějících Treex může běžet více nebo můžou běžet na jiných strojích.

Bezpečnost
Samostatný proces je možné spustit izolovaně tzv. v sandboxu a chránit tak ostatní data na serveru před potenciálně nebezpečným kódem.

Podle původního záměru měl mít `Treex::Web` dvě části – *webovou část*, která generuje HTML a komunikuje s uživatelem, a část *výpočetní*, která spouští Treexové scénáře. V průběhu vývoje se ukázalo, že nároky na interaktivitu aplikace jsou tak vysoké, že bude výhodné *webovou část* ještě rozdělit na frontend běžící u klienta (v prohlížeči) a na API (Application Programming Interface) běžící na serveru.
Implementace `Treex::Web`u se tedy skládá z celkem tří částí:

1. Frontend – postavený na frameworku AngularJS a téměř nezávislý na ostatních částech. Jedná se de facto o jediný statický HTML dokument. Veškeré dynamické aspekty, včetně komunikace se serverem, jsou řešené pomocí Javascriptu. Frontend je blíže popsaný v části 4.3.
2. API – napsané v Perlu za použití frameworku Catalyst, obsluhuje především data v databázi, zadávání scénářů a vizualizaci Treexových souborů. Více lze nalézt v části 4.4.
3. Spouštění Treexových scénářů – systém založený na knihovně Resque, která implementuje frontu požadavků – tu následně zpracovávají procesy (tzv. Workers). Dalšími podrobnostmi se zabývá část 4.5.

Frontend a API používají ke komunikaci JSON (Javascript Object Notation). API s *výpočetní* částí komunikuje prostřednictvím úložiště Redis [18]. Schéma jednotlivých částí aplikace je znázorněno na obrázku 4.1.

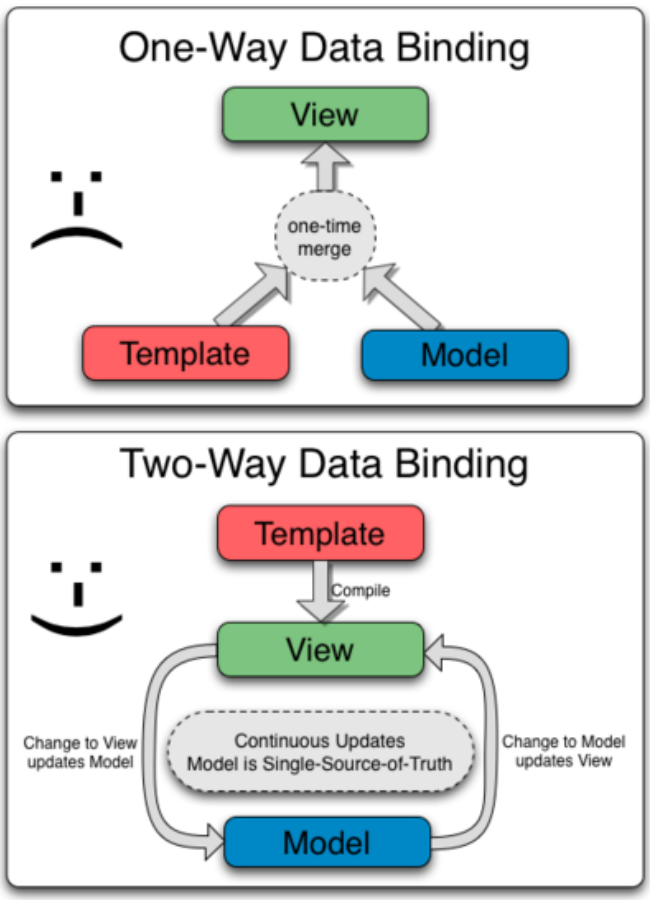


Obrázek 4.1: Schéma implementace - Části `Treex::Webu` postupně popsané v částech 4.3, 4.4 a 4.5

4.3 Frontend

AngularJS je javascriptový open-source framework určený především pro tvorbu tzv. single page aplikací (aplikace, která se skládá jen z jednoho HTML dokumentu[19, kapitola 13 a 22]).

Angular přidává do HTML další značky a umožňuje tak deklarativně napojit logiku na UI (User Interface) komponenty. Navíc implementuje two-way data-binding, a tím umožňuje synchronizaci UI a datového modelu (viz obrázek 4.2). V praxi to například znamená, že není třeba implementovat logiku, která *vyplní* formulář obsahem datového modelu a stejně tak není třeba implementovat logiku, které naopak aktualizuje datový model daty z formuláře.

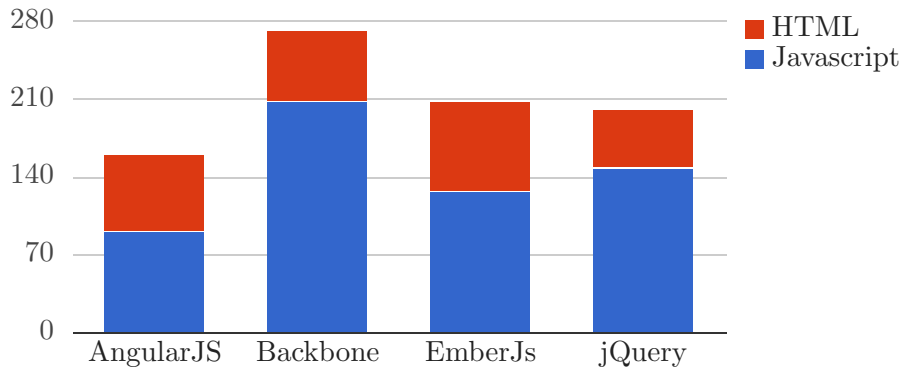


Obrázek 4.2: Ukázka principu *bindování* dat. V případě one-way databindingu je třeba implementovat kód, který se stará o transfer dat mezi View a datovým Modelem. Obrázky převzaty z dokumentace Angularu.[20]

Angular šetří při vývoji hodně času a kód aplikace je díky tomuto frameworku podstatně kratší a přehlednější. Toto lze demonstrovat například na projektu TodoMVC,[1] jehož účelem je implementovat stejnou aplikaci (Todo list) na různých platformách a nabídnout tak jejich srovnání.

	Javascript	HTML	Počet souborů
AngularJS	91	70	6
Backbone	207	64	7
EmberJs	127	80	9
jQuery	148	52	3

Tabulka 4.1: Srovnání počtu řádků jedné aplikace implementované na třech rozdílných platformách (AngularJS, Backbone[21] a EmberJs[22]) a v jQuery (knihovna usnadňující manipulaci s HTML, potažmo s DOM).



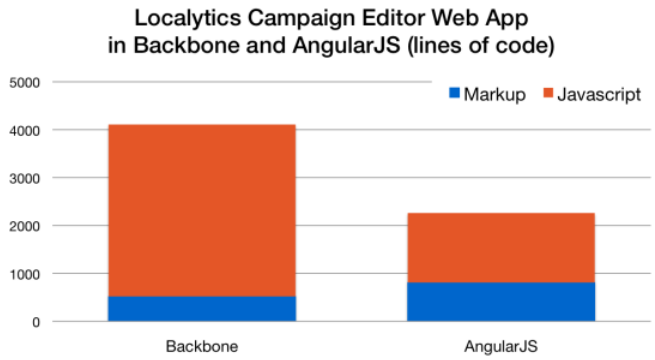
Obrázek 4.3: Graf k tabulce 4.1

Pokud porovnáme rozsah implementace Todo listu v Angularu s dvěma dalšími frameworky (Backbone[2][21] a EmberJs[3][22]), vychází Angular jako nejúspornější (tabulka 4.1 a graf na obrázku 4.3).

Přestože porovnání na základě počtu řádků kódu není vzhledem k malému rozsahu aplikace vypovídající, v praxi mohou být aplikace v Angularu skutečně co do rozsahu

[1] `http://todomvc.com/`
[2] `http://backbonejs.org/`
[3] `http://emberjs.com/`

až poloviční oproti jiným frameworkům (viz projekt společnosti Localystics[23] a graf na obrázku 4.4).



Obrázek 4.4: Srovnání Angularu a populárního Backbone. Počet řádek aplikace v Angularu je cca poloviční. Převzato z blogu společnosti Localystics[23].

4.3.1 Struktura souborů

Frontend se v adresářové struktuře `Treex::Web`u nachází v stejnojmenném adresáři `frontend`.

Všechny soubory v `frontend/app` jsou veřejně dostupné přes web server a adresář je míněný jako kořenový adresář aplikace. Kompletní adresářová struktura je na obrázku 4.5.

Soubory jsou podle typu organizovány do podadresářů. Hlavní stránka `index.html` je v kořenovém adresáři a další HTML fragmenty, které se načítají podle potřeby, jsou v podadresáři `views`.

V podadresáři `styles` jsou kaskádové styly (CSS) a v podadresáři `scripts` jsou javascriptové soubory, které tvoří nejrozsáhlejší část (viz tabulka 4.2).

	`Treex::Web`	Externí knihovny, pluginy...
HTML	780	0
CSS	200	8 500
Javascript	2 660	více než 33 000

Tabulka 4.2: Přehled počtu řádek zdrojového kódu frontendu – rozdělené na vlastní zdrojový kód `Treex::Web`u (součást této práce) a kód externích knihoven.

```
frontend/app/
├── images
├── scripts
│   ├── ace
│   ├── angular
│   ├── controllers
│   ├── directives
│   ├── filters
│   ├── services
│   ├── app.js
│   ├── config.js
│   ├── treex.js
│   └── treeview.js
├── styles
│   ├── fonts
│   └── aplication.css
├── views
│   └── ...
└── index.html
```

Obrázek 4.5: Adresářová struktura frontendu

Tabulka 4.2 také ukazuje, že frontend `Treex::Webu` neobsahuje skoro žádné vlastní kaskádové styly. Téměř kompletní vzhled je totiž postavený na knihovně Bootstrap[1][24] od společnosti Twitter.

Javascript je rozdělený na část patřící Angularu (4.3.2) a část věnující se vizualizaci závislostních stromů (4.3.4).

4.3.2 Angular

Angular má pět typů komponent:

1. Moduly (Module) – modul je druh komponenty, která v sobě obsahuje zbylé čtyři typy komponent (viz níže) a zastřešuje jejich společnou konfiguraci a inicializaci. Celý frontend je jeden modul (viz soubor `app.js`).
2. Direktivy (Directive) – direktivami se v Angularu nazývají extra atributy a značky v HTML. Lze říct, že jejich hlavní úkol je manipulace s DOM. Frontend implementuje přes 20 vlastních direktiv, které obsluhují především dynamické aspekty samotné stránky, např.:

[1] `http://twitter.github.io/bootstrap/`

- Kreslení a prohlížení stromů (4.3.4) – direktiva `twView` a další
- Stránkování (viz `pagination.js`)
- Animace načítání stránky (`twLoader`)
- Periodické spuštění funkce (`twTimer`)

3. Controllery (Controller) – controller je komponenta, která připraví data potřebná pro zobrazení stránky a následně reaguje na akce uživatele. V našem případě je co stránka, to controller – tedy každý view fragment v adresáři `views` má svůj vlastní controller.
4. Filtry (Filter) – pomocné komponenty, které slouží k formátování dat. Jejich zápis vypadá takto: `name | trim | uppercase`, což je ekvivalent pro `uppercase(trim(name))`. Většina často používaných filtrů je přímo v distribuci Angularu. My implementujeme dva: `noHtml` – escapuje HTML, a `newlines` – nahrazuje konce řádků za `<br>`.
5. Služby (Service) – úkolem služeb je poskytovat data. Služby jsou singleton objekty, takže vždy existuje jen jedna instance pro aplikaci. V našem případě služby komunikují s REST API (viz část 4.4) a jejich pojmenování tomu odpovídá:
   - Auth – přihlášení uživatele
   - Scenario – načítání a ukládání scénářů
   - Result – práce s výsledky zpracovanými Treexem
   - Input – pomocné operace pro práci se vstupem scénáře
   - Treex – načítání společných dat v podobě seznamu jazyků atd.
   - Tour – zajištění interaktivní prohlídky (jako jediná neobsluhuje komunikaci se serverem)

4.3.3 Editor scénářů

Proč vlastně mít editor scénářů? Scénář je prostý text a určitě by stačilo pro psaní jednoduché textové pole. Ovšem hlavně u delších scénářů je jednoduché se v nich ztratit, jelikož komentáře splývají s bloky a základní zvýrazňování syntaxe je v tomto případě skoro nutnost.

Nabízí se tedy použití editoru, který by se vložil do stránky a posloužil jako náhrada za textové pole. Takových editorů existuje celá řada, většina z nich se ale specializuje na

konkrétní jazyk. Z těch univerzálních jsou zajímavé především – Ace[1] a CodeMirror.[2]

Oba editory si jsou velmi podobné, a dokonce je možné jednoduše zaměnit jeden za druhý. Pro `Treex::Web` jsem nakonec zvolil Ace, jelikož implementuje další funkce, které se budou do budoucna hodit. Kromě zvýrazňování syntaxe totiž nabízí:

- Kontrolu syntaxe
- Chytré automatické doplňování
- Možnost mít přímo v kódu aktivní prvky, na které lze kliknout. Tato funkce je velice důležitá, protože dovoluje k jednotlivým blokům zobrazit např. nápovědu.

4.3.4 Vizualizace stromů

Vizualizace stromů (dále označovaná jako tisk) je pro webové rozhraní klíčová funkce. Jak jsme zmínili v kapitole 2.3, Treex tisk přímo neimplementuje a je třeba použít TrEd. Otázka, jak dostat obrázky stromů do prohlížeče, je vyřešená v PML-TQ, které k vizualizaci využívá právě TrEd. Součástí TrEdu totiž je tzv. print server, který umí exportovat stromy do SVG (Scalable Vector Graphics),[3] jež umí zobrazit webový prohlížeč. Toto řešení funguje dobře, ale má dvě zásadní úskalí:

- Perl Tk[25], ve kterém je TrEd napsaný, vyžaduje přítomnost X Window systému,[4] jinak nefunguje. To znamená, že ani print server nefunguje bez grafického interface. Na serveru to lze obejít přes virtuální X frame buffer (Xvfb), což ale skrývá jeden velký problém – lze mít jen jednu instanci print serveru (k Xvfb nelze přistupovat paralelně). Tisk stromů v tomto případě představuje slabinu PML-TQ serveru, protože k nápadnému zpomalení tisku stromů dochází už ve chvíli, kdy aplikaci používá 3–5 uživatelů současně.
- SVG export z TrEdu je vhodný jen pro zobrazování, a jelikož nemá žádnou jasnou strukturu, je jakákoliv další práce se stromy přinejmenším komplikovaná.

Vzhledem k těmto nedostatkům se jako řešení nabídlo upuštění od tisku na serveru a implementace tisku na straně prohlížeče pomocí Javascriptu a SVG.[5]

[1] `http://ace.ajax.org/`

[2] `http://codemirror.net/`

[3] Umí i export do PDF.

[4] http://www.x.org/

[5] Místo SVG by šlo použít Canvas element (součást specifikace HTML5), ovšem SVG je díky své *vektorovosti* mnohem vhodnější pro zobrazování stromů.

Prvním úkolem bylo dostat Treexový dokument do Javacriptu, aby s ním bylo možné vůbec pracovat. Javascriptový formát pro serializaci dat je de facto JSON a Perlové pole a hash odpovídá poli a objektu v JSONu. Takže konverze z Treexu do JSONu znamená převod Treexových objektů na primitivní Perlové struktury.

Dalším úkolem bylo poskládat JSON znovu do Treexového dokumentu (tentokráte v Javascriptu), aby s ním bylo možné pracovat. Za tímto účelem vznikla javascriptová knihovna nazvaná jednoduše `Treex`, jejímž hlavním úkolem je z JSONu opět vytvořit strukturu Dokument → Bundle → Zóny → Stromy (viz obrázek 2.1). Knihovna částečně kopíruje Perlový interface Treexu, ale implementuje jen nutné minimum, v žádné případě se nejedná o Treex v Javascriptu.

Posledním krokem je samotná vizualizace.

První implementace

Rozhodl jsem se vytvořit vlastní knihovnu, která by poskytla interface k nakresleným stromům a především abstrakci nad SVG, aby nebylo nutné přímo manipulovat s jednotlivými SVG elementy. Toto řešení se nakonec díky mé nezkušenosti (s SVG) ukázalo jako sice funkční, ale velmi pomalé.

Dva stromy – každý okolo 35 uzlů, trvalo nakreslit 0,2 – 0,5 s (prohlížeč Chrome verze 27, procesor 3,2 GHz). Každý strom je totiž potřeba nakreslit vlastně dvakrát. Z prvního nakreslení se zjistí rozměry všech elementů a v druhém se spočítá jejich umístění.

Klíčová chyba mojí implementace byla především v příliš mnoha operacích s DOM (operace nad SVG elementy, popř. jejich atributy) než neefektivním algoritmem. Každá změna stavu se totiž automaticky zapsala do SVG a to i v případě, že se jednalo o pomocné hodnoty.

Druhá implementace

Pokud by měla první implementace fungovat, bylo by třeba minimalizovat množství operací s DOM. Vzhledem k tomu, že šlo o poměrně hodně práce, rozhodl jsem vyzkoušet alternativní implementaci.

Ze zvědavosti jsem vyzkoušel knihovnu D3.js[26, 27] pro vizualizaci dat. Můj prototyp kreslení stromů v D3 měl cca 120 řádků kódu a výkon byl naprosto nečekaný.

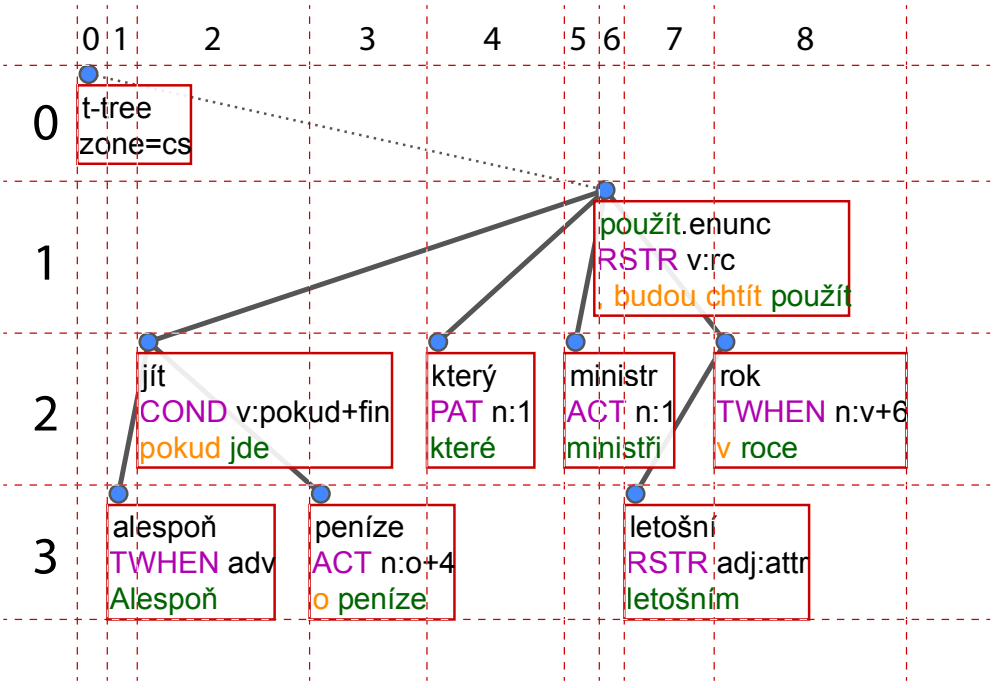
Rychlost vykreslování se na stejné konfiguraci jako výše pohybovala okolo 50 ms a velikost stromu na výkon téměř neměla vliv.

D3 totiž pracuje s DOM velice efektivně. Dokáže dokonce recyklovat jednotlivé elementy – například pokud má strom 15 uzlů a následně se má překreslit dalším stromem (řekněme 10 uzlů), D3 smaže jen 5 uzlů a zbylé použije pro nový strom.

Rozhodl jsem se tedy zahodit svoji *knihovnu* (více jak 1200 řádků CoffeeScriptu[1]) a nahradit ji implementací v D3.js s polovičním počtem řádků (okolo 600 řádků Javascriptu) a řádově vyšší rychlostí.

Algoritmus

Algoritmus kreslení stromů je velice přímočarý. Kreslení probíhá ve dvou fázích. V první fázi se všechny elementy nakreslí ve výchozí pozici, aby bylo možné zjistit jejich rozměry. Ve druhé fázi se spočítá umístění každého uzlu a provede se přesun na vypočtené souřadnice, včetně úpravy pozic jednotlivých hran.



Obrázek 4.6: Algoritmus tisku stromů

Výpočet souřadnic si lze představit jako matici $m \times n$, kde m je hloubka stromu a

[1] Malý jazyk, který se překládá do Javascriptu.

n je počet jeho uzlů (obrázek 4.6). Uzly mají ve stromě dané pořadí. Výška každého řádku se počítá jako výška nejvyššího uzlu na dané úrovni stromu. Odsazení zleva je buď odsazení předchůdce + okraj (uzel $[3, 7]$ na obrázku 4.6), nebo nejbližší uzel zleva na stejném řádku + okraj (např. uzel $[2, 8]$), podle toho, co je větší. Složitost algoritmu je tak vzhledem k počtu uzlů lineární.

4.3.5 Instalace a spuštění

Frontend vychází spíše z Javascriptového než Perlového světa. Je tedy nutné mít nainstalované `Node.js` a `npm` (Node Package Manager).

Pro nasazení na server a další úkoly se používá Grunt[1][28, strana 18 – 20], pro správu javascriptových knihoven je použit Bower.[2] První spuštění vývojového serveru vypadá takto:

```
cd frontend
npm install    # nainstaluje Grunt a Bower
bower install  # nainstaluje potrebne javascriptove knihovny
grunt server   # spusti server http://localhost:9000
```

Příkazem `grunt build` se do adresáře `dist` provede sestavení pro produkční použití (dojde k nejrůznějším optimalizacím pro provoz na síti).[3] K *běhu* frontendu pak postačí libovolný web server s konfigurací pro statické soubory.

Na oficiální adrese (kapitola 3.1) je použit web server Apache[29] a v adresáři `app` se nachází soubor `.htaccess` s konfigurací připravenou speciálně pro tento web server.

Samotný frontend bez API nic neumí, proto je pro fungování ještě potřeba API připojit. V současné době lze API připojit jen přes reverzní proxy, takže se část URL frontendu `/api/v1` směruje na server s API. V konfiguraci pro Apache je to takto:

```
<Location /treex-web/api/v1>
ProxyPass  http://treex-web/api/v1/
ProxyPassReverse  /treex-web/api/v1/ http://treex-web/api/v1/
<Location>
```

Výpis kódu 4.1: Konfigurace reverzní proxy pro web server Apache

[1] `http://gruntjs.com/`

[2] `http://bower.io/`

[3] Jedná se především o sloučení malých souborů dohromady a minifikaci Javascriptu a CSS. Navíc každý *build* přidá k názvu statických souborů kus jejich SHA1 součtu, takže si je prohlížeč nemůže splést s jinou verzí, která může být třeba ve vyrovnávací paměti.

Výpis kódu 4.1 s konfigurací říká, že všechny požadavky na adresu API[1] budou směřovány na aplikační server běžící na adrese `http://treex-web/api/v1/` (ProxyPass). Zároveň ProxyPassReverse upravuje hlavičky všech odpovědí z aplikačního serveru tak, aby vypadaly, že přišly ze stejné adresy, na kterou byly odeslány, tj. z adresy API.

4.4 RESTful API

Termín REST (Representational State Transfer) poprvé popsal ve své disertaci[30] Roy Fielding, jeden ze spoluautorů protokolu HTTP (Hypertext Transfer Protocol).

REST se téměř okamžitě stal dominantním modelem pro webová API,[2] jelikož na rozdíl od XML–RPC či SOAP je orientován spíše datově, nikoliv procedurálně.[3]

Hlavní koncept RESTu vychází z myšlenky, že každý datový zdroj (*resource*) má svůj unikátní identifikátor (URI) a manipulace s daty se provádí skrz druh metody v hlavičce HTTP požadavku: `GET`, `POST`, `PUT`, `DELETE` značící: čtení, vytvoření, update a smazání.

Ne každé webové API se může nazývat REST (nebo také RESTful). Aby API bylo tzv. RESTful, musí splňovat *omezující* podmínky, jako například jednotný interface nebo bezstavovost.[4]

API navržené pro Treex není zcela RESTful, jelikož mírně porušuje podmínku bezstavovosti. Server udržuje informace o jednotlivých sezení (`session`) – nutné pro uchovávání informace o přihlášení uživatele. Identifikátor sezení se přenáší mezi požadavky pomocí cookie. Přestože cookie jsou běžnou součástí požadavku, obecně při použití v kombinaci se `session` přestává API být RESTful.[5]

[1] `/treex-web/api/v1`

[2] Web API je stále ještě hodně obecný pojem. V tomto případě se myslí jasně definovaný interface (systém požadavek–odpověď) mezi serverem a klientem.

[3] Navíc XML–RPC i SOAP jsou protokoly, které HTTP používají jen jako transportní mechanismus.

[4] Podmínky a další principy REST jsou podrobněji popsané například v článku Principled design of the modern Web architecture[31] nebo v disertaci Roye Fieldinga[30, kapitola 5].

[5] Existují ovšem i jiné názory, například v knize RESTful Web Services Cookbook[32] v kapitole 1.3 How to Maintain Application State se kombinace `session` a cookie přímo doporučuje.

4.4.1 Perl a Catalyst

Přestože Treex je napsaný v Perlu, rozhodnutí implementovat API webového rozhraní také v Perlu nebylo samozřejmé. Mám velké zkušenosti s frameworky Ruby on Rails a Express. Výhody Ruby on Rails jsou především v dobře vyladěném ekosystému a Express těží z výhody jednoho jazyka (Javascriptu) na backendu i frontendu, což téměř eliminuje nutnost duplikování logiky.

Volba Perlu byla čistě pragmatická – jádro Treexu je v Perlu a na ÚFALu je spousta lidí, kteří mají zkušenosti s Perlem, a tedy i potenciálních udržovatelů. Pro Perl jsou k dispozici tři aktivně vyvíjené frameworky pro vývoj webových aplikací:

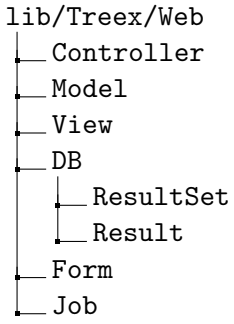
Dancer
Výhodou Danceru je především jednoduchost a minimalismus. Dancer jsem použil ve webovém frontendu pro PML-TQ[33]. Framework se pro tento projekt ukázal jako ideální řešení, ale při vývoji bylo zřejmé, že u větší aplikace by brzy došlo ke komplikacím, jelikož framework je opravu hodně jednoduchý.

Catalyst
Další z rodiny frameworků pro Perl má za sebou dlouhou historii. Je vyvíjen od roku 2005, má přes 200 registrovaných vývojářů a také opravdu velký ekosystém (viz `http://mapofcpan.org/`). Na první pohled ideální kandidát.

Mojolicious
Nenápadný framework od stejného autora jako Catalyst. Ve vývoji je od konce roku 2010 a implementuje nejnovější technologie jako podporu pro asynchronní neblokující I/O web server nebo WebSockets[34]. Navíc v sobě kombinuje jednoduchost Danceru s možnostmi Catalystu.
Vše nasvědčuje tomu, že Mojolicious je budoucnost pro vývoj webových aplikací v Perlu.

Celá implementace API pro webový frontend Treexu je v Perlovém modulu `Treex::Web`. Kromě Catalystu se opírá o modul `Catalyst::Controller::REST` (dále jen `REST`), který značně zjednodušuje celý vývoj, jelikož zajišťuje serializaci a deserializaci dat v závislosti na HTTP hlavičce `Content-Type`, která určuje datový formát, ve kterém probíhá komunikace. V našem případě půjde vždy o JSON, ovšem lze použít i XML nebo YAML (YAML Ain't Markup Language).

Catalyst striktně následuje návrhový vzor MVC a vyžaduje poměrně fixní adresářovou strukturu (viz obrázek 4.7). Všechny moduly aplikace je třeba umístit do předem připravených adresářů.



Obrázek 4.7: Adresářová struktura API

Adresář `Model` obsahuje všechny modely – objekty pro přístup k datovým zdrojům. V tomto případě jde pouze o wrappery nad existujícími knihovnami.

- Model pro databázi – implementován přes knihovnu `DBIx::Class`.[1]
- Tiskový model – zajišťuje interface pro modul `Treex::Web::View`, který serializuje Treexové struktury do JSONu a umožňuje tak tisk stromů na frontendu.
- Resque model – zajišťuje přístup k frontě požadavků na zpracování Treexových scénářů (viz část 4.5).

Adresář `View` je prázdný, jelikož díky modulu `REST` se serializace objektů děje automaticky a není třeba implementovat zvláštní view.

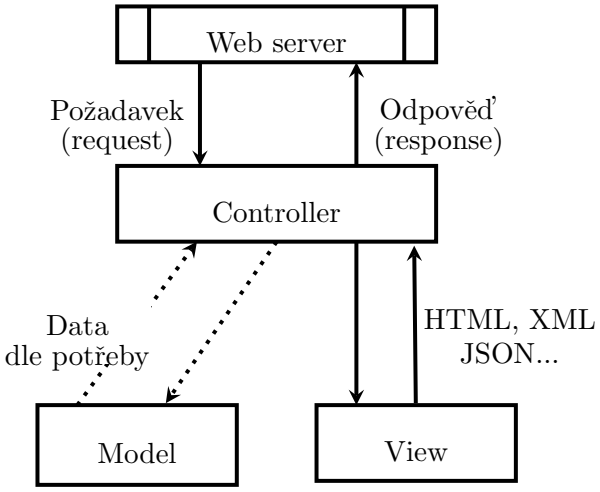
V adresáři `Controller` jsou controllery, což jsou v případě MVC moduly zodpovědné za obsluhu požadavku (jež je znázorněna na obrázku 4.8).

Klíčovou funkcí API je zpracovat požadavek na provedení scénáře a dodat uživateli výsledek. Princip přibližně znázorňuje schéma 4.9. Controller `Query` založí nový výsledek `Result`, a stejně tak vytvoří požadavek na zpracování scénáře (`Job`). V momentě zpracování scénáře je výsledek naplněn daty, a ta jsou pak dána k dispozici jako výstup. Další controllery operují především nad daty z databáze. Jedná se o operace typu vytvoř, přečti, uprav a vymaž.

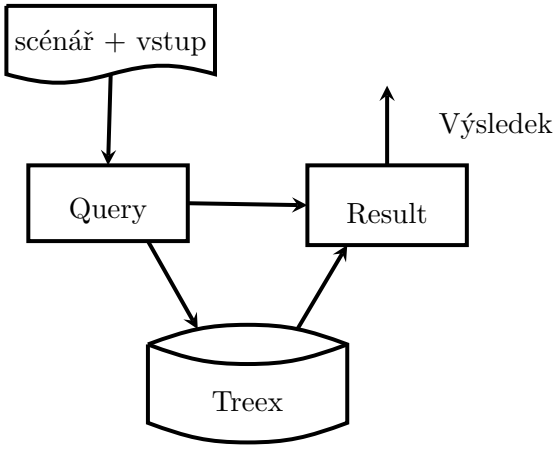
Auth je jednoduchý controller na obsluhu přihlášení uživatele.

Result zprostředkovává veškeré operace týkající se výsledku.

[1] `https://metacpan.org/module/DBIx::Class`



Obrázek 4.8: MVC v podání Catalystu - Controllery řídí obsluhu požadavků a stejně tak komunikaci mezi jednotlivými komponentami



Obrázek 4.9: Zpracování Treexového scénáře - Controller `Query` vytvoří prázdný výsledek, který je v momentě zpracování scénáře Treexem naplněn daty

Scenario zprostředkovává operace pro práci se scénáři.

Input obsluhuje alternativní možnosti vstupu – načtení dat z url nebo ze souboru.

Doc generuje dokumentaci celého API. Viz následující část 4.4.2.

Rozsah implementace vyjádřený v počtu řádek zdrojového kódu (tzv. SLOC (Source lines of code) metriku) lze nalézt v tabulce 4.3. .

	Kód	Komentáře
`Treex::Web`	2 800	820
Pomocné knihovny	920	200

Tabulka 4.3: Přehled počtu řádek zdrojového kódu API

4.4.2 Dokumentace

Kompletní dokumentace k API je k dipozici online na adrese:

`http://quest.ms.mff.cuni.cz/treex-web/api/v1/`

nebo jako příloha této práce.

Pro dokumentování webového API je použitý Swagger.[1] Ten je jednak nástrojem pro dokumentaci webových RESTful API (dokumentace se definuje na úrovni zdrojového kódu), jednak webový interface pro procházení a testování těchto API (viz adresa výše).

Jelikož Swagger není dostupný pro Perl, vytvořil jsem v rámci `Treex::Web`u svoji vlastní implementaci (modul `Swagger`), založenou na podobné knihovně pro `Node.js`.

4.5 Resque

Spouštět Treexové scénáře na serveru bez jakékoliv kontroly nejde. Ne že by to nebylo možné, ale takové řešení není v žádném případě stabilní. Scénář s větším jazykovým modelem může zabrat i více než jeden gigabajt RAM, což i při malém množství najednou puštěných scénářů spolehlivě odstaví server.

Je tedy nezbytné spouštět požadavky jeden po druhém a mít kontrolu nad jednotlivými procesy. Nabízí se řešení implementovat pro požadavky frontu, na jejímž konci by byl fixní počet procesů, které budou brát položky z fronty a postupně je zpracovávat (návrhový vzor producent / konzument).

Implementovat takovou frontu není úplně jednoduché (především proto, že veškeré operace na ní musí být atomické, aby nedocházelo k race conditions). Proto jsem se rozhodl sáhnout raději po hotovém řešení.

Jako první jsem použil Perlový balíček TheSchwartz, který implementuje jen naprosto základní a pro naše potřeby nedostačující druh fronty. Současná fronta je proto postavená na knihovně nazvané Resque.

[1] `http://swagger.wordnik.com`

Původní Resque je napsaná v Ruby, ale existuje i implementace pro Perl. Resque se skládá z dvou částí – klient a worker, přičemž klient zadává úkoly do fronty a worker je z fronty vybírá a zpracovává. Komunikace probíhá přes Redis[18] – distribuované úložiště klíč-hodnota (key-value), které zajišťuje atomicitu celé fronty.

Když Resque úkol zpracuje, uloží jen informaci, zda daný úkol proběhl v pořádku nebo skončil chybou. Pro fungování Treexu ale potřebujeme několik dalších funkcí:

- Průběžné informování o stavu zpracování
- Možnost úkoly z fronty odebrat nebo zrušit právě běžící úkol
- Timeout u každého úkolu, který by zabránil tomu, aby se zpracovával příliš dlouho

4.5.1 Status plugin

Naštěstí Resque umožňuje implementovat vlastní pluginy, kterými lze téměř kompletně změnit způsob, jakým fronta funguje. Součástí této práce je tedy plugin nazvaný jednoduše `Status` (Perlový modul `Resque::Status`).

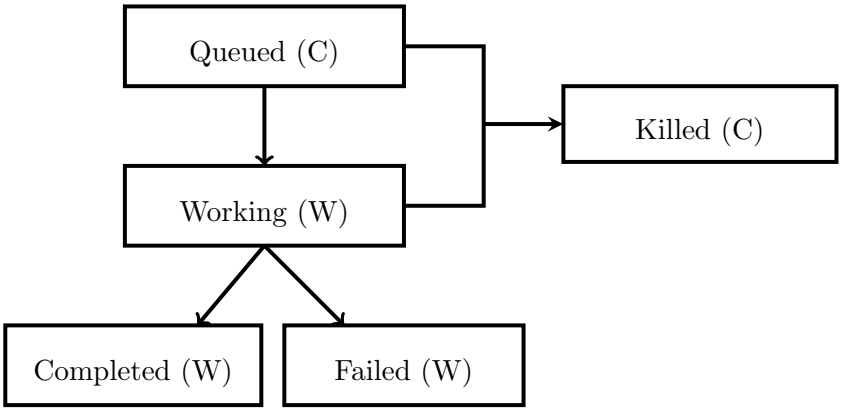
Funguje tak, že ke každému úkolu přidá do Redisu strukturu (viz tabulka 4.4), ve které je uložená informace o jeho stavu. Následně modifikuje workera, aby stav periodicky kontroloval a aktualizoval.

uuid	Unikátní identifikátor každého úkolu
name	Volitelný název úkolu (zatím se nepoužívá)
status	Aktuální stav z množiny `queued`, `working`, `completed`, `failed`, `killed`
message	Textová zpráva, například chybová hláška
time	Čas zadání úkolu

Tabulka 4.4: Struktura udržující informace o stavu úkolu

Průběh stavů je znázorněn na obrázku 4.10, přičemž úkol jde ze stavu `queued` do stavu `working`, když jej worker začne zpracovávat, a následně podle toho, jestli proběhl úspěšně skončí jako `completed` nebo `failed`.

Do stavu `killed` úkol přepne klient (vyžádá si tím zrušení úkolu). V případě, že se úkol právě zpracovává (worker při zpracovávání stav periodicky kontroluje), tak se zpracování ukončí. Je-li úkol teprve ve frontě, není hned automaticky odstraněn (Resque to totiž nepodporuje), ale když worker *vytáhne* z fronty úkol v takovémto stavu, ignoruje jej.



Obrázek 4.10: Stavy úkolu ve frontě - Průběh stavů u úkolů ve frontě. W – znamená, že stav může přiřadit worker, C – stav může přiřadit klient

4.5.2 Timeout

Resque funguje tak, že u každého úkolu je nutné uvést třídu, která implementuje metodu `perform`, jež se zavolá pro provedení úkolu. Třída spouštějící Treex je v balíčku `Treex::Web::Job::Process` a pro spouštění Treexu používá Perlový modul IPC::Run.[1] Tento modul slouží k spouštění podprocesů z Perlu a mimo jiné umí nastavit timeout.

V současné době je timeout udělaný ad hoc – není konfigurovatelný (nastavený pevně na 3 minuty) a slouží především jen jako ochrana proti pomalým scénářům, které by jinak blokovaly frontu. Takové řešení není ideální a do budoucna bude třeba implementovat možnost, aby si každý uživatel mohl timeout (v nějakém intervalu) nastavit sám, díky čemuž by mohlo existovat více front – např. fronta na potenciálně pomalé a rychlé scénáře.

4.6 Testování

Javascriptová (Angular frontend) i Perlová část mají vlastní testy. Přestože obě části mají excelentní nástroje pro testování a podporují postupy vývoje, jako je TDD (Test-driven development), při vývoji této aplikace hrály testy spíše podpůrnou roli. Testy tedy existují především pro části, které byly v průběhu vývoje nějakým způsobem kritické a testovat je ručně bylo velmi pracné.

[1] `https://metacpan.org/module/IPC::Run`

Perl

Standard testování v Perlu je rodina modulů `Test::More`, které jsou svázané dohromady protokolem TAP (Test Anything Protocol). Testy pro `Treex::Web` nejsou výjimkou a k jejich spuštění lze použít `Test::Harness`.

Stačí tedy v adresáři aplikace spustit příkaz `prove` (popř. `./bin/test.pl`, pokud není `Test::Harness` nainstalovaný) a začnou se provádět testy, které jsou v adresáři `t/*.t`. Testování je velice povrchní, testuje se především, zda lze celou aplikaci v pořádku načíst.

Angular

Celé prostředí Angularu je navržené tak, aby každá komponenta šla jednoduše otestovat. Například pro potřeby testování Angular nikde nepoužívá globální `window` objekt prohlížeče, místo toho definuje vlastní `$window` service – čímž umožňuje psát testy, které lze spustit i mimo prohlížeč.

Pro naše potřeby se testy spouští v prohlížeči. K spouštění se používá Karma[1] (*spouštěč* testů, který spustí prohlížeč) a samotné testy využívají framework Jasmine.[2] Testy jsou uložené v adresáři `frontend/test` a opět se jedná o velmi základní testy, podrobněji je otestované jen spouštění Treexového scénáře.

Testy lze spustit příkazem: `grunt test`

[1] `http://karma-runner.github.io/`
[2] `http://pivotal.github.io/jasmine/`

5

Přehled NLP platforem

Stručný přehled některých dalších NLP frameworků. Shrnutí lze nalézt v tabulce 5.1, tučně zvýrazněné frameworky mají webové rozhraní a je jim věnován větší prostor.

Tabulka 5.1: Přehled NLP platforem

	Vývoj od	Jazyk	Licence	Instituce
Apertium	2004	C++, Java	GPL	Universitat d'Alacant
Argo	2012	Java	ne	University of Manchester
Enrycher	2010	Java	ne	JSI Ljubljana
ETAP-3	$\sim$ 1980	C/C++	ne	Ruská akademie věd
GATE	1995	Java	LGPL	University of Sheffield
NLTK	2005	Python	Apache	Open-source komunita
OpenNLP	2010	Java	Apache	ASF[a]
UIMA	2006	C++, Java	Apache	ASF
WebLicht	2008	Java	ne	Universität Tübingen

[a] Apache Software Foundation

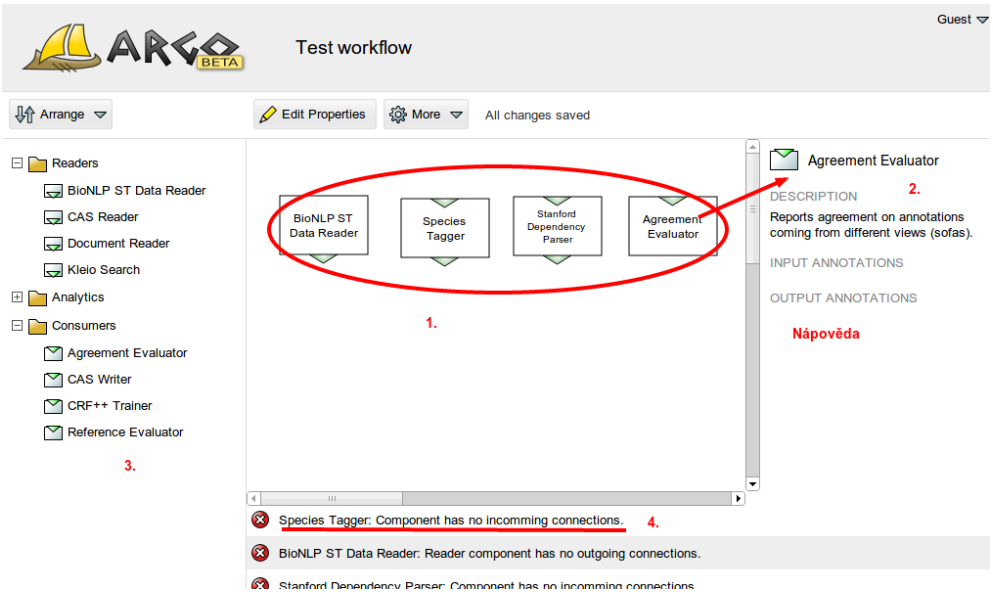
Apertinum

Open-source platforma zaměřená především na strojový překlad příbuzných jazyků. Projekt je stále ve vývoji a má desítky vývojářů.[1][35]

[1] `http://www.apertium.org/`

Argo

Webová aplikace[1] zaměřená na především na analýzu textu a tzv. text mining – tedy *dolování* informací z textu. Vznikla především za účelem moderování biomedicínské literatury.[36] Uživatel si může sestavit vlastní *proces* (workflow) velmi podobně jako Treexový scénář a následně jej testovat na datech. Oproti `Treex::Webu` má několik



Obrázek 5.1: Vytváření worflow v Argo

užitečných vlastností (obrázek 5.1):

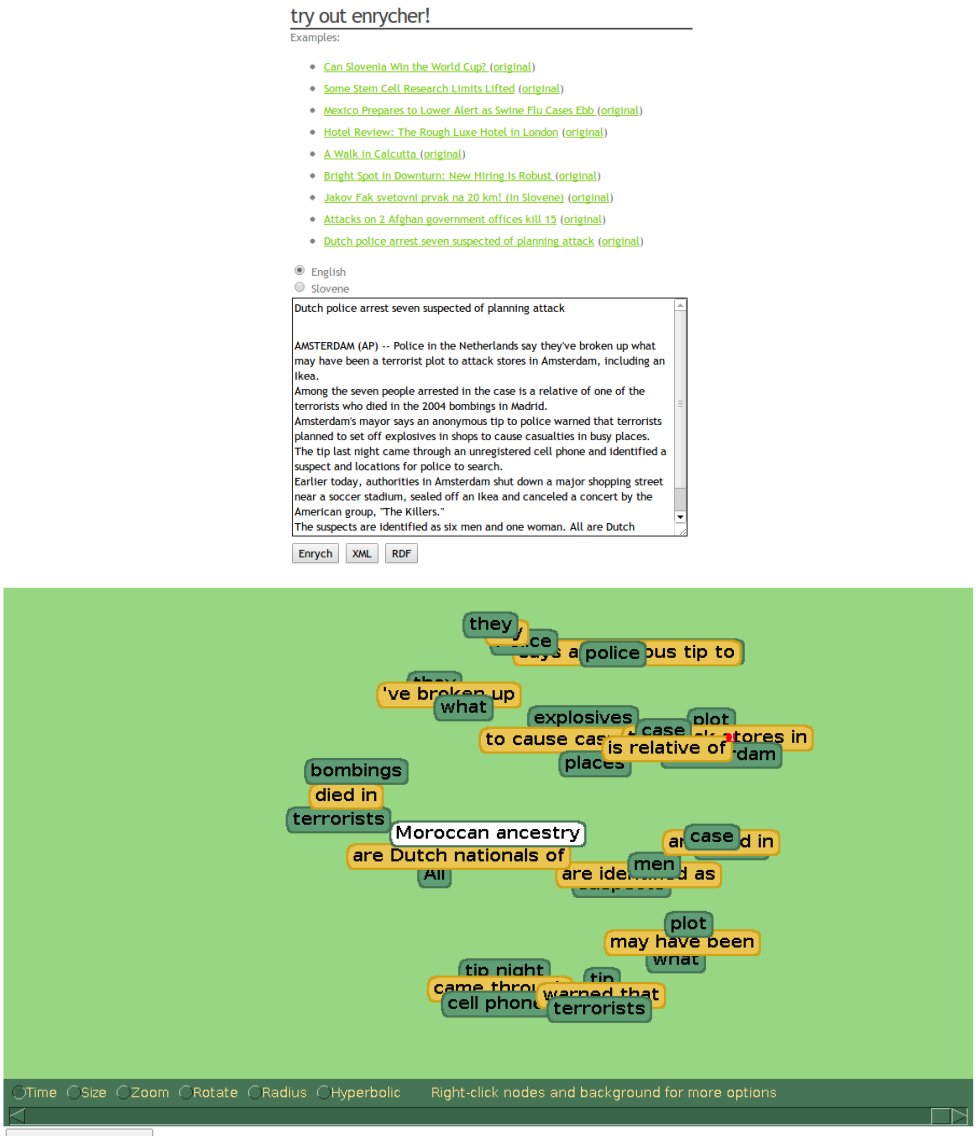
1. Jednotlivé komponenty (ekvivalent bloků v Treexu) jsou reprezentovány graficky
2. Každá komponenta má nápovědu
3. Přehledný seznam komponent
4. Průběžně aktualizovaný seznam nedostatků (chyb) ve workflow

Enrycher

Systém postavený na SOA (Service Oriented Architecture). Nejedná se o framework v pravém slova smyslu. Enrycher se skládá z deseti služeb a nabízí především roz-

[1] `http://argo.nactem.ac.uk/`

poznávání tématu (topic detection) a extrakci pojmenovaných entit (named entity extraction) z anglického nebo slovinského textu.[37] Demo je k dispozici na adrese: `http://enrycher.ijs.si/`.



Obrázek 5.2: Enrycher – demo

Demo má jednoduchý interface a jeho největší předností je sémantický graf reprezentující jednotlivé fragmenty informací vyextrahovaných z textu.

ETAP-3

Proprietární nástroj zaměřený na překlad z angličtiny do ruštiny a naopak. Online je dostupné jen malé demo - `http://cl.iitp.ru/etap3`.[38]

GATE

Nejspíše jeden z nejrozšířenějších NLP frameworků integrující v sobě i grafický interface.[39] Je vyvíjen od roku 1995 univerzitou v Sheffieldu a stojí za ním početná komunita vývojářů.

Na adrese `https://gatecloud.net/` nabízí univerzita webové rozhraní, jedná se ale o placenou službu a není žádné veřejně dostupné demo.

NLTK

Značí *Natural Language Toolkit*, framework napsaný v Pythonu. Nabízí interface k několika korpusům a spoustu nástrojů jako tokenizer, tagger, parser... Je zaměřený především na angličtinu.[40]

OpenNLP

Open-source projekt sdružující několik NLP nástrojů, které mají za úkol sloužit jako stavební kameny pro složitější aplikace.[1]

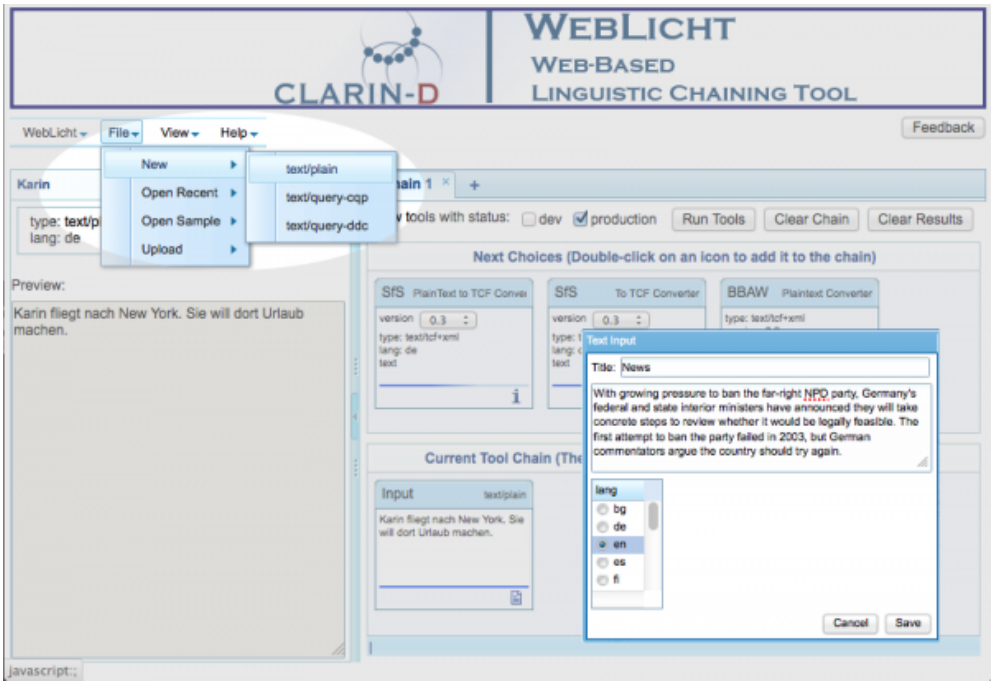
UIMA

Značí *Unstructured Information Management applications* – balík software pro analýzu velkých objemů nestrukturovaných dat a hledání informací relevantních pro koncového uživatele.[2]

[1] `http://opennlp.apache.org/`
[2] `http://uima.apache.org/`

WebLicht

Online web servis určený především pro anotaci německého textu. Bohužel web není veřejný, ale zdá se, že používá velmi podobný přístup k řešení lingvistických úloh, jako má Treex. Uživatel si sestavuje svůj vlastní řetězec procesů, které chce na text aplikovat a výsledek si může prohlédnout jako tabulku nebo vizualizovaný strom.[41, 42]



Obrázek 5.3: Weblicht – webový interface, převzato z [42]

6

Další plány

S touto prací vývoj `Treex::Web`u nekončí, spíše se jedná o začátek. Následujících několik bodů představuje vizi dalšího vývoje:

- Při každém spuštění Treexu se načítá celé prostředí znovu, a to je velice neefektivní. Mezi jeden z hlavních úkolů pro usnadnění práce s programem patří *přednačtení* nejčastěji používaných bloků do paměti. Podle odhadu by se mohla doba provádění scénářů snížit jen na několik sekund (místo až několika minut, jak je tomu teď).

- Jelikož Treex nebyl navržen pro provoz na serveru, není nijak zvlášť zabezpečený proti chybám (např. v blocích), které by mohly umožnit útočníkovi přístup na server. Bylo by vhodné Treexový proces tzv. sandboxovat, tedy spouštět v prostředí odděleném od ostatních procesů a souborů, se kterými přímo nepracuje.

- Mnohem naléhavější bezpečnostní riziko ovšem představuje absence limitů a kvót. V současné době totiž není žádný limit na množství scénářů, které může mít jeden uživatel najednou ve frontě. Lze tak napsat jednoduchý script, který zadá automaticky velké množství scénářů a tím efektivně zablokuje přístup ke službě pro ostatní uživatele. Tento problém je jeden z prvních, které bude opravovat další verze `Treex::Web`u.

- Editor scénářů neumí kromě zvýrazňování syntaxe nic dalšího. V některé z dalších verzí `Treex::Web`u by měla přibýt kontrola syntaxe, chytré automatické doplňování (autocomplete) a v neposlední řadě nápověda ke každému bloku.

- Prohlížení vět a stromů v detailu výsledku není zcela ideální. Především proto, že uživatel nemá k dispozici celý seznam vět, ale musí v nich jednotlivě listovat. Navíc prohlížení velkých stromů je nepohodlné, v budoucnu musí přibýt možnost stromy oddalovat a přibližovat (zoom).

- Použití TrEdu jako nástroje pro vizualizaci není vždycky úplně jednoduché. Hezká funkce by byla, kdyby uživatel mohl do webového rozhraní nahrát své vlastní soubory a využit `Treex::Web` jen jako nástroj pro jejich vizualizaci.

- Přestože podpora pro přihlašování a uživatelské účty existuje, bude třeba implementovat přihlašování buď přes Centrální autentizační službu UK (CAS), nebo přes Shibboleth.[15]

- Jedna z opravdu pokročilých funkcí, jejíž definitivní podoba zatím není úplně jasná, je možnost definovat scénáře a následně je měnit do podoby webové REST služby. Princip částečně souvisí s prvním bodem tohoto seznamu, totiž, že scénář definovaný jako služba by byl permanentně načtený v paměti a připravený přijímat požadavky. Zpřístupnit tak komplexní aplikace online (které už jsou v Treexu) by bylo následně velmi jednoduché.

7

Závěr

Výsledkem této práce je webová aplikace s názvem `Treex::Web`, která zastřešuje vysoce modulární NLP framework Treex.

Hlavním cílem této práce bylo spojit Treex a webové technologie tak, aby výsledná aplikace byla stabilní a snadno ovladatelná. Myslím, že uživatelské rozhraní (viz kapitola 3) tento cíl splnilo. Pro Treex tato práce znamená částečné řešení některých jeho palčivých problémů, jako je absence GUI nebo nutnost komplikované instalace. Především potenciální noví uživatelé jistě ocení, že si mohou Treex vyzkoušet online.

Jak je popsáno v kapitole 4, technologické základy pro `Treex::Web` jsou velice dobře položené a umožní do budoucna dále vylepšovat a postupně implementovat další funkce (popsané v předchozí kapitole 6). Během vývoje bylo také opraveno několik chyb v Perlové implementaci knihovny Resque, které její autor vzápětí zařadil do dalšího vydání a jsou tak k dispozici pro další uživatele na CPANu.

Osobně si na této práci cením možnosti zužitkovat své zkušenosti s vývojem webových aplikací a naučit se hodně nového.

Literatura

[1] David Mareček, Martin Popel, and Zdeněk Žabokrtský. **Maximum Entropy Translation Model in Dependency-Based MT Framework**. In *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR*, pages 201–201, Uppsala, Sweden, 2010. Association for Computational Linguistics. 2

[2] Michal Novák and Zdeněk Žabokrtský. **Resolving Noun Phrase Coreference in Czech**. *Lecture Notes in Computer Science*, **7099**:24–34, 2011. 2

[3] Martin Popel and Zdeněk Žabokrtský. **TectoMT: Modular NLP Framework**. In Hrafn Loftsson, Eiríkur Rögnvaldsson, and Sigrún Helgadóttir, editors, *Advances in Natural Language Processing*, **6233** of *Lecture Notes in Computer Science*, pages 293–304. Springer Berlin Heidelberg, 2010. Dostupné na: `http://dx.doi.org/10.1007/978-3-642-14770-8_33`. 5, 7

[4] Daniel Zeman, David Mareček, Martin Popel, Loganathan Ramasamy, Jan Štěpánek, Zdeněk Žabokrtský, and Jan Hajič. **HamleDT: To Parse or Not to Parse?** In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Mehmet Uğur Doğan, Bente Maegaard, Joseph Mariani, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey, may 2012. European Language Resources Association (ELRA). 5

[5] Drahomíra Spoustová, Jan Hajič, Jan Votrubec, Pavel Krbec, and Pavel Květoň. **The Best of Two Worlds: Cooperation of Statistical and**

Rule-Based Taggers for Czech. In *Proceedings of the Workshop on Balto-Slavonic Natural Language Processing, ACL 2007*, pages 67–74, Praha, 2007. 6

[6] Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. **Non-projective dependency parsing using spanning tree algorithms**. In *Proceedings of HLT / EMNLP*, pages 523–530, Vancouver, Canada, 2005. 6

[7] Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gulsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. **MaltParser: A language-independent system for data-driven dependency parsing**. *Natural Language Engineering*, **13**(2):95–135, 2007. 6

[8] Petr Sgall. *Generativní popis jazyka a česká deklinace*. Academia, Prague, 1967. 6

[9] Jan Hajič et al. **Prague Dependency Treebank 2.0**. CD-ROM, Linguistic Data Consortium, LDC Catalog No.: LDC2006T01, Philadelphia, 2006. 6

[10] Henry Van Styn. **Moose**. *Linux Journal*, **2011**(209):8, 2011. 8

[11] Petr Pajas and Jan Štěpánek. **XML-based representation of multi-layered annotation in the PDT 2.0**. In *Proceedings of the LREC Workshop on Merging and Layering Linguistic Information (LREC 2006)*, pages 40–47, 2006. 8

[12] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. **Building a large annotated corpus of English: The Penn Treebank**. *Computational linguistics*, **19**(2):313–330, 1993. 8

[13] Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, et al. **The CoNLL-2009 shared task: Syntactic and semantic dependencies in multiple languages**. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, pages 1–18. Association for Computational Linguistics, 2009. 8, 11

[14] Petr Pajas and Peter Fabián. **Tree Editor TrEd**, 2003. Dostupné na: http://ufal.mff.cuni.cz/tred/. 8

[15] Scott Cantor and TOM SCAVO. **Shibboleth architecture**. *Protocols and Profiles*, **10**, 2005. 16, 46

[16] Niels Provos. **libevent – an event notification library**, 2003. Dostupné na: http://libevent.org. 20

[17] The IEEE and The Open Group. **The Open Group Base Specifications Issue 7**, 2013. Specifikace funkce fork. Dostupné na: http://pubs.opengroup.org/onlinepubs/9699919799/functions/fork.html. 20

[18] Salvatore Sanfilippo and Pieter Noordhuis. **Redis**, 2010. Dostupné na: http://redis.io. 21, 36

[19] D. Flanagan. *JavaScript: The Definitive Guide*. Definitive Guide Series. O'Reilly Media, Incorporated, 2011. 22

[20] Igor Minar, Miško Hevery, and Vojta Jína. **AngularJS – Dokumentace**, 2005. Dostupné na: http://angularjs.org/. 22

[21] Addy Osmani. *Developing Backbone.js Applications*. O'Reilly, 2013. 23

[22] Marc Bodmer. *Instant Ember.Js Application Development How-To*. Packt Publishing, Limited, 2013. 23

[23] Joel Rosen. **Using AngularJS at Localytics**. Online, 2013. Dostupné na: http://www.localytics.com/blog/2013/angularjs-at-localytics/. 24

[24] Reuven M Lerner. **At the forge: Twitter bootstrap**. *Linux Journal*, **2012**(218):6, 2012. 25

[25] S. Lidie and N. Walsh. *Mastering Perl/Tk: Graphical User Interfaces in Perl*. O'Reilly Media, 2009. 27

[26] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. **D3: Data-Driven Documents**. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2011. Dostupné na: http://vis.stanford.edu/papers/d3. 28

[27] M. Dewar. *Getting Started With D3*. Number 3 in Oreilly and Associate Series. Oreilly & Associates Incorporated, 2012. 28

[28] John Raasch. *JavaScript Programming: Pushing the Limits*. Pushing the Limits. Wiley, 2013. 30

[29] Roy Thomas Fielding and Gail Kaiser. **The Apache HTTP server project**. *Internet Computing, IEEE*, **1**(4):88–90, 1997. 30

[30] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, 2000. 31

[31] Roy T. Fielding and Richard N. Taylor. **Principled design of the modern Web architecture**. *ACM Trans. Internet Technol.*, **2**(2):115–150, May 2002. Dostupné na: `http://doi.acm.org/10.1145/514183.514185`. 31

[32] Subbu Allamaraju. *RESTful Web Services Cookbook*. O'Reilly Media, 2010. 31

[33] Jan Štepánek and Petr Pajas. **Querying diverse treebanks in a uniform way**. In *Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC 2010)*, pages 1828–1835, 2010. 32

[34] Ian Fette and Alexey Melnikov. **The websocket protocol**. 2011. Dostupné na: `http://tools.ietf.org/html/rfc6455`. 32

[35] Mikel L. Forcada, Mireia Ginestí-Rosell, Jacob Nordfalk, Jim O'Regan, Sergio Ortiz-Rojas, Juan Antonio Pérez-Ortiz, Felipe Sánchez-Martínez, Gema Ramírez-Sánchez, and Francis M. Tyers. **Apertium: a free/open-source platform for rule-based machine translation**. *Machine Translation*, **25**(2):127–144, 2011. Dostupné na: `http://dx.doi.org/10.1007/s10590-011-9090-0`. 39

[36] R. Rak, A. Rowley, W.J. Black, and S. Ananiadou. **Argo: an integrative, interactive, text mining-based workbench supporting curation**. *Database: The Journal of Biological Databases and Curation*, **2012**, 2012. Dostupné na: `http://database.oxfordjournals.org/content/2012/bas010.full?keytype=ref&ijkey=i0zkIYyxcsdxhfN`. 40

[37] Tadej Štajner, Delia Rusu, Lorand Dali, Blaž Fortuna, Dunja Mladenić, and Marko Grobelnik. **A service oriented framework for natural language text enrichment**. *Informatica (Ljublj.)*, **34**(3):307–313, 2010. 41

[38] Igor Boguslavsky, Leonid Iomdin, and Victor Sizov. **Multilinguality in ETAP-3: reuse of lexical resources**. In *Proceedings of the Workshop on Multilingual Linguistic Ressources*, pages 7–14. Association for Computational Linguistics, 2004. 42

[39] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, Valentin Tablan, Niraj Aswani, Ian Roberts, Genevieve Gorrell, Adam Funk, Angus Roberts, Danica Damljanovic, Thomas Heitz, Mark A. Greenwood, Horacio Saggion, Johann Petrak, Yaoyong Li, and Wim Peters. *Text Processing with GATE (Version 6)*. 2011. Dostupné na: `http://tinyurl.com/gatebook`. 42

[40] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python*. O'reilly, 2009. 42

[41] Erhard W. Hinrichs, Marie Hinrichs, and Thomas Zastrow. **WebLicht: Web-Based LRT Services for German**. In *Proceedings of the ACL 2010 System Demonstrations*, pages 25–29, 2010. Dostupné na: `http://www.aclweb.org/anthology/P10-4005`. 43

[42] CLARIN-D/SfS-Uni. Tübingen. **WebLicht: Web-Based Linguistic Chaining Tool**. Online, 2012. Dostupné na: `https://weblicht.sfs.uni-tuebingen.de/`. 43

Seznam obrázků

Seznam tabulek

Vysvětlivky

API
V kontextu vývoje webových aplikací je API typicky nějaká definovaná množina HTTP požadavků spolu s definovanou strukturou odpovědí. K přenosu dat se většinou používá XML nebo JSON. 20, 30–32, 34, 35

CPAN
The Comprehensive Perl Archive Network (CPAN) je archiv a distribuční síť pro Perlové moduly s webovým interfacem na adrese: `http://www.cpan.org/`. Jako `cpan` se také označuje program, který funguje jako instalátor a manager pro Perlové moduly. 2

CSS
Cascading Style Sheets. 19

DOM
Document Object Model. 23, 25, 28, 29

Express
Framework na postavený na node.js (Javascript). Jedná se minimalistický (270 SLOC) a přesto velmi flexibilní framework. 31

GUI
Graphical user interface. 47

HTML
HyperText Markup Language. 19–25

HTTP
Hypertext Transfer Protocol. 31, 59

Javascript
Javascript je interpretovaný programovací jazyk. Byl původně vyvinut společností Netscape jako součást stejnojmenného webového prohlížeče a slouží jako skriptovací jazyk pro HTML dokumenty. Dnes je jeho použití mnohem širší a nachází uplatnění i mimo webové prohlížeče. 19, 20, 60

Vysvětlivky

JSON
Javascript Object Notation. 21, 27, 32, 59

MVC
Model View Controller. 19, 32, 33

NLP
Natural Language Processing – zpracování přirozeného jazyka. 1, 3, 5, 39, 47

`Node.js`
`Node.js` je softwarový systém navržený pro psaní webových aplikací postavený na V8 Javascript engine od společnosti Google. 30, 35

`npm`
Node Package Manager. 30

PML
Prague Markup Language. 8, 60

PML-TQ
PML Tree Query je vyhledávací nástroj pro nejrůznější druhy lingvisticky anotovaných treebanků. Pracuje nativně s treebanky v PML. 27, 32

REST
Representational State Transfer. 31, 46

Ruby on Rails
Framework pro webové aplikace napsaný v Ruby často označovaný jen jako Rails je ve vývoji od roku 2003 a stojí za ním společnost 37signal. Odhaduje se, že k červenci 2013 běželo na Ruby on Rails téměř 200 tisíc webových aplikací (`http://trends.builtwith.com/framework/Ruby-on-Rails`). 31

SLOC
Source lines of code. 34, 59

SVG
Scalable Vector Graphics. 27

SVN
Apache Subversion nebo zkráceně SVN je systém pro verzování zdrojových kódů. Oficiální web lze nálézt na adrese: `http://subversion.apache.org/`. 2

TAP
Test Anything Protocol. 37

TDD
Test-driven development. 37

UI
User Interface. 22

URI
Uniform Resource Identifier. 31

URL
Uniform Resource Locator. 30

XML
Extensible Markup Language. 8, 32, 59

YAML
YAML Ain't Markup Language. 32