VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

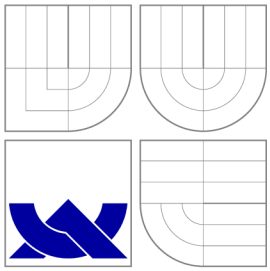
ZPRACOVÁNÍ DOKUMENTŮ SPECIFIKOVANÝCH JAZYKEM TEXY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

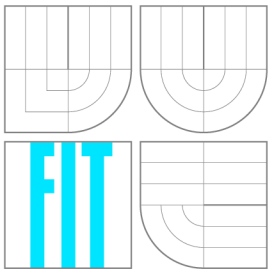
AUTOR PRÁCE
AUTHOR

MIROSLAV BODEČEK

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ZPRACOVÁNÍ DOKUMENTŮ SPECIFIKOVANÝCH JAZYKEM TEXY

PROCESSING OF TEXY LANGUAGE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MIROSLAV BODEČEK
AUTHOR

VEDOUCÍ PRÁCE — Ing. RADEK KOČÍ, Ph.D.
SUPERVISOR

BRNO 2008

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2007/2008

Zadání bakalářské práce

Řešitel: **Bodeček Miroslav**
Obor: Informační technologie
Téma: **Zpracování dokumentů specifikovaných jazykem Texy**
Kategorie: Překladače

Pokyny:

1. Seznamte se se specifikací jazyka Texy.
2. Prostudujte problematiku gramatik a lexikálních a syntaktických analyzátorů.
3. Vytvořte gramatiku jazyka Texy a navrhněte lexikální a syntaktický analyzátor.
4. Navrhněte a implementujte knihovnu tříd v Javě pro zpracování dokumentů specifikovaných jazykem Texy. Zaměřte se na možnosti transformace dokumentů Texy do jiných formátů (HTML apod.) a parametrizaci této transformace.
5. Implementujte jednoduchou aplikaci demonstrující použití navržené knihovny na reálných dokumentech. Proveďte srovnání vaší implementace s existujícími variantami.

Literatura:

- Dle pokynů vedoucího práce.

Při obhajobě semestrální části projektu je požadováno:

- První tři body zadání.

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

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

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

Vedoucí: **Kočí Radek, Ing., Ph.D.**, UITS FIT VUT
Datum zadání: 1. listopadu 2007
Datum odevzdání: 14. května 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Miroslav Bodeček**
Id studenta: 79132
Bytem: Glinkova 4, 623 00 Brno
Narozen: 24. 03. 1986, Olomouc
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
bakalářská práce

Název VŠKP: Zpracování dokumentů specifikovaných jazykem Texy
Vedoucí/školitel VŠKP: Kočí Radek, Ing., Ph.D.
Ústav: Ústav inteligentních systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:

tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☐ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

Abstrakt

Současné prostředky pro publikaci na internetu řeší problém, jak uživateli zjednodušit zadávání dokumentů, které jsou zobrazovány na webových stránkách. Tento problém v minulosti vyřešila knihovna Texy, který je postavena na platformě PHP. Práce spočívá v reimplementaci jazyka Texy pro prostředí platformy Java. V této zprávě je popsán způsob implementace překladače, vytvoření modelu jazyka a formální gramatiky. Práce se zabývá i možnostmi dalšího rozvoje knihovny, zejména pak výstupu do jiných formátů, než je HTML.

Klíčová slova

Texy, jazyk, gramatika, překladač, ANTLR, LL, Java

Abstract

One of the problems encountered by today's publishing systems for the web is making it easy for authors to enter content. This problem has been solved in the past by a PHP library called Texy. The goal of this work is to create a new implementation of the Texy language, which will be based on the Java platform. This document describes implementation details of the compiler, the model of the language and the formal grammar used. It also explores potential further development of the library, namely generating output formats other than HTML.

Keywords

Texy, language, grammar, compiler, ANTLR, LL, Java

Citace

Miroslav Bodeček: Zpracování dokumentů specifikovaných jazykem Texy, bakalářská práce, Brno, FIT VUT v Brně, 2008

Zpracování dokumentů specifikovaných jazykem Texy

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Radka Kočího, PhD. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Miroslav Bodeček
12. května 2008

Poděkování

Děkuji Ing. Radkovi Kočímu, PhD. za jeho vstřícnost a za cenné rady a podněty, které mi pomohly k vypracování této práce. Za vstřícnost a trpělivost také děkuji lidem ze společnosti oXy Online s. r. o., která tuto práci zadala.

© Miroslav Bodeček, 2008.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

Kapitola 1

Úvod

V této kapitole jsou uvedeny základní informace o práci. Obsahuje popis problematiky zadávání dokumentů zobrazovaných na webových stránkách. Pokusím se zde nastínit motivace k práci a také zdůvodnit volby implementačních prostředků.

1.1 Zpracování dokumentů pro webové stránky

Jedním z problému které mají dnešní systémy pro publikování na internetu (a nejen ony) je způsob zadávání textu dokumentů. Zatímco při zobrazování je nutné, aby byl dokument ve formátu *HTML*,[1] při zadání je pro běžného uživatele tento formát nanejvýš nevhodný.

Dnešní internetové publikační a redakční systémy nabízejí alespoň jeden ze tří způsobů zadávání dokumentů (většina umožňuje kombinaci těchto tří):

1. Dokumenty je možné zadávat přímo ve formátu **HTML**. Tento způsob s sebou nese celou řadu nevýhod. Jednak klade vysoké nároky na technickou zdatnost uživatelů. Zápis HTML také není názorný, uživatel nemá představu, jak bude dokument vypadat, dokud ho nevidí zobrazený v prohlížeči. V neposlední řadě HTML dává uživateli do rukou až příliš silný nástroj. Internetová média mají často vysoce sofistikovaný grafický design, jehož pravidla autoři dokumentů mohou snadno porušit, pokud je jim umožněno měnit přímo zobrazovaný HTML kód.

   Výhodou zadání ve formátu HTML jsou malé nároky při implementaci, nicméně ani ty nejsou nulové. Ze vstupní dokument je vždy nutné odstranit „nebezpečné“ HTML značky, je nutné jej upravit, aby byl *well-formed* a tak podobně.

2. Většina systémů nabízí více či méně pokročilou verzi ***WYSIWYG***[2] editoru. Tyto editory odstraňují dva velké problémy, které s sebou nese přímé zadávání v HTML – jsou uživatelsky mnohem přívětivější, a při jejich použití je názorně vidět, jak bude výsledný dokument vypadat.

   Nicméně i WYSIWYG editory mají své nevýhody. První z nich je, že jimi produkovaný HTML kód používá nevhodné či přímo nestandardní konstrukty jazyka HTML. Nejčastěji uváděným příkladem je použití značky `<font>`, která je v nových verzích specifikace HTML k odstranění (*deprecated*),[3] namísto použití třídy kaskádového stylu. Další nevýhoda je společná bodem 1 – výsledný dokument často porušuje zásady dané grafickým stylem celé stránky.

[1] *HyperText Markup Language*

[2] *What You See Is What You Get* – tedy způsob práce, který nabízí běžné dnešní textové procesory

3. Některé systémy umožňují zadávat dokumenty ve formátu specifickém pro zvláštní druh internetového média – tzv. ***wiki*** stránky.[7] Existují různé formáty,[3] ale většinou mají tyto společné rysy:

   (a) volně kombinují prvky, které formátují obsah, s obsahem samotným

   (b) uvolněná syntaxe – gramatika většinou neobsahuje žádné omezení, které by způsobilo, že libovolný dokument nebude akceptován. Případné chyby v syntaxi se ignorují. Toto je problematické při formalizaci gramatiky, protože daná gramatika bude na mnoha místech nejednoznačná (za předpokladu, že chceme, aby patřila do množiny bezkontextových gramatik). Toto je blíže popsáno v části 3.3.1.

   (c) abstrahují uživatele od HTML – tyto jazyky většinou neumožňují přímý kód v HTML zadávat vůbec, nebo pouze pomocí speciálního konstruktu

   Praktickou nevýhodou těchto jazyků jsou opět nároky na uživatele co do znalosti syntaxe, nicméně toto je mnohem menší problém než u přímého zadávání HTML kódu. Abstrakce od HTML je však zároveň i nevýhodou: komplikují se tak speciální případy, kdy je v dokumentu opravdu nutné použít čisté HTML.[4]

1.2 Jazyk Texy a jeho současná implementace

Tyto problémy se snaží řešit jazyk Texy,[9] který je dílem českého programátora Davida Grudla. Do jisté míry se podobá různým wiki syntaxím popsaným v 3. Hlavní odlišnost je v tom, že se snaží uživatele samoúčelně neodstiňovat od HTML, umožňuje přímý zápis HTML a také značně usnadňuje připojení dokumentu k existujícím kaskádovým stylům. Obsahuje poměrně bohatou sadu vlastností pro zápis nejběžnějších typografických konstruktů, jako jsou zvýraznění textu, seznamy, tabulky apod. Jeho další velká výhoda je, že Texy je v ČR poměrně rozšířený. To také může napovídat, že je to prostředek prakticky použitelný.

Texy bohužel chybí formální specifikace nebo gramatika. Originální implementace převaděče Texy do HTML je postavena na jazyku PHP. To prakticky znemožňuje použití v jiných programovacích jazycích. Originální implementace také neumožňuje transformaci na jiný formát, než je HTML, což je poměrně omezující. Z těchto důvodu bude vhodné sestrojit druhou implementaci jazyka postavenou na platformě Java. Tato implementace by měla být co nejvíce kompatibilní se současnou implementací, i když je zřejmé, že úplné kompatibility pro časová omezení dosáhnout nelze. Tato nová implementace je pracovně nazvána *JTexy*.

1.2.1 Rozšíření o parametrizaci výstupu

U dokumentů vystavovaných na internetu bývá nutné, aby do nich mohl publikační systém vložit určitá data, která nejsou známy v době psaní dokumentu, nebo která se mohou dynamicky měnit. Jedná se například o aktuální datum, datum publikování dokumentu, jméno zodpovědného editora, a tak podobně. Toto může být umožněno zápisem *rezervovaných míst*, do kterých bude vložena v době generování výstupu příslušná hodnota. Tato funkce bude v JTexy implementována, i když ji originální Texy nepodporuje.

[3] Jednou z nejznámějších syntaxí je *MediaWiki*, kterou používá server *Wikipedia.org*

[4] Například pro vložení externího objektu, jako je Flash.

1.3 Volba prostředků pro reimplementaci

Reimplementace Texy musí být použitelná v komerčním produktu firmy zadávající tuto bakalářskou práci. Tento produkt je postavený na platformě Java, takže logická volba je i další práci postavit na této platformě.

I když nejnovější verze platformy je 1.6, pro práci byla zvolena verze 1.5, a to z důvodu zpětné kompatibility. V době psaní této práce ještě není verze 1.6 příliš rozšířená. Naopak použít starší verzi 1.4 nemá praktický smysl, protože tato verze přestane být firmou Sun Microsystems oficiálně podporována na podzim tohoto roku (2008).[4]

Kromě samotné platformy je na projektu použito několik dalších nástrojů:

ANTLR je nástroj, který automatizuje převod formální gramatiky na lexikální a syntaktický analyzátor.[15] Vstupem tohoto nástroje je bezkontextová gramatika popsaná v jazyce podobném EBNF.[5][14, 88 s.] Výstupem je pak generovaný zdrojový kód lexikálního a syntaktického analyzátoru, který je možné přeložit standardní překladačem jazyka Java.

Tento nástroj byl zvolen pro svoji bohatou funkčnost v porovnání s podobnými nástroji, jako je *JavaCC*. Zejména se jedná o schopnost generovat $LL(*)$ lexikální analyzátor (to je podrobněji vysvětleno v části 3.2). Také integrované prostředí *ANTLRWorks*, které k němu patří, je významnou výhodou.

Apache Maven je nástroj pro správu sestavení (*build management tool*), tedy překlad, vytvoření výsledné knihovny, generování dokumentace a tak podobně.[6] Spolu s nástrojem *Apache Ant* se na tomto poli jedná o *de facto* standard. Na rozdíl od Antu má ale množství zásuvných modulů, které generují nejrůznější informační stránky o projektu, například výsledek statické analýzy kódu. Navíc také spravuje závislosti a automaticky je stahuje z internetu.

JUnit je podpůrná knihovna pro psaní *automatických testů*.[5] Automatické testy jsou blíže rozepsány v části 4.1.2. JUnit je opět standardní knihovnou s podporou v mnoha dalších systémech (integrovaná vývojová prostředí, nástroje pro analýzu pokrytí kódu testy apod.).

[5] Extended Backus-Naur Form[2]

Kapitola 2

Jazyk Texy

Následující kapitola blíže popisuje jazyk Texy (vyslovuje se „texy“[1]). Obsahuje zhodnocení jazyka a ukázku jeho použití. Tato kapitola čtenáře blíže seznámí se současnou (a jedinou) implementací překladače jazyka. Je zde také uveden model jazyka, který byl použit při sestavování formální gramatiky rozebrané v kapitole 3.

2.1 Popis jazyka

Jazyk Texy je uživatelsky poměrně přívětivý. Uživateli nabízí snadný zápis dokumentů bez specializovaného softwarového vybavení. Je pravda, že běžný uživatel počítače dává většinou přednost snadnému ovládání, jaké nabízejí textové procesory a WYSIWYG editory. Existuje však stále rostoucí skupina tzv. *pokročilých uživatelů*, kteří chápou zadávání dokumentu v textové formě s formátovacími značkami jako mnohem komfortnější.

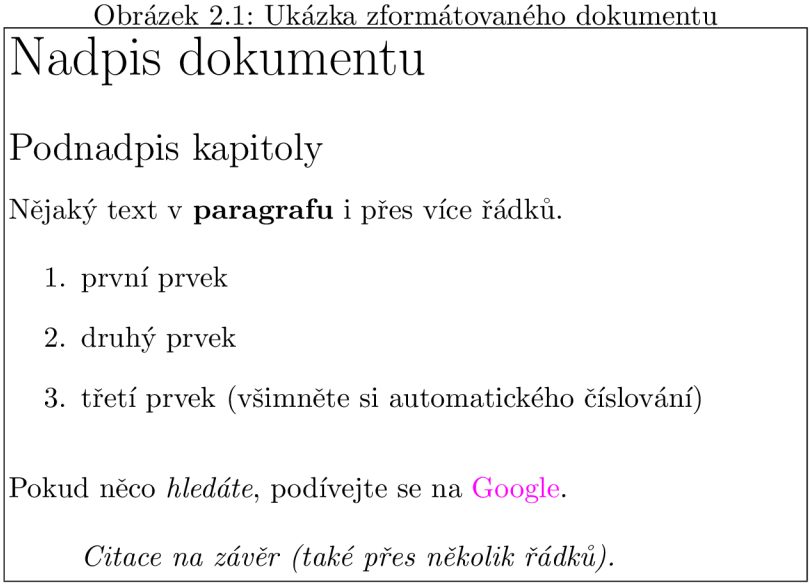
2.1.1 Ukázka

Nejnázornější bude předvést jazyk Texy na ukázce. Takto může vypadat jednoduchý dokument zadaný v jazyce Texy:

```
Nadpis dokumentu
****************

Podnadpis kapitoly
------------------

Nějaký text
v **paragrafu**
i přes více řádků.

1) první prvek
1) druhý prvek
1) třetí prvek (všimněte si automatického číslování)

Pokud něco //hledáte//, podívejte se na "Google":www.google.com

> Citace na závěr (také přes několik
> řádků).
```

Jak se takový dokument naformátuje je znázorněno na 2.1.

Obrázek 2.1: Ukázka zformátovaného dokumentu



Texy samozřejmě nabízí i mnohem pokročilejší konstrukce. Zde je například zápis tabulky:

```
|Pobočka       |Vedoucí          |Výnosy
|-----------------------------------
|Bohunice      |                 |17 mil.
|Nový Lískovec|Jeřábek Jan      ^|13 mil.
|Brno-město    |Plytký Karel     |25 mil.
|Celkem                         ||55 mil.
```

Takto zapsaná tabulka bude zformátována, jak ukazuje 2.1.

Tabulka 2.1: Ukázka zformátované tabulky

Pobočka	**Vedoucí**	**Výnosy**
Bohunice	Jeřábek Jan	17 mil.
Nový Lískovec		13 mil.
Brno-město	Plytký Karel	25 mil.
Celkem		55 mil.

Podrobný popis a mnoho dalších ukázek je uvedeno na domovské stránce Texy.[9]

2.1.2 Parametrizace výstupu

Rozšíření navrhované v 1.2.1 bude v JTexy dosaženo vkládáním rezervovaných míst pro pojmenované parametry. Rezervované místo bude zapsáno ve tvaru `${«název parametru»}`, kde název parametru je libovolný řetěz znaků (kromě znaku konce řádku nebo znaku `$`).

2.2 Současná implementace

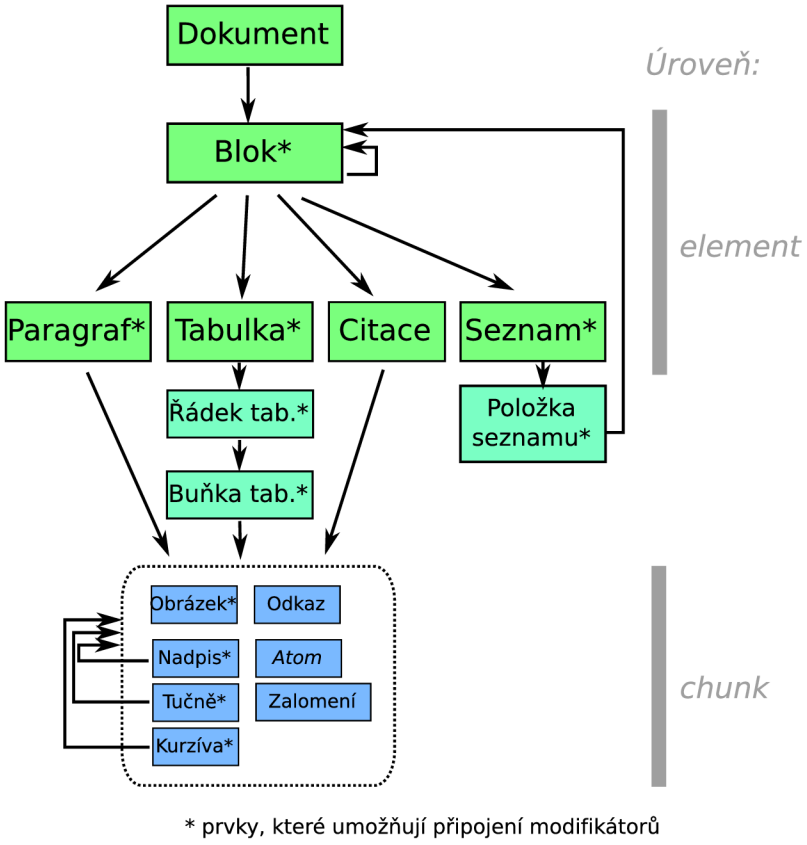
Současná implementace (nazývána v této práci také jako *referenční implementace*) je knihovna v jazyce PHP. Tato knihovna je duálně licencovaná pod GPL a pod komerční licencí. Její zdrojový kód je tedy volně přístupný ke studiu.

Knihovna se vyznačuje relativně čistým objektovým návrhem a čitelným kódem. Architektonicky je navržena jako modulární systém, kde jednotlivé rysy (*moduly*) mohou být podle potřeby vypínány, popřípadě mohou být pomocí systému *hooků* přidávány úplně nové funkce a jazykové konstrukty. Pro rozpoznání vstupu se v této knihovně většinou používají oddělené regulární výrazy. Po rozpoznání je sestaven strom HTML elementů (*DOM*), ze které je následně generován HTML výstup.

Dokumentace knihovny není příliš vyčerpávající a zaměřuje se především na výklad ukázkami.[9]

2.3 Model jazyka

K úspěšnému sestavení gramatiky a vůbec k jednoznačnému pochopení jazyka je nejprve potřeba mít k dispozici jeho *model*. Dokumentace k Texy žádný formální model neobsahuje, bylo proto nutné jej sestavit.



Obrázek 2.2: Model jazyka

Při sestavování modelu jsem vycházel z dokumentace. Ta je spíše neformální a spoléhá na výklad příkladem, takže nejasnosti bylo nutné řešit experimentálně na referenční implementaci. To mohlo přinést odchýlení se od původního záměru jejího autora. Přihlédl jsem i k tomu, že primárním výstupním formátem Texy je HTML. Proto jsou některé prvky modelu pojaté podobně jako v HTML, což může usnadnit generování výstupních dokumentů do tohoto formátu.

Na obrázku 2.2 je graficky znázorněno, z jakých prvků se jazyk skládá. Šipky na něm znázorňují relaci „obsahuje“. Tedy například blok může obsahovat více bloků, paragraf, tabulku, atp. Následuje popis jednotlivých prvků.

Dokument je nejvyšším prvkem hierarchie. Jedná se o reprezentaci jednoho dokumentu v jazyce Texy. Obsahuje právě jeden kořenový *blok*.

Elementem se nazývají všechny prvky jazyka, které mohou být obsaženy v bloku.

Blok je element, který zapouzdřuje libovolného množství dalších elementů. Na blok je možné aplikovat *modifikátory* (ty jsou podrobněji popsané dále).

Paragraf je element, který představuje klasickou jednotku textu, jak ji známe z typografie. Obsahuje libovolné množství *chunků*. Na paragraf je možné aplikovat modifikátory.

Tabulka je element sloužící k zobrazení tabulárních dat. Obsahuje jeden či více řádků. Je na ni možné aplikovat modifikátory.

Seznam je element, který zobrazuje několik po sobě jdoucích prvků seznamu. Většinou funguje jako výčet nebo vyjmenování možností, popřípadě jako seznam definicí. Seznam může mít několik vizuálních stylů:

1. nesetříděný seznam
2. setříděný číslovaný seznam
3. setříděný seznam číslovaný malými písmeny abecedy (`a-z`)
4. setříděný seznam číslovaný velkými písmeny abecedy (`A-Z`)
5. setříděný seznam číslovaný římskými číslicemi (`I.`, `II.`, `III.`, atd.)

Na seznam je možné aplikovat modifikátory.

Citace je element, který slouží k zobrazení citovaného bloku textu. Na citaci je možné aplikovat modifikátory.

Řádek tabulky je jednotka obsažená v tabulce. Obsahuje jednu nebo více buněk. Na řádek tabulky je možné aplikovat modifikátory.

Buňka tabulky je jednotka obsažená v řádku tabulky. Obsahuje libovolné množství chunků. Má dvě vlastnosti ovlivňující spojování buněk:

- horizontální přesah tabulky – kolik buněk od dané buňky doprava je spojeno do této buňky
- vertikální přesah tabulky – kolik buněk od dané buňky dolů je spojeno do této buňky

Kromě nich je na buňku tabulky možné aplikovat modifikátory.

Položka seznamu je jednotka obsažená v seznamu. Obsahuje právě jeden blok, který v sobě obsahuje další elementy. Na prvek seznamu tabulky je možné aplikovat modifikátory.

Chunkem se nazývají prvky, které představují jednotku textu.

Atom je chunk, který je dále nedělitelný a který obsahuje pouze jediný textový řetězec.

Zalomení je chunk představující vynucené zalomení řádku.

Obrázek je chunk, který je odkazem na externí grafický soubor. Obsahuje:

- cestu k externímu souboru
- šířku obrázku (v pixelech)
- výšku obrázku (v pixelech)

Na obrázek je možné aplikovat modifikátory.

Nadpis je chunk který reprezentuje nadpis v textu. Obsahuje libovolné množství dalších chunků a je definován svojí *důležitostí*, což je celé číslo od 1 do 10. 1 značí nejvyšší důležitost. Na nadpis je možné aplikovat modifikátory.

Tučná fráze je chunk představující část textu, která je znázorněna tučně. Obsahuje libovolné množství dalších chunků a je na ni možné aplikovat modifikátory.

Fráze kurzívou je chunk představující část textu, která je znázorněna kurzívou. Obsahuje libovolné množství dalších chunků a je na ni možné aplikovat modifikátory.

Odkaz je chunk, který představuje odkaz na externí dokument. Skládá se z řetězce obsahující *URL*[1] dokumentu a řetězce obsahující textový popis.

Množina modifikátorů je meta prvek dokumentu, který vždy náleží jinému prvku. Obsahuje sadu vlastností, které ovlivňují vzhled nebo umístění prvku, ke kterému náleží:

- zarovnání prvku (může být pravé, levé, na střed, do bloku nebo bez zarovnání)
- množina CSS tříd aplikovaných na generovaný HTML element
- ID generovaného HTML elementu
- atributy prvku, což jsou hodnoty které se použijí k sestavení inline CSS stylu generovaného HTML elementu

I když tento model není úplný, obsahuje všechny rysy jazyka, které byly implementovány v rámci této bakalářské práce.

[1] Uniform Resource Locator

Kapitola 3

Gramatika

Tato kapitola zdůvodní nutnost konstrukce formální gramatiky jako prostředku pro vyjádření jazyka. Představí čtenáři nástroj ANTLR sloužící ke generování lexikálního a syntaktického analyzátoru bezkontextových gramatik. Popíše také konstrukci gramatiky Texy, včetně problémů, které se v průběhu objevily.

3.1 Důvody

Formalizace gramatiky jazyka spolu nese řadu praktických výhod.

- Do značné míry usnadní další implementace. Jakkoliv může být slovní popis pro uživatele schůdnější a čitelnější, pro programátora je spíše zdrojem mnoha nejasností a dvojsmyslů.
- Celkovou strukturu jazyka má většinou původní autor či autoři intuitivně ve své mysli, ale až právě formalizace může ukázat na nedostatky nebo nelogičnosti v návrhu jazyka.
- Formální gramatika významně přispívá standardizaci. Formální definice jazyka může znamenat rozdíl mezi jazykem uzamčeným na jedinou implementaci na jediné platformě a jazykem s mnoha implementacemi na různých platformách a systémech.

3.2 Nástroj ANTLR

ANTLR je nástroj sloužící ke generování lexikálního a syntaktického analyzátoru bezkontextových gramatik.[15] I když je tento nástroj napsaný v Javě, umožňuje generovat analyzátory použitelné i v jiných programovacích jazycích, jako je C, C++ Java, C# nebo Python.

ANTLR se skládá ze dvou částí – *generátoru* a *běhové knihovny*. Generátor je program, který provede analýzu gramatiky popsané speciální syntaxí. Na základě této gramatiky vygeneruje zdrojový kód lexikálního a syntaktického analyzátoru pro cílový jazyk.[1] Tyto analyzátory je pak možné přeložit standardním překladačem daného cílového jazyka a přímo je používat z kódu aplikace nebo knihovny. Pro jejich spuštění je ovšem nutná zmíněná běhová knihovna.

[1] Tyto pojmy jsou blíže objasněny v části 4.3.

Na vstupu je proud znaků. Z něj jsou lexikálním analyzátorem vytvořeny tokeny (symboly jazyka) za pomoci definovaných pravidel. K jednotlivým pravidlům je možné připojit libovolné *akce* napsané přímo v Javě. Navíc je možné akcemi ovlivňovat i samotný výstup z lexikálního analyzátoru.

Proud tokenů z lexikálního analyzátoru je pak použit v syntaktickém analyzátoru, který aplikuje pravidla syntaktické analýzy a na základě nich vykonává akce. Zároveň může vytvářet *abstraktní syntaktický strom* (AST), pokud je v gramatice definováno, jakou strukturu má výsledný AST mít.

3.2.1 ANTLR a $LL(*)$

ANTLR generuje $LL(*)$ syntaktické analyzátory, které jsou silnější než obvyklé generované i ručně psané $LL(1)$ analyzátory. LL analyzátory mají typicky pro každé pravidlo povolujícího více alternativ jeden konečný stavový automat, který rozhoduje o použité alternativě.[8, 183 s.] Klasické $LL(k)$ parsery jsou omezeny na fixní hodnotu k proto, že tento automat mají acyklický, tedy že graf jeho stavů neobsahuje kružnici. Hodnota k je pak nejdelší možná cesta od počátečního do koncového stavu. ANTLR generuje $LL(*)$ parsery, tedy $LL(k)$ s neomezenou hodnotou k. Toho je dosaženo tím, že automat rozhodující o použité alternativě v grafu stavů může obsahovat kružnici, tedy k je potenciálně neomezené.[14, 268 s.]

3.3 Problémy

Při sestavování gramatiky jsem narazil na několik problému, které si vynutily zjednodušení gramatiky na úkor přirozeného zápisu. Některé konstrukce je proto nutné rozpoznávat programově až na základě abstraktního syntaktického stromu, nebo naopak ještě před syntaktickou analýzou rozšířením lexikálního analyzátoru. Toto je blíže popsáno v kapitole 4.

3.3.1 Nejednoznačnost gramatiky

Existuje gramatika, která jazyk Texy popisuje jednoznačně. Tato gramatika je však vyšší než bezkontextová.[2] Mějme například bezkontextovou gramatiku $G_1 = (\{S, A\}, \{X, Y\}, P, S)$, kde P je definováno jako

$$A \quad ::= \quad X\ Y{+}\ X \tag{3.1}$$
$$A \quad ::= \quad X \tag{3.2}$$
$$A \quad ::= \quad Y \tag{3.3}$$
$$S \quad ::= \quad A{+} \tag{3.4}$$

Podobný zápis se u Texy používá k zápisu fráze, konkrétně terminál X značí začátek a konec fráze, například tučného písma popsaného v 2.3. Zároveň ale tento terminál může být součástí obsahu. Tato gramatika je nejednoznačná, např. větu $X\ Y\ X$ můžeme derivovat na A použitím pravidla 3.1 nebo na $A\ A\ A$ použitím 3.2 a 3.3.

Gramatika musí být bezkontextová, abychom pro ni mohli generovat syntaktický analyzátor nástrojem ANTLR. Proto je nutné se spokojit s nejednoznačnou gramatikou. Nástroj

[2] Toto se netýká pouze jazyka Texy, ale je společné pro značkovací jazyky, kde se konstrukty jazyka volně kombinují se samotným obsahem.

ANTLR je schopen rozpoznat i nejednoznačné gramatiky. Experimentálně jsem však zjistil, že výsledný syntaktický analyzátor je pro složité nejednoznačné gramatiky natolik komplikovaný, že přesahuje pevné limity dané specifikací virtuálního stroje Java. Konkrétně je v této specifikaci uvedeno, že velikost interpretovaného kódu metody nesmí přesáhnout 65 535 bajtů.[12] Generované analyzátory některých gramatik ale obsahují metody delší – jedná se o metody, které obsluhují stavové přechody. Takový kód pak překladač odmítne přeložit. Proto je nutné pokud možno omezit místa, kde je gramatika nejednoznačná.

3.3.2 Omezení parseru

Další problém plyne z omezení parseru. Některé rysy jazyka Texy není možné přirozeným způsobem přepsat do bezkontextové gramatiky, pro kterou by byl nástroj ANTLR schopen generovat korektní parser.

Mějme bezkontextovou gramatiku $G_2 = (\{S, A\}, \{X, Y, \$\}, P, S)$, kde P je

$$A \quad ::= \quad X\ A{+}\ Y \tag{3.5}$$
$$A \quad ::= \quad Y \tag{3.6}$$
$$S \quad ::= \quad A{+}\ \$ \tag{3.7}$$

Pravidlo 3.5 je rekurzivní, což je jediný způsob, jak v gramatice zachytit konstrukt bloků v jazyce Texy. Zároveň je tato gramatika nejednoznačná z podobného důvodu, jako je popsáno v části 3.3.1 – pro větu $X\ Y\ Y\ Y$ je možná posloupnost derivací

$$(X\ Y\ Y\ Y) \rightarrow (X\ A\ A\ Y) \rightarrow (A)$$

nebo

$$(X\ Y\ Y\ Y) \rightarrow (X\ A\ Y\ Y) \rightarrow (A\ Y) \rightarrow (A\ A)$$

Při použití ANTLR jsou zřejmé tyto problémy:

1. **G_2 není $LL(*)$ gramatika.** Jak už bylo řečeno v části 3.2, ANTLR generuje $LL(*)$ parsery využívající konečného stavového automatu k rozpoznání alternativ. Je možné dokázat, že konečný stavový automat není dostatečně silný k rozhodnutí kterou z alternativ 3.5, 3.6 použít.[3][14, 307 s.]

2. **K rozpoznání G_2 nepomůže ani backtracking.** V případě, že bychom k rozpoznání pravidla místo konečného stavového automatu použili backtracking – což ANTLR umožňuje aplikací tzv. *syntaktického predikátu*[14, 331-356 s.] – nejednoznačnost jazyka stále bude způsobovat problémy při aplikaci rekurzivního pravidla. To, kde ukončit cyklus po sobě následujících neterminálů a, totiž není možné $LL(*)$ parserem rozhodnout.

   To opět nejlépe ukážu na příkladu. ANTLR generuje pro pravidlo a následující parser (v pseudokódu):

```
if input() == X and ≪backtrack a()≫: // podle 3.5
    match(X)
    while input() != $:
```

[3] Konečné stavové automaty rozpoznají pouze regulární gramatiky, což je rozpor s rekurzivním pravidlem 3.5.

```
        a()
    match(Y)
else if (input() == Y): // podle 3.6
    match(Y)
```

kde `input()` je funkce vracející terminál na vrcholu vstupního zásobníku a `match(x)` je funkce, která vyjme terminál ze vstupního zásobníku, pokud je roven `x` (v opačném případě způsobí selhání rozpoznávání).

Potom například rozpoznání věty $X\ Y\ Y\ \$$ selže. Je tomu tak proto, že cyklus uvozený podmínkou `while input() != $` je ukončen, až když je na vstupu konečný terminál \$, tedy pravidlo a příjme o jeden terminál Y více, než by mělo. Následující volání přijetí terminálu Y – `match(Y)` – způsobí selhání syntaktické analýzy. A to přesto, že gramatika G_2 zcela jistě generuje větu $X\ Y\ Y\ \$$.

Kapitola 4

Implementace

V této kapitole je podrobněji popsána implementace samotného překladače a výstupu. Je rozebráno, z jakých procesů se překlad skládá a také způsob reprezentace výsledného dokumentu. Také je zde popsána implementace výstupu do formátu HTML.

4.1 Obecné informace o implementaci

4.1.1 Poznámky k objektovému návrhu

Při implementaci jsem kladl důraz na čistý a jednoduchý návrh a časté komentáře ve zdrojovém kódu. Pro objektový návrh byly použity tyto hlavní principy:

- Jasné oddělení odpovědností jednotlivých tříd. To vede ke zvýšení čitelnosti kódu a snížení nároků na údržbu.
- Aplikace principů defenzivního programování.[10, s. 65]
- Dodržení principu tříd otevřených pro rozšíření a uzavřených pro modifikaci.[13] Tento princip mimo jiné znamená, že u tříd, které nejsou explicitně navržené pro rozšíření, je rozšiřování znemožněno modifikátorem `final`. Takto je zamezeno zneužívání dědičnosti ke změně chování tříd tam, kde by k němu nemělo docházet.
- Tam, kde to je možné, jsou třídy navrženy jako *neměnitelné* (*immutable*). To s sebou nese některé výhody: je možné je bezpečně využívat v prostředí s více vlákny bez nutnosti synchronizace a není nutné vytvářet defenzivní kopii při předávání reference na instance takové třídy.

4.1.2 Použité pomůcky

Automatické testy

Automatické testy se čím dál více prosazují v softwarovém inženýrství nejen jako nástroj pro zjednodušení práce při testování aplikace, ale také jako pomůcka programátora pro ověření správnosti algoritmu a zachování jeho správnosti během celého životního cyklu produktu. Automatickými testy se myslí sada programů, které spouští kód knihovny nebo aplikace a porovnávají její výstup nebo chování s předem daným předpokladem.[10, s. 81]

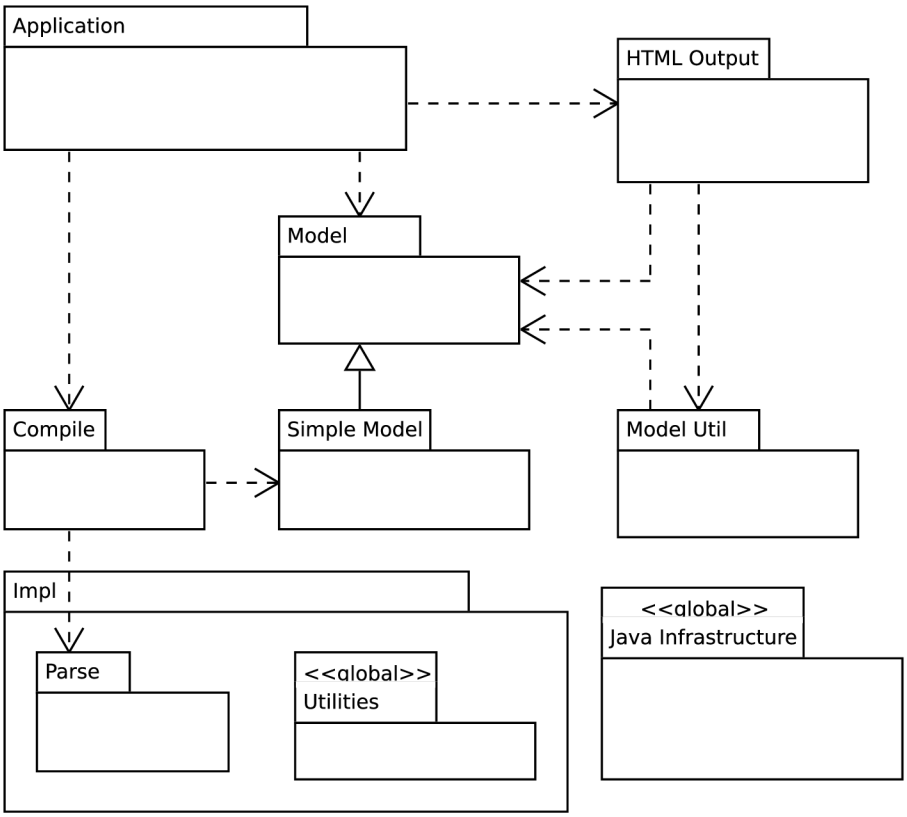
Během vývoje jsem průběžně sestavoval k většině implementovaných funkcionalit i funkční testy. Také jsem na některých místech použil techniku jednotkových testů. Automatické testy byly použity i v průběhu sestavování gramatiky. Zhruba 40% veškerého kódu jsem psal metodou *programování řízené testy.* [10, s. 94]

Tento důraz na automatické testy se zcela jistě při vývoji oplatil, jak je zhodnoceno v závěru práce (část 5.1.2).

FindBugs

Při vývoji jsem průběžně sledoval výstupy z nástroje FindBugs, který provádí statickou analýzu kódu za účelem nalezení potenciálních chyb v kódu a porušení doporučených postupů. Výsledek analýzy finální verze kódu je opět zhodnocen v závěru.

4.2 Architektura knihovny



Obrázek 4.1: Diagram balíků

Knihovna je rozdělena na balíky podle jejich odpovědností, jak znázorňuje diagram balíků 4.1.

Application představuje libovolnou aplikaci využívající JTexy.

Compile odpovídá Javovému balíku `cz.oxyonline.jtexy.compile`. Tento balík obsahuje třídy určené k překladu vstupních dokumentů. Je zodpovědný za převod AST na paměťovou reprezentaci, což je popsáno v 4.4

Model (`cz.oxyonline.jtexy.model`) obsahuje rozhraní reprezentující model Texy, viz. 4.4.1.

Simple Model (`cz.oxyonline.jtexy.model.simple`) obsahuje primitivní implementaci modelu, viz. 4.4.1.

Impl (`cz.oxyonline.jtexy.impl`) v sobě obsahuje interní třídy, které jsou použité z jiných balíku knihovny. Tento balík není určený k použití mimo kód JTexy. Obsahuje balík lexikálního a syntaktického analyzátoru (`cz.oxyonline.jtexy.impl.parse`) a balík pro pomocné funkce (`cz.oxyonline.jtexy.impl.util`).

HTML Output (`cz.oxyonline.jtexy.output.html`) obsahuje generátor výstupu do HTML, viz. 4.5.

Model Util (`cz.oxyonline.jtexy.model.util`) obsahuje třídu `ModelVisitor` popsanou v 4.5

Samotný proces překladu je znázorněn na obrázku 4.2. Tento proces je blíže popsán v následujících podkapitolách.

4.3 Vytvoření abstraktního syntaktického stromu

Prvním krokem v procesu překladu je vytvoření *abstraktního syntaktického stromu*. S tímto pojmem se běžně setkáváme u vícekrokových překladačů imperativních jazyků.[16, 8 s.]

Vytvoření syntaktického stromu je poměrně komplikovaný proces. Naštěstí pro něj existují dobře popsané a zavedené algoritmy. Při rozpoznávání naprosté většiny programovacích jazyků se dnes používá dvoufázový proces, kde první fáze se nazývá *lexikální analýza* a druhá fáze *syntaktická analýza*. Toto rozdělení má svůj počátek v teoretickém základu překladačů, tedy teorii formálních jazyků. Zde se běžně pracuje s jazykem jako množinou řetězců sestavených z určité abecedy symbolů.

Při znalosti tohoto formalizmu je možné stanovit dva podsystémy:

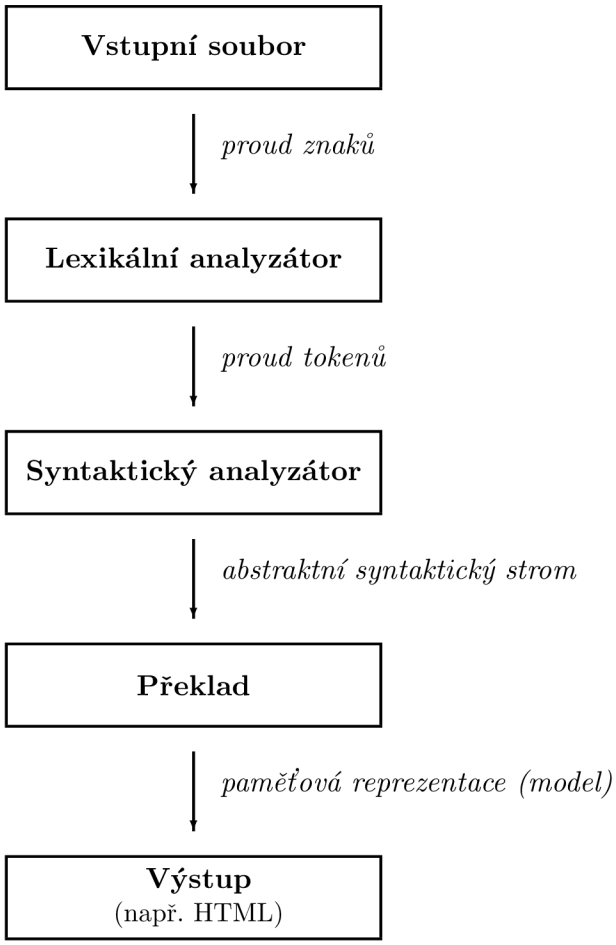
lexikální analyzátor slouží k rozpoznání posloupnosti symbolů (lexů) ve vstupním proudu znaků

syntaktický analyzátor v obecnějším pojetí rozpozná, zda posloupnost znaků patří do daného jazyka. V praktickém pojetí překladačů pak vytvoří z této posloupnosti strukturu – syntaktický strom – se kterou je možné dále pracovat.

4.3.1 Lexikální analýza

Lexikální analýzu zajišťuje třída `TexyLexer`. Tato třída je generovaná z tzv. *lexikální gramatiky* pomocí nástroje ANTLR, kterému je věnovaná část 3.2. Používá $LL(1)$ algoritmus pro rozpoznání vstupu.

Ukázka použité lexikální gramatiky:



Obrázek 4.2: Průběh překladu a výstupu

```
EOL: '\n' | '\r\n'; // Konec řádku (platforma Unix i Windows)
QUOTE: '"';
UNDERSCORE: '_';
LEFT_SQ_BRACKET: '[';
RIGHT_SQ_BRACKET: ']';
// atp.
```

Vlastní rozšíření analyzátorů

$LL(1)$ lexikální analýza sama o sobě bohužel nestačí k rozpoznání všech symbolů Texy, jako jsou HTML bloky nebo odsazení. Navíc, aby se zjednodušila samotná gramatika z důvodů popsaných v 3.3, jsou některé konstrukce rozpoznávány při lexikální analýze, i když by svým charakterem patřily spíše do syntaktické analýzy.

Z tohoto důvodu jsem vytvořil třídu `Lexer`, která slouží jako bázová třída pro generovaný `TexyLexer`. Metody této třídy jsou pak volány z `TexyLexeru` pomocí akcí uvedených v samotné gramatice. Tyto metody slouží k rozpoznání symbolů, které by nebylo možné rozpoznat $LL(1)$ analýzou.

Následuje ukázka takové metody, která rozpozná, zda znak mezery je prvním znakem

symbolu „odsazení“ („odsazení“ je posloupnost mezer, která se nachází bezprostředně na začátku řádku):

```
protected void tryMatchAndEmitIndent() {
    // not an indentation token, if this is not the first character on line
    if (getCharPositionInLine() != 1) {
        return;
    }

    // If the space is the first character on the line,
    // emit an INDENT_MARK marker instead of S.

    // consume the following spaces too
    int spaceCount = 0;
    while (input.LA(spaceCount) == ' ') {
        spaceCount++;
    }

    // consume the spaces and emit the indent mark
    consume(spaceCount);
    emitLast(TexyLexer.INDENT_MARK);
}
```

Tato metoda je v ANTLR gramatice vyvolána pravidlem

```
S: ' ' { tryMatchAndEmitIndent(); };
```

Toto pravidlo říká, že libovolná mezera odpovídá tokenu s názvem `S`. Zároveň je však při rozpoznání tohoto tokenu vyvolána metoda `tryMatchAndEmitIndent()`, která zjistí, zda se ve skutečnosti nejedná o odsazení řádku. Pokud ano, tak vyvolá metodu `emitLast()`, která do výstupu lexikálního analyzátorů přidá token `INDENT_MARK`. Token `S`, který by byl normálně rozpoznán, se následně ignoruje.

4.3.2 Syntaktická analýza

`TexyParser` je třída, která zajišťuje syntaktickou analýzu. Tato třída je generována na základě *syntaktické gramatiky*. Používá $LL(*)$ algoritmus, který je popsaný v 3.2.1, k rozpoznání syntaktické struktury. Na jejím základě pak vytvoří syntaktický strom.

Ukázka použité syntaktické analýzy (tato část slouží k rozpoznání tabulek Texy):

```
table: tableModifiers? tableRow+;

tableRow: PIPE HYPHEN HYPHEN HYPHEN+ EOL
    | tableCell+ EOL;

tableCell: PIPE ASTERISK tableCellContent
    | PIPE tableCellContent;

tableCellContent: text cellHorizontalSpan*;
```

```
cellHorizontalSpan: PIPE;

// atp.
```

Jak je vidět, zápis se nápadně podobá BNF. Z ukázky je navíc patrné, že v gramatice je také možné použít konstrukty z EBNF, jako je zápis sekvence symbolů (značky `*` a `+`).[2]

Opět platí, že ne vše je možné rozpoznat při syntaktické analýze. Některé složitější prvky jazyka, jako například seznamy, jsou ve skutečnosti rozpoznány až analýzou samotného syntaktického stromu. Proto syntaktická gramatika neobsahuje pravidla, která by tyto prvky rozpoznávala.

Kompletní text gramatiky pro ANTLR je v příloze 1.

4.4 Převod na paměťovou reprezentaci

Samotný syntaktický strom se zdá být nevhodný způsob samotné reprezentace dokumentu. Ta musí být navržená tak, aby na jejím základě bylo možné snadno stavět generátor výstupu. Rozhraní pro práci se syntaktickým stromem naopak není příliš programátorsky přívětivé a značně nevýhodná je i úzká vazba na knihovnu ANTLR. Navíc je syntaktický strom stejně nutné určitým způsobem transformovat, aby odpovídal modelu jazyka a ne pouze zjednodušené gramatice.

Proto se zdá být výhodnější pro paměťovou reprezentaci zvolit vlastní strukturu, která bude přesně kopírovat model jazyka, jak je popsaný v 2.3.

4.4.1 Struktura

Jako veřejné rozhraní pro práci s reprezentací dokumentu byly vytvořené rozhraní umístěné do balíku `cz.oxyonline.jtexy.model`. V tomto balíku mají všechny prvky jazyka svoji reprezentaci.

Byla také vytvořena sada tříd, které tyto rozhraní implementují. Tyto třídy jsou v balíku `cz.oxyonline.jtexy.model.simple`. Jsou primárně určeny k použití při překladu dokumentů, ale jsou veřejné a je možné je použít kdekoliv v kódu používajícím JTexy, tedy například jako pomůcku při implementaci jednotkových testů. Tyto třídy byly navrženy jako neměnitelné.

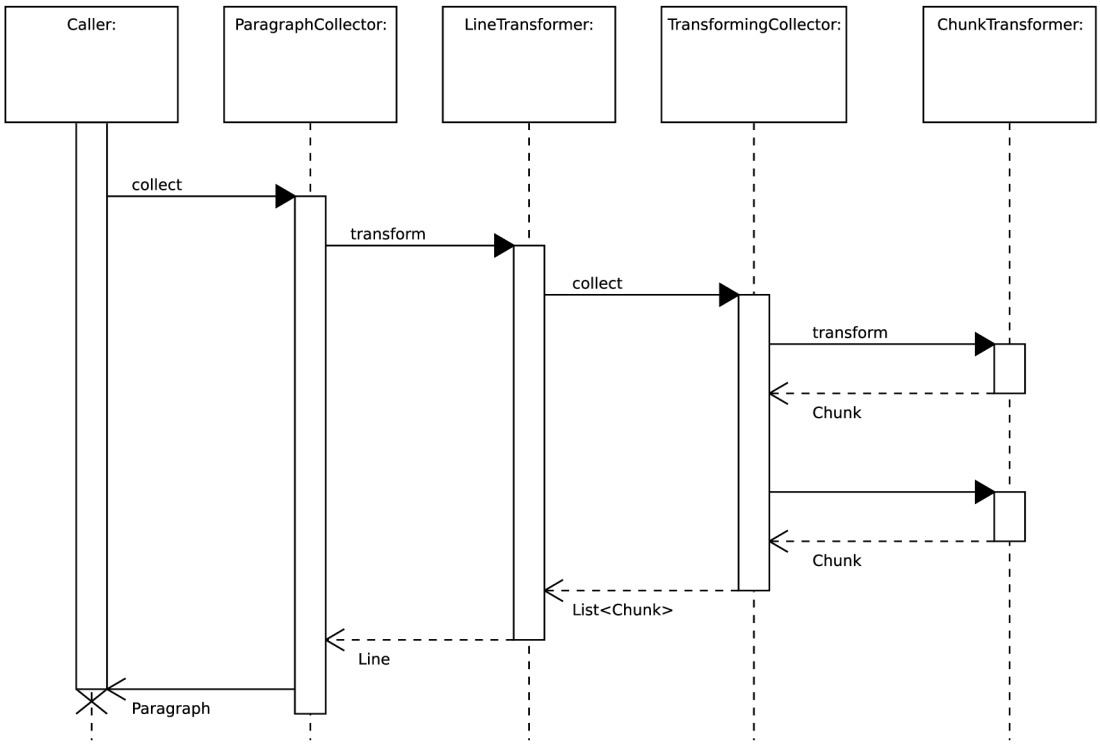
4.4.2 Transformace z AST

K dekompozici transformace z abstraktního syntaktického stromu jsem zavedl dvě abstrakce:

Transformátor (*transformer*) je třída, která na vstupu přijímá uzel syntaktického stromu a na výstupu vrací seznam vygenerovaných prvků dokumentu (tj. instancí rozhraní z `cz.oxyonline.jtexy.model`). Transformátor implementuje rozhraní `cz.oxyonline.jtexy.compile.NodeTransformer`.

Sběrač (*collector*) postupně přijímá uzly syntaktického stromu a na základě nich vygeneruje seznam prvků dokumentů. Sběrač je třída, která rozšiřuje abstraktní třídu `cz.oxyonline.jtexy.compile.NodeCollector`.

Nejčastěji použitým sběračem je `TransformerCollector`, který na každý přijatý uzel stromu aplikuje předem daný transformátor.



Obrázek 4.3: Transformace paragrafu

Tyto dvě základní abstrakce tvoří celý proces transformace vzájemným zanořováním. Toto je vidět na příkladu transformace paragrafu, jak ukazuje sekvenční diagram 4.3. Na něm je vidět, že sběrač paragrafu sestavuje výstup, který pochází z transformátoru řádků. Tento transformátor zase používá sběrač a transformátor k transformaci jednotlivých chunků.

4.5 Generátor výstupu

Konečným krokem v procesu překladu je generování výstupu. Rozhraní, které musí generátor výstupu implementovat, se jmenuje `cz.oxyonline.jtexy.OutputGenerator`. Toto rozhraní definuje pouze jednu metodu

```
public void generate(Document document, Map<String, String> parameters)
        throws OutputGenerationException;
```

která provede vytvoření výstupu. Parametr `document` udává generovaný dokument a `parameters` je mapa parametrů, které budou v dokumentu dosazené do rezervovaných míst.

Pro procházení prvků dokumentu slouží pomocná abstraktní třída `cz.oxyonline.jtexy.model.util.ModelVisitor`. Ta využívá návrhový vzor *návštěvník* (*visitor*)[11, 193 s.] pro průchod stromovou strukturou dokumentu. Je využita varianta tohoto vzoru, kdy klient i návštěvník jsou spojeni do jedné třídy. Metody `visit()` abstraktní třídy zajišťují propagaci události členům navštíveného prvku. Tyto metody mohou vypadat například takto:

```
public void visit(Table table) {
    for (TableRow row : table.getRows()) {
        visit(row);
    }
}

public void visit(TableRow row) {
    for (TableCell cell : row.getCells()) {
        visit(cell);
    }
}

public void visit(TableCell cell) {
    // ...
}
```

Při implementaci generátoru pak stačí rozšířit tuto třídu a překrýt některé nebo všechny její metody `visit()`. Při překrytí je možné upravit chování například tak, aby při průchodu prvkem bylo do výstupu zapsán libovolný text.

Příklad:

```
public void visit(TableRow row) {
    printer.println("<tr>"); // zapíše počáteční značku pro řádek tabulky

    // vyvolá původní implementaci, která prochází všechny buňky na řádku
    super.visit(row);

    printer.println("</tr>"); // zapíše koncovou značku pro řádek tabulky
}
```

Jediný současný generátor výstupu je třída `cz.oxyonline.jtexy.output.html.HtmlOutputGenerator`, která, jak název napovídá, generuje výstup do HTML formátu.

4.6 Převodní aplikace

Součást zadání této práce je i program, který je vyvolán z příkazové řádky, a který převádí vstupní dokumenty v jazyce Texy na HTML dokumenty. Ovládání tohoto programu je popsáno v příloze 3.

Kapitola 5

Závěr

V této kapitole je zhodnocena dosavadní práce. Dále jsou zde blíže přiblíženy další možnosti vývoje. Zejména je zde rozebrána možnost generování jiných výstupních formátů, než je HTML, a jaké problémy s tím mohou být spojeny.

5.1 Zhodnocení práce

S prací jsem začal před více než 7 měsíci. Můj odhad je, že celkově mi zabrala asi 150 až 200 hodin. Podařilo se mi sestavit funkční knihovnu, která začne být komerčně využívána zanedlouho po dokončení. Pravděpodobně bude také uvolněna pod některou z open source licencí a snad i přispěje k vyšší popularitě jazyku Texy jako takového. Tato knihovna má potenciál pro další rozšiřování, což je popsáno v části 5.2.

Knihovnu je možné použít v libovolné aplikaci, nebo ji spouštět pomocí příkazové řádky, jak popisuje příloha 3. Také je možné si ji přímo vyzkoušet na internetové adrese `http://thinkbox.cz/jtexy-web-demo/`.

Pro mě osobně je velkým přínosem, že jsem rozšířil svoje znalostí o formálních gramatikách. Naučil jsem se poměrně dost o třídě *LL* syntaktických analyzátorů, které se jinak podle mého názoru vyskytují poněkud na okraji zájmu akademické a odborné sféry, přestože mají některé nesporné výhody. Také jsem získal určité zkušenosti s návrhem formální gramatiky. Jako celek tedy považuji práci za úspěšnou.

5.1.1 Porovnání s referenční implementací

Ne všechny funkce referenční implementace byly z důvodu časových omezení implementovány. Tabulka 5.1 srovnává obě implementace podle uživatelských funkcí.

S dosaženou sadou funkcí je možné knihovnu považovat za použitelnou, pokud uvážíme, že všechny často užívané funkce jsou zastoupeny. JTexy navíc přidává funkci vkládání parametrů a také definuje API pro práci s dokumenty, což umožňuje vytvoření alternativních generátorů výstupu.

5.1.2 Výsledky z automatických nástrojů

FindBugs

I při nastavení tohoto nástroje na nejnižší hladinu závažnosti chyb nenašel FindBugs ve finální verzi kódu jedinou „chybu“.

Tabulka 5.1: Srovnání funkcí referenční implementace

	Implementace	
Funkce	**Texy**	**JTexy**
Odstavce	Ano	Ano
Titulky	Ano	Ano
Horizontální čáry	Ano	Ano
Vložený kód	Ano	Ano
Vypnutí formátování	Ano	Ano
Bloky	Ano	Ano
Citace	Ano	Ano[a]
Odkazy	Ano	Ano
Reference	Ano	Ne
Obrázky	Ano	Ano[b]
Fráze	Ano	Ano
Přímé HTML	Ano	Ano
Seznamy	Ano	Ano
Modifikátory	Ano	Ano
Automatická česká typografie	Ano	Ne
Tabulky	Ano	Ano
Vložené parametry	Ne	Ano
API umožňující jiné výstupy	Ne	Ano
Modularita na úrovni syntaxe	Ano	Ne

[a] Vnořené citace nejsou podporovány.
[b] Událost mouse-over není podporována.

Automatické testy

Důsledné testování pomocí automatických testů se kladně projevilo na rychlosti vývoje, ale hlavní přínos se pravděpodobně ukáže při dlouhodobém udržování této knihovny v komerčním prostředí. Tabulka 5.2 ukazuje metriky ukazující kvalitu pokrytí kódu automatickými testy.[1]

Tabulka 5.2: Pokrytí testy

Celkový počet testů	163
Celkové pokrytí testy (řádky kódu)	80 %
Celkové pokrytí testy (větvení)	73 %

5.2 Možnosti dalšího vývoje

5.2.1 Alternativní výstupní formáty

Jedním z úkolů pro reimplementaci Texy bylo umožnit implementace výstupů do jiných formátu, než je HTML. Samotný charakter tohoto jazyka jako jazyka pro zápis interne-

[1] Zjištěny pomocí pluginu *Cobertura* do Mavenu.

tových dokumentů toto sice znesnadňuje, ne však znemožňuje.

Modelovým kandidátem pro výstupní formát je PDF[2]. Tento formát je standardem pro přenášení dokumentů tak, aby si zachovávaly totožný vzhled bez ohledu na použité zobrazovací zařízení (tiskárna, obrazovka počítače, ...) či platformu. I když existují i další formáty, které by mohly být podobně atraktivní (např. PostScript), budu pro zjednodušení dále uvažovat PDF jako model pro alternativní výstupní formát.

Problém HTML

Hlavní problém implementace PDF výstupu je to, že Texy umožňuje vkládat části HTML kódu. HTML kód, který je takto vložený do stránky, není možné přímo vložit do PDF. Texy také umožňuje aplikovat na určitou část dokumentu kaskádové styly, které opět nejdou přímo převést (ještě komplikovanější je případ, kdy se autor dokumentu odkazuje na třídu kaskádového stylu definovanou v externím souboru).

Existuje několik možností, jak tuto vlastnost obejít:

Ignorování – je možné implementovat generátor výstupu prostě tak, že bude HTML značky ignorovat a pouze zobrazovat jejich obsah. Toto je bezkonkurenčně nejjednodušší řešení, ale také nejméně vhodné. HTML kód může obsahovat důležité informace o dokumentu, bez kterých by byl výsledný dokument znehodnocený. Na druhou stranu je možné uživatele alespoň v dokumentaci varovat, že použití HTML značek může znamenat, že smysluplný výstup do jiných formátů nebude možný.

Mód kompatibility – bylo by možné zavést nastavení, které by přepnulo JTexy do „módu kompatibility“. V tomto módu by Texy neakceptovalo HTML značky ani připojené kaskádové styly. Toto by také znamenalo změnu v API modelu, do kterého by byla zavedena generalizace některých prvků dokumentů. Například z rozhraní `Modifiers` by bylo možné extrahovat nadrozhraní `CompatibleModifiers`, které by obsahovalo pouze podmnožinu vlastností – těch, které mají smysl i mimo kontext HTML. Generátor výstupu by pak definoval *úroveň kompatibility*, která by určovala, zda je schopen přijímat dokumenty, které obsahují informace specifické pro formát HTML.

Přijetí HTML – poslední, pravděpodobně implementačně nejnáročnější možnost je, že by generátor výstupu byl schopen HTML značky rozpoznat, a na jejich základě se pokusit o sestavení podobného vzhledu. Ne všechny konstrukce však lze zcela přenést. Také by to znamenalo vynaložit poměrně značné úsilí, protože je nutné rozpoznat nejen vložený HTML kód, ale i vhodně aplikovat styly, a to dokonce i ty uvedené v externím souboru. Implementace takové funkcionality by mohla náročností přerůst implementaci samotné knihovny JTexy jako takové.

Pomoci by mohl podobný princip, jaký používá Texy, a to že i HTML značky jsou v dokumentu rozpoznány v době překladu. Tím by se zjednodušila práce generátoru.

Nejlepší se zdá být kompromisní řešení mezi módem kompatibility a přijímáním HTML. Pravděpodobně by bylo možné některé prostší HTML značky rozpoznat a převést na obecnější rovinu, aby byla informace o formátovaní použitelná i z generátoru PDF výstupu. Tím je možné dosáhnout relativně očekávané funkčnosti, která nebude neúnosně náročná na implementaci.

[2]Portable Document Format

5.2.2 Modularita

Námětem na další vývoj může být zavedení modulární architektury, jako má původní implementace v Texy. Tato architektura by mohla umožňovat definovat nové konstrukty jazyka jako zásuvné moduly pomocí veřejného rozhraní. Tímto je dána možnost rozšiřovat samotnou knihovnu o funkcionality, které Texy neobsahuje, a to zcela v režii třetích stran. Také to umožňuje uživateli-programátorovi zvolit, které prvky jazyka chce pro překlad dokumentu využít.

Dodatek A

Seznam příloh

1. **Texy.g** – text zjednodušené gramatiky (zdrojový kód pro nástroj ANTLR, 6 stran)
2. **INSTALL** – návod k překladu a instalaci (1 strana)
3. **README** – návod k použití převaděče JTexy spouštěného z příkazové řádky (1 strana)

Literatura

[1] Výslovnost Texy! [Online; navštíveno 5.5.2008].
URL http://forum.texy.info/viewtopic.php?id=137

[2] ISO/IEC 14977. 1996, [Online; navštíveno 7.5.2008].
URL http://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf

[3] Alignment, font styles, and horizontal rules in HTML documents. 2008, [Online; navštíveno 27.4.2008].
URL http://www.w3.org/TR/html401/present/graphics.html#edef-FONT

[4] Java Technology EOL Policy. 2008, [Online; navštíveno 7.5.2008].
URL http://java.sun.com/products/archive/eol.policy.html

[5] JUnit.org. 2008, [Online; navštíveno 28.4.2008].
URL http://junit.org/

[6] Maven. 2008, [Online; navštíveno 28.4.2008].
URL http://maven.apache.org/

[7] wiki – Britannica Online Encyclopedia. 2008, [Online; navštíveno 27.4.2008].
URL http://www.britannica.com/eb/article-9404276/wiki

[8] Aho A. V. et. al: *Compilers: Principles, Techniques and Tools*. Addison-Wesley, 1986, ISBN 0-201-10088-6.

[9] Grudl, D.: Texy. 2008, [Online; navštíveno 27.4.2008].
URL http://texy.info/cs/

[10] Gunderloy, M.: *Z kodéra vývojářem*. Computer Press, 2007, ISBN 978-80-251-1517-6.

[11] Lasater, C. G.: *Design Patterns*. Wordware Publishing, 2007, ISBN 1-59822-031-4.

[12] Lindholm T. et. al: VM Spec The class File Format. 1999, kap. 4.10 Limitations of the Java Virtual Machine [Online; navštíveno 27.4.2008].
URL http://java.sun.com/docs/books/jvms/second_edition/html/ClassFile.doc.html

[13] Object Mentor: The Open-Closed Principle. [Online; navštíveno 4.5.2008].
URL http://www.objectmentor.com/resources/articles/ocp.pdf

[14] Parr, T.: *The Definitive ANTLR Reference*. The Pragmatic Programmers, 2002, ISBN 978-09787392-4-9.

[15] Parr, T.: ANTLR Parser Generator. 2008, [Online; navštíveno 18.4.2008]. URL http://www.antlr.org/

[16] Wirth, N.: *Compiler Construction*. 2005, ISBN 0-201-40353-6.