Mendelova univerzita v Brně
Provozně ekonomická fakulta

Návrhář uživatelských stylů v aplikaci TEXonWeb

Diplomová práce

Vedoucí práce:
Ing. Jan Přichystal, Ph.D.

Bc. Pavel Potáček

Brno 2016

Stránka se zadáním práce

Děkuji vedoucímu práce Ing. Janu Přichystalovi, Ph.D. za velkou ochotu, odborné rady a cenné připomínky.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Návrhář uživatelských stylů v aplikaci TEXonWeb**
vypracoval samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací.*
Jsem si vědom, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.
Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 2. ledna 2016 .....

Abstract

Potáček, P. User styles designer in application TEXonWeb. Diploma thesis. Brno, 2016.

The diploma thesis deals with design and implementation of user styles designer in the application TEXonWeb. It contains the analysis of current state of the application and development of its components. It describes possibilities of typesetting in TEX/LATEX system, mainly in the area of creating user styles. It provides comparison of some similar oriented systems. It also includes the theory of formal languages and compilers. Part of the work is description of the interface design, implementation of styles designer and styles parser. There are mentioned possibilities of next development too.

Abstrakt

Potáček, P. Návrhář uživatelských stylů v aplikaci TEXonWeb. Diplomová práce. Brno, 2016.

Diplomová práce se zabývá návrhem a implementací návrháře uživatelských stylů v aplikaci TEXonWeb. Obsahuje analýzu současného stavu aplikace a tvorbu jejích komponent. Dále popisuje možnosti sazby v typografickém systému TEX/LATEX s ohledem na tvorbu uživatelských stylů. Poskytnuto je srovnání s dalšími podobně zaměřenými systémy. Probrána je také teorie formálních jazyků a překladačů. Součástí práce je popis návrhu rozhraní, vlastní implementace návrháře a parseru stylů. Uvedeny jsou i možnosti dalšího vývoje.

Obsah

1 Úvod a cíl práce

1.1 Úvod

Koncem 20. století došlo k hromadnému zavádění výpočetní techniky do různých oblastí lidské činnosti. Jednou z nich byla i oblast zpracování textů. Počítač s vhodným programovým vybavením dokáže nahradit dříve používané složité technologie a zároveň významně urychlit proces přípravy a tisku dokumentů.

Pro vytváření dokumentů je však zapotřebí program, který dokáže pracovat s textem. Takových programů existuje nepřeberné množství, liší se ale funkcemi i cenou. Nejjednodušším typem jsou textové editory, které se používají převážně pro zápis zdrojových kódů nebo jednoduchých textů. Mezi ně patří například Poznámkový blok dostupný v systému Microsoft Windows. O něco propracovanější jsou textové procesory, které dovolují uživateli měnit vzhled zpracovávaného textu nebo přidávat další prvky jako jsou obrázky nebo tabulky. Mezi textové procesory lze zařadit třeba Microsoft Word. Nejpokročilejší software pro zpracování textů jsou DTP[1] systémy. Ty umožňují profesionální sazbu dokumentu díky rozsáhlé kontrole nad tvořeným textem a následně jeho výstupem. Do této kategorie se řadí systémy jako TEX nebo Adobe InDesign.

K běžným úkonům, které se při tvorbě dokumentů využívají, patří úprava vzhledu textu. Jedná se v prvé řadě o vizuální odlišení různých strukturních celků, jako jsou nadpisy, odstavce, aj. Tyto úpravy lze provádět přímo v textu, kdy je možné jednotlivým elementům nastavit manuálně specifické vlastnosti. Ve většině případů je však vhodnější variantou nastavit vzhled pomocí stylů. Hlavní motivací, proč styly používat, je oddělení vizuální části od obsahu. To znamená, že při změně vlastnosti některého celku není nutné procházet celý dokument a ve všech výskytech danou vlastnost upravovat, ale stačí tuto vlastnost změnit u připojeného stylu. Díky tomu se stejné styly dají snadno použít i v jiných dokumentech. Kromě vzhledu mohou styly nést i sémantickou informaci. Např. definováním více úrovní nadpisů lze dokument rozdělit na kapitoly a podkapitoly, které pak vytváří určitou hierarchii a umožňují se v něm lépe orientovat.

Uživatelé znají styly především z textového procesoru Microsoft Word, ve kterém existuje speciální grafické rozhraní, kde je možné styly poměrně přehledně spravovat. Při úpravě textu je tak změna stylu určitého celku otázkou pouhého kliknutí. Toto ocení zvláště začínající uživatelé, pro které to znamená pouze vědět, kde je možné co nastavit.

Zcela opačná je však situace v DTP systému TEX. Jedná se o systém využívaný především pro profesionální sazbu textu, proto je třeba určitá znalost maker a jejich správného zápisu. V systému TEX/LATEX žádné grafické rozhraní pro tvorbu stylů neexistuje, je nutné si je ručně napsat definováním vlastních maker. Lze sice využít předdefinované styly, které nabízí např. nadstavba LATEX, ovšem ne vždy je jejich

[1] Zkratka pro „Desktop publishing“

podoba vyhovující. Právě tato vstupní bariéra může odradit začínající uživatele od používání stylů v TEXu, díky kterým by si mohli značně ulehčit práci.

Tento problém se snaží řešit návrhář uživatelských stylů, který je dostupný ve webové aplikaci TEXonWeb[2]. Jde se o aplikaci, která umožňuje využívat DTP systém TEX/LATEXprostřednictvím webového prohlížeče. Odpadá tak nutnost složitě instalovat a nastavovat TEX na vlastním počítači, díky čemuž se tento systém stává mnohem přístupnější uživatelům. Aplikace TEXonWeb se zaměřuje převážně na začínající uživatele, pro které je dispozici spousta nástrojů usnadňující tvorbu dokumentu v TEXu.

Návrhář stylů disponuje uživatelským rozhraním, které umožňuje jednoduchou tvorbu stylů, na jakou jsou uživatelé zvyklí právě z MS Word. Díky tomu si mohou i uživatelé, kteří úplně neovládají syntaxi TEXu, vytvářet vlastní styly a používat je ve svých dokumentech.

1.2 Cíl práce

Cílem práce je vytvoření návrháře uživatelských stylů jako komponenty pro webovou aplikaci TEXonWeb. Návrhář umožní uživatelům vytvářet nové styly s vybranými parametry, které pak mohou být vkládány do editoru nebo ukládány do souboru. Pomocí návrháře bude také možné opětovně načíst vygenerované styly a umožnit jejich další úpravu. Součástí práce bude analýza současného stavu aplikace TEXonWeb. Dále bude práce obsahovat seznámení se s možnostmi formátování textu v systému TEX/LATEX i jiných textových procesorech a DTP systémech. Na základě toho bude navržena množina vhodných parametrů pro formátování písma a odstavců. Také bude vypracován návrh rozhraní návrháře s ohledem pro použití na mobilních zařízeních. Navržené řešení bude implementováno a na závěr provedeno zhodnocení.

[2]Dostupné na `http://tex.mendelu.cz`

2 Metodika a postup řešení

Při tvorbě návrháře uživatelských stylů bude nejprve nutné prozkoumat stávající aplikaci TEXonWeb a zjistit, jak jsou řešeny všechny komponenty, které aplikace obsahuje. Vzhledem k tomu, že na aktuálních zdrojových kódech pracovalo již mnoho autorů v rámci závěrečných prací a v budoucnu budou další jistě pokračovat, bude nutné navrhnout strukturu, která bude srozumitelná a snadno udržovatelná. Navržená struktura musí být vhodná i pro souběžnou práci několika vývojářů, což je běžně se vyskytující jev. Celá aplikace využívá systém pro správu a verzování kódu SVN[3], přesto je potřeba kód vhodně rozdělit, aby nedocházelo ke zbytečným kolizím, což by mohlo výrazně komplikovat další vývoj.

Vývoj a testování návrháře bude probíhat na serveru `science.mendelu.cz`, kde má každý vývojář TEXonWeb vytvořen vlastní účet. Tato část je oddělena od hlavní aplikace, kterou uživatelé používají, proto zde provedené změny na ni nemají žádný vliv. Zdrojové kódy se ukládají do adresáře `public_html`, který je navíc dostupný z Internetu, veškeré změny tak lze testovat přímo z prohlížeče. Aktuálně se používá vývojová větev systému SVN s názvem „upgrade“, URL adresa testovací části je tedy `https://tex.mendelu.cz/~potty/upgrade/`. Veškeré provedené změny je pak po otestování nutné commitem odeslat do centrálního repozitáře SVN, odkud se mohou změny následně dostat do hlavní verze.

Po analýze současného stavu aplikace bude třeba prozkoumat způsoby formátování textu v sytému TEX/LATEX. Bude nutné zjistit, jak se v tomto systému vytvářejí a aplikují uživatelské styly, které by pak měl návrhář vytvářet. Důležité bude také podívat se na možnosti jiných DTP systémů a textových procesorů a stávající situaci porovnat. Hlavně MS Word obsahuje grafické rozhraní pro správu stylů, které je velkému množství uživatelů povědomé, proto by návrhář stylů měl vypadat obdobným způsobem.

Jelikož návrhář bude obsahovat i zpětné načítání a modifikaci stylů, bude nutné navrhnout řešení, které umožní tuto funkcionalitu realizovat. Toto řešení by mělo zvládnout analýzu stylu, zkontrolovat správnost syntaxe, případně upozornit na vyskytující se chyby. Teprve poté, co bude analýza úspěšná, dojde k načtení stylu do návrháře, kde bude možné tento styl dále modifikovat. Bude tedy potřeba navrhnout a vytvořit parser stylů. Aby jej však bylo možné sestrojit, bude nutné probrat teorii z oblasti formálních jazyků a automatů, čímž získáme potřebné informace.

Před návrhem struktury a rozhraní návrháře bude nutné vybrat vlastnosti, které bude samotný návrhář obsahovat. Vzhledem ke komplexnosti systému TEX je nemožné poskytnout veškeré dostupné vlastnosti. Vybrány proto budou pouze základní a nejpoužívanější vlastnosti, které budou rozděleny do kategorií písmo a odstavec.

Po zvolení vlastností přijde na řadu návrh struktury a rozhraní návrháře stylů. Uživatelské rozhraní bude nutné navrhnout tak, aby bylo snadno použitelné jak na desktopu, tak na mobilních zařízeních. Bude potřeba vyřešit způsob zadávání

[3] Apache Subversion

hodnot a navigaci při procházení mezi jednotlivými obrazovkami návrháře, aby byla zachována jednoduchost, přehlednost a vše bylo intuitivní.

Jako poslední krok při tvorbě návrháře bude řešena implementace, kde bude popsána tvorba klientské části, parseru stylů a jejich vzájemné propojení. Na závěr bude celé řešení zhodnoceno a probrány možnosti dalšího rozšíření.

3 TEXonWeb

3.1 Analýza současného stavu

Jak již bylo zmíněno v úvodu, webová aplikace TEXonWeb umožňuje využívat typografický systém TEX v prostředí webového prohlížeče. Hlavní výhoda tohoto přístupu je ta, že odpadá nutnost nesnadné instalace a konfigurace systému TEX na vlastním počítači. Jak uvádí (Přichystal, 2015a), TEXonWeb také umožňuje používat tento typografický systém na mobilních zařízeních s operačními systémy Android a iOS, kde jinak jeho možnosti nejsou příliš široké. Díky TEXonWeb je jednotné rozhraní a stejná funkcionalita dostupná pro všechny platformy (Přichystal, 2014).

Aplikace TEXonWeb vznikla pro potřeby předmětu Zpracování textů na počítači, který se vyučuje každý semestr na Mendelově univerzitě v Brně a zapisuje si ho přibližně 300 studentů. Díky ní je vyřešena spousta problémů s programovým vybavením pro práci se systémem TEX, jež v minulosti trápily mnoho studentů. Každý má k dispozici stejné prostředí ať už pracuje na školním počítači nebo doma. Vyřešen je také problém s přenositelností vytvořených souborů, jelikož každý uživatel může mít vlastní účet, na kterém má k dispozici úložný prostor (Přichystal, 2015b).

Hlavním elementem celé aplikace je editor zdrojového kódu dokumentu, do kterého se zapisují klasické TEXové nebo LATEXové konstrukce. Z takto vytvořeného dokumentu lze jednoduše stisknutím tlačítka PDF získat dokument v uvedeném formátu, který se prostřednictvím prohlížeče stáhne. V případě že dojde k chybě v překladu, lze pomocí tlačítka log zobrazit logový soubor, kde je možné nalézt veškeré informace o průběhu překladu.

Mimo základní editor zdrojového kódu nabízí aplikace také nástrojovou lištu, na které jsou umístěny nástroje usnadňující tvorbu dokumentu v systému TEX. Najdeme zde např. nástroj pro vyhledávání, kontrolu pravopisu, zkratky pro rychlé formátování textu (tučné písmo, kurzíva, kapitálky, strojopis) a dále také průvodce pro tvorbu tabulek, vkládání obrázků a seznamů. Velmi užitečným pomocníkem je i nástroj pro vkládání nezalomitelných mezer, které se vkládají pomocí znaku tildy za jednopísmennou předložku, aby nemohla zůstat osamocená na konci řádků. Pro usnadnění psaní speciálních znaků je k dispozici zvláštní lišta, což se hodí zejména na mobilních zařízeních. Pro tato zařízení jsou zde i tlačítka umožňující rychlou navigaci v rámci editoru a přesun kurzoru.

TEXonWeb disponuje širokou škálou dokumentů, které mají již předdefinovaný styl. Tyto dokumenty lze nalézt v nabídce šablony. Na výběr jsou styly pro dopisy, žádosti, vizitky, závěrečné práce a jejich posudky, obaly CD a DVD, kalendář, životopis a prezentace (Přichystal, 2014).

Veškeré výše uvedené funkce lze využívat zcela neomezeně i bez uživatelského účtu. Pro využití všech možností aplikace je ale vytvoření účtu potřeba. Přihlášený uživatel si může prostředí editoru a volby překladače upravit podle svých preferencí a ty jsou pak uloženy v jeho profilu. Takto je možné ovlivnit typ překladače, počet překladů, velikost písma editoru, barevný motiv a výchozí obsah.

Přihlášenému uživateli se také v hlavní nabídce zpřístupní položka soubory, pod kterou se skrývá rozhraní pro pohodlnou správu souborů. Zde jsou k dispozici běžné operace pro manipulaci se soubory, ale lze tu nalézt také možnost pro sdílení souborů s jinými uživateli pro usnadnění spolupráce více uživatelů na jednom dokumentu.

3.2 Použité technologie

Webová aplikace TEXonWeb je rozdělena na část klientskou, která zprostředkovává uživatelské rozhraní a část serverovou, která zajišťuje běh celé aplikace.

Klientská část aplikace je implementována v programovacím jazyku JavaScript. Díky tomu se celé uživatelské rozhraní a komponenty načtou pouze jednou při spuštění, což zvyšuje uživatelský komfort a navozuje obdobný pocit, jako by uživatel pracoval s klasickou desktopovou aplikací. Hojně využívána je technologie AJAX, která umožňuje zasílání asynchronních požadavků na pozadí, aniž by došlo k opětovnému obnovení stránky. Výjimku tvoří pouze otevírání souborů, při kterém dojde znovu k načtení stránky.

Pro usnadnění vývoje v JavaScriptu je ve velké míře použita knihovna jQuery. Ta odstraňuje nekompatibilitu mezi rozdílnými prohlížeči, jelikož každé JavaScriptové jádro může daný kód interpretovat různě. Mimo to také usnadňuje manipulaci s objektovým modelem dokumentu (DOM), zasílání AJAXových požadavků, zpracování událostí a tvorbu animací a efektů. Pro tvorbu uživatelského rozhraní byla využita také knihovna jQuery UI, která nabízí sadu grafických widgetů zahrnující složité ovládací prvky jako slider, dialogové okno nebo přepínač. Tyto widgety jsou implementovány pomocí jQuery. Jejich nevýhoda však spočívá v tom, že nedokáží zpracovat události vyvolané pomocí dotykové obrazovky. K tomuto účelu je využita knihovna jQuery UI Touch Punch, která zmíněný nedostatek odstraňuje a díky ní je možné používat rozhraní TEXonWeb i na mobilních zařízeních (Potáček a Přichystal, 2014).

Jako editor je použit Ace, který je napsán v JavaScriptu, snaží se napodobit vlastnosti a funkce desktopových editorů jako Vim nebo Eclipse a jeho API poskytuje široké možnosti pro manipulaci s obsahem. Jeho integrací bylo nahrazeno původní vlastní řešení, které z důvodu nekompatibility s moderními prohlížeči přestalo být dostačující (Vybíhal a Přichystal, 2014).

Dále pro tvorbu struktury dokumentu byl použit značkovací jazyk HTML5, který přináší nové elementy a výrazně rozšiřuje API např. o možnost ukládat data webu do paměti prohlížeče (Web Storage). Vzhled aplikace je tvořen pomocí CSS3, ze kterého je využito hlavně zaoblení rohů elementů, průhlednost, stíny, animace a přechody. Pro přizpůsobení aplikace mobilním zařízením byly nápomocné tzv. Media Queries, které dovolují definovat různé vlastnosti kaskádových stylů v závislosti na velikosti okna prohlížeče (Vybíhal a Přichystal, 2014).

Co se týče serverové části aplikace, na její implementaci byl použit skriptovací jazyk Perl využívající modul CGI. V této části aplikace je řešeno sestavování stránky,

správa souborů a autentizace. Probíhá zde také překlad TEXového kódu a generování dokumentu PDF o které se stará distribuce TEXLive 2013 (Přichystal, 2014).

3.3 Komponenty

Aplikace TEXonWeb obsahuje několik komponent, které lze najít v nástrojové liště. Komponentou se rozumí rozšiřující prvek, který přidává určitou funkcionalitu. Jedná se především o průvodce (wizard), což je uživatelské rozhraní, které postupně provede uživatele sérií několika kroků, ve kterých může být vykonána nějaká akce a na konci je k dispozici výsledek provedených akcí. Tito průvodci slouží ke snadné tvorbě LATEXových konstrukcí, které by se jinak musely psát kompletně manuálně. To je přínosné hlavně pro začínající uživatele, kteří ještě nemusí zápis těchto konstrukcí dokonale ovládat. Pomocí grafického rozhraní si mohou naklikat požadované parametry a průvodce na jejich základě vygeneruje výsledný kód a vloží přímo do editoru. Aktuálně aplikace obsahuje nástroje pro kontrolu pravopisu, návrh tabulek, vkládání obrázků a vkládání seznamů.

3.3.1 Kontrola pravopisu

Nástroj pro kontrolu pravopisu umožňuje odhalit pravopisné chyby v česky psaném textu. Při kontrole jsou chybná slova podtržena červeně, zároveň se zobrazí uživatelské rozhraní, pomocí kterého je možné tyto chyby opravit. Aktuálně opravované slovo je podbarveno žlutě. V levé části rozhraní je vypsán seznam navrhovaných oprav, které je možné po dvojitém kliknutí aplikovat. V pravé části jsou potom tlačítka umožňující provést další akce s aktuálním slovem. Zde je možné nástroj naučit dané slovo, kdy dojde k jeho uložení do uživatelského slovníku, případně ignorovat. Jsou zde i tlačítka pro cyklické přesouvání mezi chybami.

3.3.2 Návrhář tabulek

Návrhář tabulek je nástroj řešící problém ne úplně triviální tvorby tabulek v typografickém systému TEX. Uživatel si nejprve zvolí, kolik bude mít daná tabulka sloupců a řádků. Poté je zobrazeno rozhraní, které umožňuje navrhnout tabulku stylem WYSIWYG[4], znamenající „co vidíš, to dostaneš“. V horní části rozhraní je zobrazena tabulka, do jejíchž buněk lze vepisovat požadované hodnoty. Tabulku je dále možné upravovat přidáváním či odebíráním sloupců nebo řádků. Také lze měnit ohraničení jednotlivých buněk nebo jejich podbarvení. Nastavit jde i plovoucí prostředí, u kterého se dají vyplnit hodnoty pro parametry, kam jej vložit, text popisku a název návěští. TEXový kód tabulky si lze průběžně zobrazit v náhledu. Po skončení všech úprav pak jde kód tabulky snadno vložit do editoru.

[4]Zkratka pro „What you see is what you get“

3.3.3 Vkládání obrázků

Jak už napovídá název, tento nástroj usnadňuje vkládání obrázků do dokumentu. Pomocí tohoto rozhraní je možné nahrát obrázek z lokálního disku do úložného prostoru v rámci účtu na TEXonWeb. Z tohoto důvodu mohou vkládání obrázků využívat pouze přihlášení uživatelé. Nahraný obrázek je automaticky nastaven jako aktuálně vybraný. Poté je možné nastavit vlastnosti plovoucího prostředí jako je umístění na stránce, text popisku a návěští. Je možné definovat i rozměry obrázku nebo úhel natočení.

3.3.4 Tvorba komponent

V nedávné době prošla aplikace TEXonWeb poměrně zásadním přepracováním. Spolu se zavedením open source editoru Ace došlo také ke změně struktury kódu. Původně nebyl JavaScriptový kód nijak organizovaný a téměř vše bylo definováno na globální úrovni, což značně komplikovalo orientaci a další rozšiřování. Po zavedení úprav je kód sdružen a organizován do logických celků, které tvoří jednotlivé části aplikace. Využito bylo implementace pomocí objektového literálu, což je nejjednodušší forma zapouzdření kódu v JavaScriptu, která eliminuje používání anonymních funkcí, sjednocuje nastavení vlastností a usnadňuje znovupoužitelnost a refaktorizaci. Na globální úrovni je definován objekt pojmenovaný `TOW`, jehož vlastnosti (properties) obsahují jednotlivé komponenty TEXonWeb. Díky tomu je možné ke komponentám přistupovat globálně a zachovat volnou vazbu, kdy na sobě komponenty nejsou závislé a komunikují mezi sebou pouze pomocí systému událostí (Potáček a Přichystal, 2014).

3.4 Aktuální možnosti tvorby stylů

V současnosti je tvorba stylů v aplikaci TEXonWeb omezena jen na ruční definování vlastních maker. Toho lze docílit pomocí příkazu `\def`. Aplikace je využívána především v rámci předmětu Zpracování textů na počítači, který studuje spousta začínajících uživatelů a studentů neinformatických oborů. Pro ně je tento způsob tvorby vlastních maker komplikovaný a způsobuje jim problémy. Uživatelé se často dopouští mnoha chyb, jako je třeba zapomenutá závorka na konci definice, což znemožní překlad dokumentu. Uživatelé jsou pak z tohoto postupu frustrovaní a snaží se mu vyhnout. Při vytváření stylů pomocí grafického rozhraní v programu Microsoft Wordu naopak problémy většinou nenastávají. Návrhář stylů by tedy byl v aplikaci TEXonWeb ideálním pomocníkem, jak těmto uživatelům usnadnit tvorbu vlastních maker a zároveň sloužil jako návod, jak v této problematice postupovat. Vzhledem a funkcionalitou by připomínal rozhraní tvorby stylů v MS Word, které je spoustě uživatelů známé a dokáží s ním bez problémů pracovat (Přichystal, 2015b).

4 TEX

4.1 Charakteristika systému

TEX je volně šiřitelný typografický systém na pořizování vysoce kvalitní elektronické sazby. Byl vytvořen v roce 1978 americkým profesorem Donaldem E. Knuthem na Stanfordské univerzitě. Sám autor uvádí, že vznikl kvůli tvorbě hezkých knih, především pak technických publikací obsahujících spoustu matematických výrazů (Mittelbach a kol., 2004).

TEX se řadí mezi značkovací jazyky. Do textového souboru obsahující dokument se zapisují příkazy, které ovlivňují sazbu. Pomocí speciálních znaků se rozlišují řídicí konstrukce od běžného textu, takovým příkladem je třeba zpětné lomítko uvozující jednotlivé příkazy.

Původním záměrem bylo, aby TEX nebyl závislý na platformě a fungoval všude identicky. Proto ze vstupního dokumentu vznikl překladem soubor typu DVI, který je abstraktní, popisující stránku jako soubor znaků a příslušných fontů s údajem o jejich pozici na stránce. Díky tomu je tento soubor naprosto nezávislý na výstupním zařízení. K jeho zobrazení je však potřeba specializovaný program (DVI ovladač). V dnešní době je však upřednostňováno přímo generování souboru ve formátu PDF, což umožňují implementace TEXu jako pdfTEX nebo XETEX.

TEX obsahuje kolem 300 vestavěných příkazů (primitiv), které jsou však příliš jednoduché a sázení dokumentu pouze s jejich využitím je velmi pracné. Tento problém je vyřešen tvorbou tzv. *maker*, což jsou nové příkazy definované na základě již existujících. Základní balík maker pro TEX vytvořil sám autor Knuth a pojmenoval jej PlainTEX. Ten je již dále neměnný, dochází v něm pouze k opravám chyb. Různých balíků maker existuje velké množství, ale nejznámější a nejrozšířenější je jednoznačně LATEX. Snaží se o zjednodušení sazby dokumentů zavedením příkazů pro členění textu do kapitol, generování obsahu nebo vkládání tabulek a obrázků (Satrapa, 2011).

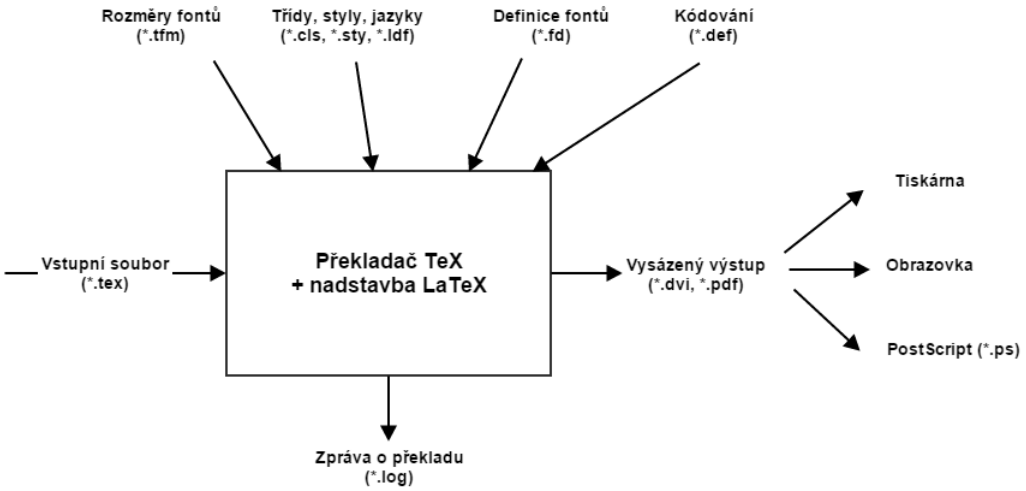
4.1.1 Práce systému

Jádro systému je tvořeno překladačem jazyka TEX (viz obrázek 1). To dostane na vstup textový soubor (obvykle s příponou `.tex`) vytvořený v libovolném editoru a obsahující text dokumentu společně s příkazy, které definují sazbu textu.

Primárním úkolem překladače je rozmístit jednotlivé znaky do sazebního zrcadla. K tomu je nutné znát rozměry jednotlivých znaků, které jsou brány jako pouhé obdélníky, jejichž rozměry jsou uloženy v souboru s příponou `.tfm`. Během překladu se vytváří logový soubor s příponou `.log`, který obsahuje veškeré informace o jeho průběhu. Celkový průběh a různá chybová hlášení se vypisují na standardní výstup. Konečným výstupem překladače je poté soubor typu DVI, jehož obsah není závislý na zobrazovaném zařízení. Pro jeho zobrazení je třeba použít speciální program nazývaný DVI ovladač, díky němuž je možné zobrazit výstup na obrazovku

nebo vytisknout. Výstup lze získat také v jazyce PostScript, který slouží pro vytváření vektorové grafiky. Používán je však také pro popis stránky v typografických systémech (Rybička, 2003).

Velmi užitečným je také výstupní formát PDF, který vyvinula firma Adobe. Ten je možné získat buď převodem z formátu PostScript nebo použitím speciálního překladače pdfTEX přímo ze vstupního souboru. Tento formát je oblíbený zejména pro svoji přenositelnost, jelikož není závislý na hardwaru ani softwaru. Navíc pro něj existuje velké množství zdarma dostupných prohlížečů a lze jej zobrazit také přímo v internetovém prohlížeči v rámci webové stránky.



Obrázek 1: Schéma datového toku v systému LATEX (Mittelbach a kol., 2004)

Celkově vzato je tvorba dokumentu v TEXu velmi podobná programování. Nejprve se připraví nebo modifikuje zdrojový text, ten se přeloží a výsledek si pak lze prohlédnout. Tuto sekvenci kroků je třeba opakovat tak dlouho, dokud není dokument vysázen ve vyhovujícím tvaru. Tento způsob tvorby má však jednu významnou nevýhodu a tou je nemožnost vidět výsledný tvar dokumentu v okamžiku provádění úprav ve zdrojovém souboru. To může odradit hlavně začínající uživatele, kteří byli dosud zvyklí používat WYSIWYG editory nebo textové procesory jako MS Word. Tato možnost přímé úpravy výsledného tvaru textu je však vykoupena tím, že neumožňuje dosáhnout takové přesnosti a kvality sazby jako právě v systému TEX. To však nemusí být zřejmé hned na obrazovce, ale až po vytisknutí na papíře. Nicméně i tento nedostatek se snaží alespoň částečně řešit některé editory TEXu jako např. Overleaf, který zobrazuje výsledný tvar textu vedle editoru kódu a překlad probíhá v průběhu psaní dokumentu. Naopak mezi výhody TEXu patří (Rybička, 2003):

- Možnost výběru libovolného textového editoru.
- Možnost automatizované úpravy zdrojového textu (např. textové filtry).
- Přizpůsobivost změnám prostředí (volba jiného překladače).

- Tvorba skriptů usnadňujících práci.
- Snadná přenositelnost textů, využití verzovacích systémů.

4.1.2 Proces zpracování vstupu

Systém TeX na svém vstupu přijímá textový soubor a výstupem je soubor DVI (případně jiný formát). Aby bylo možné lépe pochopit celé fungování tohoto procesu, je vhodné si podle (Eijkhout, 2013) TeX pomyslně rozdělit na samostatné procesory, které si mezi sebou předávají data.

- *Input procesor* – zpracovává řádky vstupního souboru do podoby nezávislé na operačním systému.
- *Token procesor* – na vstup dostane řádky z input procesoru, ze kterých vytváří posloupnost tokenů. Rozlišujeme dva druhy těchto tokenů. Prvním je uspořádaná dvojice (ASCII hodnota, kategorie), kde kategorie je číslo v intervalu 0 až 15, které určuje další chování tohoto tokenu v rámci zpracování. Druhým typem je řídicí sekvence.
- *Expand procesor* – na vstup dostane jednotlivé tokeny, které rozděluje podle toho, které se budou expandovat a které zůstanou nezměněny. Beze změn zůstanou znaky nebo primitivní řídicí sekvence. Expanze tokenu znamená nahrazení tokenu posloupností jiných tokenů. Probíhá tak dlouho, dokud nejsou všechny tokeny znaky nebo primitivní řídicí sekvence.
- *Hlavní procesor* – rozlišuje na vstupu přesně definovanou množinu tzv. povelů. Pomocí těchto povelů je pak provedena sazba. Hlavní procesor také provádí výstup do DVI.

4.2 LaTeX

Systém LaTeX je volně šiřitelná nadstavba typografického systému TeX, která byla vytvořena v roce 1985 americkým informatikem Leslie Lamportem. LaTeX poskytuje sadu příkazů umožňujících sazbu dokumentů i běžným uživatelům, kteří nemají takové znalosti jako typografičtí profesionálové, přesto chtějí využít všech výhod a možností, které nabízí TeX a produkovat dokumenty vysoké kvality.

Systém LaTeX je značkovací jazyk, pomocí kterého lze popsat spíše logickou strukturu obsahu. Znamená to tedy, že určuje, co se bude sázet, nikoliv jak. Poskytuje např. příkazy pro tvorbu nadpisů, sekcí, obsahů nebo citací. Autor dokumentu neví, jak jsou tyto strukturní prvky naformátovány. Veškeré tyto informace jsou uloženy ve třídě dokumentu, která je definovaná úvodním příkazem `\documentclass` (Kopka a Daly, 2004).

Od svého uvedení prochází LaTeX neustálým vývojem. Po mnoho let byla hojně využívána verze 2.09, která však obsahovala řadu nedostatků a v roce 1992 byl vývoj této verze ukončen. Mezitím již probíhaly práce na verzi 3. Jelikož její vývoj trval

poměrně dlouhou dobu, byl vytvořen mezistupeň s názvem LaTeX 2_ε. Nicméně ani do dnešní doby není verze 3 hotova, proto je současným standardem stále verze LaTeX 2_ε (Satrapa, 2011).

4.2.1 Struktura dokumentu

Každý LaTeXový soubor obsahuje *preambuli* a *textovou část*. V preambuli se vykytuje sada globálních příkazů, které ovlivňují celý text. Pomocí nich lze nastavit třeba rozměr stránky. Jako úplné minimum musí preambule obsahovat příkaz `\documentclass`, který specifikuje celkový styl sazby dokumentu. Tento příkaz musí být zároveň uveden jako první. Dále je zde možné uvést definice nových maker nebo připojit další pomocné balíky maker.

Balíky představují důležitou součást LaTeXu, jelikož rozšiřují výchozí nabídku dostupných příkazů. Jedná se o soubory s příponou `.sty`, které jsou načteny pomocí příkazu `\usepackage`.

Preambule končí příkazem `\begin{document}`. Vše co dále následuje je součástí textové části. Ta obsahuje samotný text dokumentu a příkazy. Oproti preambuli mají tyto příkazy pouze lokální význam, vztahují se tedy pouze na určitou část textu. Textová část je ukončena příkazem `\end{document}` (Rybička, 2003).

4.2.2 Příkazy

Příkazem se rozumí instrukce pro LaTeX, aby provedl nějakou zvláštní akci jako třeba výpis některého symbolu, změnu písma nebo jiné vlastnosti pro formátování textu. Podle (Kopka a Daly, 2004) existují tři typy příkazů:

- Samostatné znaky mající speciální význam. Jedná se o znaky `#`, `$`, `&`,`~`, `_`,`^`, `%`, `{`, `}`.
- Příkazy začínající znakem zpětného lomítka (\) a jednoho znaku, který není písmeno, např. `\$` vypíše znak `$`. Všechny výše uvedené speciální znaky lze vypsat pomocí tohoto příkazu.
- Příkazy začínající znakem zpětného lomítka (\) plus posloupnost písmen ukončená prvním znakem, který není písmeno. U názvu příkazů se rozlišuje velikost písmen, např. `\large`, `\Large` a `\LARGE` jsou odlišené příkazy pro změnu velikosti písma.

Mnoho příkazů pracuje pouze s určitou částí textu. Tento text je proto třeba zapsat jako parametr do složených závorek nacházejících se za příkazem. Např. `\textit{hello}` zobrazí text *hello* kurzívou. Těmto parametrům se říká povinné, protože musí být vždy předány. Některé příkazy však mohou přijímat i nepovinné parametry, které většinou mění jeho chování. Zapisují se pomocí hranatých závorek.

Obecný způsob zápisu příkazu je tedy `\nazev[volitelne]{povinne}`. Příkazy mohou obsahovat i více nepovinných parametrů, které se pak zapisují do stejných

hranatých závorek v daném pořadí. V případě že není poskytnut nepovinný parametr, závorky není nutné psát.

Některé příkazy mohou obsahovat i více povinných parametrů. V tomto případě musí být každý parametr zapsán do speciálních složených závorek a musí být také zachováno správné pořadí. Např. příkaz pro vytvoření černého čtyřúhelníku se zapisuje `\rule{5em}{2em}`, kde první parametr je šířka a druhý výška (Kopka a Daly, 2004).

4.2.3 Skupiny a prostředí

Systém TEXobsahuje také mechanismus nazývaný *skupiny*, který umožňuje lokální platnost nastavení. Znamená to, že když je tato skupina ukončena, nastavení se vrací do původního stavu, jaký byl před otevřením skupiny. Existují však i nastavení, které jsou vždy globální nebo je možné je tak nastavit příkazem `\global`. Skupina se v TEXu vymezuje symboly `{` a `}` nebo příkazy `\begingroup` a `\endgroup`. Využití najde třeba při změně fontu pouze určité části textu, čehož lze dosáhnout příkazem `{\Large velké písmo}`. To způsobí zvětšení písma textu uvnitř skupiny, kdežto ostatní text bude mít zase původní velikost (Eijkhout, 2013).

LATEX však zavedl nadstavbu skupin, která se nazývá *prostředí*. Prostředí stejně jako skupiny v momentě ukončení vrací nastavení a změny provedené uvnitř do původního stavu. Navíc však ovlivňuje i sazbu textu. Prostředí je vymezeno příkazy `\begin{název}` a `\end{název}`. Lze do sebe vnořovat více prostředí, což však platí i pro skupiny. Pro příklad je možné uvést prostředí `quote`, které je určeno pro sazbu citátu nebo prostředí center určené pro sazbu textu na střed (Kopka a Daly, 2004).

4.3 Formátování textu

Při popisu možností formátování textu v typografickém systému LATEX budeme vycházet ze znalostí smíšené sazby a odstavcové sazby. Nejprve je však nutné si udělat přehled o sazbě samotné.

Ve zjednodušené formě sazba znamená vytváření tiskové předlohy pomocí knižního (proporcionálního) písma, jehož znaky mají různou šířku. Má svá ustálená pravidla, která jsou definovaná normou. Vysázená tisková předloha je značně odlišná od strojopisu, který využívá neproporcionální písmo (znaky mají stejnou šířku). Toho je dosaženo různými prvky a technikami, které psací stroj neumožňuje. TEX a jeho nadstavby jsou považovány za modely sázecích strojů, proto jejich použití vyžaduje alespoň elementární znalost pravidel sazby a typografie. Rozlišujeme sazbu hladkou a smíšenou (Rybička, 2003).

Hladká sazba je sazba odstavců na určenou šířku z jednoho typu a stupně písma. Řídí se obecně typografickými pravidly a pravidly českého pravopisu. Důležitá jsou také pravidla pro sazbu různých znaků, která jsou definovaná normou ON 88 2503. Ta uvádí, že cílem je získání opticky správného vzhledu a bezchybné zhotovení sazby. Sazba má tvořit opticky ucelenou plochu a odpovídat estetice (Rybička, 2003).

Oproti tomu *smíšená sazba* znamená použití různých druhů a stupňů písma v dokumentu. Používá se především v okamžiku, kdy je potřeba některou část textu vizuálně odlišit od zbytku. Právě tímto typem sazby se budeme dále zabývat při popisu možností formátování písma.

4.3.1 Formátování písma

Stupeň písma

Systém LaTeX disponuje příkazy uvedenými v tabulce 1, pomocí kterých můžeme změnit stupeň písma v libovolné části dokumentu. Tyto příkazy mají platnost pouze do konce skupiny nebo prostředí, proto se zapisují právě do nich.

Tabulka 1: Příkazy pro změnu stupně písma (Rybička, 2003)

Velikost	Příkaz	Ukázkový text
5 pt	`\tiny`	Lorem Ipsum
7 pt	`\scriptsize`	Lorem Ipsum
8 pt	`\footnotesize`	Lorem Ipsum
9 pt	`\small`	Lorem Ipsum
10 pt	`\normalsize`	Lorem Ipsum
12 pt	`\large`	Lorem Ipsum
14,4 pt	`\Large`	Lorem Ipsum
17,28 pt	`\LARGE`	Lorem Ipsum
20,74 pt	`\huge`	Lorem Ipsum
24,88 pt	`\Huge`	Lorem Ipsum

Stupeň základního písma je nastaven na 10 pt. Toto nastavení je možné změnit parametrem příkazu `\documentclass` v preambuli zdrojového souboru, který může obsahovat hodnoty 10 pt, 11 pt nebo 12 pt. Změna stupně písma pomocí uvedených příkazů je relativní ke stupni základního písma.

Řez písma

Nastavení řezu písma slouží ke grafickému odlišení či zdůraznění určité části textu. Obecně se tento proces nazývá vyznačování a liší se jeho použití v sazbě a ve strojopisu. V sazbě je pro vyznačování používán kurzívní řez, polotučný řez případně kapitálky.

Nejčastějším způsobem vyznačování je kurzíva, která nejméně narušuje jednotné zabarvení tisku a působí dobře i z estetického hlediska. V systému LaTeX existuje příkaz `\emph`, který nastavuje vyznačovací řez automaticky podle daného

prostředí. V případě obyčejného písma je vyznačování nastaveno na kurzívu a v prostředí kurzívy se zase vyznačuje pomocí obyčejného písma. Příkazy pro změnu řezu jsou uvedeny v tabulce 2.

Tabulka 2: Příkazy pro změnu řezu písma (Rybička, 2003)

Řez	**Příkaz**
vzpřímené	`\textup{}` nebo `\upshape`
kurzíva	`\textit{}` nebo `\itshape`
skloněné	`\textsl{}` nebo `\slshape`
KAPITÁLKY	`\textsc{}` nebo `\scshape`
netučné	`\textmd{}` nebo `\mdseries`
polotučné	`\textbf{}` nebo `\bfseries`

Rodina písma

Pro práci s písmem je k dispozici balík `fontspec`, který je třeba připojit v preambuli zdrojového souboru. Tento balík definuje příkazy umožňující změnit písma pro trojici základních rodin. K nastavení základního písma v dokumentu se používá příkaz `\setmainfont` pro běžné patkové písmo, `\setsansfont` nastavuje bezpatkové písmo a `\setmonofont` pro strojopisnou variantu. Posledním parametrem těchto příkazů je vždy jméno písma, které je dostupné v systému. Například `\setmainfont{Cambria}` nastaví jako základní písmo dokumentu písmo Cambria.

Pro jednorázové přepnutí písma slouží příkaz `\fontspec`, jehož povinným parametrem je opět jméno písma. Toho lze využít pro změnu písma ve skupině nebo prostředí, např. `{\fontspec{Calibri} text}` změní písmo jen v tomto úseku textu (Satrapa, 2011).

4.3.2 Formátování odstavce

Odstavec je označován za základní stavební prvek dokumentu a jeho smyslem je ohraničení konkrétní myšlenky v textu. Sazeč má za úkol od sebe odstavce vizuálně odlišit a zajistit jejich kvalitní vysazení bez rušivých elementů. Rozlišujeme několik důležitých rozměrových parametrů odstavce: *odstavcová zarážka*, *levý okraj*, *pravý okraj*, *odsazení* a *mezera východové řádky*. Všechny tyto parametry lze samozřejmě nastavit pomocí adekvátních příkazů systému LaTeX, které budou popsány níže (Rybička, 2003).

Mezery

V LaTeXu je možné nastavit dva druhy mezer: *vodorovné* a *svislé*. Vodorovnou mezeru lze vytvořit příkazem `\hspace`, který přijímá jako parametr velikost mezery. Ta

se zadává v jednotkách podporovaných LaTeXem, například `\hspace{1em}` vytvoří vodorovnou mezeru o šířce 1 em.

Svislá mezera se dá vytvořit obdobným příkazem `\vspace`, který také obsahuje parameter velikost pro nastavení výšky mezery. Existují také příkazy, které nařídí svislou mezeru podle řádkování základního písma. Tyto příkazy jsou `\smallskip` (čtvrtina řádkování), `\medskip` (polovina řádkování) a `\bigskip` (celé řádkování). Příkazy pro vytvoření svislé mezery se zapisují v místech mezi odstavci (Rybička, 2003).

Geometrické parametry odstavce

Tyto parametry jsou uloženy v různých *délkových registrech* a jejich nastavením lze ovlivnit sazbu. Délkové registry jsou proměnné sloužící pro ukládání délkových hodnot. Zapisují se stejně jako příkazy, například `\rightskip` a je možné jejich obsah použít všude tam, kde je vyžadováno zadání nějakého rozměru, například `\hspace{\rightskip}`.

Rozlišujeme délky *pevné* a *pružné*. Do kategorie pevných délek spadají délky jako 2 pt nebo `\rightskip`. Tyto délky mají vždy přesně specifikovanou velikost. Oproti tomu pružné délky se mohou měnit podle situace a jsou využívány především v momentě, kdy je potřeba uplatnit mechanismus automatického zarovnávání. Pružná délka s nulovou přirozenou délkou, která se roztahuje na velikost danou okolím, má název `\fill`. Tuto délku je možné použít v kombinaci s horizontální mezerou jako `\hspace{\fill}`, což lze zapsat i ve zkráceném tvaru jako `\hfill`.

Pro nastavení délkových registrů ovlivňujících sazbu odstavců existují v systému LaTeX tyto příkazy (Rybička, 2003):

- `\hsize` – nastavení šířky sazby,
- `\leftskip` – levý okraj odstavce,
- `\rightskip` – pravý okraj odstavce,
- `\parident` – odstavcová zarážka,
- `\parfillskip` – mezera východové řádky,
- `\baselineskip` – řádkování (vzdálenost jednotlivých účaří),
- `\hangindent` – velikost odsazení řádků odstavce,
- `\hangafter` – počet odsazených řádků odstavce,
- `\parshape` – libovolný tvar odstavce.

Textová prostředí

V systému LaTeX je definováno několik textových prostředí, které se používají pro změnu výchozího tvaru odstavce. Pomocí těchto textových prostředí lze sázet například citáty, zarovnat text nebo vysázet text nepodléhající formátování.

Často využívané je prostředí sloužící pro zarovnání odstavce, které lze provést čtyřmi běžnými způsoby. Výchozí zarovnání je *na blok*, kdy mají všechny řádky stejnou délku. Zarovnání je dosaženo rovnoměrným rozprostřením mezer mezi slovy a také je použito dělení slov.

Pro zarovnání odstavce vlevo nebo vpravo slouží *sazba na praporek*. Když je text zarovnaný vlevo, praporek vlaje vpravo a naopak. Dělení slov se při sazbě na praporek většinou nepoužívá. Sazbu na praporek vpravo zajišťuje prostředí `flushleft`, případně příkaz `\raggedright`. Sazbu na praporek vlevo je možné zajistit pomocí prostředí `flushright` nebo příkazem `\raggedleft`.

Je možné použít také zarovnání odstavce sazbou *na střed*. To je řešeno prostředím `center` či příkazem `\centering` (Rybička, 2003).

5 Uživatelské styly

Uživatelské styly v typografických systémech poskytují funkcionalitu, díky které je možné snadněji formátovat text a toto formátování určitým způsobem uchovávat pro další použití. Pomocí stylů lze dosáhnout oddělení vizuální stránky dokumentu od jeho obsahu, což umožňuje autorovi zachovat jednotný vzhled napříč mnoha dokumenty a také usnadňují provádění změn v rámci formátování.

Výhodu stylů lze demonstrovat na jednoduchém příkladu. Autor dokumentu potřebuje změnit velikost všech nadpisů. Pokud nebude používat styly, musí projít celý dokument a postupně změnit velikost u každého nadpisu. Následně zjistí, že výsledná velikost není stále ideální, proto bude muset celý proces znovu opakovat. Už na první pohled je patrné, že tato činnost je časově velmi náročná a náchylná k chybám, v rámci velkého množství textu až nepředstavitelná. V případě použití stylů je však celá záležitost velmi jednoduchá. Stačí vyhledat styl, který je aplikován na nadpis a u něj změnit velikost písma. Následně se změna projeví u všech nadpisů v rámci celého dokumentu. Použití stylů tedy nejenom ušetří čas, ale také pomáhá eliminovat zbytečné chyby. Při ruční úpravě formátování je totiž snadné přehlédnout části textu nebo vynechat několik znaků, které měly být změněny. Díky stylům je zajištěno provedení změn důsledně v celém dokumentu (Bear, 2015).

Nastavení stylů se může lišit v jednotlivých typografických systémech. Většina systémů však umožňuje si vytvořený styl pojmenovat, což usnadňuje orientaci při jejich používání. Styly jsou obvykle rozděleny do dvou kategorií: *odstavcové* a *znakové*. Odstavcové styly se aplikují na celý odstavec, oproti tomu znakové styly se aplikují jen na vybranou část znaků. Navíc pokud je znakový styl aplikován na část, která už je formátována pomocí odstavcového stylu, tak znakový styl odstavcový přepíše.

Jak se liší v jednotlivých systémech nastavení stylů, liší se i jejich celková správa, vytváření a editace. Mnoho systémů nabízí uživatelské rozhraní, pomocí kterého lze styly jednoduše modifikovat. Mezi takové systémy patří například Adobe InDesign, Scribus nebo Microsoft Word. Na druhé straně jsou systémy, které grafické rozhraní nenabízí a uživatelé si musí vytvářet styly ručně zápisem speciálních konstrukcí. Tento způsob lze nalézt právě v systému TEX.

V následujících částech budou uvedeny programy pro tvorbu dokumentů a jejich možnosti. Nejedná se samozřejmě o kompletní přehled softwaru tohoto typu, představeny budou pouze ty nejpoužívanější programy v dnešní době.

5.1 TEX

Typografický systém TEX, přesněji jeho nadstavba LATEX, obsahuje několik předdefinovaných stylů pro formátování dokumentu. Tyto styly však často nejsou z nějakého důvodu vyhovující, proto je třeba vytvořit vlastní. Tvorba stylů je řešena pomocí nových strukturních značek, jejichž definice se obvykle ukládají do souboru s příponou `.sty`. Do dokumentu se pak tyto soubory vkládají v preambuli zdrojového

souboru pomocí příkazu `\usepackage{nazev_baliku}`. Díky tomu je zaručena jejich přenositelnost a znovupoužitelnost v různých dokumentech. Jednotlivé styly jsou vytvářeny jako vlastní makra, která obsahují příkazy pro formátování textu. Tato makra pak lze využít na libovolném místě dokumentu.

5.1.1 Tvorba vlastních maker

Nejprve je vhodné uvést, co to vlastně makro je. Makro se dá popsat jako posloupnost příkazů, které jsou zastoupeny jednou řídicí sekvencí. Lze si je představit jako procedury v běžných programovacích jazycích, protože makro může obsahovat i parametry.

Nové makro se definuje příkazem `\def`, například `\def\nadpis{titulek}` definuje makro `\nadpis`, které obsahuje titulek. Kdykoliv je tedy použita tato řídicí sekvence `\nadpis`, TEX ji expanduje na definovaný obsah *titulek*. Obsahem makra mohou být další makra, které se také expandují, dojde tedy k úplné expanzi všech maker (Olšák, 2015).

Definice makra se skládá z těchto částí:

1. libovolný počet prefixů `\global`, `\long` a `\outer`,
2. řídicí sekvence definice,
3. řídicí sekvence, která bude definována,
4. pravidlo parametrů, které specifikuje parametry daného makra,
5. tělo makra ohraničené složenými závorkami.

Prefixy mění stav definice makra. K dispozici jsou tři typy prefixů, které mohou být aplikovány na příkaz `\def` v libovolném pořadí, navíc se zde mohou vyskytovat i více než jednou. Prefix `\global` nastavuje definici jako globální, pokud se vyskytuje uvnitř skupiny. Příkaz `\def` totiž definuje ve výchozím stavu řídicí sekvenci lokálně. Jeho zápis jde zkrátit pomocí příkazu `\gdef`. Dalším prefixem je `\outer`, který neumožňuje danému makru vyskytovat se například v obsahu jiného makra nebo v parametrech. Posledním prefixem je `\long`, který dovolí načíst do parametru i několik odstavců najednou, což ve výchozím stavu není možné.

Řídicí sekvence definice může nabývat čtyř tvarů: `\def`, `\gdef`, `\edef` a `\xdef`. Řídicí sekvence `\gdef`, jak již bylo zmíněno výše, je synonymum pro `\global\def` a `\xdef` je synonymum pro `\global\edef`.

Pravidlo parametrů může být zapsáno mezi řídicí sekvenci a otevírací závorku těla makra. Je zde specifikováno, zda má makro parametry, kolik a jak jsou odděleny. Makro může mít nejvýše devět parametrů. Všechny parametry musejí být uvedeny znakem `#` a číslovány postupně od jedné do devíti. Parametry mohou být dále oddělené nebo neoddělené. Neoddělený parametr je okamžitě následován dalším parametrem nebo otevírací závorkou, například `\def\nadpis#1{…}`. U odděleného parametru se naopak za číslem vyskytují znaky, sekvence nebo celé texty, které

pak slouží jako oddělovače. Do takového parametru se tak načte veškerý text až po uvedený oddělovač, například `\def\nadpis#1!{…}`.

V těle makra se mohou vyskytovat kromě příkazů také symboly parametrů `#` následované číslem daného parametru. Tato dvojice znaků pak určuje místo, kam se má vložit přečtený parametr.

Příkaz `\def` nekontroluje, zda byla nově definovaná řídicí sekvence už dříve definována. V takovém případě řídicí sekvenci předefinuje. To může působit problémy uživatelům, kteří neznají všechny řídicí sekvence a mohou si tak přivodit nečekané problémy. Pokud předefinujeme například `\hbox`, celá sazba se v zásadě rozpadne, jelikož příkaz `\hbox` využívá mnoho maker i vnitřních rutin (Olšák, 2015).

Tento problém je vyřešen v systému LaTeX. Zde je pro vytvoření nového makra k dispozici příkaz `\newcommand`, který kontroluje, jestli je požadované jméno příkazu ještě neobsazené. Nemůže se tak stát, že by došlo k předefinování již existujícího příkazu. Pokud je však záměrem právě předefinování příkazu, slouží k tomu příkaz `\renewcommand`. Samotný zápis příkazu `\newcommand` se mírně liší od příkazu `\def`, například vytvoření makra `\newcommand{\nadpis}[1]{\emph{#1}}` ukazuje, že řídicí sekvence se zapisuje do složených závorek a parametry se udávají jako číslo do hranatých závorek, které určuje jejich počet (Satrapa, 2011).

5.2 Microsoft Word

Microsoft Word (dále jen Word) je textový procesor od firmy Microsoft. Jedná se o komerční software, který však nabízí i zkušební verzi. Word je k dispozici jako samostatný produkt nebo součástí kancelářského balíku Microsoft Office. Primárně je vyvíjen pro operační systém Windows, je však dostupný i na platformě Apple.

Word patří do rodiny editorů WYSIWYG, což v překladu značí „co vidíš, to dostaneš“. Znamená to tedy, že přímo při tvorbě dokumentu lze na obrazovce vidět, jak bude závěrečný výstup přibližně vypadat. Je třeba zdůraznit slovo přibližně, protože naprosto přesné podoby tištěného výstupu nelze dosáhnout. Word cílí převážně na běžné uživatele, kterým jde spíše o uživatelsky přívětivé rozhraní a jednoduchou tvorbu dokumentů. Pro profesionály v oblasti sazby, vyžadující naprostou kontrolu nad výslednou podobou, se jedná o zcela nevhodný nástroj.

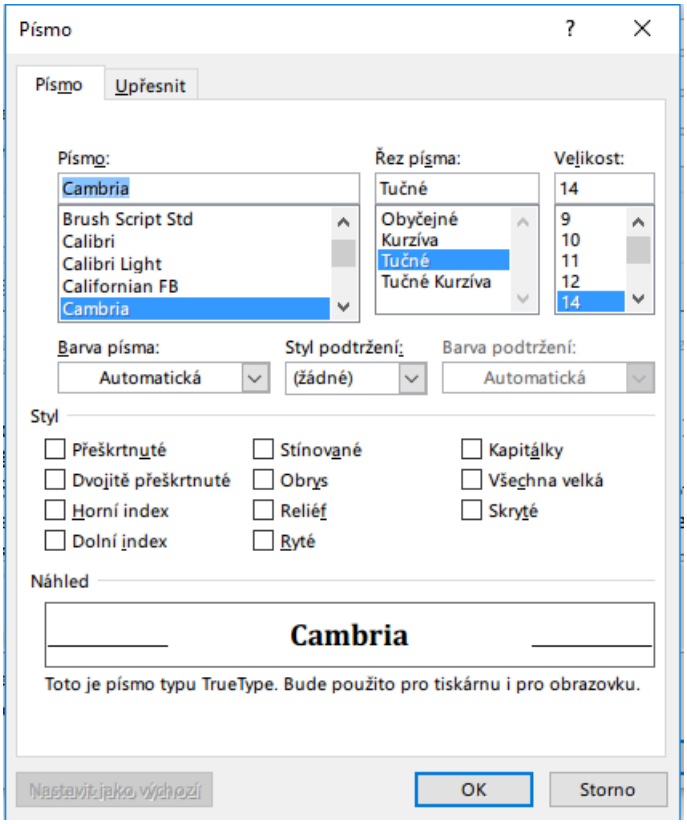
Správa a tvorba stylů ve Wordu je řešena pomocí nabízeného grafické rozhraní, které obstarává veškerou funkcionalitu. Zabývat se budeme verzí MS Word 2013. Zde si uživatel při tvorbě dokumentu může zobrazit všechny dostupné styly, u kterých zároveň vidí náhled, jak daný styl formátuje text. To přispívá k větší přehlednosti a usnadňuje orientaci při výběru vhodného stylu. Po kliknutí na styl dojde k jeho aplikaci na vybranou část textu.

Při tvorbě nového stylu je na výběr mezi pěti typy: odstavec, znak, propojený (odstavec + znak), tabulka a seznam. Vybráním některého typu se podstatně liší nabídka parametrů, které lze nastavit. Je zde také možnost založit styl na jiném stylu, což vytváří systém dědičnosti. Znamená to tedy, že nově založený styl bude mít stejné vlastnosti jako ten, na kterém je založený. Pokud dojde ke změně vlast-

nosti v rodičovském stylu, projeví se tato změna i v následném stylu. Nastavením vlastnosti následného stylu dojde k přepsání zděděné hodnoty.

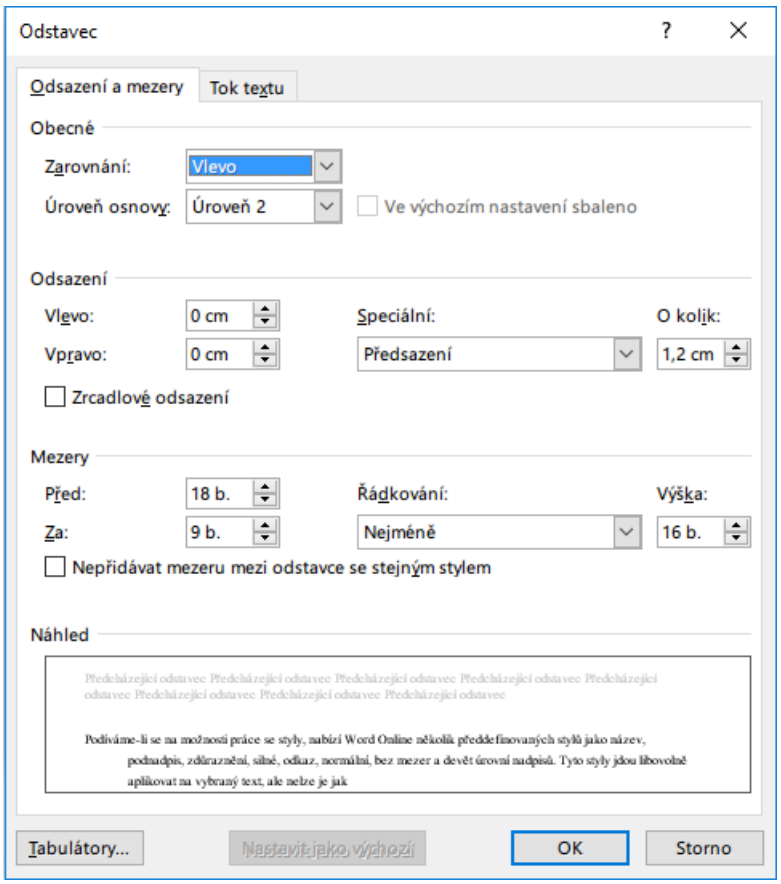
Nejvíce možností nabízí odstavcové styly, u kterých je možné měnit nastavení například písma, odstavce, tabulátorů, ohraničení, aj. K těmto nastavením lze přistoupit po kliknutí na tlačítko „Formát“, které zobrazí příslušnou nabídku. Dále se budeme podrobněji zabývat možnostmi formátování písma a odstavce, které nás nejvíce zajímají s ohledem na tvorbu návrháře stylů.

Okno s nastavením formátování písma (viz obrázek 2) nabízí v první řadě výběr rodiny písma, následně lze zvolit jeho řez, velikost, případně i barvu a podtržení. Dále v sekci „Styl“ je možné nastavit přeškrtnutí, horní a dolní index, kapitálky, všechna velká nebo skryté. Nastavení písma doplňuje náhled, kde lze vidět výslednou podobu daného formátování.



Obrázek 2: Okno s nastavením formátování písma v MS Word 2013

Nastavení formátování odstavce (viz obrázek 3) umožňuje na kartě „Odsazení a mezery“ zvolit způsob zarovnání a úroveň osnovy. Dále lze nastavit odsazení vlevo a vpravo a mezery před a za odstavcem, to vše ve vybraných rozměrech. Je možné změnit také řádkování. Na kartě „Tok textu“ jsou potom volby pro stránkování, kde je umožněno nastavit kontrolu osamocených řádků, svázání s následujícím, svázání řádků nebo vložení konce stránky před aktuální odstavec. Ve výjimkách formátování lze potlačit dělení slov. Nastavit se dá také způsob obtékání.



Obrázek 3: Okno s nastavením formátování odstavce v MS Word 2013

Celkově lze říct, že používání a tvorba stylů ve Wordu jsou poměrně jednoduché, protože se dá vše snadno nastavit pomocí grafického rozhraní, které je přívětivé i pro méně zkušené uživatele. Pomáhá k tomu i rychle dostupná nabídka s přehledem všech stylů a také různé náhledy, díky kterým uživatel hned vidí, jak daný styl vypadá. Word však při tvorbě stylů používá názvosloví, které neodpovídá typografickým pravidlům. To může být výhoda pro běžné uživatele, kteří těmto názvům rozumí více, nicméně zkušenější uživatele to může zmást. Jako příklad lze uvést velikost písma místo stupně písma nebo tučný řez místo polotučného. Takových příkladů by se dalo najít ještě více. Tato skutečnost jen potvrzuje, že Word není určen pro profesionální sazbu dokumentů, ale cílí spíše na nenáročné uživatele.

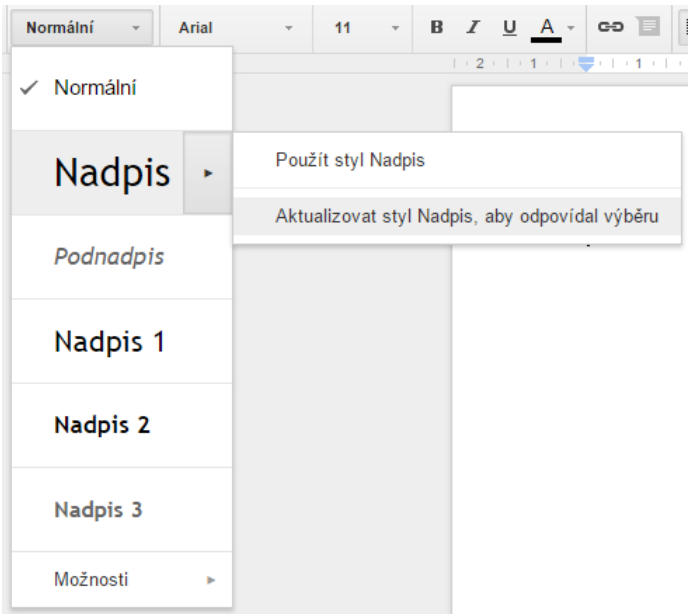
Existuje mnoho obdobných aplikací jako Word, například OpenOffice Writer. Ty však nabízí víceméně stejné možnosti, proto nebyly do výběru zahrnuty.

5.3 Google Dokumenty

Jedná se o textový procesor, který obsahuje stejně jako MS Word WYSIWYG editaci. Na rozdíl od něj je však dostupný zdarma jako webová aplikace v rámci kan-

celářského balíku nabízeného společností Google. Podmínkou pro přístup je mít zřízený vlastní Google účet. Veškeré dokumenty se ukládají do cloudové služby Google Drive, což je úložiště s limitovanou kapacitou přidělené ke každému účtu. Google dokumenty byly vybrány hlavně kvůli tomu, že se jedná o webovou aplikaci, stejně jako TEXonWeb. Můžeme tak přímo porovnat, jaké jsou dostupné možnosti tvorby stylů v aplikacích tohoto typu.

Velkou výhodou Google Dokumentů je jejich dostupnost, jelikož fungují nejen jako webová aplikace, ale také jako aplikace pro prohlížeč Chrome umožňující práci bez připojení k internetu nebo mobilní aplikace pro operační systémy Android a iOS. Další výhodu představuje možnost spolupráce více uživatelů na jednom dokumentu současně.



Obrázek 4: Seznam výchozích stylů v aplikaci Google Dokumenty

Pro správu stylů nabízí Google Dokumenty seznam dostupných stylů (viz obrázek 4), který lze najít jak v nástrojové liště, tak i v hlavní menu pod nabídkou „Formát“. Jejich výběrem se daný styl aplikuje na text. Ve výchozím stavu je k dispozici několik předdefinovaných odstavcových stylů: normální, název, podnadpis a šest typů nadpisů. Formátování textu se dá upravit pomocí příslušných ikon v nástrojové liště nebo přes nabídku „Formát“. U písma je možné vybrat rodinu, velikost, řez, podtržení, barvu a horní a dolní index. U odstavce je to potom odsazení, zarovnání a řádkování. Ručně naformátovaný text lze vybrat a aktualizovat jím některý z výchozích stylů. Tím ovšem možnosti stylů v Google Dokumentech končí, jelikož přidávat nebo mazat styly není možné. Práce se styly je zde tedy velmi výrazně omezena jen na skutečně minimální funkcionalitu.

Z výše uvedeného je patrné, že Google Dokumenty se snaží zachovat jednoduché ovládání na úkor pokročilých možností. Slouží tedy především pro snadné vytváření

menších dokumentů, které lze lehce sdílet s jinými uživateli a přistupovat k nim z různých druhů zařízení. S ohledem na dodržování typografických pravidel jsou na tom Google Dokumenty hůře než MS Word. Znovu je tedy třeba konstatovat, že profesionální sazba zde není možná.

5.4 Microsoft Word Online

Word Online patří do webového kancelářské balíku Office Online vyvíjeného společností Microsoft. Jedná se v podstatě o obdobu Google Dokumentů, má podobné vlastnosti a je také dostupný zdarma. Ve výchozím nastavení se dokumenty ukládají do cloudového úložiště Microsoft OneDrive. Stejně jako Google Dokumenty i Word Online byl vybrán z důvodu, že jde o webovou aplikaci jako TEXonWeb.

Na první pohled vypadá Word Online velmi podobně jako desktopová verze Wordu, ve skutečnosti je však jeho funkcionalita značně omezená. Vylepšené jsou naopak možnosti spolupráce více uživatelů v reálném čase na stejném dokumentu. Při srovnání s konkurenční aplikací Google Dokumenty postrádá Word Online možnost práce v offline režimu, naopak nabízí lepší podporu pro formáty Microsoftu jako je například `.docx`. Dokumenty vytvořené v tomto formátu by měly vypadat stejně i při otevření v desktopové verzi Wordu, což pro dokumenty z Google Dokumentů úplně neplatí.

Obrázek 5: Nabídka stylů v aplikaci Microsoft Word Online

Podíváme-li se na možnosti práce se styly, nabízí Word Online několik předdefinovaných stylů jako název, podnadpis, zdůraznění, silné, odkaz, normální, bez mezer a devět úrovní nadpisů (viz obrázek 5). Tyto styly jdou libovolně aplikovat na vybraný text, ale nelze je jakkoliv modifikovat ani přidávat další. Formátování lze tedy přizpůsobit jedině ručním nastavením jednotlivých vlastností přímo v textu. Zde jsou k dispozici podobné možnosti jako u Google Dokumentů. Je možné mě-

nit rodinu písma, velikost, řez a barvu. U odstavce je dovoleno upravit zarovnání, řádkování, odsazení či mezery před a za.

Je tedy jasně vidět, že pro tvorbu složitějších dokumentů není webová aplikace Word Online určena. Není zde možná jakákoliv práce se styly, kromě užití několika předdefinovaných. V tomto ohledu jsou tak možnosti této aplikace ještě menší, než u Google Dokumentů. Word Online je tedy aplikace pouze pro nenáročné uživatele.

6 Formální jazyky a automaty

Návrhář stylů bude umožňovat zpětné načítání definic maker, které byly předtím pomocí návrháře vygenerovány. Ať už byly tyto definice vygenerovány a uloženy do souboru nebo do editoru, princip načítání bude v obou případech stejný. Získá se řetězec s definicemi, který bude muset být analyzován a ověřena správnost jeho syntaxe. Aby bylo možné tento proces provést, je potřeba vytvořit syntaktický analyzátor, zkráceně parser. Předtím než budeme moci zkonstruovat parser, je zapotřebí probrat některé důležité pojmy z oblasti teorie formálních jazyků a automatů. Na základě těchto znalostí pak bude možné parser realizovat.

6.1 Abeceda a jazyk

Abychom se mohli pustit do definice jazyka, je nejprve nutné zavést pojmy abeceda a řetězec. Jedná se základní pojmy hojně využívané v teorii formálních jazyků.

Abecedu je možné definovat jako neprázdnou množinu prvků, které nazýváme symboly abecedy (případně znaky). Obvykle se značí řeckým písmenem Σ. Lze se setkat s abecedami konečnými či nekonečnými, ale pro naše účely si vystačíme pouze s konečnými. Příkladem abecedy může být binární abeceda reprezentovaná množinou $\{0, 1\}$ nebo abeceda programovacího jazyka JavaScript (Češka, 1992).

Podle (Černá, Křetínský a Kučera, 2002) se *Řetězcem* (případně slovem či větou) nad abecedou nazývá každá konečná posloupnost symbolů abecedy. Posloupnost, která neobsahuje žádný symbol, se označuje jako prázdný řetězec a zapisuje se písmenem ϵ. U řetězce lze určit jeho délku udávající počet symbolů, které obsahuje. Délka řetězce w se zapíše jako $|w|$. Nad řetězci lze také provádět některé důležité operace (Češka, 1992):

- Konkatenace (zřetězení) – připojí-li se řetězec y za řetězec x, vznikne řetězec xy.
- Reverze – symboly řetězce x^R jsou zapsány v opačném pořadí vzhledem k řetězci x.
- Podřetězec – řetězec z se nazývá podřetězcem řetězce w, jestliže existují řetězce x a y takové, že $w = xzy$.
- Prefix, sufix – řetězec x je prefix řetězce w, jestliže existuje řetězec y takový, že $w = xy$. Analogicky je možné definovat sufix.

Jazyk nad abecedou Σ je libovolná podmnožina řetězců nad Σ. Formálně lze jazyk vyjádřit jako množinu L, pro níž platí, že $L \subseteq \Sigma^*$, kde Σ^* je množina všech řetězců nad abecedou Σ včetně prázdného řetězce. Vezmeme-li pro příklad abecedu $\Sigma = \{m, n\}$, pak jazyk nad touto abecedou je $L = \{m, mn, mmn\}$ (Češka, 1992).

Nad jazyky lze definovat také operace, které je možné rozdělit na množinové a řetězcové. Množinové operace vychází ze skutečnosti, že jazyk je množina. Proto je

možné použít operace jako sjednocení, průnik, rozdíl a doplněk. Řetězcové operace jsou postavené na faktu, že prvky množin tvořící jazyky jsou řetězce. Dostupné jsou tyto operace (Češka, 1992):

- Konkatenace (případně součin): definována jako $L_1 \cdot L_2 = \{xy \mid x \in L_1 \wedge y \in L_2\}$
- Iterace: Definována jako součin jazyka se sebou samým (mocnina jazyka). $L^k = L \cdot L \cdot \ldots \cdot L$

6.2 Gramatika

V souvislosti s jazyky se nabízí jedna velmi důležitá otázka: Jakým konečným způsobem reprezentovat rozsáhlý konečný či nekonečný jazyk? Jednoduchý konečný jazyk si snadno vystačí s výčtem slov. V případě nekonečného jazyka ale vzniká problém s reprezentací. Tento problém však řeší právě gramatika. Jak uvádí (Černá, Křetínský a Kučera, 2002), je to prostředek pro reprezentaci konečných i nekonečných jazyků, jelikož splňuje požadavek *konečnosti reprezentace*. Využívá k tomu dvě konečné disjunktní abecedy (Češka, 1992):

- Množina neterminálních symbolů N: Neterminály plní roli pomocných proměnných, označují syntaktické celky (kategorie). Obvykle se značí velkými písmeny.
- Množina terminálních symbolů Σ: je shodná s abecedou, na níž je definován jazyk. Terminály jsou obvykle značeny malými písmeny. Množina vzniklá sjednocením N a Σ se nazývá *slovník gramatiky*.

Gramatika obsahuje také konečnou množinu *přepisovacích pravidel* P. Jednotlivá pravidla mají tvar uspořádané dvojice řetězců (α, β), což znamená, že je možné nahradit řetězec α řetězcem β. Řetězec α obsahuje alespoň jeden neterminální symbol a řetězec β je prvek množiny $(N \cup \Sigma)^*$. Pravidla se zapisují ve tvaru $\alpha \to \beta$, kde řetězec α se nazývá *levá strana* pravidla a řetězec β je *pravá strana* pravidla (Češka, 1992).

Podle (Černá, Křetínský a Kučera, 2002) lze formálně definovat gramatiku jako čtveřici $G = (N, \Sigma, P, S)$, kde

- N je konečná neprázdná množina neterminálních symbolů
- Σ je konečná množina terminálních symbolů, $N \cap \Sigma = \emptyset$
- P je konečná množina přepisovacích pravidel, $P \subseteq (N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$
- S je startovací symbol gramatiky náležící do množiny N

Ze startovacího symbolu S můžeme aplikací přepisovacích pravidel generovat řetězce. Takové řetězce se nazývají větné formy, pokud jsou složené ze slovníku gramatiky $(N \cup \Sigma)^*$. Jestliže větná forma obsahuje pouze terminální symboly, je

nazývána větou. V okamžiku, kdy během procesu generování dojdeme k řetězci obsahujícímu pouze terminální symboly, generování končí. Množina všech vět, které lze vygenerovat pomocí přepisovacích pravidel gramatiky G, tvoří jazyk $L(G)$ generovaný gramatikou G.

Mezi dvěma větnými formami existuje relace, která je daná pravidly gramatiky a nazývá se přímá derivace. Značí se symbolem $\Rightarrow$. Libovolná mocnina přímé derivace $\Rightarrow^n$ se nazývá nepřímá derivace. Máme-li gramatiku G a řetězce λ, μ z $(N \cup \Sigma)^*$, definice udává, že derivace $\Rightarrow_G^+$ mezi dvěma řetězci λ a μ platí tehdy, jestliže existuje posloupnost přímých derivací $v_{i-1} \Rightarrow v_i$, kde $i = 1, ..., n$ a $n \geq 1$, taková, že platí:

$$\lambda = v_0 \Rightarrow v_1 \Rightarrow \ldots \Rightarrow v_{n-1} \Rightarrow v_n = \mu$$

Tato posloupnost se pak označuje jako derivace délky n (Češka, 1992).

6.2.1 Chomského klasifikace gramatik

Jak může napovědět název, o rozdělení gramatik do čtyř skupin se zasloužil americký lingvista Noam Chomsky. Důvodem pro jejich rozdělení byla různá omezení na tvar přepisovacích pravidel. Gramatiky jsou tak rozděleny podle popisné síly, která je u každé skupiny jiná. Podle (Češka, 1992) a (Černá, Křetínský a Kučera, 2002) rozlišujeme tyto čtyři typy:

- *Typ 0*: Pravidla mají tvar, který se shoduje s obecnou definicí gramatiky. Na jejich tvar tedy nejsou kladené žádné omezující požadavky, jedná se tak o neomezené gramatiky. Tyto gramatiky jsou akceptovány Turingovým strojem.
- *Typ 1*: Tyto gramatiky se nazývají kontextové. V jejich pravidlech může být neterminál A nahrazen řetězcem γ pouze tehdy, je-li jeho pravým kontextem řetězec β a levým kontextem řetězec α. Neterminál v pravidlech nemůže být nahrazen prázdným řetězcem. Výjimku tvoří pouze pravidlo $S \rightarrow \epsilon$, kde S je počáteční symbol. S se pak ale nesmí vyskytnout na pravé straně žádného pravidla. Tyto gramatiky akceptuje lineárně ohraničený automat.
- *Typ 2*: Každé pravidlo je ve tvaru:

  $$A \rightarrow \gamma, \quad A \in N, \quad \gamma \in (N \cup \Sigma)^*$$

  Tyto gramatiky se nazývají také bezkontextové, protože substituci je možné provádět bez ohledu na kontext, ve kterém je neterminál A uložen.
- *Typ 3*: Gramatiky označované také jako regulární. Obsahují pravidla ve tvaru $A \rightarrow aB$ nebo $A \rightarrow a$ s případnou výjimkou pravidla $S \rightarrow \epsilon$, pokud S není na pravé straně žádného pravidla. Tyto gramatiky jsou akceptovány konečným automatem.

Podle těchto typů gramatik lze také určit příslušný typ jazyka. Jazyk L je regulární (případně bezkontextový, kontextový nebo typu 0), jestliže je generovatelný regulární (případně bezkontextovou, kontextovou nebo typu 0) gramatikou

G. Každá další třída jazyků je obsažena v předchozí, tedy každý regulární jazyk je také bezkontextový, bezkontextový jazyk je také kontextový a kontextový jazyk je i jazyk typu 0. Ovšem neplatí to obráceně.

6.3 Jazyky typu 3

Existuje několik způsobů, jak lze formálně reprezentovat jazyky typu 3. První možností reprezentace je pomocí gramatiky (lineární, regulární), což bylo uvedeno i výše. Další způsoby reprezentace představují regulární výrazy a konečný automat. Všechny tyto způsoby jsou vzájemně ekvivalentní a lze je mezi sebou převádět (Černá, Křetínský a Kučera, 2002).

6.3.1 Regulární výrazy

Regulární výrazy slouží jako notace *regulárních množin*. Regulární množiny obsahují řetězce složené ze symbolů abecedy Σ a jsou konstruované pomocí určitých pravidel. Regulární množina nad abecedou Σ je definována následujícím způsobem (Češka, 1992):

1. $\emptyset$ je regulární množina nad Σ.
2. $\{\epsilon\}$ je regulární množina nad Σ.
3. $\{a\}$ pro všechna $a \in \Sigma$ je regulární množina nad Σ.
4. Jsou-li P a Q regulární množiny nad Σ, pak také $P \cup Q$, $P \cdot Q$ a P^* jsou regulární množiny nad Σ.
5. Regulární množiny jsou právě ty množiny, které lze získat aplikací pravidel 1–4.

Regulární výrazy jsou složené z řetězců nad abecedou Σ (operandů) a operátorů (zřetězení, varianta, iterace).

Regulární výrazy jsou velmi užitečným prostředkem při tvorbě překladače programovacího jazyka. Jak udává (Sipser, 2012), mohou být pomocí nich popsány základní prvky programovacích jazyků, které se nazývají tokeny. Takto lze popsat třeba názvy proměnných, klíčová slova nebo čísla. Například číslo, které může obsahovat desetinnou část nebo znaménko, se dá takto vyjádřit jako:

$$(+ \cup - \cup \epsilon)(D^+ \cup D^+.D^* \cup D^*.D^+)$$

kde $D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ je abeceda obsahující číslice.

6.3.2 Konečný automat

Konečný automat představuje abstraktní model pro systémy s konečným počtem stavů. Je vybaven konečně stavovou řídicí jednotkou a čtecí hlavou, která čte vstupní slovo z pásky. Na počátku výpočtu se čtecí hlava nachází na nejlevějším políčku

pásky. Automat následně přečte symbol. Na základě něj a aktuálního stavu změní svůj stav a posune čtecí hlavu o jedno políčko vpravo. Výpočet končí v okamžiku, kdy automat přečte celé vstupní slovo nebo když v některém stavu není kam jít. Pokud je celé slovo na pásce přečteno a výsledný stav je některý z koncových stavů, pak je automatem akceptováno. Množina všech slov, která daný konečný automat M akceptuje, tvoří jazyk akceptovaný automatem M Jazyk, který je rozpoznatelný konečným automatem, se nazývá regulární (Černá, Křetínský a Kučera, 2002).

Konečné automaty se dělí na nedeterministické a deterministické. Nedeterministický konečný automat je dle (Češka, 1992) pětice $M = (Q, \Sigma, \delta, q_0, F)$, kde:

- Q je neprázdná konečná množina vnitřních stavů,
- Σ je konečná množina vstupních symbolů (abeceda),
- $\delta : Q \times \Sigma \to 2^Q$ je přechodová funkce,
- $q_0 \in Q$ je počáteční stav automatu,
- $F \subseteq Q$ je množina koncových stavů.

Deterministický konečný automat (DKA) je speciální případ nedeterministického konečného automatu (NKA), protože následující stav jednoznačně určen. Pro každý stav a čtený symbol existuje jen jedna možnost, do kterého dalšího stavu lze přejít.

NKA představuje abstraktní reprezentaci algoritmu pro rozpoznávání jazyků, kdežto DKA je přímo konkrétní algoritmus pro rozpoznávání jazyků. Každý regulární výraz nebo NKA lze převést na DKA přijímající stejný jazyk (Aho a kol., 2006).

6.4 Bezkontextové jazyky

Bezkontextové gramatiky jsou při popisování jazyků daleko silnější než regulární gramatiky. Lze totiž pomocí nich popsat většinu rysů současných programovacích jazyků. Například párovost závorek nebo blokovou strukturu bychom pomocí regulárních gramatik nedokázali popsat. Bezkontextové gramatiky jsou také velmi často používány při konstrukci překladačů, konkrétně při syntaktické analýze. Syntaktický analyzátor totiž dokáže analyzovat věty bezkontextových jazyků (Sipser, 2012).

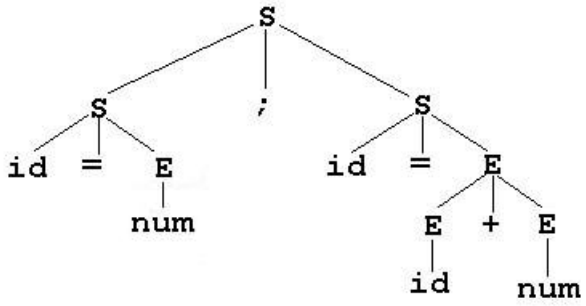
6.4.1 Derivační stromy

Derivační strom je důležitý prostředek, který slouží ke grafickému vyjádření struktury věty, čili její derivace (viz obrázek 6). Rozlišujeme levou a pravou derivaci. Pokud v každém kroku derivace byl ve větné formě přepsán nejlevější neterminál, pak se jedná o levou derivaci. V případě přepsání nejpravějšího neterminálu jde o derivaci pravou.

Jak udává definice, strom T nazveme derivačním stromem bezkontextové gramatiky G, pokud platí tyto podmínky (Černá, Křetínský a Kučera, 2002):

1. Každý uzel je ohodnocen symbolem z $N \cup \Sigma \cup \{\epsilon\}$.
2. Kořen má ohodnocení S.
3. Má-li vnitřní uzel ohodnocení A, pak $A \in N$.
4. Má-li uzel n ohodnocení A a všichni jeho následníci $n_1, ..., n_k$ mají v uspořádání zleva doprava ohodnocení $X_1, ..., X_k$, pak $A \rightarrow X_1 ... X_k \in P$.
5. Má-li uzel n ohodnocení ϵ, pak n je list a jediným následníkem svého rodičovského uzlu.

Slovo, které vznikne zřetězením ohodnocení listů v uspořádání zleva doprava, je výsledkem derivačního stromu T.



Obrázek 6: Příklad derivačního stromu

6.4.2 Zásobníkový automat

Zásobníkový automat je abstraktní nedeterministické zařízení, které představuje přirozený model syntaktického analyzátoru bezkontextových jazyků. Ke každé bezkontextové gramatice G je totiž možné sestrojit zásobníkový automat P tak, aby platilo $L(G) = L(P)$ a obráceně. Zásobníkový automat má oproti nedeterministickému konečnému automatu navíc paměť v podobě zásobníku, jejíž velikost není omezena, tudíž je nekonečná.

Čtecí hlava může symboly ze vstupní pásky pouze číst a může se posouvat jen doprava (jednocestný automat). Zásobníkový automat může ukládat symboly do zásobníku a kdykoliv pak může být symbol ze zásobníku přečten a vyjmut. Při ukládání i vybírání lze pracovat pouze s vrcholem zásobníku. Jedná se tedy o paměť typu LIFO (last in, first out) (Černá, Křetínský a Kučera, 2002).

Zásobníkový automat je obecně nedeterministický. Existují však i deterministické zásobníkové automaty, jejich rozpoznávací síla je ale nižší, než u nedeterministických. Nedeterministické zásobníkové automaty tak rozpoznají některé jazyky,

které deterministické rozpoznat nedokáží. Nelze ani pomocí obecného algoritmu převést nedeterministický na deterministický (Sipser, 2012).

Nedeterministický zásobníkový automat je podle (Češka, 1992) definován jako sedmice $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, kde:

- Q je neprázdná konečná množina vnitřních stavů automatu,
- Σ je konečná vstupní abeceda, jejímiž prvky jsou vstupní symboly,
- Γ je konečná množina zásobníkových symbolů,
- δ je zobrazení z množiny $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ do konečné množiny podmnožin množiny $Q \times \Gamma^*$ popisující přechodovou funkci,
- $q_0 \in Q$ je počáteční stav automatu,
- $Z_0 \in \Gamma$ je počáteční symbol v zásobníku,
- $F \subseteq Q$ je množina koncových stavů.

Dále definujeme konfiguraci zásobníkového automatu, což je trojice $(q, w, \alpha) \in Q \times \Sigma^* \times \Gamma^*$, ve které q je momentální stav, w je dosud nepřečtená část vstupního slova a α je obsah zásobníku. Počáteční konfigurace je (q_0, w, Z_0), kde w je vstupní slovo. Koncová konfigurace má tvar (q, ϵ, α), kde $q \in F$ je koncový stav a $\alpha \in \Gamma^*$ (Vavrečková, 2015).

Přechodem zásobníkového automatu P se rozumí relace na množině všech konfigurací automatu P a značí se symbolem $\vdash_P$. Relaci přechodu $(q, aw, Z\alpha) \vdash (q', w, \gamma\alpha)$ lze intuitivně interpretovat tak, že zásobníkový automat P může ze stavu q po přečtení vstupního symbolu $a \neq \epsilon$ a symbolu Z z vrcholu zásobníku přejít do stavu q'. Zároveň se čtecí hlava posune doprava a na vrchol zásobníku se po odstranění symbolu Z uloží řetězec γ. Jestliže $\gamma = \epsilon$, pak je odstraněn pouze vrchol zásobníku. V případě že $a = \epsilon$, čtecí hlava se neposouvá, přechod do nové konfigurace se neurčuje vstupním symbolem. Jedná se tak o ϵ-přechod (Češka, 1992).

Můžeme rozlišit dvě základní varianty zásobníkových automatů podle způsobu akceptování vstupní věty. Buď ji lze přijímat přechodem do koncového stavu nebo vyprázdněním zásobníku. Obě tyto varianty jsou ekvivalentní a lze je mezi sebou převádět.

6.5 Překladače

Překladač je obecně program, který dokáže převést zdrojový program napsaný ve zdrojovém jazyku do cílového programu v cílovém jazyku. To vše při zachování stejného významu. Překladače se dělí podle typu cílového programu na tyto druhy (Vavrečková, 2008):

- *Kompilátor* (generační překladač) – překládá vyšší programovací jazyk (Pascal, C, C++) do strojového jazyka nebo jazyka symbolických instrukcí (JSI, Assembler).
- *Interpret* (interpretační překladač) – netvoří generovaný program, pouze zdrojový program interpretuje, čímž vytváří vnitřní reprezentaci programu pro svou potřebu. Patří sem třeba systémové shelly, skriptovací jazyky (PHP, JavaScript) či HTML.
- *Hybridní překladač* – zapadá někam mezi předchozí dva typy. Nejprve je vygenerován mezikód nezávislý na operačním systému a ten je poté interpretován příslušným interpretem. Tento způsob překladu se používá například u programovacího jazyku Java.

Každý z těchto typů je uzpůsoben jinému použití, mají rozdílné vlastnosti, z čehož plynou i jejich výhody a nevýhody. Obecně je u kompilovaných programů lepší rychlost běhu a spuštění. Interpretované programy jsou zase rychleji přeloženy a jejich kód lze snadno přenášet mezi platformami.

6.5.1 Části překladače

Překladač je možné rozdělit na několik částí podle toho, co daná část v průběhu překladu vykonává. Tyto části však nemusí být striktně odděleny, některé se totiž mohou vzájemně propojovat. Podle (Vavrečková, 2008) jsou to:

1. *Lexikální analyzátor* – jako vstup přijímá zdrojový text a jeho úkolem je převést ho na posloupnost symbolů (tokenů) daného významu (např. číslo, klíčové slovo), kterým rozumí ostatní části překladače. Odstraňuje také nevýznamné části jako bílé znaky nebo komentáře.
2. *Syntaktický analyzátor* – vytváří strukturu překládaného programu reprezentující derivační strom. Skládá symboly lexikálního analyzátoru k sobě, čímž tvoří příkazy, bloky a jiné struktury.
3. *Sémantický analyzátor* – zpracovává derivační strom a přiřazuje význam každé skupině symbolů získané při syntaktické analýze. Provádí se zde například typová kontrola a konverze nebo kontrola deklarace proměnných.
4. *Optimalizátor kódu* – provádí optimalizace intermediárního kódu pro zrychlení běhu nebo snížení jeho velikosti.
5. *Generátor cílového kódu* nebo *interpretace* – podle typu překladače se buď vytváří kód v jazyce symbolických instrukcí, v jazyce sestavujícího programu (interpretační překladač) nebo strojovém jazyce (strojový kód).

Překladač obsahuje také funkce jako hlášení chyb nebo logování průběhu překladu do souboru. Tyto funkce bývají obvykle součástí výše uvedených částí, ale mohou se vykytovat i samostatně.

Dvě základní části překladače tvoří *přední část* (front end) a *zadní část* (back end). V přední části je obsažena lexikální, syntaktická a sémantická analýza. Tato část vytváří vnitřní formu programu a je nezávislá na cílovém systému. Zadní část se stará o optimalizaci kódu a generuje cílový program. Oproti předchozí části je však závislá na cílovém systému. Pokud tedy chceme cílový program generovat pro více platforem, můžeme použít stejnou přední část překladače, ale zadní část už musíme použít jinou.

V následujících částech si podrobněji představíme princip tvorby lexikálního a syntaktického analyzátoru, jelikož budou v rámci této práce zkonstruovány pro analýzu definic maker vygenerovaných návrhářem stylů.

6.5.2 Lexikální analýza

Lexikální analýza představuje první fázi při zpracování zdrojového programu. Úkolem lexikálního analyzátoru je převedení zdrojového programu na posloupnost symbolů (tokenů) s daným významem. Symbol může tvořit například číslo, klíčové slovo, operátor nebo název proměnné. Symbol obsahuje informaci o svém typu a skutečnou hodnotu (lexém). Užitečné mohou být i další informace jako třeba číslo řádku, kde se daný symbol nachází. Lexikální analyzátor se také stará o odstranění nepodstatných částí, které tvoří bílé znaky a komentáře. Je potřeba tedy specifikovat podobu těchto částí a odebrat je, aby se nedostaly mezi symboly.

Většina konstrukcí běžně používaných programovacích jazyků patří do třídy regulárních jazyků. Pro popis struktury jednotlivých symbolů se tak obvykle využívá regulární gramatika, regulární výrazy nebo přechodový graf (Habiballa, 2005).

Způsob implementace lexikálního analyzátoru se odvíjí od toho, jaké možnosti nám nabízí zvolený programovací jazyk. Velmi důležitá je úroveň podpory regulárních výrazů nebo práce s řetězci. Implementaci lze tedy realizovat buď přímo s využitím všech dostupných prostředků daného programovacího jazyka nebo zkonstruováním konečného automatu. Třetí variantou je vytvoření lexikálního analyzátoru pomocí generátoru. Mezi takové nástroje patří například Lex či jeho varianta Flex, kde stačí specifikovat strukturu tokenů regulárními výrazy a dostaneme vygenerovaný kód analyzátoru v jazyku C, případně C++. Tato metoda je sice velmi snadná, ale máme jen minimální kontrolu nad její implementací a optimalizací (Aho a kol., 2006).

Při implementaci lexikálního analyzátoru pomocí konečného automatu je nejprve nutné vytvořit regulární gramatiku popisující zkoumaný jazyk. Pro ní je třeba sestrojit konečný automat, který musí být deterministický. Postup je pak takový, že se v každém stavu automatu načte znak a podle něj se rozhodne, kterou větví dále pokračovat. Je-li automat v koncovém stavu a nelze jinam pokračovat, načetl právě jeden symbol a přesune se do počátečního stavu. V koncovém stavu se navíc testuje načtením následujícího znaku, zda je načtený symbol správně ukončen. Pokud automat není v koncovém stavu a nemá kam pokračovat, načtený znak je chybný a došlo k lexikální chybě (Vavrečková, 2008).

6.5.3 Syntaktická analýza

Syntaktický analyzátor (jinak také parser) má za úkol analyzovat vstupní bezkontextový jazyk, aby rozpoznal, jestli je program správně zapsán. K tomu využívá posloupnost symbolů získaných z lexikálního analyzátoru. Během analýzy se vytváří syntaktická struktura programu, která je reprezentována derivačním stromem.

Podle způsobu konstrukce derivačního stromu rozlišujeme dvě základní metody syntaktické analýzy (Habiballa, 2005):

- *Metoda shora dolů* – derivační strom se konstruuje od kořene, který značí startovací symbol, až dolů k listům. Konstrukce probíhá zleva doprava, podle levé derivace. Jedná se o proces nalezení levého rozkladu dané věty. Tedy jde o získání posloupnosti čísel pravidel použitých při levé derivaci.
- *Metoda zdola nahoru* – derivační strom se konstruuje zdola od listů ke kořeni. Postupuje se opět zleva doprava, ale podle pravé derivace. Jde o proces nalezení pravého rozkladu dané věty. Získáváme tedy posloupnost čísel pravidel použitých při pravé derivaci, ale v opačném pořadí.

Občas nastane moment, kdy nelze jednoduše vybrat vhodné přepisovací pravidlo. U metody shora dolů je to v okamžiku, kdy jeden neterminální symbol má více pravidel začínajících stejným podřetězcem. U metody zdola nahoru je to v situaci, kdy pro aktuálně přepisovaný řetězec existuje více pravidel s odpovídající pravou stranou. Tento problém lze obecně řešit dvěma způsoby (Habiballa, 2005):

- *Analýza s návratem* (backtracking) – v případě výběru nevhodného pravidla se vrátí o krok zpět a zkouší použít jiné pravidlo. Kvůli své časové i paměťové složitosti se v praxi příliš nepoužívá.
- *Deterministická analýza* – pro výběr správného pravidla používá další informace získané během překladu (např. informace o dosud nepřečtené části vstupní věty). Pro tuto analýzu se využívají deterministické bezkontextové gramatiky: LL pro analýzu shora dolů a LR pro analýzu zdola nahoru.

6.5.4 LL(k) gramatiky

LL gramatiky generují LL jazyky, které spadají do podtřídy deterministických bezkontextových jazyků. Pro tyto jazyky lze sestrojit deterministický syntaktický analyzátor, který provádí analýzu metodou shora dolů. Označení $LL(k)$ znamená čtení vstupní věty zleva doprava (left to right), vytváří se levý rozklad (left parse) a při výběru pravidla je potřeba znát nejvýše k symbolů z dosud nepřečtené části vstupní věty. Ačkoliv LL analyzátory nemají takové vlastnosti jako LR analyzátory, přesto jsou velmi oblíbené a používané kvůli menší náročnosti realizace (Češka, 1992).

Hlavním problémem při syntaktické analýze metodou shora dolů je rozhodnout, jaké použít pravidlo, když máme u jednoho neterminálu na výběr několik možností

na jeho přepis. Existují proto speciální typy gramatik, které se snaží eliminovat nejednoznačnosti při přepisu zavedením určitých omezení pro tvar pravidel.

Jednoduché LL(1) gramatiky

Nejjednodušším typem LL gramatik je *jednoduchá* $LL(1)$ gramatika neboli $SLL(1)$. Omezení tvaru jejích pravidel spočívá v tom, že pravá strana každého pravidla musí začínat terminálem a pro dvě stejné levé strany musí pravé strany začínat různými terminály. Není tedy možné používat ϵ-pravidla, díky čemuž nelze popsat některé důležité struktury jako např. prázdný řetězec (Habiballa, 2013).

Q-gramatiky

Tento nedostatek řeší q-gramatika, která rozšiřuje $SLL(1)$ gramatiku o možnost použití ϵ-pravidel. Má to ovšem jednu podmínku. Obsahuje-li gramatika pravidlo $X \rightarrow \epsilon$, pak terminály, kterými začínají pravé strany ostatních X-pravidel, nesmí patřit do $FOLLOW(X)$. $FOLLOW(X)$ je funkce, jejímž výsledkem je množina všech terminálů, které mohou následovat za symbolem X, včetně ϵ, pokud za X už nemusí následovat žádný symbol (Habiballa, 2013).

Přestože má q-gramatika již vysokou míru expresivity, může být zvlášť u složitějších gramatik problém s přehledností a systematičností. Tento menší nedostatek řeší gramatika $LL(1)$.

6.5.5 LL(1) gramatiky

Gramatika $LL(1)$ umožňuje zapsat pravidlo tak, aby jeho pravá strana začínala neterminálem. Přesto se i zde mohou vyskytnout kolize. Abychom je dokázali odhalit, musíme využít mimo jiné i funkci $FIRST(\alpha)$. Jejím výsledkem je množina terminálů, kterými mohou začínat řetězce odvozené z α, resp. ϵ, pokud se z α dá odvodit ϵ.

Podle definice (Vavrečková, 2008) lze gramatiku nazvat $LL(1)$, když každá množina pravidel se stejnou levou stranou $A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$ má tyto vlastnosti:

- $FIRST(\alpha_1) \cap FIRST(\alpha_2) = \emptyset$.
- Pokud z řetězce α_1 je možné generovat prázdný řetězec a z řetězce α_2 není možné generovat prázdný řetězec, pak $FOLLOW(A) \cap FIRST(\alpha_2) = \emptyset$.

Transformace na LL(1) gramatiku

Může se stát, že námi vytvořená bezkontextová gramatika není $LL(1)$. V takovém případě existuje několik metod, pomocí kterých se můžeme pokusit danou gramatiku transformovat na $LL(1)$. Nedá se ovšem říct, že by tyto transformace vedly vždy spolehlivě k cíli, jedná se nedeterministický postup. Pro některé bezkontextové

jazyky totiž ani $LL(1)$ gramatika sestrojit nejde. Transformaci lze provést pomocí těchto metod (Habiballa, 2013):

- *Odstranění levé rekurze* – zavedením nového neterminálu lze odstranit případ, kdy se neterminál přepisuje na řetězec začínající jím samým.
- *Levá faktorizace* – pokud dvě nebo více pravidel začíná stejným řetězcem, tento řetězec se vytkne a zavede nový neterminál, který bude přepisovat na zbytky řetězců. Řeší $FIRST$-$FIRST$ kolize.
- *Rohová substituce* – všechny výskyty neterminálu jsou nahrazeny všemi jeho pravými stranami.
- *Pohlcení terminálního symbolu* – řeší situaci, kdy za neterminálem následuje řetězec, kterým může některé pravidlo od tohoto neterminálu začínat. Vznikne tak nový neterminál, který reprezentuje kolidující terminál s daným neterminálem. Řeší $FIRST$-$FOLLOW$ kolize.
- *Pohlcení řetězce* – tato metoda je podobná jako pohlcení terminálu, jen místo jednoho terminálu je pohlcen řetězec.

Překladový automat

Překladový automat je model syntaktického analyzátoru pracující metodou shora dolů. Jedná se o zásobníkový automat, který navíc obsahuje výstupní pásku a je definován *rozkladovou tabulkou*. Rozkladová tabulka určuje pro každý neterminál a následující symbol, jaké pravidlo máme při expanzi použít.

Konstrukce rozkladové tabulky se liší pro různé typy LL gramatik, uvedeme si tedy postup jejího vytvoření pro gramatiku $LL(1)$. Nejprve ohodnotíme řádky tabulky neterminály a sloupce vstupními symboly (terminály a ϵ). Pro pravidla, která přepisují na neprázdné řetězce spočítáme $FIRST$ těchto řetězců a poté do buněk příslušných sloupců toto pravidlo vložíme. Pro pravidla přepisující na prázdné řetězce spočítáme $FOLLOW$ neterminálu, který přepisují a následně vložíme toto pravidlo do příslušných sloupců symbolů resp. ϵ pro nalezené prvky $FOLLOW$. V ostatních případech nastane chyba. Každá buňka rozkladové tabulky může obsahovat nejvýše jedno pravidlo, jinak nebude možné provést deterministickou analýzu (Habiballa, 2013).

Po sestrojení rozkladové tabulky je možné provést rozpoznání daného slova s využitím algoritmu pro $LL(1)$ syntaktickou analýzu. Algoritmus provádí přechody mezi konfiguracemi. Konfigurace je zde trojice (x, α, π), kde x je dosud nepřečtená část slova, α obsah zásobníku a π je posloupnost čísel pravidel. Přechody mezi konfiguracemi se provádí podle následujících kroků 1. a 2., dokud nenastane krok 3. nebo 4 (Habiballa, 2013).

1. *Expanze* – neterminál na vrcholu zásobníku se nahradí příslušným řetězcem a číslo pravidla je připojeno k posloupnosti znázorňující levý rozklad.

2. *Porovnání* – porovná se symbol vstupu se symbolem na vrcholu zásobníku. Pokud jsou totožné, symbol ze vstupu se přečte a symbol ze zásobníku odstraní.
3. *Přijetí* – nastane v konfiguraci $(\epsilon, \epsilon, \pi)$, řetězec je rozpoznán a analýza končí.
4. *Chyba* – v ostatních případech analýza končí chybou.

Výstupem tohoto algoritmu je tedy levý rozklad vstupního slova pokud $w \in L(G)$, jinak je výstupem chyba.

Pro implementaci $LL(1)$ překladového automatu je potřeba $LL(1)$ gramatika, vypočtené množiny $FIRST$ a $FOLLOW$ a případně rozkladová tabulka. K dispozici máme dle (Vavrečková, 2008) dvě metody, pomocí kterých můžeme implementaci provést: metoda přepisu rozkladové tabulky a metoda rekurzivního sestupu.

Metoda přepisu rozkladové tabulky obsahuje všechny tři analýzy (lexikální, syntaktickou i sémantickou). Při její implementaci je potřeba zásobník, který na začátku obsahuje symbol konce zásobníku # a startovací symbol gramatiky. Roli lexikálního analyzátoru plní funkce `Lex()`, která po zavolání vrací jeden symbol. Při programování syntaktické analýzy jsou potřeba funkce reprezentující příslušné akce z rozkladové tabulky: `expand(číslo_pravidla)`, `pop()`, `accept()` a `error()`. Navíce obsahuje funkci `Akce()`, která pracuje tak, jak bychom sami pracovali s „papírovou“ rozkladovou tabulkou.

Metoda rekurzivního sestupu je velmi oblíbená a přehledná metoda. Je založena na používání rekurze, takže není potřeba zásobník. Není nutné vytvářet ani rozkladovou tabulku, ale je třeba stejně vytvořit všechny množiny potřebné k její konstrukci. Funkce `Lex()` znovu provádí lexikální analýzu a vrací vždy jeden symbol. Dále se postupuje tak, jako bychom vytvářeli derivační strom „ručně“. Rekurze je řešena pomocí volání neterminálů. Proto jsou vytvořeny funkce, které dané neterminály reprezentují a jsou i stejně pojmenované. Na místě neterminálu tedy voláme tyto funkce, oproti tomu terminály se jen porovnávají se vstupem. Uvnitř funkcí neterminálů zpracováváme řetězec pravé strany pravidla. Například uvnitř funkce `A()` vyhodnocujeme všechny symboly řetězce α přesně podle pravidla $A \to \alpha$. Postupujeme vždy zleva doprava. U terminálních symbolů dojde jen k porovnání a následnému načtená dalšího vstupního symbolu. Když rekurze skončí a je přečten celý vstup, dojde k akceptaci. Výhody této metody jsou podle (Grune a Jacobs, 2007) tyto:

- jednoduché navázání sémantických akcí,
- vysoká efektivita, často větší než u přepisu rozkladové tabulky.

Nevýhodou může teoreticky být velikost takového parseru a s tím související vyšší paměťová náročnost. V dnešní době by to však nemělo působit žádné problémy.

7 Implementace

7.1 Výběr parametrů

Prvním krokem při tvorbě návrháře stylů je výběr parametrů, které bude moct uživatel pomocí grafického rozhraní nastavit. Tyto parametry vychází z předchozí kapitoly věnující se formátování písma a odstavce, což spadá pod smíšenou a odstavcovou sazbu. Do těchto dvou kategorií budou také parametry členěny.

Základním kritériem pro výběr parametrů je množina znalostí vyučovaných v předmětu Zpracování textů na počítači. Jelikož návrhář stylů by měl podle (Přichystal, 2015b) v podstatě plnit funkci výukového nástroje, jde o to zvolit parametry, které jsou při výuce tohoto předmětu nejvíce používané pro tvorbu vlastních strukturních značek. Smyslem není poskytnout komplexní řešení, které umožní nastavit všechny vlastnosti částí dokumentu. Takové řešení by bylo velmi komplikované, což v je v rozporu v hlavní myšlenkou návrháře stylů, že by mělo jít o jednoduchý a uživatelsky přívětivý nástroj určený hlavně začínajícím uživatelům. Výběr parametrů byl navíc konzultován s Ing. Janem Přichystalem, Ph. D., který je autorem aplikace TEXonWeb i vyučujícím předmětu Zpracování textů na počítači. Přehled vybraných parametrů pro změnu formátování písma a odstavce je uveden v tabulce 3, případně v tabulce 4. Parametry jsou doplněny zápisem příslušného příkazu v TEXu.

Tabulka 3: Vybrané parametry pro změnu formátování písma

Parametr	**Příkaz**
rodina písma	`\fontspec`
neproporcionální písmo	`\texttt`
tučný řez písma	`\textbf`
skloněný řez písma	`\textsl`
kurzívní řez písma	`\textit`
kapitálky	`\textit`
stupeň písma	`\tiny` až `\Huge`

Tabulka 4: Vybrané parametry pro změnu formátování odstavce

Parametr	**Příkaz**
sazba na praporek vpravo	prostředí `flushleft`
sazba na praporek vlevo	prostředí `flushright`
sazba na střed	prostředí `center`
levý okraj	`\leftskip`
pravý okraj	`\rightskip`
vodorovná mezera	`\hspace`

Samostatným bodem je pojmenování parametrů tak, aby byly dostatečně srozumitelné pro většinu uživatelů. Z tohoto důvodu došlo k určitým zjednodušením v názvech některých parametrů či jejich hodnot namísto přesných pojmů z oblasti sazby a typografie. Na základě konzultace byly vybrány názvy, které uživatelé znají z jiných programů, hlavně tedy z MS Word, což by mělo zajistit lepší přístupnost pro začínající uživatele. Parametr „stupeň písma“ byl tedy pojmenován jako „velikost“, „polotučný řez“ jako „tučný“, zarovnání na praporek vlevo nebo vpravo jako „zarovnání vlevo“ a „zarovnání vpravo“, „vodorovná mezera“ jako „vertikální odsazení“.

7.2 Návrh uživatelského rozhraní

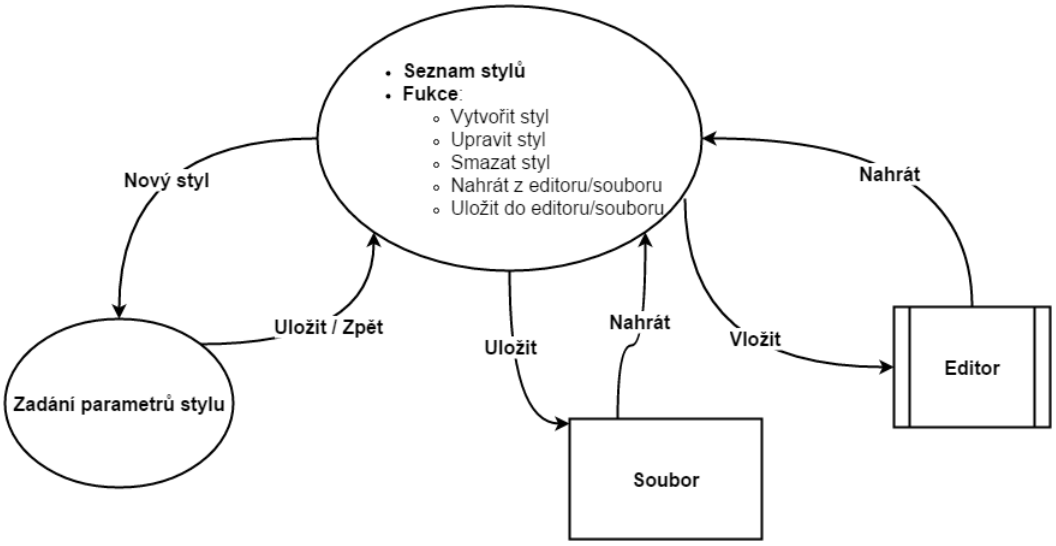
Poté co byly vybrány parametry, které budou obsaženy v návrháři stylů, přistoupí se k návrhu uživatelského rozhraní. Prvním krokem návrhu je podle (Saffer, 2010) definování účelu a funkcionality. Vytvořený návrhář stylů bude komponenta pro webovou aplikaci TEXonWeb, která umožní zejména začínajícím uživatelům tvorbu vlastních strukturních značek v typografickém systému TEX pomocí grafického rozhraní. Funkcionalita návrháře bude následující:

- vytváření nových strukturních značek (maker),
- editace a mazání těchto značek,
- ukládání značek do souboru,
- vkládání značek přímo do editoru aplikace TEXonWeb,
- načítání definic maker vytvořených návrhářem ze souboru nebo z editoru pro možnost jejich zpětné editace.

7.2.1 Konceptuální model

Funkcionalita je tedy definována, přesto ještě není ideální začít kreslit návrh grafického rozhraní. Zaměřit se na konkrétní fyzickou implementaci grafických komponent nás může hodně svázat. Lepším postupem je soustředit se nejprve na abstraktní návrh. Většinu aplikací a jejich částí je totiž možné charakterizovat pomocí popisů jako *seznam objektů*, *seznam operací*, *seznam kategorií* nebo *seznam nástrojů*. Tyto popisy jasně specifikují svůj účel, přesto jsou naprosto abstrahovány od konkrétního vzhledu (Tidwell, 2010).

Pro abstraktní návrh lze použít model, který se nazývá *konceptuální*. Ten popisuje, co může uživatel s daným systémem provádět a co od něj může očekávat. Konceptuální model pak slouží jako základ pro návrh uživatelského rozhraní. Díky tomuto procesu je výsledná aplikace jednodušší a lépe pochopitelná. Konceptuální model by měl být co nejjednodušší a obsahovat pouze důležitou funkcionalitu (Henderson a Johnson, 2013). Pro návrháře stylů byl vytvořen konceptuální model, který lze vidět na obrázku 7.



Obrázek 7: Konceptuální model návrháře uživatelských stylů

7.2.2 Struktura

Nyní je třeba se zamyslet nad samotnou implementací uživatelského rozhraní. Existují v podstatě tři základní typy aplikací (Saffer, 2010):

- aplikace využívající více oken,
- aplikace s jedním rozděleným oknem,
- aplikace s jedním oknem měnícím obsah.

Návrhář stylů sice není samostatná aplikace, jedná se o komponentu v rámci aplikace TEXonWeb, ale při návrhu jejího uživatelského rozhraní budeme používat stejné principy, jako by o aplikaci šlo. Dopustíme se tak drobného zjednodušení.

Výběr typu aplikace je velmi závislý na platformě, na které bude použita. Návrhář stylů bude obsahovat strukturu s jedním oknem, které mění svůj obsah. Na tomto modelu jsou postaveny webové stránky, tudíž jsou na něj uživatelé zvyklí. Jeho použití je vhodné také pro menší displeje, kde je kladen důraz na přehlednost a intuitivitu (Saffer, 2010). To je užitečné zejména z toho důvodu, že návrhář stylů bude využíván i na mobilních zařízeních.

Obecnou strukturu návrháře máme popsánu, nyní je třeba zvolit vzor (pattern), který danou strukturu implementuje. Jak uvádí (Tidwell, 2010), existuje několik vzorů pro strukturu aplikace. Tyto vzory různými způsoby poskytují uživateli informace nebo nástroje. Pro příklad lze uvést vzory:

- *Two-panel selector* – dva panely, vybraním elementu v prvním zobrazí jeho obsah do druhého.
- *Canvas Plus Palette* – používáno hlavně v grafických nástrojích.

- *One-Window Drilldown* – aplikace zabírá vždy celé okno.
- *Alternative views* – přizpůsobení zobrazení odlišnou formou.

Pro návrháře stylů bude zvolen vzor *One-Window Drilldown*. Ten v každém okamžiku zabírá vždy celé okno, takže pokud uživatel vybere některou volbu, ta se zobrazí opět přes celé okno. V našem případě přes celé okno návrháře. Tento přístup je vhodný zejména pro použití na zařízeních s malou obrazovkou, kdy není k dispozici příliš prostoru, případně v aplikacích pro méně zkušené uživatele. Obě podmínky návrhář splňuje. Hlavní podstata tohoto vzoru spočívá v linearizaci činnosti uživatele, k tomu je třeba poskytnout jasná tlačítka pro akce typu vpřed, zpět či zrušit (Tidwell, 2010).

7.2.3 Navigace

Po volbě struktury je potřeba vybrat způsob navigace. Ta je důležitá k tomu, aby uživatel dosáhl cíle, ke kterému byla aplikace vytvořena a také aby v každém okamžiku věděl, na kterém místě se zrovna nachází. V jednom okně obvykle nelze zobrazit vše, to by bylo pro uživatele příliš matoucí. Některé elementy je potřeba schovat, důležité však je, aby cesta k nim byla co nejkratší.

Návrhář stylů bude využívat navigační vzor *Hub And Spoke*, který patří mezi vzory řešící přechod ze stavu do stavu. Tento vzor je hojně používán na mobilních zařízeních či v aplikacích využívajících celé okno. Uživateli je poskytnut centrální vstup (Hub), ze kterého se lze dostat k jednotlivým akcím (Spoke), zde provést požadovaný úkon a vrátit se zpět do centrálního vstupu (Tidwell, 2010). Návrhář stylů bude tedy obsahovat úvodní panel, ze kterého se bude dát přepnout na další panely jako vytváření nebo editace stylu a po uložení se vrátit zpět. Podobně bude fungovat také načtení stylů.

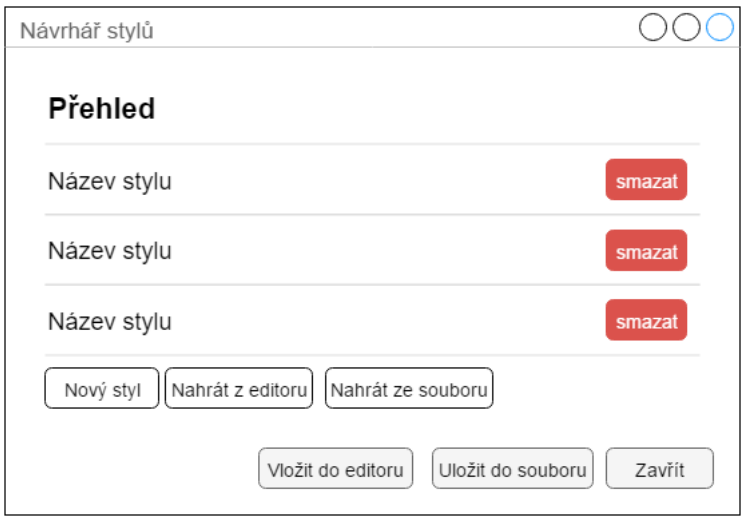
7.2.4 Úvodní panel

Na úvodním panelu bude seznam vytvořených stylů, kde budou uvedeny jejich názvy. U každého stylu bude k dispozici tlačítko pro jeho smazání. Po kliknutí na název stylu dojde k otevření panelu s editací. Dále zde budou přítomny tlačítka pro vytvoření stylu, vložení do editoru, uložení do souboru, nahrání z editoru, nahrání ze souboru a zavření okna návrháře. Návrh úvodního panelu lze vidět na obrázku 8.

V případě vzoru Hub And Spoke, který má omezené navigační možnosti, je třeba na každou obrazovku umístit element umožňující návrat do známé části aplikace. Tento přístup se označuje jako *Escape Hatch*. V návrháři bude implementován tlačítkem zpět, které bude vracet uživatele z různých obrazovek znovu na úvodní.

Je třeba se také zaměřit na přechody mezi jednotlivými stavy, v tomto případě přepnutí z jedné obrazovky na druhou. Náhlá změna totiž může uživatele zmást. Mnohem přirozenější je, když změna stavu probíhá postupně a plynule. Tomuto vzoru se říká *Animated Transitions*. Přidání vhodně zvolené animace nejenže působí

dobrým vizuálním dojmem, ale také udržuje orientaci uživatele. Animace však nesmí být příliš dlouhá, aby uživatele neobtěžovala. Ideální délka animace je 300–1000 ms (Tidwell, 2010). Do návrháře stylů budou tedy přidány animace pro lepší znázornění přechodů mezi obrazovkami.



Obrázek 8: Návrh úvodního panelu návrháře uživatelských stylů

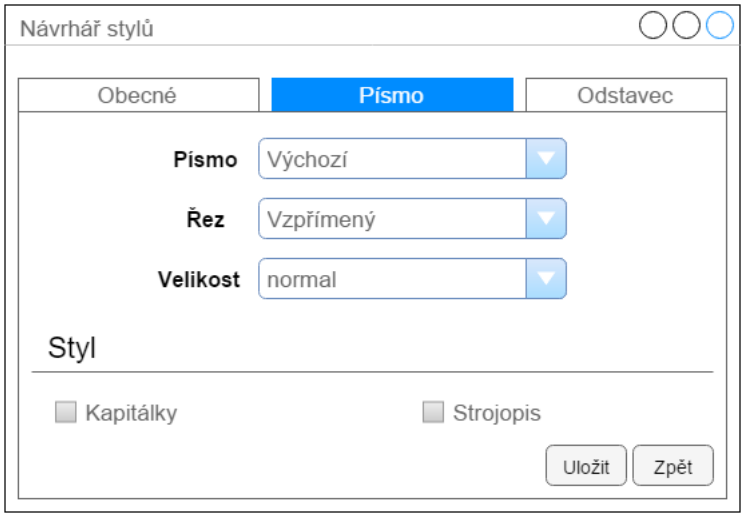
7.2.5 Panel pro tvorbu stylu

Další důležitou obrazovkou bude panel pro vytvoření nového stylu. Ten bude rozdělen na tři části, kde se bude zadávat název stylu, parametry písma a parametry odstavce. Každá část bude mít svou záložku, ale v jeden okamžik bude viditelný obsah pouze jedné z nich. Kliknutím na název záložky se obsah přepne. Tento vzor je specifikován jako *Modular Tabs*. Slouží hlavně k vizuálnímu oddělení obsahu a udržení přehledného rozhraní se spoustou elementů (Tidwell, 2010).

Nastavení parametrů bude řešeno pomocí klasických formulářových prvků (viz obrázek 9). Název stylu, rozměry okrajů a vertikálního odsazení budou implementovány jako jednoduché textové pole, písmo, řez, velikost a zarovnání budou vytvořeny jako výběr z několika možností (Select), kapitálky a strojopis budou nastavovány přes zaškrtávací políčko (Checkbox). Pro uložení stylu bude sloužit tlačítko vpravo dole, které by mělo být na konci tzv. „visual flow“, tedy na místě kam uživatel směřuje svůj pohled po vyplnění všech parametrů stylu a očekává potvrzení své akce. Název tohoto vzoru je *Done Button* (Tidwell, 2010).

Formulářové prvky pro nastavení parametrů budou obsahovat co nejvíce předvyplněných hodnot, které vedou k úspoře času, jak udává vzor *Good Defaults* (Tidwell, 2010). Parametry písma, řezu, velikosti a zarovnání budou nastaveny podle výchozích hodnot pro sazbu dokumentu. Pro rozměry parametrů okraje a vertikálního odsazení je výchozí hodnota nastavena na 0 em, aby uživatelům bylo jasné, v jakém formátu mají tuto hodnotu zadávat.

Dále je třeba kontrolovat, zda jsou hodnoty parametrů správně vyplněny. To je důležité hlavně u textových polí, kde může uživatel zadat cokoliv. Proto je potřeba uživatele upozornit, pokud do pole vloží nepovolený znak. Ideální způsob řešení je pomocí vzoru *Same-Page Error Messages* (Tidwell, 2010), kdy je kontrola validity prováděna již v době vyplňování a chybné pole je hned vyznačeno červenou barvou. Uživatel tedy okamžitě vidí, zda zadává hodnotu správně nebo špatně.



Obrázek 9: Návrh panelu pro tvorbu stylu v návrháři uživatelských stylů (sekce písmo)

7.3 Návrhář uživatelských stylů

Aplikační logika návrháře stylů se nachází v souboru `js/styles_designer.js`. Pro implementaci byl zvolen programovací jazyk JavaScript, který se v aplikaci TEXonWeb používá pro tvorbu skriptů na straně klienta. Pro usnadnění práce s tímto jazykem je navíc hojně využívána i knihovna jQuery. Mezi její hlavní přednosti patří konzistentní API fungující napříč různými prohlížeči, usnadnění práce při manipulaci s DOM nebo zasílání požadavků metodou AJAX.

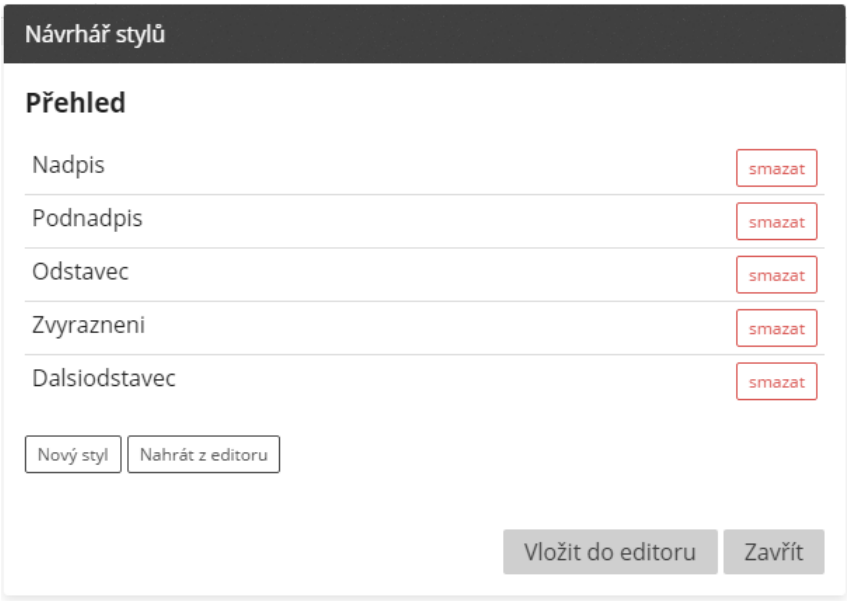
Aplikace TEXonWeb definuje na globální úrovni JavaScriptový objekt `TOW`, který obsahuje všechny komponenty a další specifické vlastnosti s nastavením. Návrhář stylů je přidán do tohoto objektu a přistoupit k němu lze pomocí zápisu `TOW.StylesDesigner`. Zapouzdření kódu návrháře stylů je řešeno skrze objektový literál, který obsahuje jeho vlastnosti a metody. Ty jsou zapsány ve formátu `key:value` a oddělené čárkou.

Návrhář stylů se inicializuje pomocí metody `TOW.StylesDesigner.init()`, která je volána v okamžiku, kdy je načten DOM. To je zajištěno funkcí `$(document).ready()` poskytovanou knihovnou jQuery (Chaffer a Swedberg, 2013). Metoda `init()` nastartuje inicializační proces, který je rozdělen na tři části a vykonává se v těchto metodách:

- `startup()` – nastavuje výchozí hodnoty vlastností,
- `initElements()` – ukládá vybrané elementy z DOM do proměnných kvůli znovupoužitelnosti a lepší výkonnosti,
- `bindEvents()` – nastaví daným elementům funkce pro zpracování událostí.

Je vhodné si blíže představit metodu `bindEvents()`. Ta je důležitá hlavně tím, že obsahuje funkci pro zpracování události, ve které se načítá šablona uložená v souboru `js/templates/styles-designer.hbs`. V této šabloně se vyskytuje HTML kód popisující strukturu návrháře stylů. V JavaScriptovém kódu se totiž velmi špatně manipuluje s HTML. Celá struktura se stává velice nepřehlednou, proto je potřeba od sebe JavaScriptový kód a HTML oddělit. To je zvláště důležité, je-li HTML kódu velké množství. K tomuto účelu byl využit šablonovací systém Handlebars[5], který pomocí speciálních výrazů uzavřených mezi dvojité složené závorky {{…}} umožňuje třeba výpis proměnných nebo zápis kontrolní logiky. Šablona se načte při prvním otevření okna návrháře pomocí AJAXu. Následně se její obsah zkompiluje do čistého HTML, které se vloží do elementu představujícího okno návrháře. Poté co dojde k jejímu načtení, je spuštěna inicializační metoda `initTemplateElements()`, která uloží veškeré elementy obsažené v šabloně do proměnných.

Šablona obsahuje HTML elementy, které mají přiřazené jednotlivé třídy a identifikátory. Pro ně je pak nastaven vzhled pomocí kaskádových stylů (CSS). Vzhled a rozmístění prvků se řídí návrhem z části Návrh uživatelského rozhraní. Výslednou podobu rozhraní je možné vidět na obrázku 10.



Obrázek 10: Uživatelské rozhraní návrháře uživatelských stylů

[5]Dostupné na `http://handlebarsjs.com/`

7.3.1 Správa stylů

Samotné styly jsou uloženy v objektu `StylesDesigner` jako vlastnost `styles`. Jedná se o pole objektů, jejichž struktura reprezentuje parametry vybrané v předcházející části (viz ukázka kódu 1). Ve výchozím stavu je toto pole prázdné.

```
{
  name: "Nadpis",
  font: {
    caps: true,
    face: "Calibri",
    size: "normalsize",
    style: "textbf",
    type: false
  },
  par: {
    align: "flushleft",
    hspaceLeft: "0em",
    hspaceRight: "0em",
    vspaceAfter: "0em",
    vspaceBefore: "0em"
  }
}
```

Ukázka kódu 1: Objekt reprezentující vytvořený styl

Kliknutím na tlačítko „Nový styl“ na úvodním panelu se zobrazí panel pro tvorbu nového stylu. Tento panel je rozdělen na tři části, ve kterých se nachází formulářové prvky pro nastavení hodnot jednotlivých parametrů. Většinu prvků tvoří výběr z několika možností, případně zaškrtávací políčka. Jsou zde ovšem také textová pole pro zadání názvu a rozměrů. U těchto elementů je potřeba kontrolovat, co zde uživatel vyplňuje, protože jejich hodnoty musí být ve specifickém formátu. Pro tyto prvky tedy byla vytvořena validace kontrolující hodnotu již během psaní (tzv. „live validace“).

Validace je spuštěna při vyvolání události `input` nebo `propertychange`. Ty reagují na změnu hodnoty příslušného elementu, takže pokrývají jak psaní na klávesnici, tak přímé vložení textu ze schránky. Událost `input` je součástí specifikace HTML5, takže moderní prohlížeče ji plně podporují. Událost `propertychange` je zde uvedena kvůli starším verzím prohlížečů z rodiny Internet Explorer.

Při validaci názvu stylu se po každé změně hodnoty zavolá metoda `onStyleNameChange()`. Ta kontroluje, zda název obsahuje pouze malá nebo velká písmena bez diakritiky. Pokud název není validní, nastaví se rodičovskému elementu obalujícímu daný formulářový prvek třída `.has-error`, díky které je okraj textového pole a jeho obsah obarven červenou barvou. Při vyplnění správné hodnoty se tato třída zase odstraní a zmizí tak červené vyznačení.

Validaci rozměru řeší metoda `onDimensionChange()`, která je volána při změně hodnoty textového pole, stejně jako u názvu. Pole pro zadání rozměrů se vyskytují

u parametrů levého a pravého okraje a vertikálního odsazení před a za odstavcem. Kontrolu validního formátu řeší metoda `validateDimension()`, která testuje, zda má rozměr nepovinné znaménko mínus na začátku, poté libovolný počet číslic, nepovinnou desetinnou část a na konci jednotku podporovanou systémem TEX. Implementaci validace rozměru lze vidět v ukázce kódu 2. Pokud rozměr není validní, stejně jako u názvu je okraj textového pole a jeho text označen červeně.

```
// fce vyvolana pri zmene rozmeru
onDimensionChange: function (e) {
  var value = $(e.target).val();
  var parent = $(e.target).parent();
  if (!this.validateDimension(value)) {
    // pridani tridy .has-error, oznaci input cervene
    parent.addClass('has-error');
  } else {
    parent.removeClass('has-error');
  }
},

// validace rozmeru
validateDimension: function (dimension) {
  dimension = dimension.replace(/\s/g, '');
  if (dimension === '') {
    return true;
  }
  return /^-?\d+(\.\d+)?(pt|pc|in|bp|cm|mm|dd|cc|sp|ex|em)$/.test(
      dimension);
}
```

Ukázka kódu 2: Validace formulářového prvku pro zadání rozměru

Po kliknutí na tlačítko „Uložit“ je zavolána metoda `saveStyle()`. Ta nejdříve uloží všechny hodnoty formulářových prvků do proměnných a z nich je poté sestaven objekt `style`, jehož struktura byla uvedena výše. Následně je tento objekt podroben validaci pomocí metody `validateStyle()`. Tato validace zabraňuje uložení stylu s hodnotami v nepovoleném formátu. Opět se zde kontrolují hodnoty rozměrů, název a také zda styl se stejným názvem již neexistuje. V okamžiku výskytu chyby v jednom z těchto případů se zobrazí vyskakovací okno s popisem chyby a styl se neuloží. Pokud validace skončí úspěchem, objekt style je uložen do pole `styles` a zobrazí se úvodní panel s přehledem stylů, tentokrát už se zobrazeným nově vytvořeným stylem.

Pokud máme vytvořený styl, po kliknutí na něj jej můžeme upravit. Editace stylu probíhá téměř stejně jako vytváření. Rozdíl je v tom, že jsou zde nastavené hodnoty tak, jak je daný styl měl uložené. Také je nastavena vlastnost `StylesDesigner.editMode` na název aktuálně upravovaného stylu, což slouží k ověření, že je zapnut režim úpravy stylu a ne vytváření. To najde využití zejména při ukládání úprav, kde se používá metoda `saveStyle()`. Díky nastavenému `editMode`

nedojde k přidání nového stylu, ale v poli `styles` se vyhledá objekt se stejným názvem a ten se přepíše.

Na úvodním panelu je v přehledu u každého stylu dostupné tlačítko pro smazání. Po kliknutí na něj se uživateli zobrazí vyskakovací okno s dotazem, zda chce opravdu styl smazat. To slouží jako prevence před případným nechtěným smazáním stylu.

7.3.2 Ukládání dat do paměti prohlížeče

Nyní můžeme uvažovat následující situaci: V přehledu návrháře vidíme několik vytvořených stylů. Pokud bychom však z nějakého důvodu obnovili stránku v prohlížeči nebo otevřeli jiný soubor v aplikaci TEXonWeb, vytvořené styly budou smazány a museli bychom je znovu vytvořit. Takové chování rozhodně nepřispívá uživatelské přívětivosti. Jelikož TEXonWeb neobsahuje žádnou databázovou vrstvu, je potřeba se podívat na dostupná řešení na straně prohlížeče. Potřebnou funkcionalitu nabízí technologie *Web Storage*.

Technologie Web Storage byla dříve součástí specifikace HTML5, nyní už se však jedná o samostatnou část. Poskytuje aplikační rozhraní pro uchovávání dat v paměti prohlížeče napříč HTTP požadavky. K podobnému účelu slouží i cookies, jenže ty mají omezenou velikost na 4 kB a přenáší se při každém požadavku, což zvyšuje objem přenesených dat. Navíc data uložená v cookies jsou viditelná, pokud nejsou šifrovaná, což může představovat bezpečnostní riziko (Lubbers, Albers a Salim, 2011).

Web Storage umožňuje ukládat data jako dvojice klíč:hodnota, kde jako hodnota mohou být JavaScriptové objekty. Poskytovaný prostor se liší podle prohlížeče, ale jak uvádí specifikace, minimálně by mělo jít o 5 MB pro jednu doménu (W3C, 2015). Použití Web Storage je tak mnohem intuitivnější a mocnější než je tomu v případě cookies.

V rámci Web Storage existují dva typy úložných prostorů: *session storage* a *local storage*. Ty se liší hlavně tím, jak dlouho drží uložená data v paměti prohlížeče a pro jakou část jsou viditelná (viz tabulka 5).

Tabulka 5: Rozdíly mezi úložišti session storage a local storage (Lubbers, Albers a Salim, 2011)

session storage	**local storage**
Data jsou uložena do zavření okna nebo záložky prohlížeče.	Data zůstávají uložena i po zavření prohlížeče.
Data jsou viditelná pouze v rámci okna nebo záložky, ve které byla vytvořena.	Data jsou sdílena mezi všemi okny a záložkami v rámci jedné domény.

Pro účely návrháře stylů bylo vybráno jako vhodnější úložiště local storage, které dokáže uchovat data i po zavření okna prohlížeče. Uživatel by tak o vytvořené styly neměl přijít, pokud je sám nesmaže.

Implementace je provedena v objektu `StylesDesigner.storage`, který má definovaný `namespace` *tow.styles*, což je klíč, pod kterým jsou data návrháře stylů uložena. Objekt `storage` obsahuje dvě metody starající se o manipulaci s daty: `load()` a `store()`. Jak vyplývá z názvu, metoda `load()` zajišťuje načtení dat z local storage do návrháře a je volána v metodě `startup()`, hned při samotné inicializaci návrháře. Metoda `store()` ukládá data do local storage a volá se při vytvoření nového stylu, jeho úpravě nebo smazání.

Jelikož ne všechny prohlížeče mohou podporovat technologii Web Storage, jedná se hlavně o starší verze prohlížečů, je potřeba tuto podporu kontrolovat v rámci implementace. K tomu slouží knihovna *Modernizr*[6], která informuje o tom, jestli prohlížeč danou technologii podporuje. V rámci návrháře byla použita právě detekce local storage použitím vlastnosti `Modernizr.localstorage`. Díky tomu v nepodporovaných prohlížečích nedojde k chybě a data se do local storage neuloží.

7.3.3 Vytváření definic maker

Ze stylů vytvořených v návrháři lze vygenerovat definice maker, které pak jdou použít v jednotlivých dokumentech. Návrhář stylů obsahuje dvě možnosti, jak je možné definice maker vygenerovat: Buď je možné uložit je do stylového souboru s příponou `.sty` nebo je vložit přímo do editoru.

Vložení vygenerovaného kódu do editoru je řešeno v metodě `onInsert StyleToEditor()`. Nejprve je pomocí metody `createStyleContent()` vytvořen řetězec, který obsahuje definice maker na základě parametrů vytvořených stylů. Poté se s využitím aplikačního rozhraní editoru Ace hledají v jeho obsahu pozice řetězců *%begin-styles%* a *%end-styles%*. Tyto dva řetězce ohraničují blok definic maker vygenerovaných návrhářem stylů. Pokud jsou v obsahu editoru nalezeny, definice jsou vloženy na toto místo a původní obsah je přepsán. V opačném případě se definice maker vloží na místo aktuálního kurzoru. Tato funkcionalita je užitečná zejména k tomu, když máme v editoru již vygenerované definice, v návrháři provedeme nějaké úpravy a chceme tyto definice aktualizovat. Návrhář tedy automaticky staré definice nahradí a my je nemusíme mazat ručně.

Pro uložení definic maker do souboru je potřeba, aby byl uživatel přihlášen. Proto nepřihlášeným uživatelům se tato možnost na úvodním panelu návrháře ani nezobrazí. Po kliknutí na tlačítko „Uložit do souboru“ se zavolá metoda `onCreateStyleFile()`. Ta zavře okno návrháře a vyvolá správce souborů v režimu pro uložení souboru, kde je v políčku pro název předvyplněna přípona `.sty`. Pro uložení souboru s definicemi bylo potřeba modifikovat skript `js/spravce_souboru.js`, kde byla přidána metoda `saveStyles()`. Ta zajistí uložení souboru a poté vyvolá pomocí funkce `trigger()` událost *stylesSaved*, jejíž para-

[6]Dostupné na `https://modernizr.com/`

metr je název uloženého souboru. Tato událost je zpracována v návrháři stylů skrze metodu `onStylesFileSaved()`, která pomocí metody `insertPackage()` vloží do editoru příkaz `\usepackage{balík}` pro připojení stylového souboru, kde „balík“ je název tohoto souboru bez přípony.

7.3.4 Načítání vytvořených definic maker

Součástí funkcionality návrháře stylů je i načítání definic maker, které jím byly vygenerované, aby je bylo možné zpětně modifikovat. To znamená, že z takto načtených definic maker se opět sestaví JavaScriptové objekty stylů, které se objeví v přehledu na úvodním panelu návrháře a mohou být pomocí grafického rozhraní znovu upraveny podle potřeby. Definice maker je možné načíst jak ze souboru, tak i z editoru. Obě tyto možnosti jsou dostupné na úvodním panelu pomocí tlačítek „Nahrát z editoru“ a „Nahrát ze souboru“.

Načtení definic maker z editoru řeší metoda `loadStyle()`. Ta nejdříve hledá v textu editoru blok s definicemi ohraničený řetězci *%begin-styles%* a *%end-styles%*. V případě nalezení je získán obsah tohoto bloku a předán jako parametr metodě `handleLoadStyle()`. Tato metoda vytvoří instanci třídy `Parser`, který ověří správnost syntaxe daného textu a pokud je vše v pořádku, vytvoří na základě něj objekty stylů. Tvorba parseru bude podrobně popsána v další části. Následně probíhá kontrola, zda už styl se stejným názvem není vytvořen. Pokud již styl existuje, uživatel jej může přepsat nebo přejmenovat. Poté je styl uložen do pole `styles` a zobrazí se v přehledu stylů, kde může být opět modifikován. V případě že se během tohoto procesu vyskytne chyba, uživateli je zobrazena chybová hláška s popisem. Pokud tedy uživatel upraví kód definic maker a způsobí v něm překlep nebo syntaktickou chybu, návrhář mu oznámí, o jakou chybu se jedná, případně jak ji opravit.

Načtení definic maker ze souboru je implementováno v metodě `loadStyleFile()`. Funguje velmi podobně jako předchozí varianta načítání z editoru, pouze je zapotřebí vybrat soubor s definicemi maker ze správce souborů. Získaný obsah souboru se pak předá metodě `handleLoadStyle()`, jejíž popis je uveden výše.

7.4 Parser uživatelských stylů

Aby bylo možné realizovat načítání již vytvořených definic maker, bylo nutné vytvořit syntaktický analyzátor (parser), který kontroluje syntaktickou správnost načítaných definic, z nichž je poté možné vytvořit požadované objekty.

Zdrojový kód parseru se nachází v souboru `js/styles_parser.js`. Pro jeho implementaci byl použit programovací jazyk JavaScript. Jelikož se jedná o objektově-orientovaný jazyk, kód byl z hlediska uspořádání rozdělen do několika tříd. Definice třídy je zde však řešena jiným způsobem, než klasicky pomocí klíčového slova `class` jako třeba v jazyku Java nebo PHP. V JavaScriptu je vlastní třída definována funkcí, která se označuje jako konstruktor. Například `var Auto = function(){};` definuje novou třídu `Auto`. Vytvořit instanci této třídy pak lze po-

mocí klíčového slova `new`, například `var octavia = new Auto()`. Pro přidání metody je třeba využít vlastnosti `prototype`, která je dostupná ve všech objektech. Zápis `Auto.prototype.nastartuj = function(){};` definuje ve třídě `Auto` metodu `nastartuj()`.

7.4.1 Lexikální analyzátor

Nejprve je nutné implementovat lexikální analyzátor (lexer), který dokáže ze vstupního textu vytvořit posloupnost tokenů (lexikálních symbolů) a ty jsou poté předány na vstup parseru. Důležitým úkolem lexikálního analyzátoru je také odstranění bílých znaků (whitespaces), které jsou pro analýzu zbytečné.

Ve zdrojovém kódu je lexikální analyzátor reprezentován třídou `Lexer`. Ta obsahuje metodu `input()`, která zajišťuje načtení vstupního řetězce, v našem případě to jsou definice maker, a zároveň také odstraňuje komentáře, které jsou pro analýzu rovněž nepodstatné. Třída `Lexer` uchovává v poli `tokens` tokeny, které jsou vytvořeny v průběhu tokenizace. Tokeny jsou vytvářeny jako instance třídy `Token`. Ta má vlastnost `type` určující druh tokenu, `value` obsahující řetězec, který odpovídá danému pravidlu (lexém) a `position` specifikující pozici tokenu v řetězci.

Specifikace pravidel jednotlivých druhů tokenů je řešena pomocí regulárních výrazů. Ty definují, jakou posloupnost znaků mohou různé tokeny obsahovat. Díky tomu dokáže lexikální analyzátor rozpoznat přípustný řetězec a vytvořit příslušný druh tokenu. Množina definovaných druhů tokenů je uvedena v tabulce 6. Pravidla jsou uložena ve vlastnosti `rules` třídy `Lexer` jako objekty ve formátu `type: Lexer.EQUALS, pattern: '='`, kde `type` je druh tokenu a `pattern` je řetězec použitý v regulárním výrazu. Rozpoznávání tokenů podle těchto pravidel se vykonává postupně shora dolů v pořadí, jak jsou uvedena. Je tedy důležité uvádět nejprve konkrétní pravidla a poté až obecná. Mohlo by totiž dojít k nesprávnému určení tokenu, jelikož obecné pravidlo by mohlo pohltit řetězec, který měl být správně rozpoznán specifičtějším pravidlem. Proto obecná pravidla `IDENTIFIER` a `TEXT` jsou uvedena až jako poslední.

Je potřeba uvést, že množina těchto pravidel stačí pro rozpoznání takových druhů tokenů, které se vyskytují v definicích maker vygenerovaných pomocí návrháře stylů. Tyto druhy tokenů rozhodně nepokrývají celou množinu tokenů, jenž se mohou v typografickém systému TEX vyskytovat. Pro obsažení dalších druhů tokenů bychom museli tuto množinu dále rozšiřovat.

Hlavní metoda třídy `Lexer` zajišťující tokenizaci se nazývá `tokenize()`. Ta přidává do pole `tokens` tokeny, které jí vrací metoda `getToken()`. Metoda `getToken()` pracuje s vlastnostmi `buffer` a `pos`. Vlastnost `buffer` obsahuje načtený vstupní řetězec, vlastnost `pos` obsahuje index prvního dosud nezpracovaného znaku. Nejprve jsou odstraněny bílé znaky globálně v celém řetězci pomocí regulárního výrazu `RegExp('\\S', 'g');`. Následně se prochází cyklem pravidla pro určení druhu tokenu uložená v poli `rules`. Pro každé pravidlo je vykonán regulární výraz, který se snaží najít podřetězec vstupního řetězce, který tomuto výrazu odpovídá. V případě

Tabulka 6: Druhy tokenů použité při lexikální analýze

Druh tokenu	Regulární výraz
equals	`=`
def_param	`#[0-9]`
kw_begingroup	`\\begingroup`
kw_begin	`\\begin`
kw_def	`\\def`
kw_endgroup	`\\endgroup`
kw_end	`\\end`
kw_fontspec	`\\fontspec`
kw_shape	`\\(textsc|texttt|textit|textbf|textsl)`
kw_skip	`\\(leftskip|rightskip)`
kw_space	`\\vspace`
number	`-?\d+(\.\d+)?`
unit	`(pt|pc|in|bp|cm|mm|dd|cc|sp|ex|em)`
left_cbracket	`{`
right_cbracket	`}`
identifier	`\\[a-zA-Z]+`
text	`[a-zA-Z ]+`

výskytu je vytvořena nová instance třídy `Token`, která obsahuje informaci o druhu tokenu, hodnotu vyjádřenou nalezeným podřetězcem a pozici specifikovanou vlastností `pos`. Poté dojde k přenastavení indexu `pos` na hodnotu rovnající se součtu aktuální hodnoty a délky nalezeného podřetězce. Díky tomu se další token bude hledat až od tohoto místa, které ještě nebylo zpracováváno. Na závěr je nově vytvořený token vrácen metodě `tokenize()`. V případě že není nalezena shoda u žádného pravidla, dostáváme se do situace, kdy čtený řetězec obsahuje neočekávaný znak. V takové situaci je na úrovni implementace vyhozena výjimka a uživateli se zobrazí chybová hláška. Ta specifikuje tento nedefinovaný znak a jeho pozici v řetězci. Následně je celá tokenizace přerušena a ukončena.

7.4.2 Návrh gramatiky

Dříve než je možné implementovat samotný parser, je potřeba navrhnout příslušnou gramatiku. Formální gramatika totiž dokáže popsat syntaxi jazyka pomocí pravidel pro tvorbu vět. V našem případě bude gramatika popisovat syntaxi jazyka definic maker v systému TEX vygenerovaných návrhářem stylů. Na základě vytvořené gramatiky je pak možné sestrojit parser, který analyzuje větu a dokáže rozhodnout, zda patří do daného jazyka. Tím se ověří správnost syntaxe.

Pro ručně psaný parser se velmi často využívá gramatika LL(1), která se zpracovává pomocí metody rekurzivního sestupu. Jedná se o deterministickou metodu pracující shora dolů. Parser se tak rozhoduje na základě prvního dosud nezpracovaného tokenu, které další pravidlo má vybrat.

Jak udává definice zmíněná v teoretické části, gramatika je čtveřice obsahující *množinu neterminálních symbolů*, *množinu terminálních symbolů*, *množinu přepisovacích pravidel* a *startovací symbol*. Množina terminálních symbolů naší gramatiky je tvořena lexikálními symboly (tokeny), jejichž seznam je uveden výše. Při tvorbě pravidel gramatiky si lze vypomoct *syntaktickými diagramy*, které slouží k jejich grafickému vyjádření. Představují tak daleko přehlednější a intuitivnější formu zápisu než pravidla v čistě textovém tvaru. V těchto diagramech se vyskytují dva typy uzlů: Uzly ve tvaru obdélníku reprezentují neterminální symboly a uzly v oválném tvaru znázorňují terminální symboly. Jeden diagram definuje jeden neterminální symbol. Cesta spojující jednotlivé symboly znázorňuje jejich možnou posloupnost. Například lze uvést syntaktický diagram reprezentující neterminální symbol *def* (viz obrázek 11). Tento neterminál je složen z posloupnosti symbolů `kw_def` (klíčové slovo `def`), `identifier` (identifikátor), `def_param` (parametr) a `DEFCONTENT` ohraničený složenými závorkami.

Obrázek 11: Syntaktický diagram neterminálního symbolu DEF

Na základě těchto syntaktických diagramů lze sestrojit přepisovací pravidla pro naši gramatiku. Tento postup spočívá v postupném procházení cest zleva doprava po směru šipky tak, abychom vybrali všechny přípustné varianty posloupností symbolů. Jeden průchod reprezentuje jedno pravidlo, kde levá strana pravidla obsahuje symbol pojmenovaný podle názvu diagramu a pravá strana pravidla obsahuje posloupnost symbolů ve směru cesty. Vytvořená gramatika obsahuje tyto přepisovací pravidla:

$$
\begin{aligned}
STYLE &\rightarrow DEF\ STYLE \\
STYLE &\rightarrow \epsilon \\
DEF &\rightarrow kw\_def\ identifier\ def\_param\ \{ \\
&\qquad DEFCONTENT\ \} \\
DEFCONTENT &\rightarrow COMMAND\ DEFCONTENT \\
DEFCONTENT &\rightarrow def\_param\ DEFCONTENT \\
DEFCONTENT &\rightarrow \epsilon \\
COMMAND &\rightarrow identifier \\
COMMAND &\rightarrow kw\_shape\ SHAPEPARAM \\
COMMAND &\rightarrow kw\_space\ SPACEPARAM \\
COMMAND &\rightarrow kw\_font\ TEXTPARAM \\
COMMAND &\rightarrow kw\_skip\ SKIPPARAM \\
COMMAND &\rightarrow GROUP \\
COMMAND &\rightarrow ENVIRONMENT \\
SHAPEPARAM &\rightarrow \{BRACKETSCONTENT\ \} \\
SPACEPARAM &\rightarrow \{DIMENSION\ \} \\
TEXTPARAM &\rightarrow \{text\ \} \\
SKIPPARAM &\rightarrow = DIMENSION \\
GROUP &\rightarrow \{\ DEFCONTENT\ \} \\
GROUP &\rightarrow kw\_begingroup\ DEFCONTENT\ kw\_endgroup \\
ENVIRONMENT &\rightarrow kw\_begin\ TEXTPARAM\ DEFCONTENT \\
&\qquad kw\_end\ TEXTPARAM \\
BRACKETSCONTENT &\rightarrow COMMAND\ BRACKETSCONTENT \\
BRACKETSCONTENT &\rightarrow def\_param\ BRACKETSCONTENT \\
BRACKETSCONTENT &\rightarrow \epsilon \\
DIMENSION &\rightarrow number\ unit
\end{aligned}
$$

Aby gramatika mohla být LL(1), musí splňovat určitá kritéria. Transformace uvedené gramatiky z bezkontextové na LL(1) proběhla již během návrhu odstraněním levé rekurze a provedením levé faktorizace. Abychom však ověřili, že výsledná gramatika je skutečně LL(1), musíme spočítat množiny FIRST a FOLLOW. Na základě těchto množin je možné zkonstruovat rozkladovou tabulku, která definuje překladový automat. Ten už pak lze implementovat pomocí zvolené metody rekurzivního sestupu. Je třeba však dávat pozor, aby rozkladová tabulka neobsahovala kolize. To znamená, že v jedné buňce tabulky nesmí být více jak jeden záznam. V takové situaci by totiž nešlo deterministicky zjistit, jaká expanze se má provést. Gramatika by tak nebyla LL(1) a bylo by nutné ji upravit. K výpočtům množin

FIRST a FOLLOW a konstrukci rozkladové tabulky velmi dobře poslouží webová aplikace Desátomat[7]. Tyto výpočty je možné najít v příloze.

7.4.3 Syntaktický analyzátor

Úkolem syntaktického analyzátoru je určit, zda syntaxe vstupní věty odpovídá pravidlům definovaných gramatikou. Během této analýzy vzniká struktura nazývaná derivační strom. Jak již bylo uvedeno v předchozí části, syntaktický analyzátor je implementován metodou rekurzivního sestupu. Při této metodě se derivační strom konstruuje shora dolů a zleva doprava. Čtení vstupní věty probíhá také zleva doprava a vytváří se její levá derivace.

Implementace metody rekurzivního sestupu spočívá v rekurzivním zanořování funkcí, které představují neterminální symboly a jsou i stejně pojmenované. Uvnitř těchto funkcí se nachází všechna pravidla pro daný neterminál. Výběr pravidla závisí na prvním dosud nepřečteném tokenu. Při zpracování pravidla se postupuje po jednotlivých symbolech. Jedná-li se o terminál, pouze se přečte. Jde-li o neterminál, zavolá se stejnojmenná funkce a tím dojde k onomu zanoření. Po projití všech symbolů funkce skončí.

Syntaktický analyzátor je ve zdrojovém kódu reprezentován třídou `Parser`. Důležitou vlastností této třídy je `lexer`, která obsahuje instanci třídy `Lexer`. Tímto způsobem je lexikální analyzátor propojen s parserem. Hlavní metodu této třídy představuje `parse()`, která spouští celý proces syntaktické analýzy. Metoda `parse()` obdrží jako parametr vstupní řetězec, který následně předá jako vstup lexikálnímu analyzátoru. Poté se zavolá metoda `consume()`, která přiřadí vlastnosti `Parser.lookahead` následují nezpracovaný token z lexeru metodou `Lexer.nextToken()`. Následně se zavolá metoda `style()`, která reprezentuje startovací symbol. Metoda `style()` ovšem není volána přímo, ale pomocí metody `rule('style')`. Metoda `rule()` je vytvořena kvůli sjednocení kódu, protože při každém zavolání metody reprezentující neterminál se přidá uzel do derivačního stromu, který se v průběhu vytváří. Jeho kořen je uložen ve vlastnosti `root`. Všechny metody neterminálů se tedy volají přes `rule('')`, aby před samotným zavoláním metody `()` mohl být vykonán kód společný pro všechny tyto metody.

Pro příklad si můžeme podrobněji popsat fungování metody neterminálu `def()` (viz ukázka kódu 3). Nejprve se kontroluje pomocí metody `isLookahead(Lexer.KW_DEF)`, zda je první dosud nezpracovaný token typu `KW_DEF` (klíčové slovo `def`). V kladném případě se postupuje podle příslušného pravidla a zpracovávají se terminální symboly metodou `expect(Lexer.KW_DEF)`. Ta kontroluje typ tokenu, který se pak přidá do derivačního stromu. Metodou `consume()` se poté token zpracuje, čímž je nastaven další dosud nepřečtený token. V případě špatného typu tokenu při kontrole v metodě `expect()` je vyhozena výjimka, která uživateli zobrazí chybovou hlášku s popisem dané chyby. Metoda `def()` následně zpracuje další terminály `IDENTIFIER`, `DEF_PARAM` a `LEFT_CBRACKET`. Poté je zavo-

[7]Dostupné na http://desatomat.cz/

lána metoda pro neterminál `rule('defContent')`, čímž dojde k zanoření a začnou se zpracovávat její pravidla. Po návratu zpět do metody `def()` se ještě zpracuje terminál `RIGHT_CBRACKET`. Tím je tato metoda ukončena. Pokud by hned v úvodu při kontrole tokenu metodou `isLookahead` byl zjištěn špatný typ tokenu, vyvolá se výjimka a chybová hláška stejně jako v metodě `expect()`.

```
def: function () {
  // kontrola zda je token na vstupu KW_DEF (klicove slovo \def)
  if (this.isLookahead(Lexer.KW_DEF)) {
    this.expect(Lexer.KW_DEF); // overeni terminalu KW_DEF
    this.expect(Lexer.IDENTIFIER);
    this.expect(Lexer.DEF_PARAM);
    this.expect(Lexer.LEFT_CBRACKET);
    this.rule('defContent'); // fce neterminalu defContent
    this.expect(Lexer.RIGHT_CBRACKET);
  } else {
    this.error('\\def');
  }
}
```

Ukázka kódu 3: Implementace neterminálu def

Po úspěšném dokončení syntaktické analýzy máme vytvořený derivační strom, jehož kořen je uložen ve vlastnosti `root`. Nyní je potřeba tento strom projít, abychom získali informace o načtených definicích maker. Procházení je řešeno metodou `dfs()`, která spustí prohledávání stromu do hloubky. Následně je každý uzel prozkoumán metodou `inspectNode()`. Ta vytváří nové objekty stylů a zjišťuje jejich vlastnosti. Po projití všemi uzly jsou objekty stylů uloženy do pole `styles`, které je vráceno jako návratová hodnota metody `parse()` zpět do příslušné metody návrháře stylů.

8 Diskuze

Návrhář uživatelských stylů přináší do webové aplikace TEXonWeb novou funkcionalitu, kterou nedisponují ani vyspělé webové aplikace umožňující sazbu v typografickém systému TEX/LATEX jako Overleaf nebo ShareLATEX. V době psaní této práce jsem v uvedených aplikacích nenalezl nic, co by umožňovalo nebo alespoň usnadňovalo tvorbu vlastních stylů. Obdobná situace je také u desktopových editorů zaměřených na práci s TEXem, mezi které patří třeba TEXmaker nebo TEXstudio. Tyto editory nabízí velké množství průvodců a jiných nástrojů pro usnadnění sazby, nicméně nic z toho se netýká tvorby stylů. Návrhář stylů tak může poskytnout zajímavou konkurenční výhodu oproti těmto aplikacím.

Můžeme také porovnat návrháře stylů s řešeními uvedenými v kapitole Uživatelské styly. Podíváme-li se na oblast tvorby stylů v textovém procesoru MS Word, pak je jasné, že návrhář stylů takových možností nedosahuje. Přesto se alespoň částečně snaží jeho vlastnosti přenést do prostředí typografického systému TEX. Ve srovnání s podobnými webovými aplikacemi jako Google Dokumenty nebo MS Word Online však návrhář stylů poskytuje v této oblasti podstatně více možností, jelikož tyto aplikace obsahují podporu tvorby stylů jen v minimální míře.

Návrhář stylů cílí na začínající uživatele, hlavně na ty, kteří se v rámci předmětu Zpracování textů na počítači budou učit vytvářet vlastní styly. Pomocí grafického rozhraní si mohou nastavit vlastnosti stylu a poté hned vygenerovat příslušné definice, které jim umožní využívat vlastní strukturní značky při sazbě. Návrhářem vygenerované definice mohou posloužit jako názorná ukázka, jak takové styly vytvářet. Vytvořené styly lze pomocí návrháře rovněž upravovat a mazat. Umožněno je také načítání vygenerovaných definic. Pro tento účel byl vytvořen parser stylů, který kontroluje správnost syntaxe definic a dokáže upozornit na případné chyby. Po úspěšném načtení se dané styly objeví v přehledu návrháře a je s nimi možné dále pracovat. S pomocí návrháře stylů tak uživatelům odpadne složité vytváření vlastních definic ručním psaním kódu, jelikož syntaxe TEXu jim způsobovala časté problémy. Díky tomu tak třeba někteří uživatelé na typografický systém TEX nezanevřou kvůli uživatelské nepřívětivosti.

Sada vlastností stylů, které lze pomocí návrháře nastavit, se nemusí zdát příliš velká, nicméně je třeba počítat s tím, že vlastností v systému TEX je velké množství a v rámci zachování jednoduchosti nelze pojmout všechny. Vybrány proto byly pouze základní vlastnosti, které jsou jasně srozumitelné uživatelům a byly vyhodnoceny jako nejpoužívanější v rámci výuky.

Rozšiřování vlastností, které bude možné pomocí návrháře nastavit, je každopádně jednou z možností, jak návrháře stylů dále vylepšit. Mohly by jimi být vlastnosti, které se do finálního výběru nedostaly jako: barva písma, odstavcová zarážka, řádkování nebo křížové odkazy. Pro přidání nových vlastností by bylo potřeba náležitě upravit uživatelské rozhraní, aby obsahovalo příslušný formulářový prvek pro zadání hodnoty. Měla by však zůstat zachována jednoduchost rozhraní a intuitivnost. Například prvek pro výběr barvy písma by mohl obsahovat barevnou paletu,

ve které by šlo jednoduše zvolit požadovanou barvu. V případě potřeby by však zůstala zachována možnost vložit barvu pomocí kódu. Po přidání nových vlastností by bylo nutné taky zrevidovat parser stylů a připadně přidat nové konstrukce, pokud se odlišují od stávajících, aby bylo možné vytvořené styly znovu načíst a modifikovat.

Další možností, jak návrháře stylů rozšířit, by bylo přidání podpory pro napovídání vytvořených značek, které by se zobrazovaly uživateli při psaní kódu v editoru. Aplikace TEXonWeb využívá editor Ace, který funkci pro napovídání slov již ve výchozím stavu obsahuje. Je ale potřeba tuto funkci nazvanou completer povolit a poskytnout seznam slov, které bude nabízet. Editor Ace je napsaný stejně jako návrhář stylů v jazyku JavaScript a poskytuje potřebné API pro přidání nového completeru, pomocí něhož se dá definovat seznam nabízených slov. Completeru by tedy bylo třeba předat názvy návrhářem vytvořených značek. Taková funkce by aplikaci TEXonWeb přinesla významné zlepšení uživatelského komfortu a zrychlení psaní kódu.

Návrhář stylů by se dal také vylepšit o náhled, který by zobrazoval aktuální podobu vytvářeného stylu. Podobnou funkcí disponuje třeba MS Word. Při zadávání vlastností stylu, ať už se jedná o vytváření či editaci, by se do určené oblasti v rámci rozhraní návrháře generoval odpovídající PDF náhled. Náhled by obsahoval pouze předem definovaný výchozí text, který by byl naformátován podle zvolených vlastností. Po každé změně hodnoty některé vlastnosti by došlo k aktualizaci tohoto náhledu. Uživatelé by tak hned viděli, jak se daná vlastnost projevuje, což by jim usnadnilo vytváření stylů.

Styly vytvořené návrhářem jsou poměrně omezené a zdaleka nedosahují takových možností, kterých docílí zkušený uživatel ručním psaním vlastních definic s využitím všech prostředků nabízených TeXem. Do těla definice je totiž možné psát libovolné TEXové konstrukce, navíc lze využívat velké množství dostupných balíčků. Vytvořit tedy komplexní nástroj podporující všechny možnosti je prakticky nemožné, vždy bude muset obsahovat spoustu omezení a podporovat jen malou část funkcí. Myslím si tedy, že vytvořený návrhář stylů je užitečný nástroj, který uživatelům dokáže usnadnit a zpříjemnit práci v aplikaci TEXonWeb.

9 Závěr

Cílem práce bylo vytvořit návrháře uživatelských stylů v aplikaci TEXonWeb, který by umožňoval uživatelům vytvářet nové styly. Takto vytvořené styly by pak bylo možné ukládat do editoru či do souboru. Uložené styly by pak bylo možné zpětně načíst a znovu modifikovat. Dále bylo nutné analyzovat současný stav aplikace, probrat možnosti formátování textu a tvorby stylů v systému TEX/LATEX a jiných textových procesorech.

Práce obsahuje analýzu současného stavu aplikace TEXonWeb, která zahrnuje popis použitých technologií a postupů. Zmíněny jsou i některé komponenty aplikace a způsob jejich tvorby. Součástí práce je dále charakteristika typografického systému TEX/LATEX a jeho možnosti formátování textu. Dále jsou blíže představeny uživatelské styly, je popsán jejich význam a přínos. Uvedeny jsou i jednotlivé typografické systémy a textové procesory a prozkoumány jejich možnosti v oblasti tvorby stylů. Kvůli možnosti zpětné modifikace stylů bylo třeba probrat teorii formálních jazyků a automatů, aby bylo možné navrhnout příslušnou gramatiku a sestrojit parser stylů. Dále byly představeny parametry ovlivňující formátování textu, které by mohl návrhář obsahovat a vybrány ty nejdůležitější. Byl popsán detailní průběh návrhu uživatelského rozhraní. Probrán je způsob implementace návrháře stylů v programovacím jazyku JavaScript. Popis tvorby parseru je rozdělen na návrh gramatiky a implementaci lexikálního a syntaktického analyzátoru. V závěru je provedeno shrnutí celé práce, jsou popsány její přednosti a nedostatky. Nastíněny jsou také možnosti dalšího vývoje návrháře stylů.

Vytvořením návrháře uživatelských stylů, který obsahuje požadovanou funkcionalitu, byl splněn cíl práce. Návrhář stylů byl otestován na zkušebním serveru, což pomohlo odhalit spoustu chyb a nedostatků, které byly následně opraveny. Testování probíhalo ve spolupráci s Ing. Janem Přichystalem, Ph.D., se kterým byly veškeré úpravy konzultovány. Momentálně je už návrhář stylů nasazen a využíván v rámci výuky předmětu Zpracování textů na počítači. Během toho budou od uživatelů shromažďovány poznatky a připomínky, na základě kterých pak bude možné provádět další případná vylepšení. Veškeré zdrojové kódy návrháře stylů, včetně návrhu gramatiky, jsou obsaženy v příloze odevzdané v Univerzitním informačním systému.

10 Literatura

Aho, A V. a kol. *Compilers: Principles, Techniques, and Tools*. 2nd ed. Addison Wesley, 2006. ISBN 978-0321486813.

Bear, J H. *Using Style Sheets in Desktop Publishing* [online]. 2015 [cit. 2015-11-23]. Dostupné z: http://desktoppub.about.com/od/uselayoutsoftware/a/use_styles.htm.

Černá, I., Křetínský, M., Kučera, A. *Automaty a formální jazyky I*. Brno: Masarykova univerzita, 2002. 159 s..

Češka, M. *Gramatiky a jazyky* [online]. 1992 [cit. 2015-12-17]. Dostupné z: http://kifri.fri.uniza.sk/~bene/vyuka/kompilatory/pomocne-materialy/Gramatiky%20a%20jazyky-Ceska.3.pdf.

Eijkhout, V. *TEX by Topic* [online]. 2013 [cit. 2015-11-23]. Dostupné z: https://bitbucket.org/VictorEijkhout/tex-by-topic/raw/16b5340c1b9a19ac36165fa6f0d367608d50e995/TeXbyTopic.pdf.

Grune, D., Jacobs, C J H. *Parsing Techniques: A Practical Guide*. 2nd ed. Springer, 2007. 662 s. ISBN 978-0-387-20248-8.

Habiballa, H. *Překladače* [online]. 2005 [cit. 2015-12-17]. Dostupné z: http://www1.osu.cz/home/habibal/kurzy/xprek.pdf.

Habiballa, H. *Gramatiky a jazyky* [online]. 2013 [cit. 2015-12-19]. Dostupné z: http://www1.osu.cz/home/habibal/kurzy/GRAJA.pdf.

Henderson, A., Johnson, J. *Conceptual Models in a Nutshell* [online]. 2013 [cit. 2015-12-1]. Dostupné z: http://boxesandarrows.com/conceptual-models-in-a-nutshell/.

Chaffer, J., Swedberg, K. *Mistrovství v jQuery: Kompletní průvodce vývojáře*. 1. vyd. Brno: Computer Press, 2013. 384 s. ISBN 978-80-251-4103-8.

Kopka, H., Daly, P W. *LATEX: Podrobný průvodce*. 1. vyd. Brno: Computer Press, 2004. 576 s. ISBN 80-7226-973-9.

Lubbers, P., Albers, B., Salim, F. *HTML5: programujeme moderní webové aplikace*. 1. vyd. Brno: Computer Press, 2011. 304 s. ISBN 978-80-251-3539-6.

Mittelbach, F. a kol. *The LATEX Companion*. 2nd ed. Addison-Wesley Professional, 2004. ISBN 978-0201362992.

Olšák, P. *TEX pro pragmatiky* [online]. 2015 [cit. 2015-11-23]. Dostupné z: http://petr.olsak.net/ftp/olsak/tpp/tpp.pdf.

Potáček, P., Přichystal, J. *Refaktorizace aplikace TEXonWeb*. In PEFnet 2014. 1. vyd. Brno: Mendel University in Brno, 2014, s. 1–5. ISBN 978-80-7509-152-9.

Přichystal, J. *TEXonWeb Wiki* [online]. 2014 [cit. 2015-11-23]. Dostupné z: https://aurel.mendelu.cz/projects/texonweb/wiki.

Přichystal, J. *Možnosti tvorby dokumentu v TEXu pomocí webového prohlížece*. In Otvorený softvér vo vzdelávaní, výskume a v IT riešeniach. Zborník príspevkov medzinárodnej konferencie OSSConf 2014. 1. vyd. Bratislava: Spolocnost pre otvorené informacné technológie, 2014, s. 23–30. ISBN 978-80-970457-4-6.

Přichystal, J. *Návrhář stylů v aplikaci TEXonWeb*. In Otvorený softvér vo vzdelávaní, výskume a v IT riešeniach. Zborník príspevkov medzinárodnej konferencie OSSConf 2015. 1. vyd. Žilina: Žilinská univerzita v Žilině, 2015, s. 71–76. ISBN 978-80-970457-7-7.

Rybička, J. *LATEX pro začátečníky*. 3. vyd. Brno: Konvoj, 2003. ISBN 80–7302–049–1.

Saffer, D. *Designing for Interaction: Creating Innovative Applications and Devices*. 2nd ed. Berkeley, CA: New Riders, 2010. 223 s. ISBN 978-0-321-64339-1.

Satrapa, P. *LATEX pro pragmatiky* [online]. 2011 [cit. 2015-11-22]. Dostupné z: http://www.nti.tul.cz/~satrapa/docs/latex/latex-pro-pragmatiky.pdf.

Sipser, M. *Introduction to the Theory of Computation*. 3rd ed. Cengage Learning, 2012. 504 s. ISBN 978-1-133-18779-0.

Tidwell, J. *Designing interfaces*. 2nd ed. Sebastopol: O'Reilly, 2010. 547 s. ISBN 978-1-4493-7970-4.

Vavrečková, Š. *Programování překladačů*. Slezská univerzita Opava, 2008. 218 s. ISBN 978-80-7248-493-5.

Vavrečková, Š. *Teorie jazyků a automatů I* [online]. 2015 [cit. 2015-12-15]. Dostupné z: http://vavreckova.zam.slu.cz/obsahy/tja/skripta1/tja1.pdf.

Vybíhal, J., Přichystal, J. *Tvorba nového uživatelského rozhraní aplikace TEXonWeb*. In PEFnet 2014. 1. vyd. Brno: Mendel University in Brno, 2014, s. 1–10. ISBN 978-80-7509-152-9.

W3C. *Web Storage (Second Edition)* [online]. 2015 [cit. 2015-12-06]. Dostupné z: http://www.w3.org/TR/webstorage/.