Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Generování dokumentů z uživatelských šablon

Plzeň, 2022 Bc. Zdeněk Častorál

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2021/2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Zdeněk ČASTORÁL**
Osobní číslo:	**A19N0026P**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Téma práce:	**Generování dokumentů z uživatelských šablon**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s problematikou automatizované tvorby dokumentů dosazováním dat do uživatelských šablon.
2. Navrhněte řešení definice uživatelských šablon pro generování dokumentů. Můžete využít existující řešení nebo navrhnout vlastní implementaci.
3. Implementujte editor šablon a generátor dokumentů nad připravenou sadou dat.
4. Zdokumentujte způsob a pravidla tvorby šablon.
5. Řešení otestujte a zhodnoťte.

Rozsah diplomové práce: **doporuč. 50 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

dodá vedoucí diplomové práce

Vedoucí diplomové práce: **Ing. Petr Přibyl**
CCA Group, a.s.

Konzultant diplomové práce: **Ing. Martin Zíma, Ph.D.**
Katedra informatiky a výpočetní techniky

Datum zadání diplomové práce: **10. září 2021**
Termín odevzdání diplomové práce: **19. května 2022**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 11. října 2021

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 19. května 2022

Bc. Zdeněk Častorál

Poděkování

Tímto bych rád poděkoval svému vedoucímu diplomové práce Ing. Petru Přibylovi za odborné vedení, za pomoc a cenné rady při zpracování této práce. Dále bych chtěl poděkovat svému konzultantovi diplomové práce Ing. Martinu Zímovi, Ph.D. za cenné rady, věcné připomínky a vstřícnost při konzultacích této práce.

Abstract

This Master's thesis explores the issues surrounding the automated generation of documents by inserting data into user templates. The objective of this paper is to develop a new software product, or use and modify an existing software product made by a third party, which could be used to create and edit document templates and subsequently generate documents in the Microsoft Word format over a prepared data set. The work presented here has been commissioned by CCA Group a.s.

The thesis opens with an overview of the general issues that are encountered in document automation, primarily focusing on the automated generation of documents. This is followed by an analysis of the existing solution used by the company that has commissioned this work, including any shortcomings of that solution. The next part of the paper presents in great detail the specific requirements that need to be met as part of a new solution. The paper then explores the reasons why the particular variant of the new software product has been selected, which is then followed by a description of the proposal and implementation of the new product. Finally, the paper presents the course of tests performed when the software product was being implemented.

Abstrakt

Tato diplomová práce se zabývá problematikou automatického generování dokumentů dosazováním dat do uživatelských šablon. Cílem této práce je vytvořit nový softwarový produkt či použít a upravit již existující softwarový produkt třetí strany, který umožňuje vytvářet a editovat šablony dokumentů a následně generovat dokumenty nad připravenou sadou dat do formátu Microsoft Word. Zadavatelem této práce je společnost CCA Group a.s.

Práce nejprve shrnuje obecnou problematiku automatizace dokumentů, především se zaměřuje na automatické generování dokumentů. Dále je analyzováno stávající řešení zadavatele včetně jeho nedostatků a poté jsou detailně specifikovány požadavky na nové řešení. V další části práce je zdůvodněn výběr varianty vytvoření nového softwarového produktu, poté je popsán jeho návrh a implementace. Na závěr je uveden průběh testování implementovaného softwarového produktu.

Obsah

1 Úvod

Automatizace dokumentů a jejich správa se stává v dnešní době čím dál více potřebnou, pokud chce společnost uspět v konkurenčním prostředí. Díky automatizaci dokumentů lze zkrátit dobu tvorby dokumentů, zrychlit proces získávání dat z dokumentů, snížit počet chyb při tvorbě dokumentů a celkově zredukovat náklady zejména na lidské zdroje, které lze alokovat v jiných oblastech podnikového procesu.

Zadavatelem této diplomové práce je společnost CCA Group a.s., která se zabývá dodáváním řešení a služeb v oblasti informačních technologií. Konkrétně se jedná například o konzultace návrhů a architektur, zakázkový vývoj software, provoz, servis a vývoj aplikací. Společnost úzce spolupracuje s Ministerstvem spravedlnosti, pro které realizuje řadu projektů. Jeden z nich se zabývá správou a řízením dokumentů. Právě z tohoto důvodu vznikl požadavek na produkt, který bude umožňovat automatizovanou tvorbu dokumentů prostřednictvím dosazování dat do uživatelských šablon. Společnost CCA Group a.s. takový produkt řadu let vyvíjí a provozuje. V současné době uvažuje o jeho dalším rozvoji a portaci do nových technologií, nebo o kompletně novém softwarovém řešení.

Cílem této práce je vytvořit nový softwarový produkt či použít a upravit již existující softwarový produkt třetí strany, který umožňuje vytvářet a editovat šablony dokumentů a následně generovat dokumenty nad připravenou sadou dat do formátu Microsoft Word (`doc` či `docx`). Příprava datových sad není předmětem této práce.

Přínosy tohoto řešení jsou: urychlení vytváření dokumentů se stejnou strukturou (šablonou), eliminace chyb způsobených lidskou chybou, snížení nákladů na lidské zdroje. *Primární cílovou skupinu* tvoří zákazníci firmy CCA Group a.s., kteří v rámci své práce generují ze svých aplikací dokumenty z uživatelských šablon. Řešení předpokládá základní znalost kancelářského balíku Microsoft Office uživatelem.

V této práci nejprve popisuji obecnou problematiku automatizace dokumentů. V další části popisuji stávající produkt řešící danou problematiku, věnuji se rozboru specifikace požadavků zadavatele na produkt nový a analyzuji možné varianty řešení včetně možnosti použití již existujících produktů. Poté navrhuji definici šablon dokumentů a popisuji implementaci výsledného řešení, které odpovídá požadavkům zadavatele. Na závěr se zabývám testováním výsledného produktu a celkovým zhodnocením práce.

2 Problematika automatizace dokumentů

Automatizace dokumentů zahrnuje širokou oblast problematiky a řešení. Zabývá se zejména těmito *dvěma* základními hledisky:

- **automatické zpracování dokumentů** (Document Process Automation) – je proces, při kterém se automaticky získávají požadovaná data z rozdílných typů dokumentů, kontroluje se jejich validita a následně se využívají v dalších procesech,
- **automatické generování dokumentů** (Automated Document Generation) – je proces, při kterém se vytvářejí nové dokumenty (např. faktury, smlouvy) prostřednictvím předem získaných dat [7].

V následujícím textu se zaměřím pouze na část *automatického generování dokumentů*.

2.1 Automatické generování dokumentů

Jak jsem již zmínil, *automatické generování dokumentů* je proces vytváření nových dokumentů prostřednictvím dat. V literatuře se můžeme setkat s řadou definic automatického generování dokumentů. Pro účely této práce zde uvádím dvě, dle mého názoru, vhodné definice:

1. *Automatické generování dokumentů* je souhrn systémů a postupů, které napomáhají vytváření elektronických dokumentů. Tyto systémy používají již existující texty a data k sestavení nového dokumentu [7].
2. *Automatické generování dokumentů* je proces rychlého a efektivního generování elektronických dokumentů za použití softwarových řešení. Generování těchto dokumentů je založeno na získání dat a textu z různorodých zdrojů a jejich následném zakomponování do nových dokumentů prostřednictvím předdefinovaných šablon [8].

Podniky používají pro automatické generování dokumentů specifické softwarové nástroje, které zajišťují získávání požadovaných dat, a následně tato data transformují do požadovaných vzorů dokumentů [7]. Posloupnost kroků těchto softwarových nástrojů je následovná:

1. **získání vstupních dat** – nejprve je nutné specifikovat a získat data pro generované dokumenty, lze realizovat manuálně či automaticky prostřednictvím softwarových nástrojů, např. ze systémů ERP (*Enterprise Resource Planning*, podnikové plánování zdrojů) a CRM (*Customer Relationship Management*, řízení vztahů se zákazníky),

2. **výběr/definice šablony dokumentu** – po získání vstupních dat je nutné vybrat či nově vytvořit specifické šablony dokumentů pro každý typ dokumentu,

3. **transformace dat do šablony dokumentu** – následuje příprava vstupních dat do formátu specifické šablony dokumentu. Tento proces může obsahovat transformaci dat ze vstupního formátu do formátu šablony. V této fázi jsou data připravena na vkládání na předem definovaná místa v šabloně (viz obrázek 2.1 – vyznačeno červenými rámečky). V rámci této fáze se kontroluje také konzistence šablony (např. validita použitých značek pro generování),

4. **kontrola konzistence dokumentu** – v dalším kroku automatický softwarový nástroj kontroluje možný výskyt nekonzistentností dokumentu. Jedná se například o případy, kdy řetězec vstupních dat je delší než maximální povolená délka řetězce pro danou značku,

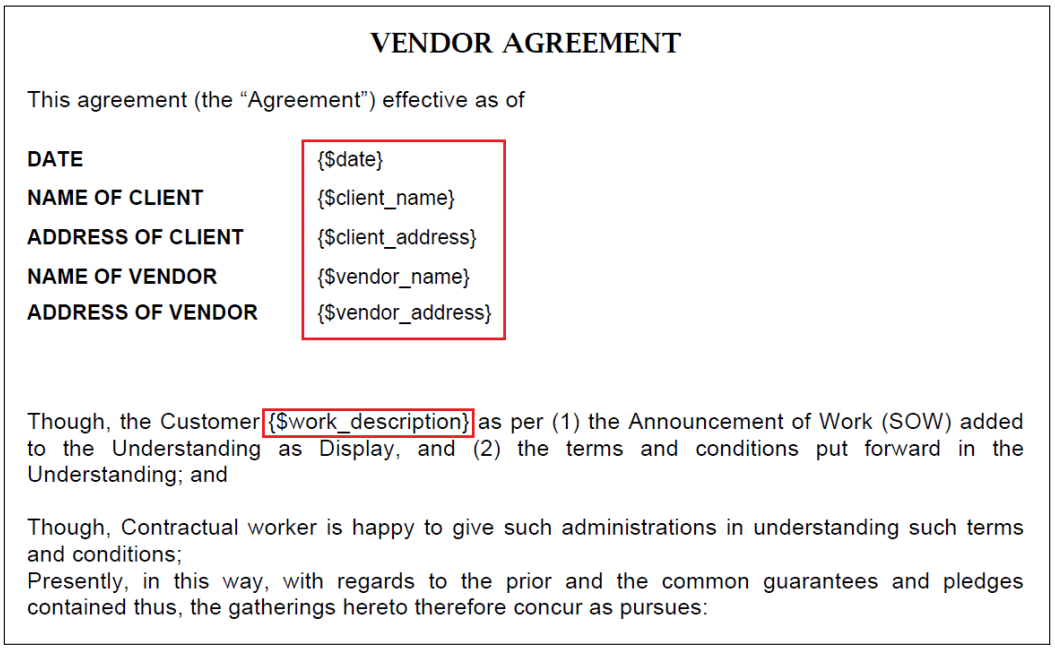
5. **vygenerování dokumentu** – poslední fáze, při které dochází k samotnému vygenerování nového dokumentu [7].

Na obrázku 2.2 je znázorněn možný průběh vygenerování nového dokumentu za předpokladu již předem získaných vstupních dat a navržené šablony dokumentu.
Mezi hlavní výhody automatického generování dokumentů se řadí zejména:

- **zrychlení tvorby dokumentů** – s použitím automatického softwarového řešení lze eliminovat pracné a zdlouhavé procesy při vytváření rutinních dokumentů se stejnou strukturou [8],

- **snížení nákladů** – další výhodou je snížení nákladů, zejména na lidské zdroje a jejich možná alokace na jiné potřebné aktivity,

- **snížení počtu chyb** – manuální vytváření dokumentů je náchylné na chyby způsobené lidským faktorem. Software pro automatické generování dokumentů nekonzistentní informace odhalí a informuje uživatele,



VENDOR AGREEMENT

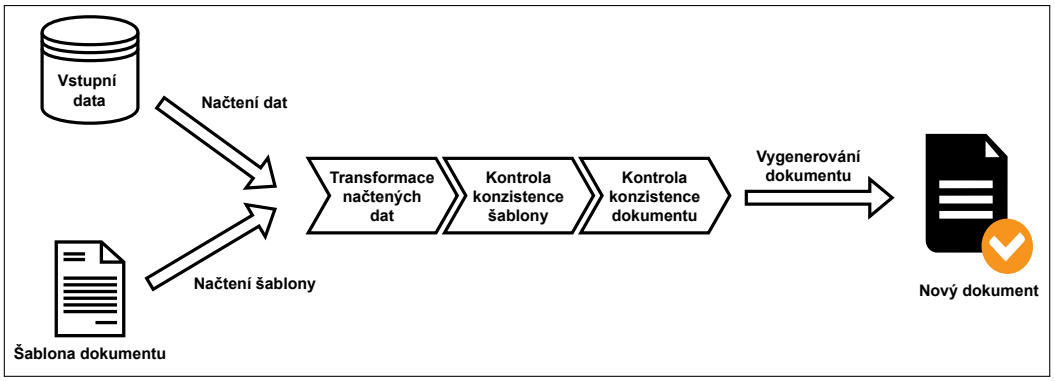
This agreement (the "Agreement") effective as of

DATE	{$date}
NAME OF CLIENT	{$client_name}
ADDRESS OF CLIENT	{$client_address}
NAME OF VENDOR	{$vendor_name}
ADDRESS OF VENDOR	{$vendor_address}

Though, the Customer {$work_description} as per (1) the Announcement of Work (SOW) added to the Understanding as Display, and (2) the terms and conditions put forward in the Understanding; and

Though, Contractual worker is happy to give such administrations in understanding such terms and conditions;
Presently, in this way, with regards to the prior and the common guarantees and pledges contained thus, the gatherings hereto therefore concur as pursues:

Obrázek 2.1: Šablona dokumentu; Zdroj: [20] (upraveno)



Obrázek 2.2: Průběh vygenerování dokumentu z uživatelské šablony; Zdroj: vlastní zpracování

- **standardizované šablony** – při manuálním vytváření dokumentů může docházet k vytváření různorodých stylů vzhledu šablon uživatelem, ale prostřednictvím softwarového řešení je vzhled jednotlivých typů dokumentů definován šablonou dokumentu a je tedy stejnorodý [7].

3 Výchozí stav softwarového řešení zadavatele

Jednou z hlavních činností společnosti CCA Group a.s. je úzká spolupráce s Ministerstvem spravedlnosti a dalšími organizacemi, pro které zajišťuje správu a řízení dokumentů. Z tohoto důvodu zde vznikla potřeba softwarového produktu, který zajišťuje automatizovanou tvorbu dokumentů prostřednictvím dosazování dat do uživatelských šablon. Takovým produktem již společnost CCA Group a.s. disponuje.

V této kapitole popisuji stávající softwarové řešení pro automatické generování dokumentů a jeho nedostatky.

3.1 Popis softwarového produktu

Softwarové řešení je vyvíjeno a spravováno přímo společností CCA Group a.s. Interní název produktu je *Modul Dokumenty*. Tento produkt umožňuje vytvoření a editaci šablon dokumentu v prostředí Microsoft Word a následné generování nového dokumentu s dosazenými vstupními daty. Daná aplikace se skládá z technologií:

- *Oracle Forms* – obsahuje funkce pro definici vzorů,
- *Visual Basic for Applications* (VBA) – obsahuje makra v rámci aplikace Microsoft Word pro dosazení vstupních dat a generování dokumentů.

Tento produkt vyžaduje připojení k SQL (*Structured Query Language*) databázovému serveru.

3.1.1 Postup použití aplikace

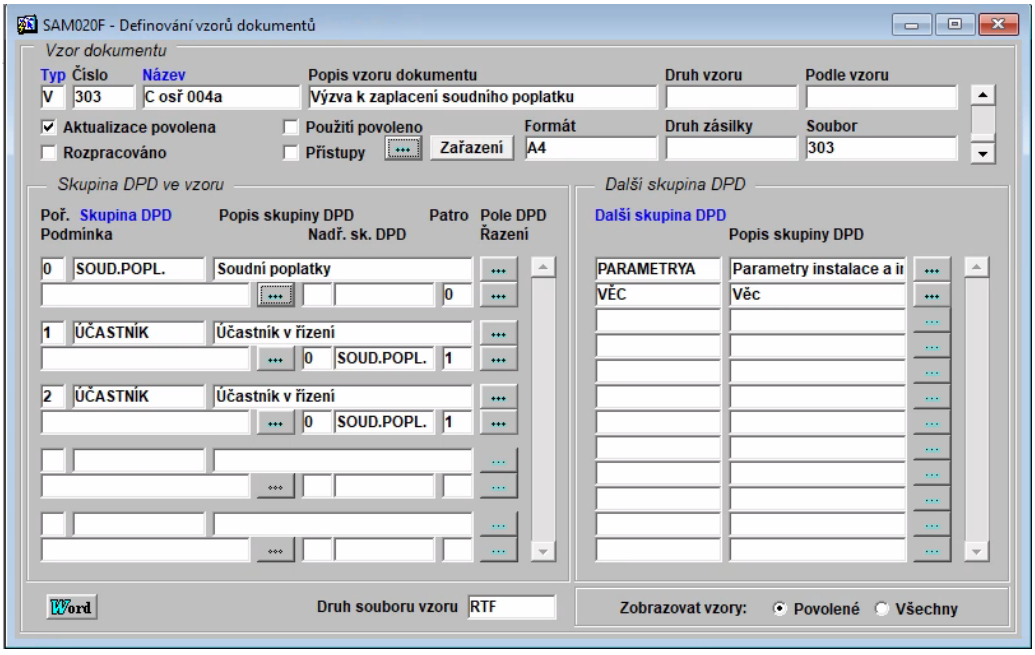
Postup vytvoření šablony dokumentu, dosazení vstupních dat a vygenerování nového dokumentu je následovný:

1. uživatel spustí aplikaci a založí v ní šablonu dokumentu, případně vybere již existující šablonu (v případě tohoto produktu je šablonou dokumentu tzv. *vzor*),
2. v rámci založení vzoru uživatel připojí tzv. *datová pole dokumentu* (DPD) – jedná se o obraz databázových tabulek a pohledů,

3. uživatel spustí aplikaci Microsoft Word prostřednictvím tlačítka `Word` (založení vzoru je ilustrováno na obrázku 3.1),

4. uživatel definuje obsah vzoru v prostředí Microsoft Word – ten prostřednictvím prostředků hromadné korespondence doplňuje tzv. *slučovací pole* (viz obrázek 3.2 - vyznačena červenými rámečky), do kterých se při generování nového dokumentu automaticky doplňují vstupní data,

5. uživatel spustí generování nového dokumentu – aplikace sestaví z definovaného vzoru dotaz do SQL databáze vstupních dat, která následně Microsoft Word doplní na místo slučovacích polí.

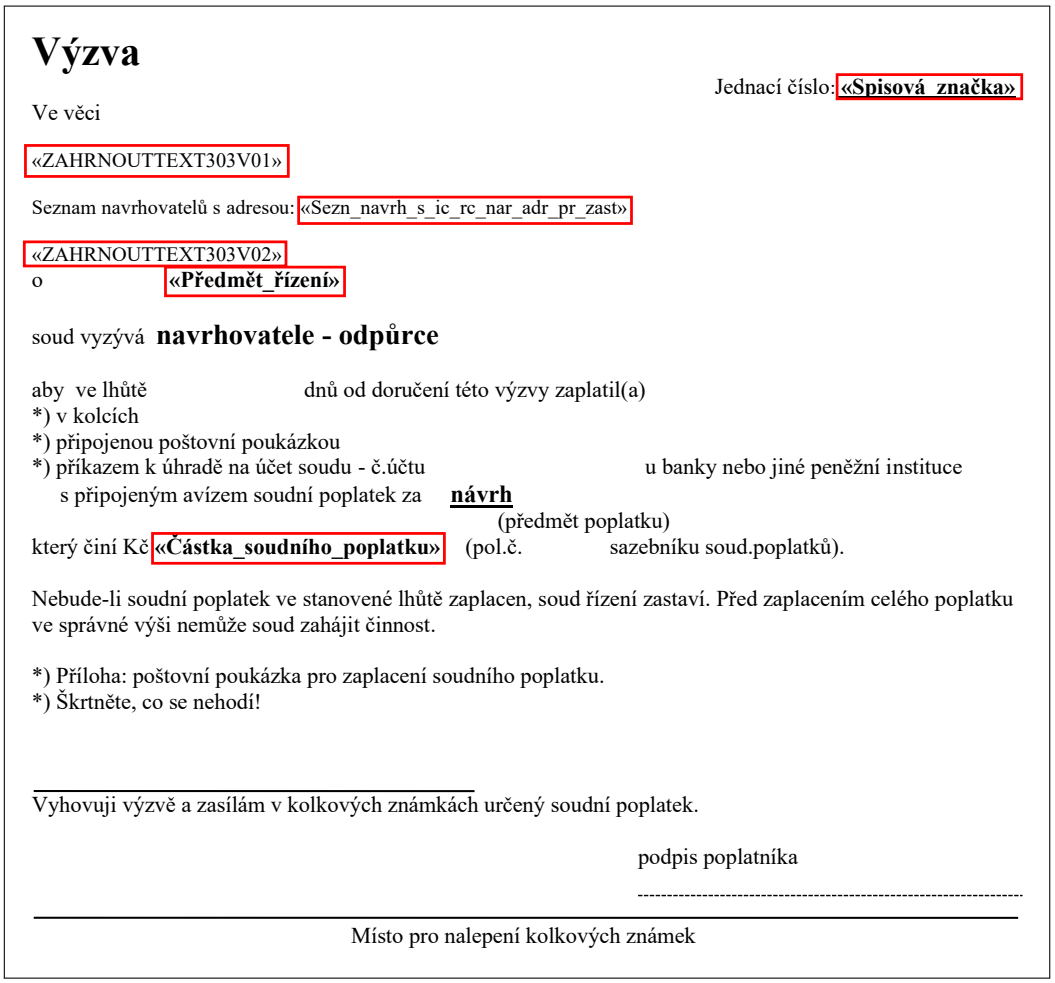


Obrázek 3.1: Definování vzorů dokumentů; Zdroj: CCA Group a.s.

3.2 Nedostatky stávajícího řešení

Nedostatky stávajícího softwarového produktu jsou zejména:

- implementace produktu v zastaralých technologiích – *Oracle Forms* a *VBA*,
- obtížně udržovatelný a rozšiřitelný programový kód,
- striktní vázanost produktu na *Oracle Forms*,



Výzva

Jednací číslo: **«Spisová_značka»**

Ve věci

«ZAHRNOUTTEXT303V01»

Seznam navrhovatelů s adresou: «Sezn_navrh_s_ic_rc_nar_adr_pr_zast»

«ZAHRNOUTTEXT303V02»
o **«Předmět_řízení»**

soud vyzývá **navrhovatele - odpůrce**

aby ve lhůtě dnů od doručení této výzvy zaplatil(a)
*) v kolcích
*) připojenou poštovní poukázkou
*) příkazem k úhradě na účet soudu - č.účtu u banky nebo jiné peněžní instituce
s připojeným avízem soudní poplatek za **návrh**
(předmět poplatku)
který činí Kč **«Částka_soudního_poplatku»** (pol.č. sazebníku soud.poplatků).

Nebude-li soudní poplatek ve stanovené lhůtě zaplacen, soud řízení zastaví. Před zaplacením celého poplatku ve správné výši nemůže soud zahájit činnost.

*) Příloha: poštovní poukázka pro zaplacení soudního poplatku.
*) Škrtněte, co se nehodí!

Vyhovuji výzvě a zasílám v kolkových známkách určený soudní poplatek.

podpis poplatníka

Místo pro nalepení kolkových známek

Obrázek 3.2: Vzor dokumentu se slučovacími poli (vyznačena červenými rámečky); Zdroj: CCA Group a.s.

- silná závislost na Microsoft Word, zejména z důvodu doplňování slučovacích polí,
- nemožnost načítání jiných formátů vstupních dat než z SQL serveru,
- nemožnost zpracování hierarchicky (víceúrovňově) strukturovaných dat – je nutné v tomto případě vnořovat vzory do sebe.

4 Specifikace požadavků zadavatele na nové řešení

V této kapitole definuji požadavky zadavatele na realizaci projektu, který je součástí této diplomové práce. Zadavatelem je společnost CCA Group a.s.

Cílem projektu je vytvořit nový softwarový produkt či použít a upravit již existující softwarový produkt třetí strany, který řeší automatické generování dokumentů z šablon definovaných uživatelem. Produkt musí umožňovat vytváření a editaci šablon dokumentů a následné generování dokumentů nad připravenou sadou dat do formátu Microsoft Word (doc či docx). Příprava datových sad není předmětem této práce.

4.1 Základní shrnutí

4.1.1 Současný stav

Zadavatel disponuje softwarovým produktem, který danou problematiku řeší (viz kapitola 3). Toto řešení má ovšem svá omezení a nedostatky. Vývoj nového softwarového produktu proto probíhá „na zelené louce“, případně lze využít a upravit již existující řešení třetí strany.

4.1.2 Cílová skupina

Cílovou skupinu pro toto softwarové řešení tvoří zejména zákazníci firmy CCA Group a.s., kteří v rámci své práce generují ze svých aplikací dokumenty z uživatelských šablon. Řešení předpokládá základní znalost kancelářského balíku Microsoft Office uživatelem.

4.1.3 Rozsah projektu

Tento projekt se zabývá implementací nového softwarového produktu či použitím a upravením již existujícího softwarového produktu třetí strany, který má za cíl řešit automatické generování dokumentů z šablon definovaných uživatelem nad připravenou sadou vstupních dat.

Výsledný softwarový produkt musí umožňovat:

- doplňování dat na určená místa v šabloně dokumentu,

- definici direktiv, které ovlivňují způsob generování dokumentu z vytvořené šablony,
- definici metod, které modifikují doplňovaná data, nebo doplňují jiná data.

Celé toto řešení musí být připraveno na rozšíření o další direktivy či metody. Vstupní data budou poskytována ve formátu *JavaScript Object Notation* (JSON). Jejich struktura není zadavatelem stanovena. Definovaná struktura vstupních dat musí být dostatečně obecná, aby bylo možné jejím prostřednictvím popsat různorodě strukturovaná data. Konkrétní struktura pro jednu sadu vstupních dat bude popsána *metadatovým souborem* těchto vstupních dat taktéž ve formátu JSON.

Výstupem projektu bude softwarový produkt s následujícími funkcionalitami:

- **načtení metadat:**
  - editor šablon bude umožňovat načtení metadat vstupních dat,
  - tato metadata budou obsahovat strukturu vstupních dat,
  - metadata budou zobrazována ve stromové struktuře uživateli,
- **elementární dosazování dat:**
  - v šabloně dokumentu bude umožněno definovat specifická pole zastupující hodnoty vstupních dat,
  - tato pole budou při generování nového dokumentu nahrazena odpovídajícími hodnotami vstupních dat,
- **opakovací blok:**
  - v definované šabloně dokumentu bude možné vytvořit konstrukci pro opakovací blok,
  - při generování nového dokumentu budou na místo konstrukce iterativně dosazeny všechny hodnoty atributu typu `kolekce` ze vstupních dat,
- **opakovací blok s tabulkou:**
  - bude možné definovat konstrukci pro opakovací blok s tabulkou v šabloně dokumentu,

  - v průběhu generování nového dokumentu bude na místě této konstrukce vytvořena tabulka, kde na každý řádek této tabulky budou dosazeny hodnoty atributu typu `kolekce` ze vstupních dat,

- **vložení podšablony:**
  - do šablony dokumentu bude možné vložit konstrukci pro vložení podšablony dokumentu definované v externím souboru,
  - generátor při tvorbě nového dokumentu pro každou takovou konstrukci najde příslušnou podšablonu dokumentu, doplní ji o příslušné hodnoty vstupních dat a vloží její obsah do šablony dokumentu,

- **definice metod:**
  - šablona dokumentu bude umožňovat definovat metody pro *vložení aktuálního data* a *formátování dat*,
  - při generování dokumentu bude dosazeno aktuální datum v případě použití metody pro *vložení aktuálního data*,
  - v případě metody pro *formátování data* dojde v rámci generování nového dokumentu ke zformátování data prostřednictvím zadaného formátu.

4.1.4 Přínosy nového řešení

Přínosy nového řešení oproti stávajícímu:

- softwarový produkt bude implementován v moderním používaném prostředí,
- řešení bude snadno rozšiřitelné a udržovatelné,
- grafické uživatelské rozhraní bude přívětivé a snadno ovladatelné.

4.1.5 Kontext systému

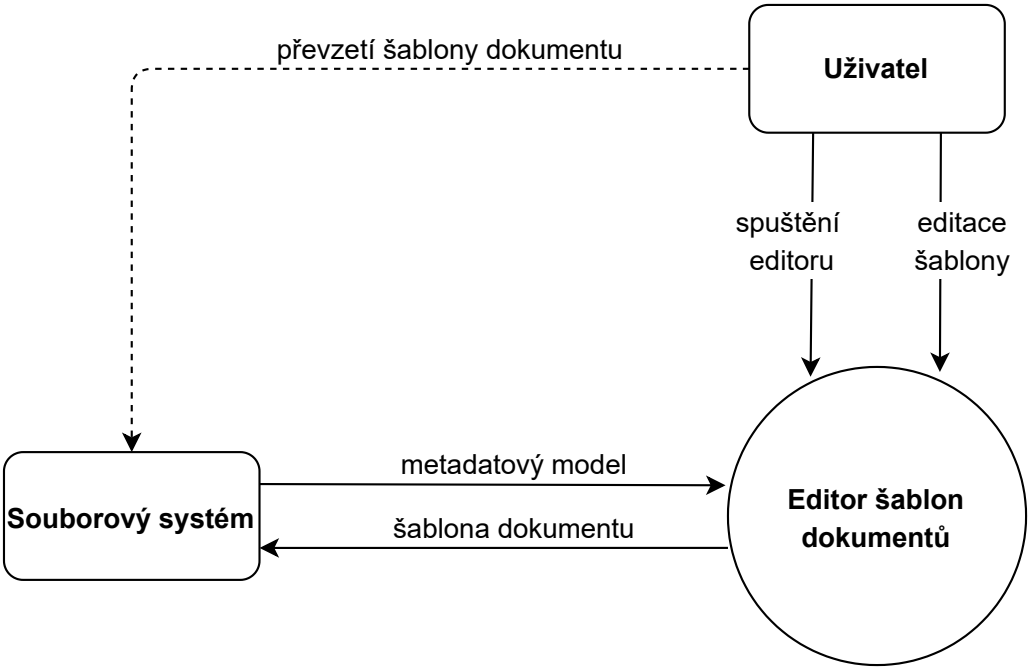
Softwarové řešení zahrnuje dvě hlavní části:

- *editor šablon dokumentů* – vytvoření a editace šablony dokumentu,
- *generátor dokumentů z uživatelských šablon* – samotný generátor nového dokumentu z předem definované šablony a vstupních dat.

Kontext řešení pro každou z těchto dvou částí je popsán níže.

Editor šablon dokumentů

Kontext řešení pro část *editoru šablon dokumentů* znázorňuje obrázek 4.1.



Obrázek 4.1: Kontextový diagram – editor šablon dokumentů; Zdroj: vlastní zpracování

Externí entity *editoru šablon dokumentů* tedy jsou:

- Souborový systém,
- Uživatel.

Souborový systém v tomto kontextu obsahuje metadata vstupních dat pro budoucí generování nového dokumentu a dále slouží jako úložiště pro vytvořenou šablonu dokumentu. Entitou *Uživatel* v tomto kontextu je běžný uživatel používající editor šablon s cílem vytvořit šablonu dokumentu, která bude sloužit pro budoucí generování dokumentů. Uživatel v této roli spouští editor šablon a prostřednictvím jeho nástrojů vytváří a edituje šablonu dokumentu.

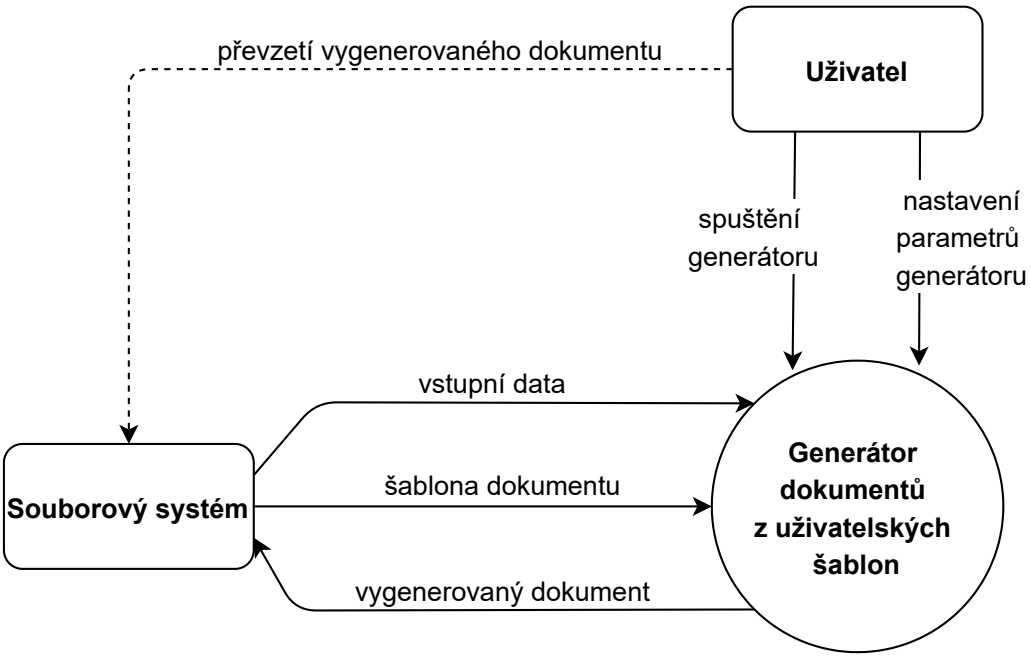
Vytvoření a editace šablony probíhá následovně:

- uživatel spustí editor šablon dokumentů,
- uživatel vytvoří novou šablonu nebo načte již existující šablonu dokumentu ze souborového systému prostřednictvím nástrojů editoru šablon dokumentů,

- uživatel načte soubor s metadaty vstupních dat prostřednictvím nástrojů editoru šablon dokumentů,
- editor šablon dokumentů zpřístupní uživateli nástroje pro editaci šablony,
- uživatel edituje šablonu dokumentu,
- po skončení editace uživatel vyvolá akci pro uložení šablony dokumentu,
- editor šablon dokumentů uloží šablonu na souborový systém,
- uživatel může převzít šablonu dokumentu ze souborového systému.

Generátor dokumentů z uživatelských šablon

Kontext řešení pro část *generátoru dokumentů z uživatelských šablon* znázorňuje obrázek 4.2.



Obrázek 4.2: Kontextový diagram – generátor dokumentů z uživatelských šablon; Zdroj: vlastní zpracování

Externí entity *generátoru dokumentů z uživatelských šablon* tedy jsou:

- Souborový systém,
- Uživatel.

Souborový systém v tomto kontextu obsahuje nadefinovanou šablonu dokumentu a vstupní data pro generování nového dokumentu. Také slouží jako úložiště pro vygenerovaný dokument. Entitou *Uživatel* je v tomto kontextu běžný uživatel, který používá generátor dokumentů s cílem vytvořit nový dokument prostřednictvím nadefinované šablony dokumentu a vstupních dat. Uživatel v této roli nastavuje parametry pro spuštění generátoru dokumentů a následně generátor spouští. Tento proces lze automatizovat, pro nastavení a spouštění generátoru může být použito jiné softwarové řešení.

Generování dokumentu z uživatelské šablony probíhá následovně:

- uživatel nastaví vstupní parametry spuštění generátoru dokumentů, kterými jsou:
  - definovaná šablona dokumentu,
  - vstupní data,
  - cesta k výstupnímu souboru (vygenerovanému dokumentu) na souborovém systému,
- uživatel spustí generátor dokumentů s definovanými parametry,
- generátor na základě šablony dokumentu a vstupních dat vytvoří nový dokument a uloží jej na souborový systém,
- uživatel může převzít vygenerovaný dokument ze souborového systému.

4.1.6 Provozní prostředí

Softwarový produkt je určen výhradně pro společnost CCA Group a.s. Tento produkt vyžaduje *64-bitové prostřední Windows* a nainstalovaný produkt *Microsoft Word 2019*. Softwarový produkt bude kompatibilní také s produktem *Microsoft Word 365*, ale vývoj softwarového produktu bude cílen zejména na *Microsoft Word 2019*. Níže jsou blíže specifikovány minimální softwarové a hardwarové požadavky pro bezchybnou funkčnost tohoto produktu.

Minimální požadavky

- operační systém: Windows 10 Home (64-bit.) nebo vyšší,
- procesor (CPU): architektura x86-64 (64-bitový), frekvence 2 GHz nebo vyšší,

- volné místo na pevném disku: 1 GB nebo více,
- operační paměť (RAM): 4 GB nebo více,
- ostatní software: Microsoft Word 2019 (nebo Microsoft Word 365), .NET Framework Runtime 4.7.2 (pokud nebude použito kompletně již existující řešení).

4.1.7 Omezení návrhu a implementace

Zákazníkem bylo rozhodnuto, že části vlastní implementace softwarového řešení musí být provedeny v prostředí `.NET Framework 4.7.2` v jazyce `C#` (pokud nebude použito řešení třetí strany). Implementací se detailně zabývá kapitola 7. Generovaný dokument musí být ve formátu `Microsoft Word` (`doc` nebo `docx`). Část softwarového produktu, která plní funkci generování dokumentů, musí být spustitelná samostatně prostřednictvím jiné aplikace.

4.1.8 Uživatelská dokumentace

K softwarovému produktu bude dodána uživatelská dokumentace, která bude popisovat uživatelské ovládání aplikace. Dále bude popisovat vytváření a editaci uživatelských šablon.

4.2 Funkce softwarového produktu

V následující kapitole jsou popsány funkce softwarového produktu definované zadavatelem projektu.

Nejprve je nutné definovat datové typy, které budou vstupní data obsahovat a se kterými bude tento softwarový produkt pracovat. Jedná se o 3 základní datové typy:

- `string` – jedná se o řetězec znaků,
- `integer` – jedná se o celé číslo,
- `datetime` – datový typ pro datum a čas.

Dále se jedná o datový typ `kolekce`, který může obsahovat libovolný počet výše zmíněných základních datových typů.

4.2.1 Načtení metadat

Popis

Část softwarového produktu zabývající se editací šablon dokumentů bude umožňovat načtení metadat vstupních dat ve formátu JSON. Tato metadata budou obsahovat strukturu uložení vstupních dat včetně datových typů, hloubky zanoření jednotlivých elementů atd. Struktura metadat, stejně jako struktura vstupních dat, není zadavatelem stanovena.

Scénář užití

Prostřednictvím grafického uživatelského rozhraní (detailněji v kapitole 4.3.1) v rámci *editace šablon dokumentů* uživatel vyvolá funkci pro načtení metadat, která následně příslušná metadata načte a zobrazí uživateli datovou strukturu.

Funkční požadavky

1. Funkce zobrazí dialogové okno, do kterého uživatel vybere příslušný soubor obsahující metadata ve formátu JSON.
2. Funkce načte po vybrání příslušného souboru metadata vstupních dat.
3. Funkce zobrazí uživateli strukturu vstupních dat, včetně datových typů, ve stromové struktuře.
4. Funkce umožní uživateli výběr dostupných atributů a vložení specifických polí do šablony dokumentu (viz dále).

4.2.2 Elementární dosazování dat

Popis

Softwarový produkt bude umožňovat definování specifických polí v šabloně dokumentu, do kterých budou v procesu generování nového dokumentu podle dané šablony vkládána data ze zdroje vstupních dat. Tato pole bude možné vkládat také prostřednictvím editoru šablon dokumentů po načtení metadatového souboru.

Scénář užití

Uživatel vloží do šablony dokumentu specifické pole pro vložení hodnoty vstupních dat. Pokud uživatel zadává pole ručně, předpokládá se jeho znalost struktury dat (či metadat). Pole lze vložit prostřednictvím grafického

uživatelského rozhraní editoru šablon dokumentů. Při vytváření nového dokumentu generátor dosadí pro každé specifické pole odpovídající hodnoty vstupních dat. Generátor použije pro formátování cílového textu stejný formát, který byl použit pro specifické pole.

Funkční požadavky

1. Funkce předpokládá načtení a zobrazení stromové struktury vstupních dat (na základě metadat) – viz kapitola 4.2.1.
2. Funkce vloží specifické pole do šablony dokumentu na základě volby uživatele.
3. Funkce v rámci generování nového dokumentu dosadí na pozice specifických polí odpovídající data.
4. Funkce použije pro formátování cílového textu stejný formát, který byl použit pro specifické pole.

Body 1. a 2. lze přeskočit, pokud uživatel má znalost vstupních dat a zadá specifické pole manuálně.

4.2.3 Opakovací blok

Popis

Softwarový produkt bude umožňovat definovat v šabloně dokumentu prostřednictvím specifické konstrukce opakovací blok, který umožní iterativně dosadit všechny hodnoty atributu typu `kolekce`. Opět bude možné vložit předdefinovaný opakovací blok prostřednictvím grafického rozhraní editoru šablon.

Scénář užití

Uživatel vloží do šablony dokumentu specifickou konstrukci opakovacího bloku. Při manuálním zadávání této konstrukce uživatelem se opět předpokládá jeho znalost struktury dat (či metadat). Konstrukci lze vložit prostřednictvím grafického uživatelského rozhraní editoru šablon dokumentů. Generátor dokumentů dosadí pro každou takovou konstrukci odpovídající hodnoty kolekce ze vstupních dat. Generátor použije pro formátování cílového textu stejný formát, který byl použit pro specifickou konstrukci.

Funkční požadavky

1. Předpokladem této funkce je načtení a zobrazení stromové struktury vstupních dat (na základě metadat) – viz kapitola 4.2.1.
2. Funkce vloží specifickou konstrukci opakovacího bloku do šablony dokumentu na základě volby uživatele.
3. Při generování nového dokumentu tato funkce zpracuje konstrukci opakovacího bloku a dosadí na pozici tohoto bloku odpovídající hodnoty kolekce ze vstupních dat.
4. Funkce použije pro formátování cílového textu stejný formát, který byl použit pro specifickou konstrukci.

Body 1. a 2. lze přeskočit, pokud uživatel má znalost vstupních dat a zapíše specifickou konstrukci opakovacího bloku do šablony dokumentu manuálně.

4.2.4 Opakovací blok s tabulkou

Popis

Obdobně jako u opakovacího bloku (kapitola 4.2.3), bude tento softwarový produkt umožňovat definovat opakovací blok s tabulkou prostřednictvím specifické konstrukce. Oproti opakovacímu bloku (bez tabulky) však vytvoří tabulku, do které následně bude iterativně vkládat všechny hodnoty atributu typu `kolekce` (jeden prvek kolekce bude odpovídat jednomu řádku tabulky). Konstrukci pro opakovací blok bude možné taktéž vložit prostřednictvím grafického rozhraní editoru šablon.

Scénář užití

Uživatel vloží do šablony dokumentu specifickou konstrukci pro opakovací blok s tabulkou. Při manuálním zadávání této konstrukce se předpokládá jeho znalost struktury dat (či metadat). Opět lze tato konstrukce vložit prostřednictvím grafického uživatelského rozhraní editoru šablon dokumentů. Pro každou takovou konstrukci vytvoří generátor tabulku a následně do ní vloží hodnoty kolekce vstupních dat. Jeden prvek kolekce odpovídá jednomu řádku vytvořené tabulky. Formát tabulky bude nadefinovaný v dané konstrukci.

Funkční požadavky

1. Funkce předpokládá načtení a zobrazení stromové struktury vstupních dat (na základě metadat) – viz kapitola 4.2.1.

2. Na základě volby uživatele vloží funkce konstrukci opakovacího bloku s tabulkou do šablony dokumentu.

3. Funkce při generování nového dokumentu zpracuje konstrukci opakovacího bloku s tabulkou tak, že vytvoří tabulku v šabloně dokumentu a pro každý prvek kolekce ze vstupních dat dosadí jeho hodnoty na unikátní řádek tabulky. Formát tabulky definuje konstrukce opakovacího bloku s tabulkou.

I v tomto případě lze body 1. a 2. přeskočit v případě, že uživatel disponuje znalostí struktury vstupních dat a vloží konstrukci pro opakovací blok s tabulkou manuálně.

4.2.5 Vložení podšablony

Popis

Softwarový produkt bude umožňovat definovat specifickou konstrukci pro vložení podšablony dokumentu definované v externím souboru do šablony dokumentu. Vkládaná podšablona musí obsahovat stejnou strukturu dat jako šablona, do které se daná podšablona vkládá. Uživatelské rozhraní editoru šablon dokumentů umožní vkládat specifickou konstrukci pro vložení podšablony.

Scénář užití

Uživatel vloží do šablony dokumentu specifickou konstrukci pro vložení podšablony. Při manuálním zadávání konstrukce se předpokládá uživatelova znalost struktury dat (či metadat). Konstrukce lze také vložit prostřednictvím grafického rozhraní editoru šablon dokumentů.

Generátor při tvorbě nového dokumentu pro každou takovou konstrukci najde příslušnou podšablonu dokumentu, kterou zpracuje stejným způsobem jako šablonu (tzn. doplní všechna pole a všechny konstrukce příslušnými daty). Následně vloží obsah takto zpracované podšablony na místo v šabloně, které je definováno zápisem specifické konstrukce pro vložení podšablony.

Funkční požadavky

1. Funkce zprostředkuje uživateli výběr podšablony dokumentu.
2. Funkce vloží specifickou konstrukci pro vložení podšablony dokumentu do šablony dokumentu na základě volby uživatele.
3. Při generování nového dokumentu tato funkce zpracuje konstrukci pro vložení podšablony dokumentu tak, že:
   (a) najde odpovídající podšablonu dokumentu,
   (b) do této podšablony doplní všechna specifická pole a všechny specifické konstrukce,
   (c) poté vloží obsah zpracované podšablony na definované místo v šabloně dokumentu.

Opět lze body 1. a 2. přeskočit v případě, že uživatel má znalost struktury vstupních dat uživatelem a zadá specifickou konstrukci pro vložení podšablony dokumentu ručně.

4.2.6 Definice metod

Popis

Tento softwarový produkt bude umožňovat definovat metody pro:

- *vložení aktuálního data*,
- *formátování data*.

Definice těchto metod se bude realizovat prostřednictvím specifických polí v šabloně dokumentu obsahující identifikátor dané metody. Opět bude možnost vkládání těchto polí pomocí grafického rozhraní editoru šablon dokumentů.

Metoda pro *vložení aktuálního data* nebude závislá na vstupních datech. Metoda pro *formátování data* bude umožňovat formátování jak aktuálního data vloženého prostřednictvím metody, tak data vloženého prostřednictvím specifického pole pro elementární dosazování dat (dle kapitoly 4.2.2). Je možné formátovat pouze atributy typu `datetime`.

Scénář užití

Uživatel vloží do šablony dokumentu specifické pole s identifikátorem dané metody. Pole lze vložit buď prostřednictvím grafického rozhraní editoru šablon dokumentů, nebo manuálně.

V případě použití metody pro *vložení aktuálního data* generátor pro každé takové pole dosadí na místo tohoto pole aktuální datum (při generování) ve stejném formátu, v jakém bylo toto specifické pole.

Pro případ použití metody pro *formátování data* generátor dosadí zformátované datum pro každé takové specifické pole na jeho místo.

Funkční požadavky

1. Funkce na základě volby uživatele vloží do šablony dokumentu specifické pole s identifikátorem metody (v případě *metody pro formátování data* zprostředkuje uživateli možnost výběru formátu data).

2. Funkce při generování nového dokumentu:

   - dosadí na pozici specifického pole aktuální datum (ve stejném formátu, v jakém bylo specifické pole) v případě použití metody pro *vložení aktuálního data*,
   - dosadí zformátované datum na pozici specifického pole v případě použití metody pro *formátování data*.

Bod 1. lze v případě manuálního zadání specifického pole přeskočit.

4.3 Požadavky na vnější rozhraní

4.3.1 Uživatelské rozhraní

Pro část *editoru šablon dokumentů* bude implementováno grafické uživatelské rozhraní, které musí být dostatečně jednoduché a intuitivní pro běžného uživatele se základní znalostí kancelářského balíku Microsoft Office.

Prostřednictvím grafického uživatelského rozhraní musí mít uživatel možnost využít všech jazykových konstrukcí, které podporuje *generátor dokumentů z uživatelských šablon*.

4.3.2 Hardwarové rozhraní

Softwarový produkt neobsahuje hardwarové rozhraní.

4.3.3 Softwarové rozhraní

Část *generátoru dokumentů z uživatelských šablon* musí disponovat softwarovým rozhraním pro funkční volání z jiných aplikací. Kompletní řešení má softwarové rozhraní vůči souborovému systému pro načítání a ukládání souborů.

4.4 Další mimofunkční požadavky

4.4.1 Výkonnostní požadavky

Softwarový produkt bude nasazen na běžných PC (Personal Computer, osobní počítač) stanicích, jejichž minimální požadavky jsou stanoveny v kapitole 4.1.6. Maximální deklarovaná doba generování jednoho dokumentu z uživatelské šablony je stanovena na 60 sekund na 25 stránek šablony dokumentu.

4.4.2 Bezpečnostní požadavky

Softwarové řešení nebude vyžadovat autentizaci, ani autorizovaný přístup. Není nutné řešit ani požadavky zákona č. 181/2014 Sb. o kybernetické bezpečnosti.

4.4.3 Kvalitativní parametry

Dle zadání zadavatelské společnosti musí být kód softwarového produkt napsán dle stanovené firemní kultury. Programový kód musí být jednoduše udržovatelný a rozšiřitelný.

5 Varianty řešení

Cílem této kapitoly je identifikovat, zhodnotit a vybrat nejvhodnější řešení pro stávající situaci zadavatele.

Dle specifikace požadavků zadavatele (viz kapitola 4) je možné pro realizaci softwarového produktu použít buď existující softwarové řešení třetí strany a upravit jej dle požadavků zadavatele, nebo vytvořit nový softwarový produkt.

V této kapitole nejprve analyzuji existující software, který řeší danou problematiku, a poté popisuji specifika vlastního řešení.

5.1 Analýza existujících produktů

V této kapitole analyzuji již existující softwarové produkty, které umožňují uživateli vytvářet šablony dokumentů a následně automatizovaně generovat výsledné dokumenty prostřednictvím dosazení požadovaných dat. Softwarová řešení, která budu v této kapitole analyzovat, jsou:

- `Stimulsoft BI Designer` [18],
- `Windward Core` [25],
- `Formstack Documents` [13],
- `Apache Velocity` [21].

5.1.1 Stimulsoft BI Designer

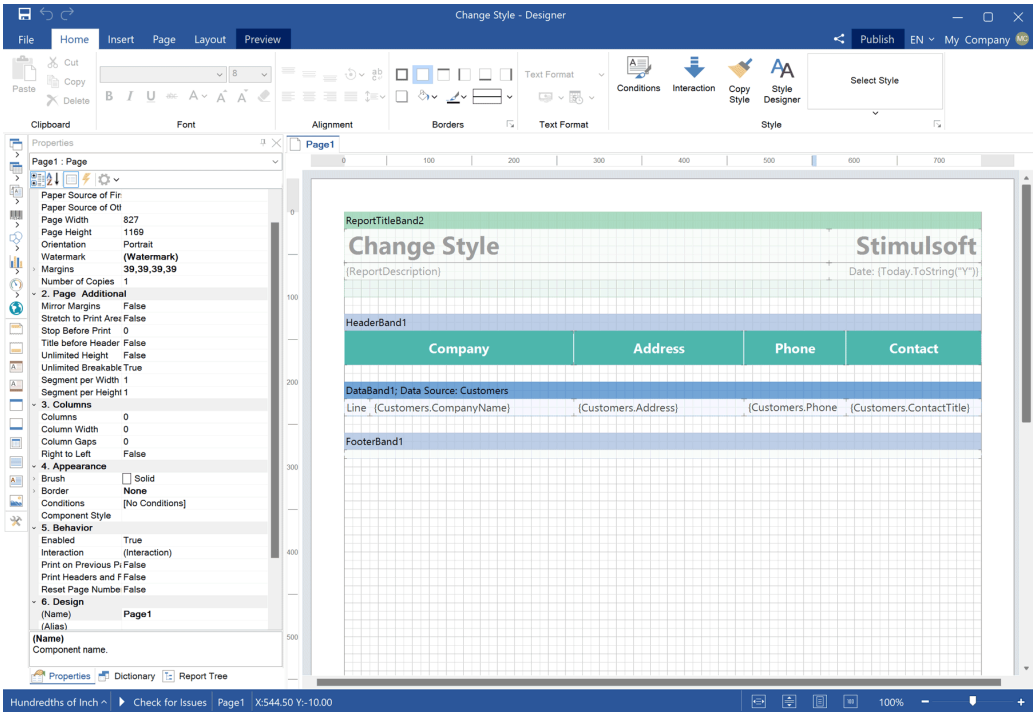
Stimulsoft BI Designer je jeden ze softwarových produktů Estonské společnosti Stimulsoft OÜ. Toto řešení umožňuje vytvářet a editovat uživatelské šablony dokumentů a následně se současným připojením datového zdroje generovat a ukládat finální dokumenty do formátů `PDF`, `Microsoft Excel`, `Microsoft Word` atd. Aplikace dále umožňuje analyzovat data a vytvářet jejich vizualizace [18]. Řešení je publikováno jak verze pro desktop (viz obrázek 5.1), tak pro webový prohlížeč.

Použití tohoto softwarového produktu je omezenou licenční smlouvou. Ceny licencí bez časového omezení jsou uvedeny v tabulce 5.1.

Po hlubší analýze se toto softwarové řešení jeví jako nepříliš vhodné pro potřeby zadavatele CCA Group, a.s. Aplikace je zaměřena spíše na podporu

Tabulka 5.1: Ceny licencí – Stimulsoft BI Designer; Zdroj: [19]

Název licence	**Počet uživatelů**	**Počet geografických lokalit licence**	**Cena**
Single License	1	1	$599,95
Team License	4	1	$1 499,95
Site License	neomezeno	1	$3 999,95
WorldWide License	neomezeno	neomezeno	$11 999,95



Obrázek 5.1: Stimulsoft BI Designer – verze pro desktop; Zdroj: [6]

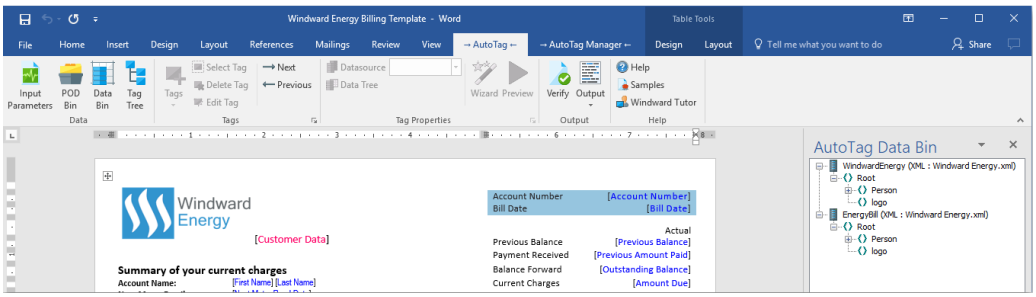
BI (Business intelligence) procesy a na vytváření tabulek a vizualizací dat prostřednictvím grafů a diagramů. Další nevýhodou je složitější ovládání pro běžného cílového uživatele a cena jednotlivých licencí.

5.1.2 Windward Core

Windward Core je jedním ze softwarových řešení americké společnosti Windward Studios, LLC. Jedná se o produkt, který je zaměřený na vytváření reportů a generování dokumentů. Umožňuje vytvářet a upravovat uživatelské šablony dokumentů přímo v prostředí Microsoft Office, připojit různorodé

datové zdroje a následně generovat výstupní dokumenty ve formátech Microsoft Word, Microsoft Excel, PDF a mnoho dalších [27]. Windward Core je rozdělen do dvou komponent:

- `Designer` – uživatelské rozhraní – publikované jako rozšíření produktové řady Microsoft Office (viz obrázek 5.2),
- `Engine` – logika aplikace – obsahuje nativní kód pro jazyky Java a .NET, pro ostatní jazyky lze využít REST API (*Application Programming Interface*, aplikační programové rozhraní).



Obrázek 5.2: Windward Core – Designer; Zdroj: [27]

Windward Core je tedy možné používat jak prostřednictvím uživatelského rozhraní v produktech Microsoft Office, tak volat jeho funkce programově. [25].

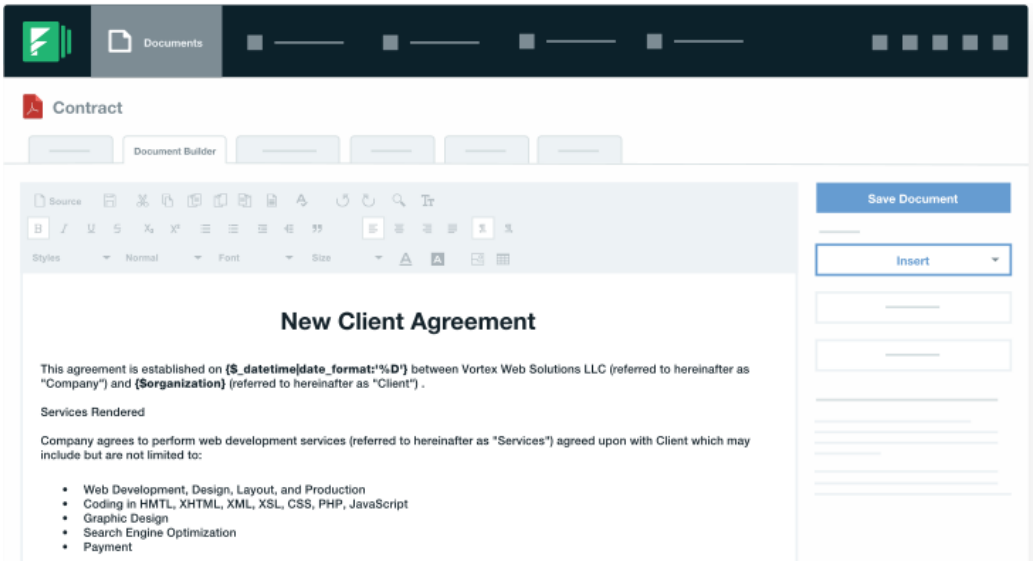
Cena základní licence (5 vývojářů, 3 000 stránek/měsíc) omezující použití daného produkutu je vyčíslena na $6 171 ročně [26].

Výhodou tohoto řešení je jednoduchá integrace do prostředí Microsoft Office a pestrá škála obsahovaných funkcí. Nevýhodou jsou vysoké licenční poplatky, a proto bylo toho řešení vyhodnoceno jako nevhodné pro zadavatele.

5.1.3 Formstack Documents

Dalším podobným softwarovým řešením je *Formstack Documents*. Jedná se o jeden z produktů americké společnosti Formstack, LLC, která se zabývá vývojem platformy napomáhající zefektivnit organizacím podnikové procesy prostřednictvím digitalizace a automatizace [10].

Samotný produkt Formstack Documents umožňuje vytvořit nebo nahrát uživatelskou šablonu, nahrát poskytnutá data a poté vygenerovat dokument. Řešení je realizováno jako webová aplikace [12] (viz obrázek 5.3).



Obrázek 5.3: Formstack Documents – Designer; Zdroj: [12]

Ceny standardních licencí se pohybují od $92 za měsíc (*Starter*) do $367 za měsíc (*Pro*). Nicméně i licence *Pro* povoluje pouze 50 uživatelských šablon. Pro vyšší počet těchto šablon existuje licence *Enterprise*, jejíž cenu výrobce neuvádí (kalkulace se provádí na základě individuální poptávky) [11].

Problémem tohoto produktu z hlediska potřeb zadavatele je poměrně nízká možnost úprav softwarového řešení a formátu přijímaných dat. Z tohoto důvodu se toto řešení jeví jako nevhodné pro potřeby zadavatele.

5.1.4 Apache Velocity

Apache Velocity je šablonovací systém založený na programovacím jazyce *Java*, vývojářem je americká společnost The Apache Software Foundation [21].

Pomocí tohoto systému lze vytvářet webové stránky v souladu se softwarovou architekturou *Model-View-Controller* (MVC) a díky tomu jej lze použít jako alternativu například k technologiím *Java Server Pages* (JSP) nebo PHP (*Hypertext Preprocessor*). Také může být tento produkt použit například ke generování SQL skriptů a XML (*Extensible Markup Language*) souborů z předem definovaných šablon. V neposlední řadě lze Velocity použít jako samostatnou aplikace pro generování reportů. Na rozdíl od předchozích řešení je tento produkt zdarma k použití (pod licencí *Apache Software License*) [21]. Součástí řešení Apache Velocity jsou projekty:

- `Velocity Engine` – jedná se o výkonnou část šablonovacího systému,

- `Velocity Tools` – obsahuje pomocné nástroje k vytváření webových či jiných aplikací [22].

Pro definici šablon v rámci Velocity Engine je vytvořen jazyk `Velocity Template Language` (VTL). Ten umožňuje například deklaraci proměnných (variables), vlastností (properties), metod (methods), dále umožňuje zápis direktiv atd. [5]. Podrobněji je jazyk popsán v kapitole 5.2.2.

Výhodou tohoto řešení je vysoká možnost úprav, jednoduchá integrace do dalších aplikací a licence, která je zdarma. Nevýhodou tohoto řešení je skutečnost, že produkt neobsahuje grafické uživatelské rozhraní pro vytvoření šablony dokumentu, nativně neumožňuje zpracování formátovaného textu pro Microsoft Word a není kompatibilní s prostředím `.NET Framework`. V této podobě řešení není vhodné pro potřeby zadavatele, nicméně je možné jej použít jako inspiraci pro vytvoření nového produktu, zejména gramatiky jazyka `VTL`.

5.1.5 Shrnutí

Analyzovaná softwarová řešení umožňují uživateli vytvořit šablonu dokumentu, následně připojit data a nakonec vygenerovat finální dokument. První *tři* zmíněné produkty obsahují grafické uživatelské rozhraní pro tvorbu a editaci šablon dokumentů a jsou pro komerční využití omezeny zpoplatněnými licencemi. Posledním uvedeným produktem je `Apache Velocity`, který nedisponuje uživatelským rozhraním pro tvorbu šablon, ale licence není zpoplatněna ani pro komerční potřeby.

Z analýzy těchto řešení jsem vyvodil, že první tři produkty nejsou vhodné pro potřeby zadavatele (jednotlivé důvody jsou popsány výše). Poslední analyzované řešení lze použít jako inspirace pro vytvoření nového softwarového produktu.

Je tedy nutné vytvořit zcela *nové softwarové řešení*, které bude plnit potřeby zadavatele dle specifikace požadavků popsaných v kapitole 4.

5.2 Aplikace vlastního řešení

V této kapitole popisuji nejprve oblasti požadovaného řešení. Dále se zaměřuji na softwarové nástroje a knihovny, které pokrývají problematiku oblastí řešení a pomocí kterých budu realizovat softwarový produkt.

5.2.1 Oblasti řešení

K realizaci softwarového produktu je nutné vyřešit problematiku následujících oblastí:

- formální jazyk pro definici šablon dokumentů,
- lexikální a syntaktická analýza šablon a jejich parsování,
- dosazení vstupních dat do šablony dokumentu,
- uživatelské rozhraní editoru šablon.

Jako inspirací pro vytvoření gramatiky formálního jazyka pro definici šablon dokumentů jsem zvolil *Velocity Template Language (VTL)*. Lexikální a syntaktickou analýzu šablon a následné parsování budu realizovat prostřednictvím nástroje *ANother Tool for Language Recognition (ANTLR)* z důvodu snadné rozšiřitelnosti o další jazykové konstrukce.

Pro uživatelské rozhraní editoru šablon dokumentů použiji prostředí Microsoft Word, v rámci kterého implementuji rozšíření pro vytvoření a editaci těchto šablon pomocí nástroje *Visual Studio Tools for Office (VSTO)*. Pro dosazování dat do šablony dokumentu využiji knihovnu *Microsoft Office Interop Word*.

Všechny tyto technologie popisuji dále v textu.

5.2.2 Velocity Template Language

V této podkapitole podrobně popisuji Velocity Template Language, zejména jeho jazykové konstrukce.

Kompletní specifikace VTL uvádí dokumentace (viz [5]). Zápis je uváděn ve formě blízké EBNF (*extended Backus–Naur form*).

Jazykové konstrukce jsou rozděleny na:

- reference *(references)*, kam spadají proměnné (variables), vlastnosti (properties) a metody (methods),
- direktivy *(directives)* – jedná se například o direktivy `set`, `if`, `foreach`, a mnoho dalších.

Reference (References)

V této podkapitole jsou uvedeny všechny jazykové konstrukce spadající do referencí. Nejprve je zde popsán zápis identifikátorů.

Každý identifikátor musí začínat malým nebo velkým písmenem (bez diakritiky) nebo podtržítkem. Poté může pokračovat libovolným počtem písmen, číslic či podtržítek. Definice je následovná:

```
( a..z | A..Z | _ ) [ ( a..z | A..Z | 0..9 | _ ),
( a..z | A..Z | 0..9 | _ ), ... ]
```

.

Proměnné (Variables)

Proměnné jsou definovány následovně:

```
$ [ ! ] [ { ] identifier [ [ | alternate value ] } ]
```

,

kde:

- *identifier* je identifikátor (popsaný výše),
- *alternate value* je alternativní výraz pro případ, že reference, na kterou identifikátor odkazuje, je `null`, prázdná, `false`, nebo 0.

Vlastnosti (Properties)

Vlastnosti jsou definovány následovně:

```
$ [ { ] identifier . identifier [ [ | alternate value
] } ]
```

,

kde:

- *identifier* je identifikátor (popsaný výše),
- *alternate value* je alternativní výraz pro případ, že reference, na kterou identifikátor odkazuje, je `null`, prázdná, `false`, nebo 0.

Metody (Mehods)

Zápis metod je následovný:

```
$ [ { ] identifier . identifier ( [ parameter list...
] ) [ [ | alternate value ] } ]
```

,

kde:

- *identifier* je identifikátor,
- *alternate value* je alternativní výraz pro případ, že reference, na kterou identifikátor odkazuje, je `null`, prázdná, `false`, nebo 0,
- *parameter list* je volitelný parametr, který obsahuje seznam parametrů dané metody oddělených čárkou.

Direktivy (Directives)

V této podkapitole je uveden pouze výběr některých direktiv, které VTL obsahuje (kompletní výčet je uveden v dokumentaci jazyka – viz [5]). Všechny direktivy jsou uvozeny znakem `#`.

Direktiva `set`

Direktiva `set` slouží k přiřazení hodnoty reference. Je definována následovně:

```
# [ { ] set [ } ] ( $ref = [ ", ' ] arg [ ", ' ] )
```

kde:

- *`$ref`* je levá strana výrazu – musí se jednat o proměnnou nebo vlastnost,
- *`arg`* je pravá strana výrazu – pokud je uvozena dvojitými uvozovkami, je parsovaná, pokud je uvozena jednoduchými uvozovkami, parsovaná není. V případě, že je hodnota `null`, k přiřazení nedojde.

Příklady použití této direktivy jsou uvedeny v ukázce kódu A.1 přílohy A.

Direktivy `if, elseif, else`

Podmínky se v jazyce VTL realizují obvykle jako ve většině formálních jazyků prostřednictvím direktiv `if, elseif, else`. Jsou definovány následovně:

```
# [ { ] if [ } ] ( condition ) output [ # [ { ] elseif
[ } ] ( condition ) output ] [ # [ { ] else [ } ]
output ] # [ { ] end [ } ]
```

přičemž:

- *`condition`* je výraz, který se vyhodnocuje,
- *`output`* je blok kódu jazyka VTL.

Příklady použití těchto direktiv jsou uvedeny v ukázce kódu A.2 přílohy A. VTL podporuje obvyklé operátory ve vyhodnocovacích výrazech podmínek, jejich konkrétní výčet je uveden v tabulce A.1 přílohy A.

Direktiva `foreach`

Direktiva `foreach` slouží k iterování přes všechny objekty dané kolekce a je definována následovně:

```
# [ { ] foreach [ } ] ( $ref in arg ) statements
[ # [ { ] else [ } ] alternate statements ]
# [ { ] end [ } ]
```
,

kde:

- *$ref* je proměnná, do které je v rámci každé iterace vložen prvek z kolekce,
- *arg* je kolekce prvků, přes které je iterováno prostřednictvím direktivy `foreach`,
- *statements* je kód v jazyce VTL, pokud je kolekce prvků `arg` validní,
- *alternate statements* je kód v jazyce VTL, pokud není kolekce prvků `arg` validní (tzn. je `null`, prázdná nebo přes ni nelze iterovat).

Příklady použití této direktivy jsou uvedeny v ukázce kódu A.3 přílohy A.

5.2.3 ANother Tool for Language Recognition

ANother Tool for Language Recognition (ANTLR) je nástroj implementovaný v programovacím jazyce *Java* pro lexikální a syntaktickou analýzu, který umožňuje tvorbu parserů strukturovaných textů či binárních souborů [1]. Hlavními vývojáři tohoto produktu jsou Terence Parr a Sam Harwell. Software je opatřený licencí BSD (použití je zdarma, kód je dostupný ve veřejném repozitáři na serveru *GitHub*) [3]. Nástroj ANTLR poskytuje *aplikační programové rozhraní* (API) pro volání jeho jednotlivých funkcí, pro verzi ANTLR 4 je dostupné pro programovací jazyky: Java, C# a C++ [4].

Velmi často se ANTLR používá k tvorbě překladačů formálních jazyků. Ty musí být popsány bezkontextovou gramatikou typu LL dle pravidel tvorby ANTLR gramatiky (viz ANTLR dokumentace – [2]). Z takto popsané gramatiky generuje rekurzivním sestupem rozpoznávač a poskytne rozhraní pro vytvoření parsovacího stromu [16].

5.2.4 Visual Studio Tools for Office

Visual Studio Tools for Office (VSTO) je soubor vývojářských nástrojů v rámci produktu *Microsoft Visual Studio*, prostřednictvím kterých lze spouštět *.NET Framework* aplikace v prostředí produktů řady *Microsoft Office* [24]. Díky tomu lze přizpůsobit Microsoft Office aplikace a přidávat specifické funkce.

VSTO lze v rámci produktu Microsoft Visual Studio použít dvěma způsoby:

- *Document-level customization* – přizpůsobení a úprava souborů formátů Microsoft Office na úrovni specifického dokumentu (např. právě otevřeného),
- *VSTO Add-ins* – přizpůsobení a úprava na úrovni celé Microsoft Office aplikace (např. přidání funkcí do aplikace Word) [15].

V rámci této práce použiji rozšíření celé aplikace Microsoft Word (tzn. VSTO Add-ins).

5.2.5 Microsoft Office Interop Word

Microsoft Office Interop Word je knihovna (DLL - *Dynamic-link library*) poskytující rozhraní v jazyce C# pro přístup k objektům aplikace Microsoft Word [14]. Vyžaduje instalaci produktu Microsoft Word na zařízení, na kterém je spouštěna. Umožňuje například vkládat text do Word dokumentu, upravovat jej, formátovat a mnoho dalšího.

V rámci realizovaného softwarového řešení použiji tuto knihovnu zejména ke vkládání textu do šablony dokumentu v rámci generování nového dokumentu.

6 Návrh softwarového řešení

V analytické části práce jsem vybral realizaci vlastního řešení, tzn. implementaci nového softwarového produktu. Na základě tohoto rozhodnutí navrhuji v této kapitole nejprve architekturu vlastního softwarového řešení. Dále navrhuji definici uživatelských šablon, strukturu vstupních dat a jejich metadat a grafické uživatelské rozhraní editoru šablon dokumentů. Následně popisuji návrh funkcí softwarového produktu, které pokrývají požadavky zadavatele na tyto funkce (viz kapitola 4.2), a rozšiřujících funkcí vlastního softwarového řešení.

6.1 Architektura řešení

Dle specifikace požadavků zadavatele (kapitola 4) je softwarové řešení rozděleno na dvě hlavní části:

- editor šablon dokumentů,
- generátor dokumentů z uživatelských šablon.

V souladu s kontextem systému (kapitola 4.1.5) ilustruji na obrázku B.1 přílohy B preskriptivní architekturu daného softwarového řešení v modelovacím jazyce *ArchiMate*.

V příloze C rámcově zobrazuji prostřednictvím UML (*Unified Modeling Language*) diagramů aktivit workflow editoru šablon dokumentů (obrázek C.1) a generátoru dokumentů z uživatelských šablon (obrázek C.2).

6.2 Definice uživatelských šablon

Uživatelské šablony (šablony dokumentů) slouží ke generování nového dokumentu prostřednictvím dosazení vstupních dat. Taková šablona bude definována ve formátu Microsoft Word (tzn. `doc` nebo `docx`). Jazykové konstrukce, které je možné v šabloně použít k dosazení vstupních dat, jsou inspirovány jazykem VTL (viz kapitola 5.2.2). Parsování takové šablony bude realizováno v souladu s předchozí analýzou prostřednictvím nástroje ANTLR. Z tohoto důvodu je nutné nadefinovat příslušnou gramatiku jazyka pro tvorbu těchto šablon.

V této kapitole popisuji jazykové konstrukce, které jsou nezbytné pro realizaci funkcí specifikovaných zadavatelem v kapitole 4.2, a gramatiku jazyka šablon.

6.2.1 Jazykové konstrukce

Definici jazykových konstrukcí uvádím ve stejném zápisu jako u jazyka *VTL* pro snadnější porovnání. Stejně jako *VTL*, i zde jsou jazykové konstrukce rozděleny na:

- *reference* – sem patří proměnné, vlastnosti a metody,
- *direktivy* – konkrétně se jedná o direktivy `foreach`, `tableforeach` a `include`.

Reference

V této kapitole definuji jazykové konstrukce spadající do referencí pro jazyk šablon. Nejprve popisuji definici identifikátorů.

Každý identifikátor musí začínat malým nebo velkým písmenem, pokračovat může libovolným počtem písmen nebo číslic:

```
( a..z | A..Z ) [ ( a..z | A..Z | 0..9 ),
( a..z | A..Z | 0..9 ), ... ]
```

.

Proměnné

Proměnné jsou uvozeny znakem `$`, následuje název proměnné, který může být uzavřen ve složených závorkách. Jsou nadefinovány následujícím způsobem:

```
$ [ { ] var_identifier [ } ]
```

,

kde:

- *`var_identifier`* je identifikátor proměnné.

Vlastnosti

Vlastnosti jsou aplikovány nad proměnnou, od které se oddělují tečkou. Vlastnosti mohou mít další vlastnosti, které jsou opět odděleny tečkou. Jsou definovány následovně:

```
$ [ { ] var_identifier . prop_identifier
[ ( . prop_identifier ), ... ] [ } ]
```

,

kde:

- *var_identifier* je identifikátor proměnné,
- *prop_identifier* je identifikátor vlastnosti.

Metody

Metody mohou být použity samostatně, nad proměnnou či její vlastností (záleží na charakteru metody). Opět jsou vzájemně odděleny tečkou. Zápis metod pro volání samostatně:

```
$ [ { ] method_identifier ( [ parameter ] ) [ } ]
```

,

a pro volání nad proměnnou:

```
$ [ { ] var_identifier . method_identifier
( [ parameter ] ) [ } ]
```

,

kde:

- *var_identifier* je identifikátor proměnné,
- *method_identifier* je identifikátor metody,
- *parameter* je volitelný parametr, který obsahuje parametr dané metody.

Direktivy

Definované direktivy jazyka pro definici šablon jsou:

- `foreach`,
- `tableforeach`,
- `include`.

Všechny direktivy jsou uvozeny znakem `#`.

Direktiva `foreach`

Direktiva `foreach` definuje opakovací blok, v rámci kterého umožňuje iterativně dosadit všechny hodnoty dané kolekce na specifická místa v těle této direktivy. Je definována následovně:

```
# [ { ] foreach [ } ] ( $ref in arg ) statements
# [ { ] end [ } ]
```

,

kde:

- *$ref* je reference, do které je v rámci každé iterace vložen prvek z kolekce,
- *arg* je kolekce prvků, přes které je iterováno prostřednictvím této direktivy,
- *statements* je blok textu či kódu v jazyce šablon (aktuální hodnotu z kolekce lze dosadit použitím reference *$ref*).

Příklad použití této direktivy v prostředí Microsoft Word je uveden na obrázku F.1 přílohy F.

Direktiva `tableforeach`

Direktiva `tableforeach` definuje opakovací blok s tabulkou, v rámci kterého iterativně přidává řádky do tabulky a dosazuje do nich hodnoty dané kolekce. V těle této direktivy je nutné předem definovat tabulku o dvou řádcích, kde v prvním řádku bude hlavička tabulky a v druhém řádku specifická pole pro iterativní dosazení hodnot kolekce. Direktiva je definována následovně:

```
# [ { ] tableforeach [ } ] ( $ref in arg ) init_table
# [ { ] end [ } ]
```

,

kde:

- *$ref* je reference, do které je v rámci každé iterace vložen prvek z kolekce,
- *arg* je kolekce prvků, přes které je iterováno prostřednictvím této direktivy,
- *init_table* je tabulka o dvou řádcích, kde v prvním řádku je hlavička tabulky a v druhém řádku jsou proměnné či vlastnosti dostupné prostřednictvím reference *$ref*.

Příklad použití této direktivy v prostředí Microsoft Word je uveden na obrázku F.2 přílohy F.

Direktiva `include`

Direktiva `include` definuje vložení podšablony dokumentu, která je uložena v externím souboru. Je vyžadováno, aby podšablona byla ve formátu Microsoft Word. Pokud podšablona obsahuje jazykové konstrukce v jazyce šablon,

musí tyto konstrukce odpovídat stejnému formátu dat jako nadřazená šablona. V rámci této direktivy je nutné specifikovat cestu uložené podšablony. Direktiva je definována následovně:

```
# [ { ] include [ } ] (" sub_template_path ")
```

kde:

- *sub_template_path* je cesta v souborovém systému k dané podšabloně.

Příklad použití této direktivy v prostředí Microsoft Word je uveden na obrázku F.3 přílohy F.

6.2.2 Gramatika jazyka šablon

Lexikální a syntaktickou analýzu a následné parsování šablony dokumentu budu realizovat prostřednictvím nástroje ANTLR 4. Díky tomu bude možné jednoduše rozšiřovat gramatiku jazyka šablon o další jazykové konstrukce.

Definice gramatiky jazyka šablon je rozdělena na dvě části:

- *lexer* – ze vstupního proudu znaků vytvoří *symboly*,
- *parser* – ze symbolů vytvoří rekurzivním sestupem *rozpoznávač* a poskytne rozhraní pro vytvoření *parsovacího stromu*.

Lexer gramatiky jazyka šablon je uveden v ukázce kódu G.1 a *parser* je uveden v ukázce kódu G.2 přílohy G.

Gramatika jazyka šablon bude v rámci realizace této práce zpracovávat pouze datové typy `string`, tzn. všechny datové typy uvedené v kapitole 4.2 budou kontrolovány na úrovni zpracování *editorem šablon dokumentů* a do šablony dokumentu budou vkládány jazykové konstrukce zpracovávající pouze datový typ `string`. V gramatice jazyka šablon jsou uvedena pravidla pro datový typ `integer` k pozdější implementaci rozšíření tohoto softwarového produktu.

Rozsah viditelnosti proměnných (*scope*) v jazyce šablon dokumentů bude definován v rozsahu pravidla `block` (pravidlo definováno v ukázce kódu G.2 přílohy G), tzn. při zanoření jednotlivých bloků mohou vnitřní identifikátory překrýt identifikátory nadřazeného bloku, a pokud bude použit identifikátor, který není definován v daném bloku, bude použit identifikátor bloku nadřazeného atp. V implementaci této práce se bude rozsah viditelnosti proměnných týkat pouze řídících proměnných (v kapitole 6.2.1 uvedeny jako *$ref*) pro direktivy `foreach` a `tableforeach`.

6.3 Struktura vstupních dat a jejich metadat

Ve specifikaci požadavků zadavatele (kapitola 4) je definován formát vstupních dat – jedná se o formát JSON. Struktura vstupních dat není zadavatelem stanovena, pouze je definováno, že struktura dat musí být dostatečně generická, aby bylo možné jejím prostřednictvím popsat různorodě strukturovaná data. Konkrétní struktura pro každou sadu vstupních dat je popsána metadaty.

V této kapitole definuji nejprve strukturu vstupních dat a poté strukturu jejich metadat.

6.3.1 Definice struktury vstupních dat

Aby byla struktura vstupních dat dostatečně generická, skládá se ze *dvou* obecných typů objektů:

- `DataObject`,
- `DataAttribute`.

`DataObject`

Jedná se o hlavní typ objektů, který *musí* být kořenovým objektem JSON souboru vstupních dat. Obsahuje tři pole:

- `name` – jedná se o pole typu `string`, které nese název daného datového objektu, v případě kořenového objektu se jedná o název celé datové sady, musí být unikátní v rámci jedné úrovně hierarchie,
- `descendats` – je kolekce typu `DataObject`,
- `attributes` – je kolekce typu `DataAttribute`.

Příklad použití objektu `DataObject` je uveden v ukázce kódu D.1 přílohy D.

`DataAttribute`

Objekt `DataAttribute` zastupuje jeden atribut daného objektu `DataObject`. Kolekce objektů `DataAttributes` se vkládá do pole `attributes` objektu `DataObject`. Objekt `DataAttributes` obsahuje dvě pole:

- `name` – jedná se o pole typu `string`, které nese název daného atributu, musí být unikátní v rámci jedné úrovně hierarchie,
- `value` – je pole typu `string`, které obsahuje hodnotu atributu.

Příklad použití objektu `DataAttribute` je uveden v ukázce kódu D.2 přílohy D.

6.3.2 Definice struktury metadat

Metadata konkrétní vstupní datové sady popisují strukturu těchto vstupních dat (např. hierarchii objektů, datové typy atributů atd.). Metadata jsou taktéž ve formátu JSON a hierarchie zanoření jednotlivých objektů přesně odpovídá hierarchii zanoření objektů vstupních dat. Metadatový soubor používá také dva typy objektů:

- `MetadataObject` – odpovídá objektu vstupních dat `DataObject`,
- `MetadataAttribute` – odpovídá objektu vstupních dat `DataAttribute`.

Oba tyto objekty odpovídají objektům vstupních dat, liší se pouze v konkrétních polích.

MetadataObject

Jedná se o hlavní typ objektů metadatového souboru, musí být taktéž kořenovým objektem. Obsahuje pět polí:

- `name` – pole typu `string`, nese název daného objektu (v případě kořenového objektu nese název celé datové sady), musí být shodné s polem `name` objektu `DataObject` vstupních dat pro odpovídající datovou sadu a musí být unikátní v rámci jedné úrovně hierarchie,
- `label` – pole typu `string`, nese popisný štítek daného objektu (pro zobrazení uživateli v grafickém uživatelském rozhraní – GUI),
- `isArray` – příznak typu `boolean` – `true` pokud mají být následovníci interpretováni jako kolekce (ve vstupních datech se očekává kolekce), jinak `false`,
- `descendats` – je kolekce typu `MetadataObject`,
- `attributes` – je kolekce typu `MetadataAttribute`.

Příklad použití objektu `MetadataObject` je uveden v ukázce kódu E.1 přílohy E. V tomto příkladu je definováno, že objekt `ucastnici` je kolekce a ve vstupních datech se očekává jeden nebo více stejných typů objektů uložených v poli `descendats` objektu `ucastnici` (viz ukázka kódu D.1 přílohy D).

MetadataAttribute

Objekt `MetadataAttribute` zastupuje strukturu jednoho atributu objektu `MetadataObject` (obdobně jako `DataAttrbitute` pro vstupní data). Kolekce objektů `MetadataAttribute` se vkládá do pole `attributes` objektu `MetadataObject`. Objekt `MetadataAttribute` obsahuje čtyři pole:

- `name` – pole typu `string`, nese název daného atributu, musí být shodné s polem `name` objektu `DataAttrbiute` vstupních dat pro odpovídající datovou sadu a musí být unikátní v rámci jedné úrovně hierarchie,
- `dataType` – pole typu `string`, jedná se o datový typ odpovídajícího atributu,
- `label` – pole typu `string`, jedná se o popisný štítek daného atributu (pro zobrazení v GUI),
- `example` – pole typu `string`, uvádí příklad hodnoty daného atributu.

Příklad použití objektu `MetadataAttribute` je uveden v ukázce kódu E.2 přílohy E.

6.3.3 Podmínky pro vstupní data v šabloně

Nutnou podmínkou pro správné načtení vstupních dat, resp. jejich metadat, je dodržení definované struktury uvedené v kapitole 6.3.1, resp. 6.3.2. Šablona dokumentu musí obsahovat jazykové konstrukce, které jsou v souladu se vstupními daty (resp. metadaty). Například pokud je v šabloně definována *proměnná* a její *vlastnosti*:

```
${spis.soud.nazev}
```

musí ve vstupních datech tato proměnná a její vlastnosti existovat a poslední z vlastností musí být *atributem*, který obsahuje hodnotu – viz ukázka kódu 6.1.

K vytvoření a editaci šablony je doporučeno použít *editor šablon dokumentů*, který zpřístupňuje pouze jazykové konstrukce, které je možné použít. Tím se výrazně snižuje riziko nekonzistentně definované šablony a selhání generování nového dokumentu.

Ukázka kódu 6.1: Příklad použití objektu `DataObject`;
Zdroj: vlastní zpracování

```
{
   "name":"spis",
   "descendants":[
      {
         "name":"soud",
         "attributes":[
            {
               "name":"nazev",
               "value":"Okresní soud Plzeň – město"
            },
            ...
```

6.4 Grafické uživatelské rozhraní

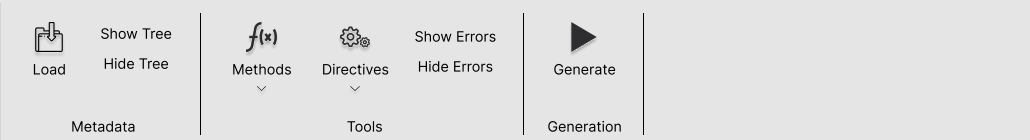
Grafické uživatelské rozhraní editoru šablon dokumentů budu implementovat jako rozšíření produktu Microsoft Word. Funkcionality tohoto rozšíření budou dostupné v horní liště produktu Microsoft Word, konkrétně na záložce s názvem `TemplateBuilder`.

V této kapitole ilustruji návrh grafického uživatelského rozhraní v softwarovém nástroji *Figma* [9]. Tento návrh je rozdělen na části:

- lišta nástrojů,
- stromová struktura metadat,
- formulářová okna,
- panel syntaktických chyb.

6.4.1 Lišta nástrojů

Lišta nástrojů (ilustrována na obrázku 6.1) se bude nacházet v horní části okna a bude hlavní součástí záložky `TemplateBuilder`.



Obrázek 6.1: GUI – lišta nástrojů; Zdroj: vlastní zpracování

Lišta je rozdělena na tři sekce:

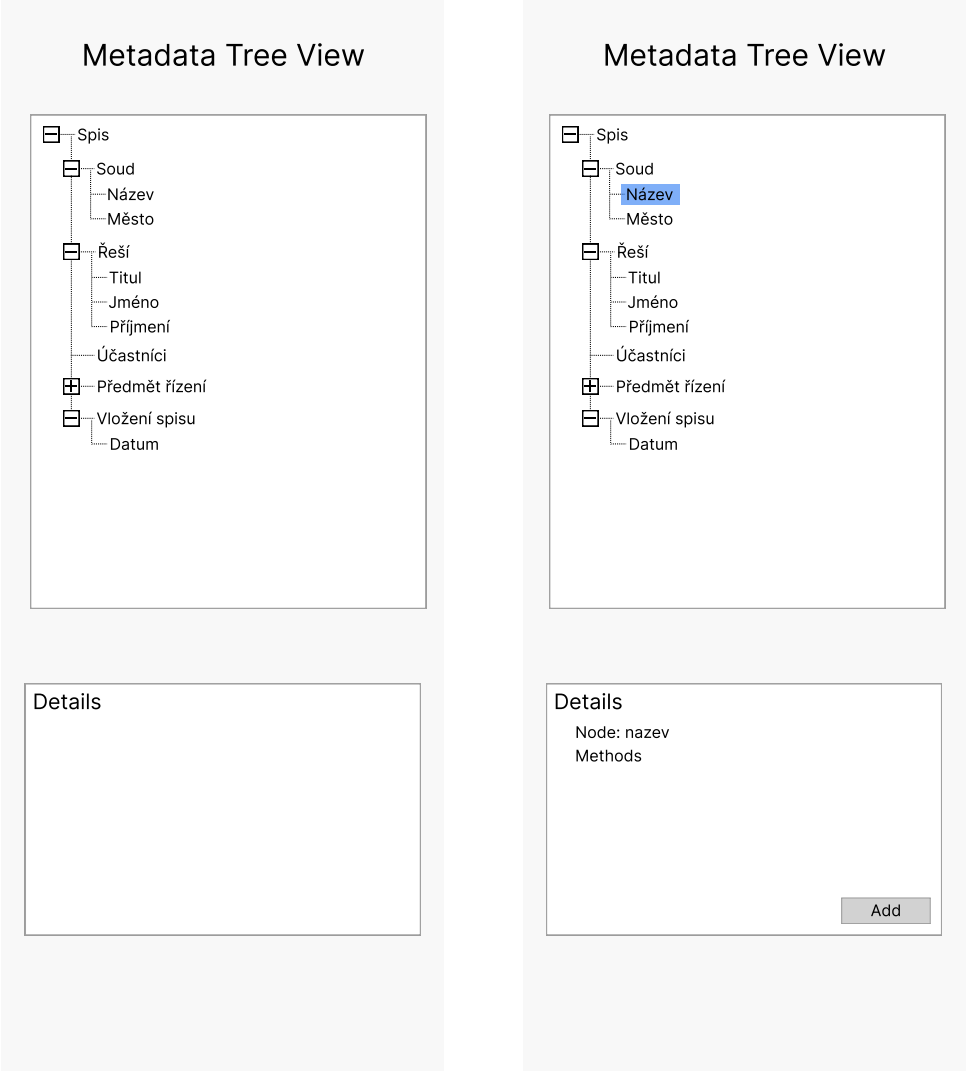
- `Metadata`:
  - tlačítko `Load` – načtení metadatového souboru,
  - tlačítko `Show Tree` – zobrazení stromové struktury metadat,
  - tlačítko `Hide Tree` – skrytí stromové struktury metadat,
- `Tools`:
  - galerie `Methods` – galerie metod, které nejsou závislé na vstupních datech,
  - galerie `Directives` – galerie direktiv, které nejsou závislé na vstupních datech,
  - tlačítko `Show Errors` – zobrazení panelu syntaktických chyb,
  - tlačítko `Hide Errors` – skrytí panelu syntaktických chyb,
- `Generation`:
  - tlačítko `Generate` – vygenerování nového dokumentu z šablony.

6.4.2 Stromová struktura metadat

Stromová struktura metadat se bude nacházet v pravé části okna. Bude obsahovat jak samotný strom metadat, tak sekci podrobností *objektů* a *atributů* metadat.
Stromová struktura metadat je ilustrována na obrázcích 6.2 a 6.3, konkrétně se jedná o stavy, kdy:

- není vybrán žádný atribut ani objekt – obrázek 6.2a,
- je vybrán atribut `Název` (typu `string`) – obrázek 6.2b,
- je vybrán atribut `Datum` (typu `datetime`) – obrázek 6.3a,
- je vybrán objekt `Účastníci` (`kolekce`) – obrázek 6.3b.

V sekci podrobností (`Details`) jsou zobrazeny detaily aktuálního atributu či objektu ve stromové struktuře. Pokud bude možné pro vybraný atribut či objekt přidat jazykovou konstrukci do šablony dokumentu, bude zobrazeno tlačítko `Add`, které toto přidání realizuje. Pro atributy, nad kterými mohou být volány metody, jsou tyto metody v sekci podrobností zobrazeny (obrázek 6.3a). Pro objekty typu *kolekce* jsou dostupné direktivy v této sekci též zobrazeny (obrázek 6.3b).

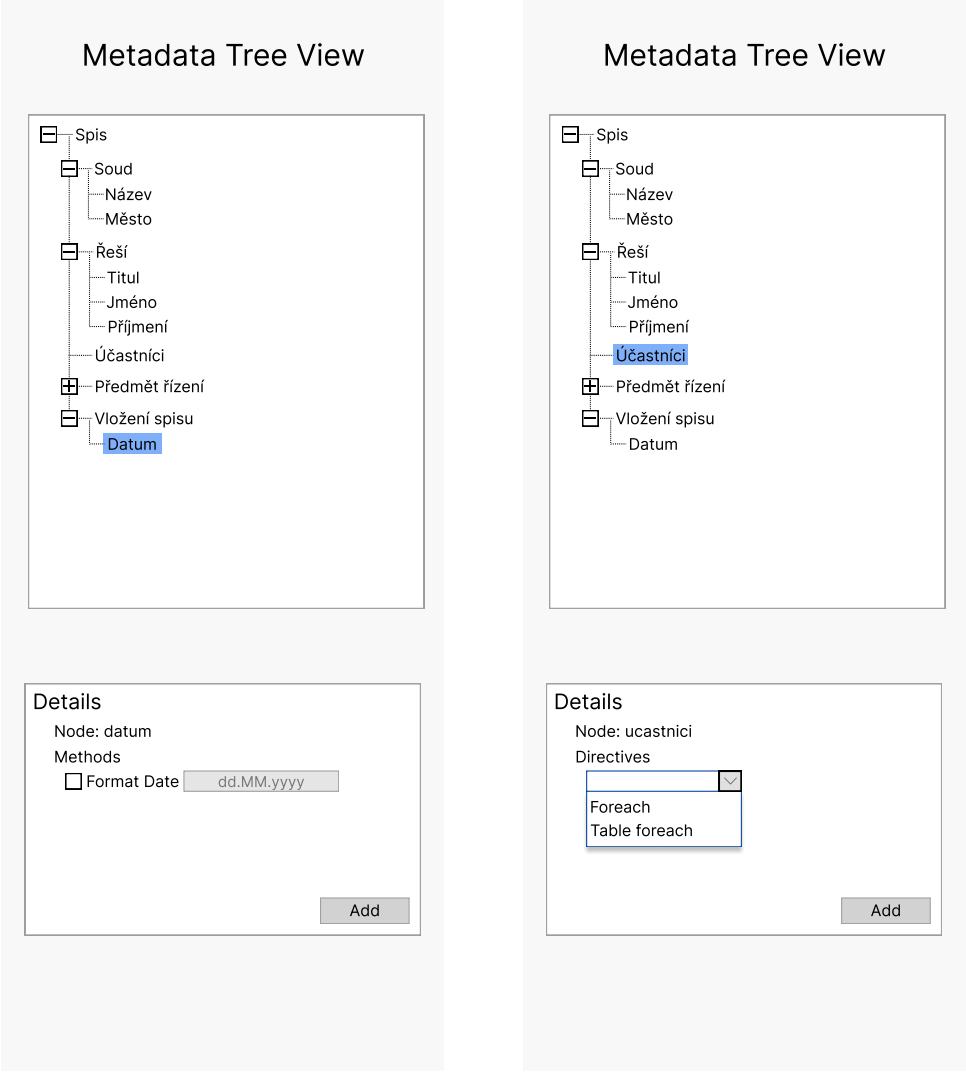


(a) Stav: nevybrán atribut ani objekt (b) Stav: vybrán atribut Název

Obrázek 6.2: GUI – stromová struktura metadat, 1. část;
Zdroj: vlastní zpracování

6.4.3 Formulářová okna

Galerie `Methods` v sekci `Tools` lišty nástrojů (viz kapitola 6.4.1) bude obsahovat v rámci implementace této práce metodu `Current Date` pro vložení jazykové konstrukce aktuálního data do šablony dokumentu. Po zvolení této metody se zobrazí formulářové okno pro vložení této konstrukce do šablony s možností formátování data (podrobněji v kapitole 6.5.6). Vložení jazykové konstrukce do šablony se bude provádět tlačítkem `Add`. Návrh tohoto okna je ilustrován na obrázku 6.4.

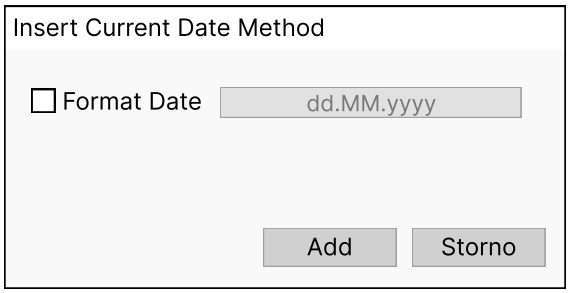


(a) Stav: vybrán atribut Datum (b) Stav: vybrán objekt Účastníci

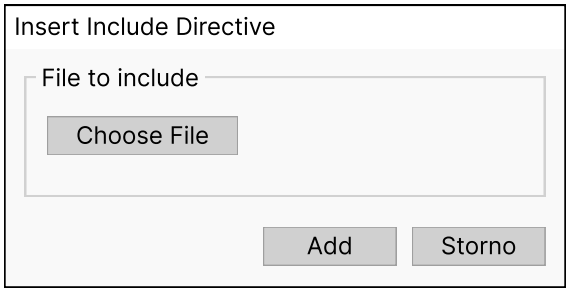
Obrázek 6.3: GUI – stromová struktura metadat, 2. část;
Zdroj: vlastní zpracování

Galerie `Directives` v sekci `Tools` lišty nástrojů (viz kapitola 6.4.1) bude obsahovat v rámci implementace této práce direktivu `Include` pro vložení jazykové konstrukce podšablony dokumentu. Po zvolení této direktivy bude vyvolána akce pro zobrazení formulářového okna pro vložení této direktivy do šablony dokumentu. V tomto okně bude tlačítko `Choose File` pro výběr požadované podšablony dokumentu. Po výběru podšablony se vloží jazyková konstrukce tlačítkem `Add`. Návrh okna je zobrazen na obrázku 6.5.

Tlačítko `Generate` v sekci `Generation` lišty nástrojů (viz kapitola 6.4.1) bude umožňovat vygenerování nového dokumentu z aktuálně otevřené (příp. editované) šablony (viz kapitola 6.6). Po kliknutí na toto tlačítko dojde k ote-

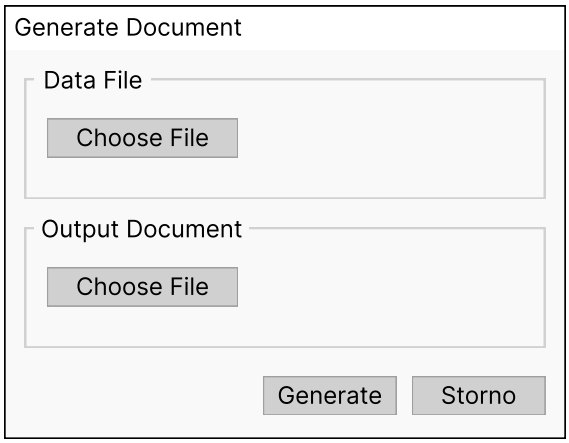


Obrázek 6.4: GUI – formulářové okno pro vložení aktuálního data;
Zdroj: vlastní zpracování



Obrázek 6.5: GUI – formulářové okno pro vložení podšablony dokumentu;
Zdroj: vlastní zpracování

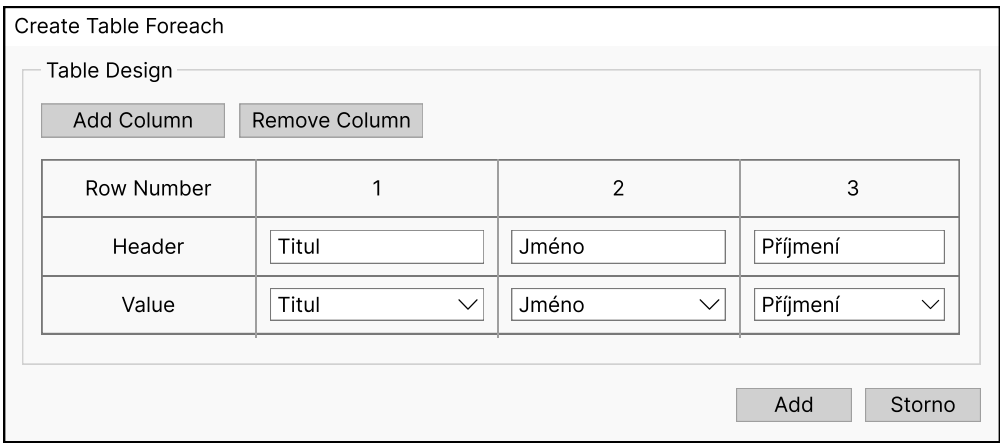
vření formulářového okna, které bude obsahovat tlačítka pro vybrání souboru vstupních dat a zvolení souboru pro uložení nového dokumentu. Dále v tomto okně bude tlačítko `Generate` pro spuštění generování. Návrh tohoto okna zobrazuje obrázek 6.6.



Obrázek 6.6: GUI – formulářové okno pro vygenerování nového dokumentu;
Zdroj: vlastní zpracování

Při vybrání direktivy `Table foreach` ve stromové struktuře metadat pro objekt typu *kolekce* (viz obrázek 6.3b) se otevře formulářové okno pro vy-

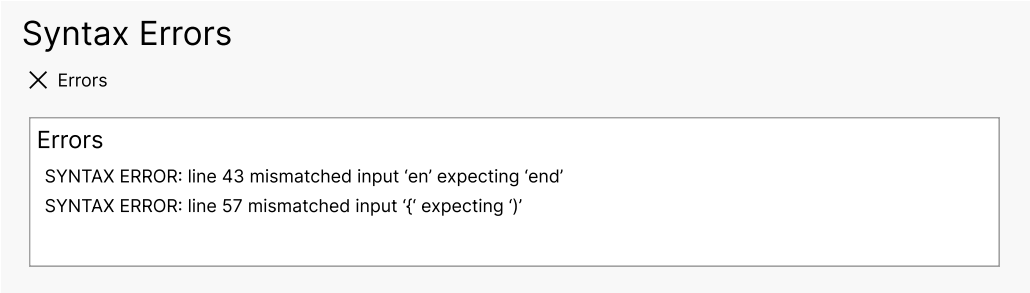
tvoření návrhu a konfiguraci tabulky. Okno bude obsahovat definici hlavičky tabulky a atributů, které mají být do generované tabulky dosazovány. Dále bude okno obsahovat tlačítko `Add Column`, resp. `Remove Column` pro přidání sloupce tabulky, resp. odebrání sloupce tabulky. Pro přidání jazykové konstrukce direktivy `tableforeach` vytvořené dle navržené tabulky v tomto formulářovém okně bude sloužit tlačítko `Add`. Návrh tohoto formulářového okna je ilustrován na obrázku 6.7.



Obrázek 6.7: GUI – formulářové okno pro vytvoření návrhu tabulky;
Zdroj: vlastní zpracování

6.4.4 Panel syntaktických chyb

Panel syntaktických chyb se bude nacházet ve spodní části okna. Po načtení metadat bude možné tento panel zobrazit kliknutím na tlačítko `Show Errors` v liště nástrojů (viz kapitola 6.4.1). Panel bude zobrazovat syntaktické chyby v aktuálně upravované šabloně dokumentu. Grafický návrh tohoto panelu je ilustrován na obrázku 6.8.



Obrázek 6.8: GUI – panel syntaktických chyb;
Zdroj: vlastní zpracování

6.5 Funkce softwarového produktu

V této kapitole navrhuji funkce daného softwarového produktu, které jsou v souladu se specifikací funkcí zadavatele (kapitola 4.2).

6.5.1 Načtení metadat

Editor šablon dokumentů bude umožňovat načtení metadat vstupních dat ve formátu JSON, požadovaná struktura těchto metadat je uvedena v kapitole 6.3.2. V grafickém uživatelském rozhraní editoru šablon dokumentů:

1. uživatel klikne na záložku `TemplateBuilder`, čímž se mu zobrazí lišta nástrojů modulu pro editaci šablon (podrobněji v kapitole 6.4.1),
2. na liště uživatel vybere tlačítko `Load` pro načtení metadat vstupních dat,
3. editor šablon dokumentů vyvolá akci pro otevření okna s výběrem metadatového souboru,
4. uživatel vybere soubor s metadaty a potvrdí svůj výběr,
5. pokud je metadatový soubor validní, dojde k jeho načtení,
6. uživatel klikne na tlačítko `ShowTree` na liště nástrojů,
7. v pravé části okna editoru se mu zobrazí náhled metadat ve stromové struktuře (podrobněji v kapitole 6.4.2).

6.5.2 Elementární dosazování dat

V šabloně dokumentu bude možné definovat jazykové konstrukce, do kterých bude *generátor dokumentů z uživatelských šablon* v rámci procesu generování vkládat data ze zdroje vstupních dat. Jazykové konstrukce musí být v souladu se vstupními daty (viz kapitola 6.3.3).

Jazykové konstrukce pro účely realizace *elementárního dosazování dat* jsou *proměnné* a jejich *vlastnosti* (blíže popsané v kapitole 6.2.1). Tzn. například pro proměnnou a její vlastnost:

```
${spis.soud.nazev}
```

bude dosazen název soudu pro daný spis (např. *Okresní soud Plzeň – město*).

Editor šablon dokumentů bude *po validním načtení metadat* umožňovat vložit tyto konstrukce pro elementární dosazování dat. V grafickém rozhraní:

1. uživatel klikne na tlačítko `ShowTree` v liště nástrojů,
2. v pravé části okna editoru se mu zobrazí náhled metadat ve stromové struktuře (podrobněji v kapitole 6.4.2), jednotlivé položky stromové struktury odpovídají *objektům* a *atributům* vstupních dat,
3. uživatel klikne na zvolený atribut ve stromové struktuře,
4. v pravé dolní části pod stromovou strukturou se mu zobrazí vybraný atribut, který může pomocí tlačítka `Add` přidat do šablony dokumentu,
5. do šablony dokumentu se na základě definice vstupních dat přidá validní jazyková konstrukce pro zvolený atribut na místo aktuální polohy kurzoru v šabloně.

6.5.3 Opakovací blok

Generátor dokumentů z uživatelských šablon bude umožňovat zpracování jazykových konstrukcí pro realizaci opakovacího bloku k iterativnímu dosazení všech hodnot atributu typu `kolekce` ze zdroje vstupních dat. Opět je nutné, aby jazykové konstrukce byly v souladu se vstupními daty (viz kapitola 6.3.3).

Konstrukcí pro účely realizace *opakovacího bloku* je direktiva `foreach` (blíže popsána v kapitole 6.2.1). Například pokud uživatel bude požadovat výpis jmen a příjmení všech účastníků soudního řízení, jazyková konstrukce může vypadat následovně:

```
#foreach ($ucastnik in $spis.ucastnici)
${ucastnik.jmeno} ${ucastnik.prijmeni}
#end
```

,

přičemž je nutné, aby v souboru vstupních dat existovala struktura objektů `spis.ucastnici` a objekt `ucastnici` obsahoval *kolekci* objektů stejné struktury reprezentující jednotlivé účastníky soudního řízení (viz ukázka kódu D.1 přílohy D). Odpovídající metadata k takovým vstupním datům nesou informaci, že následovníci objektu `ucastnici` mají být interpretováni jako kolekce – konkrétně se jedná o atribut `isArray`, který je nastavený na hodnotu `true` (viz ukázka kódu E.1 přílohy E).

V rámci *editoru šablon dokumentů* bude po validním načtení odpovídajících metadat uživateli umožněno jednoduše vkládat tuto konstrukci. Na základě metadat *editor šablon* pozná, zda vybraný objekt je typu *kolekce* a pokud ano, umožní uživateli vložení direktivy `foreach`. V grafickém rozhraní:

1. uživatel klikne na tlačítko `ShowTree` v liště nástrojů,
2. v pravé části okna editoru se mu zobrazí náhled metadat ve stromové struktuře (podrobněji v kapitole 6.4.2), jednotlivé položky stromové struktury odpovídají *objektům* a *atributům* vstupních dat,
3. uživatel klikne na zvolený objekt ve stromové struktuře,
4. pokud je objekt typu *kolekce*, zobrazí se uživateli v pravé dolní části pod stromovou strukturou nabídka použití direktivy `foreach` pro opakovací blok,
5. při vybrání této direktivy si uživatel zvolí název řídící proměnné cyklu a tlačítkem `Add` přidá tuto jazykovou konstrukci do šablony dokumentu,
6. do šablony dokumentu se na základě definice vstupních dat přidá validní jazyková konstrukce `foreach` pro zvolený objekt na místo aktuální polohy kurzoru v šabloně (bez těla této direktivy),
7. po přemístění kurzoru uživatelem do těla této direktivy v šabloně dokumentu dojde k přegenerování stromové struktury metadat, aby tato struktura zobrazovala pouze ty jazykové konstrukce, které je možné v direktivě `foreach` použít (viz kapitola 6.6),
8. následující postup je shodný s postupem pro *elementární dosazování dat* (kapitola 6.5.2), tzn. volba a přidání atributu.

6.5.4 Opakovací blok s tabulkou

Obdobně jako u opakovacího bloku bude *generátor dokumentů z uživatelských* šablon umožňovat zpracování jazykových konstrukcí pro realizaci opakovacího bloku s tabulkou. V rámci této konstrukce budou iterativně přidávány řádky do tabulky a následně do nich dosazovány hodnoty atributu typu *kolekce* ze zdroje vstupních dat. V rámci této konstrukce je nutné předem definovat tabulku o dvou řádcích, kde v prvním řádku bude hlavička tabulky a v druhém řádku budou jazykové konstrukce pro dosazení hodnot *kolekce*. Opět je nutné, aby jazykové konstrukce byly v souladu se vstupními daty (viz kapitola 6.3.3).

Jazykovou konstrukcí pro účely realizace *opakovacího bloku s tabulkou* je direktiva `tableforeach` (blíže popsána v kapitole 6.2.1). Zde uvádím stejný příklad jako pro opakovací blok (kapitola 6.5.3) pouze s tím rozdílem, že výsledný dokument bude obsahovat výpis jmen a příjmení všech účastníků

soudního řízení *v tabulce s hlavičkou*. Jazyková konstrukce může vypadat následovně:

```
#tableforeach ($ucastnik in $spis.ucastnici)
| Jméno | Příjmení |
| ${ucastnik.jmeno} | ${ucastnik.prijmeni} |
#end
```

,

přičemž je opět nutné, aby v souboru vstupních dat existovala struktura objektů `spis.ucastnici` a objekt `ucastnici` obsahoval *kolekci* objektů stejné struktury reprezentující jednotlivé účastníky soudního řízení (viz ukázka kódu D.1 přílohy D). Stejně jako u *opakovacího bloku*, i zde odpovídající metadata ke vstupním datům nesou informaci, že následovníci objektu `ucastnici` mají být interpretováni jako kolekce – atribut `isArray`, který je nastavený na hodnotu `true` (viz ukázka kódu E.1 přílohy E).

Pokud budou validně načtena odpovídající metadata a pokud bude vybrán objekt typu *kolekce*, editor šablon dokumentů umožní uživateli vložit direktivu `tableforeach`. V uživatelském rozhraní:

1. uživatel klikne na tlačítko `ShowTree` v liště nástrojů,
2. v pravé části okna editoru se mu zobrazí náhled metadat ve stromové struktuře (podrobněji v kapitole 6.4.2), jednotlivé položky stromové struktury odpovídají *objektům* a *atributům* vstupních dat,
3. uživatel klikne na zvolený objekt ve stromové struktuře,
4. pokud je objekt typu *kolekce*, zobrazí se uživateli v pravé dolní části pod stromovou strukturou nabídka použití direktivy `tableforeach` pro opakovací blok,
5. uživatel vybere tuto direktivu, zvolí název řídící proměnné cyklu a stiskne tlačítko `Add`,
6. *editor šablon dokumentů* vyvolá akci otevření okna pro vytvoření a konfiguraci tabulky (podrobněji v kapitole 6.4.3),
7. uživatel zvolí hlavičku tabulky a atributy, které mají být v tabulce opakovány,
8. pro dokončení stiskne uživatel tlačítko `Add`,
9. do šablony dokumentu se na základě vstupních dat přidá jazyková konstrukce `tableforeach` pro zvolený objekt na místo aktuální polohy kurzoru v šabloně (včetně definované základní tabulky s hlavičkou).

6.5.5 Vložení podšablony

V šabloně dokumentu bude možné definovat jazykovou konstrukci pro vložení podšablony dokumentu definované v externím souboru (formátu Microsoft Word). *Generátor dokumentů z uživatelských šablon* v rámci procesu generování tuto konstrukci zpracuje tak, že obsah podšablony dokumentu vloží do dané šablony dokumentu *na místo jazykové konstrukce pro vložení podšablony*. Jazykové konstrukce šablony musejí být v souladu se vstupními daty (viz kapitola 6.3.3). Pokud vkládaná podšablona obsahuje jazykové konstrukce, musejí být v souladu se vstupními daty nadřazené šablony.

Jazyková konstrukce pro účely realizace *vložení podšablony* je direktiva `include` (blíže popsána v kapitole 6.2.1). Například pokud uživatel bude chtít vložit podšablonu dokumentu definovanou v umístění:

```
C:\docs_gen\data\sub_template.docx
```

,

jazyková konstrukce může vypadat následovně:

```
#include("C:\docs_gen\data\sub_template.docx")
```

.

Do šablony dokumentu bude vložen obsah této podšablony a daná šablona dokumentu bude následně zpracována běžným způsobem. Podšablony dokumentu mohou obsahovat také direktivy `include`, jejich zpracování pak probíhá *rekurzivně*.

Direktiva `include` není volána nad konkrétním objektem či atributem vstupních dat, proto není nutné mít ve chvíli jejího vložení načten metadatový soubor v rámci *editoru šablon dokumentů*. V grafickém uživatelském rozhraní:

1. uživatel klikne na záložku `TemplateBuilder`, čímž se mu zobrazí lišta nástrojů modulu pro editaci šablon (podrobněji v kapitole 6.4.1),
2. na liště uživatel vybere tlačítko `Directives`,
3. uživatel klikne na tlačítko `Include Template`,
4. otevře se okno pro vložení direktivy `include`, kde uživatel klikne na tlačítko `Choose File` (podrobněji v kapitole 6.4.3),
5. editor šablon dokumentů vyvolá akci pro otevření okna s výběrem souboru s podšablonou dokumentu,
6. uživatel vybere soubor podšablony dokumentu a potvrdí svůj výběr,
7. do šablony dokumentu se přidá validní jazyková konstrukce pro vložení podšablony na místo aktuální polohy kurzoru v šabloně.

6.5.6 Definice metod

Generátor dokumentů z uživatelských šablon bude umožňovat zpracování jazykových konstrukcí pro realizaci metod. Konstrukce pro tyto účely jsou *metody*, které jsou blíže popsány v kapitole 6.2.1. Metody lze v závislosti na jejich charakteru volat buď nad proměnnou či její vlastností, nebo je použít samostatně. Dle specifikace této funkcionality zadavatelem (kapitola 4.2.6) budou implementovány metody pro:

- vložení aktuálního data,
- formátování data.

Metoda pro vložení aktuálního data

Metoda pro vložení aktuálního data bude volána samostatně (tzn. ne nad proměnnou či její vlastností). Identifikátor této metody je `currentDate()` a je volána bez parametrů. Výchozí formátování data je `dd.MM.yyyy` (tzn. například 18.04.2022). Do šablony dokumentu, kde je tato metoda volána, bude dosazeno aktuální datum v době generování nového dokumentu. Zápis této metody v šabloně dokumentu může být následovný:

`${currentDate()}`.

V grafickém rozhraní editoru šablon dokumentů:

1. uživatel klikne na záložku `TemplateBuilder`, čímž se mu zobrazí lišta nástrojů modulu pro editaci šablon (podrobněji v kapitole 6.4.1),
2. na liště uživatel vybere tlačítko `Methods`,
3. uživatel klikne na tlačítko `Current Date`,
4. otevře se okno pro vložení metody `currentDate()` (podrobněji v kapitole 6.4.3),
5. uživatel má možnost zvolit formátování data (viz dále) nebo ponechat výchozí formátování data této metody,
6. uživatel potvrdí přidání metody tlačítkem `Add`,
7. do šablony dokumentu se přidá validní jazyková konstrukce pro metodu aktuálního data `currentDate()` na místo aktuální polohy kurzoru v šabloně.

Metoda pro formátování data

Metodu pro formátování data bude možné volat jak nad proměnnou či její vlastností, tak nad metodou pro vložení aktuálního data. Metodu bude možné volat nad proměnnou či její vlastností pouze v případě, že odpovídají atributu typu `datetime` (tato skutečnost je uvedena v metadatech). Identifikátor této metody je `formatDate(<parameter>)`, kde *parameter* je jediným parametrem této metody, obsahuje informaci o požadovaném formátu data a je typu `string`. Metoda zformátuje datum dle zadaného formátu, kterým je například:

- `dd.MM.yyyy`, např. 18.04.2022,
- `MM/dd/yyyy`, např. 04/18/2022,
- `dd. MMMM yyyy`, např. 18. dubna 2022.

Kompletní přehled možných formátů data jsou uvedeny v publikaci [17]. Zápis této metody v šabloně dokumentu může být následovný:

```
${currentDate().formatDate("dd. MMMM yyyy")}
${spis.datumVlozeni.datum.formatDate("dd.MM.yyyy")}
```

.

V grafickém rozhraní editoru šablon dokumentů uživatel:

- v případě volání nad metodou pro vložení aktuálního data:
  1. po otevření okna pro vložení metody `currentDate()` zvolí uživatel formát data prostřednictvím zaškrtávacího políčka a definuje formát data,
  2. tlačítkem `Add` uživatel přidá do šablony dokumentu aktuální datum s metodou pro formátování data,
- v případě volání nad proměnnou či její vlastností:
  1. při přidávání atributu v rámci *elementárního dosazování dat* či obou typů *opakovacích bloků* se uživateli zobrazí zaškrtávací políčko metody `formatDate(...)`, pokud je daný atribut typu `datetime`,
  2. při zaškrtnutí tohoto políčka uživatel definuje formát data,
  3. tlačítkem `Add` uživatel přidá do šablony dokumentu vybraný atribut s metodou pro formátování data.

6.6 Rozšiřující funkce řešení

V této kapitole navrhuji další funkcionality daného softwarového produktu, které nejsou přímo uvedeny ve specifikaci požadavků zadavatele. Jedná se zejména o:

- generování nového dokumentu z aktuálně otevřené šablony v *editoru šablon dokumentů*,
- přegenerování stromové struktury metadat podle umístění kurzoru v šabloně dokumentu v rámci *editoru šablon dokumentů*,
- analýza syntaktických chyb v aktuálně upravované šabloně dokumentu v rámci *editoru šablon dokumentů*.

Editor šablon dokumentů bude umožňovat generování nového dokumentu z aktuálně otevřené (příp. editované) šablony dokumentu. Tato funkcionalita slouží k průběžnému kontrolování generovaného výstupu šablony. Pro vygenerování nového dokumentu je nutné zvolit vstupní data k dosazení do šablony dokumentu a cestu k výstupnímu souboru formátu Microsoft Word. Tato funkcionalita je z pohledu GUI popsána v kapitole 6.4.3.

Další funkcionalitou editoru šablon dokumentů bude přegenerování stromové struktury metadat podle umístění kurzoru v šabloně dokumentu. Editor bude kontrolovat polohu kurzoru v šabloně dokumentu a na základě této polohy vyhodnocovat relevanci zobrazované stromové struktury metadat. Tato funkcionalita je stěžejní pro vkládání jazykových konstrukcí do těla direktivy `foreach`, aby tato struktura zobrazovala pouze ty jazykové konstrukce, které je možné v direktivě `foreach` použít.

Poslední z výše zmíněných rozšiřujících funkcionalit editoru šablon dokumentů je analýza syntaktických chyb v aktuálně upravované šabloně dokumentu. Editor bude periodicky analyzovat aktuálně upravovanou šablonu dokumentu, zda obsahuje syntaktické chyby. V případě, že daná šablona syntaktické chyby obsahuje, zobrazí tyto chyby v panelu syntaktických chyb (viz kapitola 6.4.4).

7 Implementace softwarového řešení

Implementaci softwarového řešení jsem provedl v souladu se specifikací požadavků zadavatele (kapitola 4) a návrhem softwarového řešení (kapitola 6). V analytické části práce jsem zvolil realizaci vlastního řešení (kapitola 5.2), tudíž výstupem této práce je zcela *nový softwarový produkt*, který řeší požadovanou problematiku. Softwarový produkt je rozdělen na dvě hlavní části:

- editor šablon dokumentů,
- generátor dokumentů z uživatelských šablon.

V této kapitole uvádím přehled implementovaných funkcí, popisuji implementaci obou hlavních částí softwarového řešení včetně použitých technologií, rozšíření gramatiky stávajícího řešení, omezení implementovaného řešení a rozdělení zdrojových souborů softwarového řešení.

7.1 Přehled implementovaných funkcí

V rámci této práce jsem implementoval všechny funkce uvedené v kapitolách 6.5 a 6.6. Jedná se o funkce:

- načtení metadat (viz kapitola 6.5.1),
- elementární dosazování dat (viz kapitola 6.5.2),
- opakovací blok (viz kapitola 6.5.3),
- opakovací blok s tabulkou (viz kapitola 6.5.4),
- vložení podšablony (viz kapitola 6.5.5),
- definice metod (viz kapitola 6.5.6),
- rozšiřující funkce (viz kapitola 6.6):
  - generování nového dokumentu z aktuálně otevřené šablony v editoru šablon dokumentů,

– přegenerování stromové struktury metadat podle umístění kurzoru v šabloně dokumentu v rámci editoru šablon dokumentů,
– analýza syntaktických chyb v aktuálně upravované šabloně dokumentu v rámci editoru šablon dokumentů.

7.2 Editor šablon dokumentů

Editor šablon dokumentů slouží k vytvoření a editaci šablony dokumentu pro účely generování nových dokumentů z takovéto šablony. Při použití tohoto editoru se výrazně snižuje riziko nekonzistentně definované šablony a selhání generování nového dokumentu oproti ručnímu definování šablony.

Jako editor šablon dokumentů jsem použil produkt *Microsoft Word*, který jsem rozšířil o nástroje pro vytváření a editaci šablon dokumentů. Toto rozšíření jsem implementoval prostřednictvím souboru nástrojů VSTO (popsán v kapitole 5.2.4) v jazyce `C#` pro platformu `.NET Framework 4.7.2`.

Nástroje pro vytváření a editaci šablon dokumentů jsou dostupné na záložce s názvem `TemplateBuilder` v grafickém uživatelském rozhraní Microsoft Word (viz kapitola 6.4). Funkce editoru šablon dokumentů jsou implementovány dle návrhu uvedeném v kapitolách 6.5 a 6.6.

V této kapitole dále popisuji použití souboru nástrojů VSTO (Visual Studio Tools for Office) a grafické uživatelské rozhraní rozšiřující části Microsoft Word, která obsahuje nástroje pro vytváření a editaci šablon dokumentů.

7.2.1 Visual Studio Tools for Office

Jak jsem již zmínil v kapitole 5.2.4, v této práci používám způsob *VSTO Add-ins*, tzn. přizpůsobení a úpravu na úrovni celé aplikace Microsoft Office (v tomto případě Microsoft Word).

VSTO vytvoří v jazyce `C#` třídu `ThisAddIn`, která je vstupním bodem programu rozšíření. V této třídě vytvoří požadované metody pro inicializaci rozšíření, hlavní z nich je metoda `ThisAddIn_Startup(...)`, která je volána při ihned při spuštění. Další důležitou třídou je `TemplateRibbon`, která definuje lištu nástrojů pro vytvoření a editaci šablony dokumentu (podrobněji v kapitole 6.4.1).

7.2.2 Grafické uživatelské rozhraní

Grafické uživatelské rozhraní jsem implementoval v souladu s návrhem uvedeným v kapitole 6.4.

Implementace je realizována prostřednictvím technologií:

- *VSTO* – lišta nástrojů (viz výše),
- *Windows Forms* – okno se stromovou strukturou metadat a ostatní formulářová okna.

Pro úplnost uvádím v příloze H implementovanou podobu grafického uživatelského rozhraní editoru šablon dokumentů na obrázcích:

- H.1 – lišta nástrojů,
- H.2 a H.3 – stromová struktura metadat,
- H.4, H.5, H.6 a H.7 – formulářová okna,
- H.8 – panel syntaktických chyb.

7.3 Generátor dokumentů z uživatelských šablon

Generátor dokumentů z uživatelských šablon slouží k samotnému generování nového dokumentu z uživatelských šablon a příslušných vstupních dat. Implementace je realizována v jazyce `C#` pro prostředí `.NET Framework 4.7.2` a tato část softwarového produktu je publikována jako knihovna *DLL*. Tato knihovna je spouštěna:

- *z konzolového spouštěče* (`GenTrigger`) – samostatná konzolová aplikace, která volá metody knihovny,
- *z editoru šablon dokumentů* – prostřednictvím tlačítka `Generate` v liště nástrojů (viz kapitola 6.4.1).

Funkce editoru šablon dokumentů jsem implementoval dle návrhu uvedeném v kapitolách 6.5 a 6.6. Hlavní třídou DLL knihovny je `DocumentManager`, která obsahuje metody pro řízení celého postupu generování nového dokumentu.

V této kapitole dále popisuji implementaci parsování šablony dokumentu a generování nového dokumentu a způsob volání generátoru dokumentů (DLL knihovny) z externích aplikací.

7.3.1 Parsování šablony dokumentu

Parsování šablony dokumentu je realizováno pomocí nástroje ANTLR na základě definované gramatiky jazyka šablon (viz kapitola 6.2.2). Šablona dokumentu musí být definována v jazyce šablon (viz kapitola 6.2) ve formátu Microsoft Word. ANTLR vytvoří rozpoznávač a poskytne rozhraní pro vytvoření *parsovacího stromu*.

V této kapitole popisuji vygenerované rozhraní nástrojem ANTLR a následnou implementaci vytvoření parsovacího stromu.

Rozhraní pro vytvoření parsovacího stromu

Rozhraní pro vytvoření parsovacího stromu je generováno nástrojem ANTLR na základě definované gramatiky (viz příloha G). ANTLR vytvoří následující třídy v jazyce `C#`:

- `DocsGenLexer` – třída vytvořená z definice gramatiky části *lexer* (viz ukázka kódu G.1),
- `DocsGenParser` – třída vytvořená z definice gramatiky části *parser* G.2,
- `DocsGenParserListener` – rozhraní pro vytvoření parsovacího stromu – metoda *listener* (vytváření řízeno nástrojem ANTLR),
- `DocsGenParserVisitor` – rozhraní pro vytvoření parsovacího stromu – metoda *visitor* (vytváření řízeno manuálně voláním metod).
- `DocsGenParserBaseListener` – automaticky vytvořená prázdná implementace rozhraní `DocsGenParserListener`, obsahuje *virtuální* metody,
- `DocsGenParserBaseVisitor` – automaticky vytvořená prázdná implementace rozhraní `DocsGenParserVisitor`, obsahuje *virtuální* metody.

Vytvoření parsovacího stromu

Hlavní třídou, která obsahuje metody pro spuštění vytvoření parsovacího stromu a následné generování nového dokumentu, je třída `Generator`. Metody této třídy:

- vytvoří z neformátovaného vstupního textu šablony dokumentu proud (*stream)* znaků,
- následně pomocí třídy `DocsGenLexer` vytvoří proud symbolů (tokenů),

- poté pomocí třídy `DocsGenParser` vytvoří *parsovací kontext*, díky kterému je možné prostřednictvím tříd `DocsGenParserBaseListener` nebo `DocsGenParserBaseVisitor` vytvořit parsovací strom.

V rámci práce vytvářím parsovací strom metodou *visitor*, tzn. při vytváření stromu v rámci jednotlivých tříd programu používám dědičnost vzhledem ke třídě `DocsGenParserBaseVisitor` a manuálně volám metodu `Visit(...)` pro další prvky.

Pro každé pravidlo z části *parser* gramatiky jazyka šablon (viz ukázka kódu G.2), tj. `parse`, `block`, `atom`, `formal`, `variable` atd., jsem vytvořil samostatnou třídu podle vzoru:

`<název pravidla>Visitor`,

tzn. `ParseVisitor`, `BlockVisitor`, `AtomVisitor`, `FormalVisitor`, `VariableVisitor` atd. a tomu odpovídající entitu `Parse`, `Block`, `Atom`, `Formal`, `Variable` atd. pro uložení potřebných dat.

Každá třída s příponou (*suffixem*) `Visitor` implementuje metodu:

`Visit<název pravidla>(<kontext pravidla>)`,

kde kontext pravidla je parsovací kontext pro dané pravidlo a tato metoda je volána z jiné třídy metodou:

`Visit(<kontext pravidla>)`.

Například třída `BlockVisitor` obsahuje metodu `VisitBlock (DocsGenParser.BlockContext context)` s parsovacím kontextem pro pravidlo `block` a tato metoda je volána ze třídy `ParseVisitor` metodou `Visit(context)` s kontextem pro pravidlo `block`. Voláním této metody `Visit(context)` je řízeno sestavování parsovacího stromu. Kořenem (*root*) parsovacího stromu je instance třídy `Parse` a metoda `Visit(context)` pro tuto třídu je volána ze třídy `Generator`.

V příloze I uvádím ukázku kódu I.1, která obsahuje implementaci metody `Generate(...)` třídy `Generator`. Tato metoda obsahuje kroky pro vytvoření parsovacího stromu metodou `visitor` a dále volá metodu `Visit(context)` pro kořen parsovacího stromu `Parse`. V této příloze dále uvádím ukázku kódu I.2 obsahující `ParseVisitor` a obrázek I.1 s UML diagramem tříd zobrazujícím třídy `Visitor`.

Třídy `Visitor` v rámci metod `Visit(<kontext pravidla>)` ukládají *pro každou jazykovou konstrukci*, která je obsažena v šabloně dokumentu, její rozsah (index prvního a posledního znaku dané konstrukce; instance třídy

`ElementRange`) do atributů odpovídajících entit. Tzn. pokud například rozpoznávač nalezne v šabloně dokumentu jazykovou konstrukci odpovídající pravidlu `block`, v rámci metody `VisitBlock(...)` třídy `BlockVisitor` dojde k uložení rozsahu této konstrukce do entity `Block`. První znak šablony dokumentu má index 0. Toto je nutné pro pozdější dosazování vstupních dat v rámci generování nového dokumentu.

7.3.2 Generování nového dokumentu

Po *vytvoření parsovacího stromu* následuje jeho procházení. Kořenem (*root*) tohoto stromu je instance třídy `Parse`. V rámci průchodu parsovacím stromem dochází ke generování nového dokumentu prostřednictvím dosazování vstupních dat.

V této kapitole popisuji implementaci procházení parsovacího stromu, metodu dosazování vstupních dat a běh generátoru dokumentů z uživatelských šablon z pohledu implementace.

Procházení parsovacího stromu

Procházení parsovacího stromu je realizováno prostřednictvím tříd s příponou (*suffixem*) `Builder`, např. `ParseBuilder`, `BlockBuilder`, `AtomBuilder`, `VariableBuilder` atd. Každá z těchto tříd obsahuje metodu `Build()`, která realizuje procházení stromu. Řídící třídou je `DocumentBuilder`, která se prostřednictvím metody `BuildText()` stará o spuštění procházení parsovacího stromu. Tato metoda je volána ze třídy `Generator` metodou `Generate(...)`, která vytváří instanci třídy `DocumentBuilder` s parametry: *kořen parsovacího stromu*, *kořen stromu vstupních dat*.

Všechny třídy s příponou `Builder`, kromě třídy `DocumentBuilder`, dědí od abstraktní třídy `Builder`, která obsahuje společné atributy a metody pro tyto třídy, kterými jsou zejména:

- atribut `dataRoot` – kořen stromu vstupních dat,
- atribut `controllingVariables` – tabulka symbolů, v rámci této práce slouží pouze pro uložení řídících proměnných cyklů *foreach* a *tableforeach*,
- metoda `AddControllingVariables(...)` – slouží k uložení nové proměnné do tabulky symbolů,
- metoda `RemoveLastControllingVarsList(...)` – slouží k odstranění poslední proměnné z tabulky symbolů.

V příloze J uvádím na obrázku J.1 UML diagram těchto tříd.

Jak jsem již zmínil, *tabulka symbolů* slouží v rámci této práce pouze pro uložení řídících proměnných cyklů. K ukládání a odebírání řídících proměnných z tabulky symbolů dochází v rámci třídy `BlockBuilder` v metodě `Build()`.

Dosazování dat

Před samotným dosazováním dat se vytvoří dočasná *kopie šablony dokumentu*, prostřednictvím které bude vygenerován nový dokument. Následně se vytvoří *nový dokument* v zadaném umístění, do kterého se zkopíruje obsah *kopie šablony dokumentu*. Do tohoto dokumentu se dosazují vstupní data – *generování dokumentu*. Generování dokumentu je realizováno tímto způsobem z důvodu přesného zachování formátování všech částí šablony dokumentu. Šablona, kopie šablony i nově vygenerovaný dokument jsou ve formátu Microsoft Word (`doc` nebo `docx`). Práce se soubory formátu Microsoft Word je realizována pomocí knihovny Microsoft Office Interop Word (viz kapitola 5.2.5).

Dosazování dat, včetně například vytvoření tabulky v rámci direktivy *tableforeach*, je implementováno v rámci metod třídy `DocumentManager`. Vždy je nutné znát rozsah dané jazykové konstrukce v šabloně dokumentu (viz kapitola 7.3.1). Metody pro dosazování dat do dokumentu jsou volány v rámci metod `Build()` ve třídách s příponou `Builder` při průchodu parsovacím stromem. Díky skutečnosti, že třídy s příponou `Builder` dědí od abstraktní třídy `Builder`, mají přístup ke stromu vstupních dat pro dosazení.

Běh generátoru dokumentů

Běh generátoru dokumentů z uživatelských šablon z pohledu implementace je následovný:

1. načtení definované šablony dokumentu ve formátu Microsoft Word z určené cesty na souborovém systému,
2. vytvoření dočasné kopie šablony dokumentu ,
3. vytvoření nového dokumentu pro dosazování vstupních dat v určené cestě na souborovém systému,
4. načtení vstupních dat do stromové struktury,

5. inicializace nástroje ANTLR pro vytvoření rozhraní k tvorbě parsovacího stromu,

6. vytvoření parsovacího stromu a uložení rozsahu jednotlivých jazykových konstrukcí (třídy s příponou `Visitor`),

7. procházení parsovacího stromu, vytvoření tabulky symbolů a dosazování dat do nového dokumentu (třídy s příponou `Builder`),

8. uložení finální verze vygenerovaného nového dokumentu.

7.3.3 Způsob volání generátoru dokumentů z externích aplikací

Jelikož je generátor dokumentů z uživatelských šablon realizován jako DLL knihovna, lze jej volat z externích aplikací. Předpokladem je, že tyto aplikace jsou implementovány v jazyce `C#` a kompatibilní s `.NET Framework 4.7.2`.

Jak jsem již zmínil v úvodu této kapitoly, hlavní třídou generátoru dokumentů (DLL knihovny) je `DocumentManager`. Tato třída poskytuje metody pro řízení celého postupu generování nového dokumentu z definované šablony dokumentu a vstupních dat. Třída je realizována podle návrhového vzoru *Jedináček* (*Singleton*). Pořadí volání jednotlivých metod nad instancí této třídy za účelem vygenerování nového dokumentu je následovné:

1. `LoadTempTemplateFromFile(templateFilePath)` – tato metoda načte šablonu dokumentu (ta je definována cestou `templateFilePath`), vytvoří kopii této šablony a následně tuto kopii šablony načte,

2. `Preprocess()` – tato metoda předzpracuje načtenou kopii šablony dokumentu,

3. `CreateDocument(documentFilePath)` – tato metoda vytvoří nový dokument pro dosazování vstupních dat v umístění předaném cestou `documentFilePath`,

4. `GenerateDocument(dataFilePath)` – tato metoda zajišťuje generování obsahu nového dokumentu (dosazováním dat) na základě souboru vstupních dat definovaném cestou `dataFilePath`,

5. `Close()` – tato metoda zavře všechny používané soubory,

6. `CloseWordProcesses()` – tato metoda ukončí všechny procesy Microsoft Word.

Příklad způsobu volání generátoru dokumentů z externích aplikací je uveden v ukázce kódu K.1 přílohy K.

7.4 Rozšíření gramatiky jazyka šablon

Gramatiku jazyka šablon jsem implementoval dle návrhu uvedeného v kapitole 6.2.2 prostřednictvím nástroje `ANTLR 4`. Soubory s definicí gramatiky jsou:

- `DocsGenLexer.g4` – obsahuje definici *lexeru*,
- `DocsGenParser.g4` – obsahuje definici *parseru*.

Lexer, resp. *parser* gramatiky jazyka šablon je uveden v ukázce kódu G.1, resp. G.2 přílohy G.

7.4.1 Postup rozšíření gramatiky

Při rozšiřování aktuální gramatiky jazyka šablon a jejím následném parsování je postup následovný:

1. přidání lexikálního pravidla či pravidel do lexeru (soubor `DocsGenLexer.g4`),
2. přidání parsovacího pravidla či pravidel do parseru (soubor `DocsGenParser.g4`),
3. přegenerování rozhraní pro vytvoření parsovacího stromu nástrojem ANTLR,
4. přidání `Visitor` třídy pro nové pravidlo parseru (pokud je nutné – při úpravě pouze části *lexer* se tato třída nepřidává),
5. přidání entity pro nové pravidlo parseru (pokud je nutné – záleží na charakteru nového pravidla),
6. přidání `Builder` třídy pro nové pravidlo parseru (pokud je nutné – záleží na charakteru nového pravidla),
7. úprava stávajících tříd `Visitor` a `Builder` ostatních pravidel, aby došlo k validnímu vytvoření a procházení parsovacího stromu a správnému vygenerování nového dokumentu.

7.4.2 Příklad rozšíření gramatiky

V tomto příkladě ilustruji rozšíření gramatiky jazyka šablon a vytvoření či úpravu příslušných tříd pro tvorbu a procházení parsovacího stromu o direktivu `if` (v tomto příkladě bez konstrukcí `else` či `elseif`). Tato jazyková konstrukce slouží k zadání a vyhodnocení *podmínkového výrazu*. Pokud bude podmínka vyhodnocena jako splněná (hodnota `true`), dojde ke zpracování těla této direktivy, v opačném případě ke zpracování nedojde. Direktiva bude definována následovně:

```
# [ { ] if [ } ] ( expression ) block
# [ { ] end [ } ]
```
,

kde:

- ***expression*** je podmínkový výraz k vyhodnocení,
- ***block*** je tělo direktivy, tzn. blok textu či kódu v jazyce šablon.

Příklad rozšíření je pouze ilustrativní a *nezahrnuje* rozšíření o související gramatická pravidla potřebná k fungování této direktivy (např. pravidlo `expression` v parseru gramatiky). Tato související pravidla je nutno definovat obdobným způsobem jako ilustrovaná pravidla.

Rozšíření lexeru

Nejprve je tedy nutné rozšířit *lexer* gramatiky (soubor `DocsGenLexer.g4`) o tuto novou jazykovou konstrukci. Rozšíření může vypadat následovně:

- do sekce *Directive mode* (`mode DIR_;`) se přidá pravidlo uvedené v ukázce kódu 7.1,
- do pole symbolů (`tokens`) se přidá symbol `IF` – toto pole po přidání tohoto symbolu je uvedeno v ukázce kódu 7.2.

Ukázka kódu 7.1: Přidání direktivy `if` – lexer;
Zdroj: vlastní zpracování

```
1 DIR_IF
2  : ( 'if' | '{if}' ) SPACES? '(' -> type(IF), popMode, pushMode(CODE_)
3  ;
```

Ukázka kódu 7.2: Přidání symbolu `IF` – lexer;
Zdroj: vlastní zpracování

```
1 tokens {
2   OPAR, CPAR, OBRACK, CBRACK, OBRACE, CBRACE, STRING, INTEGER, ID, REFERENCE, DOT,
        FOREACH, TABLEFOREACH, INCLUDE, END, K_IN, IF
3 }
```

Rozšíření parseru

Poté je nutné rozšířit *parser* gramatiky (soubor `DocsGenParser.g4`) o tuto novou jazykovou konstrukci. Rozšíření může vypadat následovně:

- přidá se pravidlo `if_directive` uvedené v ukázce kódu 7.3,
- pravidlo `directive` se rozšíří o pravidlo `if_directive` – pravidlo `directive` po rozšíření je uvedeno v ukázce kódu 7.4.

Ukázka kódu 7.3: Přidání direktivy `if` – parser;
Zdroj: vlastní zpracování

```
1 if_directive
2  : IF expression CPAR block end
3  ;
```

Ukázka kódu 7.4: Rozšíření pravidla `directive` – parser;
Zdroj: vlastní zpracování

```
1 directive
2  : foreach_directive
3  | tableforeach_directive
4  | include_directive
5  | if_directive
6  ;
```

Vytvoření a úprava příslušných tříd

Po přidání nových pravidel do lexeru a parseru gramatiky jazyka šablon je nutné přegenerovat rozhraní pro vytvoření parsovacího stromu nástrojem ANTLR a následně vytvořit příslušné třídy pro direktivu `if`. Třídy mohou být následovné:

- `IfDirective` – entita pro direktivu `if`,
- `IfDirectiveVisitor` – `Visitor` třída pro vytvoření parsovacího stromu, dědí od třídy `DocsGenParserBaseVisitor` a obsahuje implementaci metody `VisitIf_directive(DocsGenParser.If_directiveContext context)` s parsovacím kontextem pro pravidlo `if_directive`,
- `IfDirectiveBuilder` – `Builder` třída pro procházení parsovacího stromu a generování nového dokumentu, dědí od abstraktní třídy `Builder` a obsahuje metodu `Build()` pro průchod parsovacím stromem a dosazování dat.

Poté je nutné upravit stávající třídy `Visitor` a `Builder` ostatních pravidel. V tomto případě se jedná o třídu `AtomVisitor`, do které je nutné přidat podmínku pro direktivu `if`. Podmínku pro direktivu `if` je také nutné přidat do třídy `DirectiveBuilder`.

7.4.3 Příklad rozšíření uživatelského rozhraní

Tento příklad navazuje na kapitolu 7.4.2, jedná se tedy o rozšíření uživatelského rozhraní editoru šablon dokumentů o direktivu `if`. Tato direktiva bude přidávána do lišty nástrojů (viz kapitola 6.4.1), konkrétně do galerie `Directives` v sekci `Tools`.

Lištu nástrojů definuje třída `TemplateRibbon`. V té je definován atribut `directivesGallery` typu `RibbonGallery`, který odpovídá galerii `Directives`. Pro přidání nové položky do galerie je nutné vytvořit atribut typu `RibbonDropDownItem`, např. s názvem `ribbonDropDownItemIf`, přiřadit mu minimálně vlastnost `Label` (např. `If Directive`) a tento atribut přidat do galerie následovně:

```
directivesGallery.Items.Add(ribbonDropDownItemIf)
```
.

Následně je nutné přidat akci pro kliknutí na tuto položku. To se realizuje v metodě `directivesGallery_Click(...)` třídy `TemplateRibbon`. V této metodě je již realizována akce pro direktivu `include`, stačí tedy přidat další podmínku i pro direktivu `if`. Pro vložení jazykové konstrukce je připravena metoda `InsertMark(string markToInsert)` ve třídě `WordManager`, které obsahuje parametr `markToInsert` odpovídající dané jazykové konstrukci.

7.5 Omezení implementovaného řešení

Implementované softwarové řešení má následující omezení:

- při vložení plovoucího *textového pole* do šablony dokumentu nedojde k validnímu vygenerování finálního dokumentu,
- generátor dokumentů z uživatelských šablon nezpracovává *záhlaví a zápatí* šablony dokumentu.

Zadavatel je s touto skutečností obeznámen a tato omezení nebrání zadavateli v použití tohoto softwarového produktu vzhledem k charakteru jeho potřeb. Vyřešení těchto omezení bude probíhat v rámci budoucího vývoje softwarového produktu.

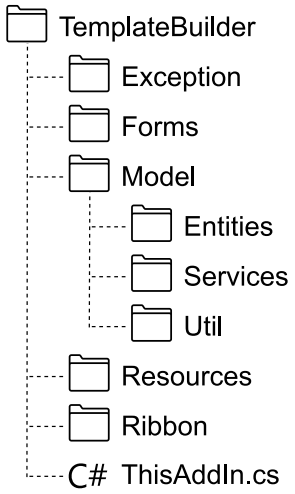
7.6 Rozdělení zdrojových souborů softwarového řešení

Zdrojové soubory softwarového řešení jsou rozděleny do tří projektů:

- `TemplateBuilder` – projekt obsahující rozšíření produktu Microsoft Word pro vytváření a editaci šablon dokumentů (*editor šablon dokumentů*),
- `GenLib` – projekt obsahující DLL knihovnu pro generování nového dokumentu z uživatelské šablony a vstupních dat (*generátor dokumentů z uživatelských šablon*),
- `GenTrigger` – projekt obsahující konzolový spouštěč pro spouštění knihovny `GenLib` z příkazové řádky.

7.6.1 Projekt `TemplateBuilder`

Rozdělení hlavní zdrojových souborů projektu `TemplateBuilder` je ilustrováno na obrázku 7.1.



Obrázek 7.1: Struktura projektu `TemplateBuilder`;
Zdroj: vlastní zpracování

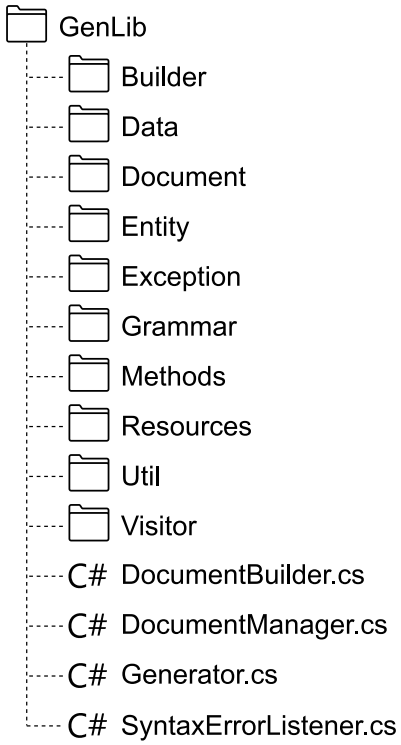
Význam hlavních adresářů a tříd:

- adresář `Exception` – obsahuje třídy s definovanými výjimkami (*exceptions*), které mohou nastat při vykonávaní programu,
- adresář `Forms` – obsahuje *Windows Forms* formuláře použité v grafickém uživatelském rozhraní v rámci editoru šablon dokumentů,

- adresář `Model` – obsahuje modelové třídy, které jsou rozděleny do adresářů:
  - `Entities` – obsahuje entity použité v programovém kódu,
  - `Services` – obsahuje třídy s logikou programu,
  - `Util` – obsahuje pomocné třídy,
- adresář `Resources` – obsahuje zdroje (*resources*) pro potřeby vykonávaní programu,
- adresář `Ribbon` – obsahuje třídy pro definici nástrojové lišty v grafickém uživatelském rozhraní,
- třída `ThisAddIn.cs` – hlavní třída programu, inicializuje *rozšíření produktu Microsoft Word*.

7.6.2 Projekt `GenLib`

Rozdělení hlavní zdrojových souborů projektu `GenLib` je ilustrováno na obrázku 7.2.



Obrázek 7.2: Struktura projektu `GenLib`;
Zdroj: vlastní zpracování

Význam hlavních adresářů a tříd:

- adresář `Builder` – obsahuje třídy s příponou `Builder` pro procházení parsovacího stromu (viz kapitola 7.3.2),
- adresář `Data` – obsahuje třídy pro načtení vstupních dat a entity, které odpovídají struktuře vstupních dat,
- adresář `Document` – obsahuje třídy pro práci s dokumentem ve formátu `doc` nebo `docx`,
- adresář `Entity` – obsahuje entity, které jsou součástí parsovacího stromu,
- adresář `Exception` – obsahuje třídy s definovanými výjimkami (*exceptions*), které mohou nastat při vykonávaní programu,
- adresář `Grammar` – obsahuje soubory, které definují gramatiku (`DocsGenLexer.g4` a `DocsGenParser.g4`) a třídy vygenerované nástrojem ANTLR,
- adresář `Methods` – obsahuje třídy pro realizaci *metod v šabloně dokumentu*,
- adresář `Resources` – obsahuje zdroje (*resources*) pro potřeby vykonávání programu,
- adresář `Util` – obsahuje pomocné třídy,
- adresář `Visitor` – obsahuje třídy s příponou `Visitor` pro vytvoření parsovacího stromu (viz kapitola 7.3.1),
- třída `DocumentBuilder.cs` – zajišťuje spuštění procházení parsovacího stromu,
- třída `DocumentManager.cs` – hlavní třída knihovny, obsahuje metody pro řízení celého postupu generování nového dokumentu (tzn. spuštění načtení šablony dokumentu, načtení vstupních dat, generování a uložení vygenerovaného dokumentu),
- třída `Generator.cs` – zajišťuje spuštění generování nového dokumentu ve formátu `doc` nebo `docx`,
- třída `SyntaxErrorListener.cs` – slouží jako posluchač (*listener*) pro výjimky (*exceptions*), které jsou vyvolány v průběhu parsování šablony dokumentu.

7.6.3 Projekt `GenTrigger`

Projekt `GenTrigger` obsahuje jednu třídu: `Program.cs`. Tato třída načte z příkazové řádky vstupní parametry: cestu k šabloně dokumentu, cestu k datovému souboru a cestu, na kterou vygenerovat nový dokument. Dále spouští metody knihovny `GenLib` pro vygenerování dokumentu z šablony a vstupních dat.

8 Testování softwarového řešení

Implementované softwarové řešení jsem testoval:

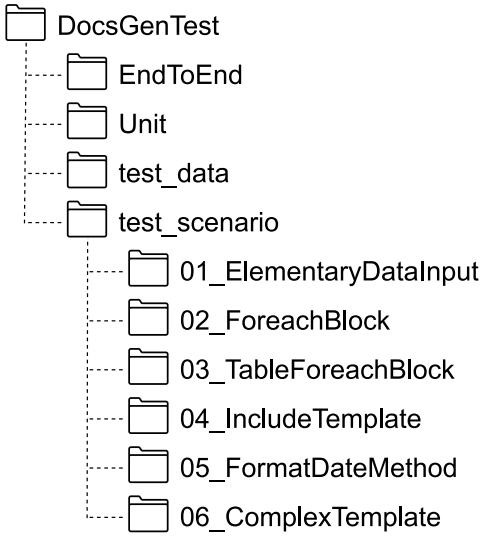
- *jednotkovými testy* – automatické,
- *end-to-end testy* – automatické,
- *výkonnostními testy* (performance testy) – manuální.

Automatické testy (tj. jednotkové testy a end-to-end testy) jsem realizoval prostřednictvím nástroje *MSTest* [23]. Zdrojové soubory těchto testů a testovací scénáře jsou součástí projektu `DocsGenTest`.

V této kapitole nejprve uvádím rozdělení zdrojových souborů automatických testů a poté popisuji průběh jednotlivých částí testování implementovaného softwarového řešení.

8.1 Rozdělení zdrojových souborů automatických testů

Rozdělení zdrojových souborů automatických testů ilustruje obrázek 8.1.



Obrázek 8.1: Struktura projektu `DocsGenTest`;
Zdroj: vlastní zpracování

Význam hlavních adresářů:

- adresář `EndToEnd` – obsahuje testovací třídy pro realizaci end-to-end testů,
- adresář `Unit` – obsahuje testovací třídy pro realizaci jednotkových testů,
- adresář `test_data` – obsahuje testovací data,
- adresář `test_scenario` – obsahuje testovací scénáře pro end-to-end testy.

8.2 Jednotkové testy

První částí testování softwarového řešení jsou *jednotkové testy* (v adresáři `Unit` projektu `DocsGenTest`), které jsem realizoval pro hlavní části softwarového řešení. Konkrétně se jedná o:

- `GeneratorTest.cs` – testuje metodu `Generate(...)` třídy `Generator` pro jazykové konstrukce jazyka šablon,
- `DataLoaderTest.cs` – testuje metodu `LoadDataJson(...)` třídy `DataLoader` pro validní a nevalidní soubor vstupních dat (soubory dat uloženy v adresáři `test_data`),
- `MetadataLoaderTest.cs` – testuje metodu `LoadMetadataJson(...)` třídy `MetadataLoader` pro validní a nevalidní soubor metadat (soubory metadat uloženy v adresáři `test_data`),
- `MethodsLibTest.cs` – testuje metody `MethodFormatDate(...)` a `MethodCurrentDate(...)` třídy `MethodsLib` pro validní a nevalidní vstupy.

Testovací metody jsou spouštěny s různými vstupními daty. V testech se porovnává očekávaná hodnota s reálnou hodnotou, případně se očekává vyvolání konkrétní výjimky při zpracování.

Testování jednotkovými testy bylo finálně úspěšné, všechny jednotkové testy vyšly s pozitivním výsledkem.

8.3 End-to-end testy

Druhou částí testování softwarového řešení jsou *end-to-end* testy (v adresáři `EndToEnd` projektu `DocsGenTest`). Tyto testy pokrývají celý průběh generování nového dokumentu z definované šablony a vstupních dat.

Pro každý testovací scénář je nadefinována šablona dokumentu, odpovídající vstupní data ve formátu JSON a *očekávaný finální dokument*. V rámci testu proběhne vygenerování *nového dokumentu* z definované šablony a vstupních dat. Vygenerovaný dokument se poté porovná s *očekávaným finálním dokumentem*, případně se očekává vyvolání konkrétní výjimky při zpracování.

Testovací data jsou uložena v adresáři `test_data` a nadefinované šablony s odpovídajícími očekávanými dokumenty jsou uloženy v adresáři `test_scenario`. Testovací třídy pro realizaci end-to-end testů jsou:

- `ElementaryDataInputTest.cs` – testuje elementární dosazování dat; šablony a očekávané dokumenty jsou umístěny v adresáři: `test_scenario\01_ElementaryDataInput`,
- `ForeachBlockTest.cs` – testuje opakovací blok; šablony a očekávané dokumenty jsou umístěny v adresáři: `test_scenario\02_ForeachBlock`,
- `TableForeachBlockTest.cs` – testuje opakovací blok s tabulkou; šablony a očekávané dokumenty jsou umístěny v adresáři: `test_scenario\03_TableForeachBlock`,
- `IncludeTemplateTest.cs` – testuje vložení podšablony dokumentu; šablony, podšablony a očekávané dokumenty jsou umístěny v adresáři: `test_scenario\04_IncludeTemplate`,
- `FormatDateMethodTest.cs` – testuje metodu pro formátování data; šablony a očekávané dokumenty jsou umístěny v adresáři: `test_scenario\05_FormatDateMethod`,
- `ComplexTemplateTest.cs` – testuje komplexní šablonu dokumentu se všemi možnými jazykovými konstrukcemi; šablony, podšablony a očekávané dokumenty jsou umístěny v adresáři: `test_scenario\06_ComplexTemplate`.

V příloze L uvádím příklad souborů pro testování komplexní šablony dokumentu (třída `ComplexTemplateTest.cs`):

- *vstupní data* – ukázka kódu L.1,
- *šablona dokumentu* – obrázek L.1,
- *podšablona dokumentu* – obrázek L.2,
- *očekávaný finální dokument* – obrázek L.3,
- *vygenerovaný finální dokument* – obrázek L.4.

Testování end-to-end testy bylo finálně úspěšné, všechny tyto testy vyšly s pozitivním výsledkem.

8.4 Výkonnostní testy

Třetí částí testování jsou *výkonnostní* (performance) testy. Dle výkonnostních požadavků zadavatele (kapitola 4.4.1) je maximální deklarovaná doba generování jednoho dokumentu z uživatelské šablony stanovena na 60 sekund na 25 stránek šablony dokumentu (při dodržení minimálních požadavků na provozní prostředí – viz kapitola 4.1.6).

Výkonnostní testování jsem prováděl na běžném PC, jehož parametry jsou uvedeny v tabulce 8.1. Testování jsem realizoval nad daty, šablonami a podšablonami dokumentů, které jsou shodné jako pro end-to-end testy (uvedeny v příloze L).

Testoval jsem *dobu generování finálního dokumentu* v závislosti na *počtu stránek šablony dokumentu* (počet stránek podšablony dokumentu byl konstantní – viz obrázek L.2 přílohy L). V případě vícestránkových šablon dokumentů jsem použil původní jednostránkovou šablonu (viz obrázek L.1 přílohy L) a zduplikoval jsem obsah této stránky na požadovaný počet stránek. Výsledky výkonnostního testování jsou uvedeny v tabulce 8.2.

Tabulka 8.1: Parametry testovacího PC; Zdroj: vlastní zpracování

Typ PC	Lenovo ThinkPad T450 20BV001CMC
Operační systém	Windows 10 Home (64-bit.)
Procesor (CPU)	Intel Core i5-5200U (2 GHz)
Operační paměť (RAM)	4 GB
Microsoft Office	Microsoft Word 2019
Ostatní software	.NET Framework Runtime 4.7.2

Tabulka 8.2: Výsledky výkonnostního testování; Zdroj: vlastní zpracování

Počet stránek šablony	**Doba generování (s)**
1	7, 51
10	11, 82
25	**19,13**
50	31, 60
100	62, 13

Doba generování jednoho dokumentu ze šablony v rozsahu 25 stran byla 19, 13 sekund.

Testování výkonnostními testy bylo finálně úspěšné, maximální deklarovaná doba generování nebyla překročena.

9 Závěr

Cílem této práce bylo vytvořit nový softwarový produkt či použít a upravit již existující softwarový produkt třetí strany, který umožňuje vytvářet a editovat šablony dokumentů a následně generovat dokumenty nad připravenou sadou dat do formátu Microsoft Word (doc či docx).

V první části práce jsem analyzoval obecnou problematiku automatizace dokumentů, díky čemuž jsem získal širší rozhled v této oblasti a informace o výhodách automatického generování dokumentů. Dále jsem se zabýval analýzou stávajícího řešení zadavatele (společnost CCA Group a.s.), při které jsem vyhodnotil jeho nedostatky. Na základě analýzy stávajícího řešení jsme společně se zadavatelem specifikovali požadavky na nové řešení, které se vyvaruje nedostatků řešení stávajícího, bude rozšiřitelné a snadno udržovatelné.

V další části práce jsem analyzoval existující produkty řešící danou problematiku a vyhodnotil jsem, že žádný z nich není vhodný pro potřeby zadavatele. Z tohoto důvodu jsem vybral realizaci vlastního softwarového řešení, tzn. implementaci nového softwarového produktu.

V práci jsem dále navrhl nové softwarové řešení pro danou problematiku. Konkrétně se jedná o návrh: řešení definice šablon dokumentů včetně pravidel a způsobů tvorby těchto šablon, struktury vstupních dat, grafického uživatelského rozhraní editoru šablon a všech požadovaných funkcí specifikovaných zadavatelem.

Dle návrhu řešení jsem implementoval softwarový produkt, který se skládá z editoru šablon a generátoru dokumentů. V práci jsem popsal implementaci obou těchto částí softwarového produktu, postup rozšíření produktu o nové funkce a omezení implementovaného řešení. Následně jsem provedl testování výsledného řešení. V práci jsem popsal průběh testování, které se skládalo z automatických jednotkových testů, automatických end-to-end testů a manuálních výkonnostních testů. Testování bylo úspěšné, všechny výsledky testů byly vyhodnoceny pozitivně.

Cíl práce byl splněn. Výstupem je *nový softwarový produkt*, který umožňuje vytvářet a editovat šablony dokumentů a následně generovat dokumenty nad vstupními daty do formátu Microsoft Word. *Softwarový produkt splňuje požadavky zadavatele* a je připraven k nasazení do testovacího prostředí zadavatele.

Do budoucna je plánováno nasazení do provozního prostředí a nahrazení stávajícího řešení zadavatele. V rámci dalšího vývoje budou vyřešena omezení tohoto řešení.

Seznam zkratek

.NET – neboli „dotnet“, představuje soubor technologií v softwarových produktech, které tvoří platformu.

ANTLR (ANother Tool for Language Recognition) – neboli jiný nástroj pro rozpoznávání jazyka, je nástroj pro lexikální a syntaktickou analýzu, který umožňuje tvorbu parserů strukturovaných textů či binárních souborů.

API (Application Programming Interface) – neboli aplikační programové rozhraní, je soubor funkcí, procedur, tříd či protokolů aplikace, které může programátor využívat.

BI (Business Intelligence) – je manažerský informační systém.

CPU (Central processing unit) – neboli centrální procesorová jednotka, je označení základní elektronické součásti v počítači, která umí vykonávat strojové instrukce, ze kterých je tvořen počítačový program a obsluhovat jeho vstupy a výstupy.

CRM (Customer Relationship Management) – je řízení vztahu se zákazníkem.

DLL (Dynamic-link library) – neboli dynamicky linkovaná (připojovaná) knihovna, je implementace konceptu sdílených knihoven společnosti Microsoft pro operační systém Microsoft Windows.

DPD (Datová pole dokumentu) – je interní zkratka společnosti CCA Group a.s., jedná se o pole ve vzoru dokumentu, která odpovídají databázovým tabulkám a pohledům připojené databáze.

EBNF (Extended Backus–Naur Form) – neboli rozvinutá Backusova–Naurova forma, je rodina notací využívaných pro zápis bezkontextových gramatik.

ERP (Enterprise Resource Planning) – neboli plánování podnikových zdrojů, je kategorie podnikových informačních systémů, jimiž podnik za pomoci počítače řídí a integruje všechny nebo většinu oblastí své činnosti (např. plánování, zásoby, nákup, prodej, marketing, finance, personalistika atd.).

GUI (Graphic User Interface) – neboli grafické uživatelské rozhraní, je uživatelské rozhraní, které uživateli poskytuje ovládání zařízení pomocí grafických prvků na obrazovce.

JSON (JavaScript Object Notation) – je formát zápisu dat nezávislý na počítačové platformě určený pro přenos dat.

JSP (JavaServer Pages) – je webová technologie založená na jazyce Java, která umožňuje dynamicky vkládat obsah do HTML stránek.

MVC (Model-view-controller) – je softwarová architektura, která rozděluje datový model aplikace, uživatelské rozhraní a řídicí logiku do tří nezávislých komponent.

PC (Personal computer) – neboli osobní počítač, je víceúčelový mikropočítač, který je díky své velikosti, možnostem a ceně vhodný pro individuální použití.

PDF (Portable Document Format) – neboli přenosný formát dokumentů, je formát souborů pro výměnu elektronických dokumentů.

PHP (Hypertext Preprocessor) – neboli hypertextový preprocesor, je skriptovací programovací jazyk určený především pro programování dynamických webových stránek a webových aplikací.

PUB – je formát dokumentů produktu Microsoft Publisher.

RAM (Random Access Memory) – je operační paměť počítače, která umožňuje číst nebo zapisovat data do libovolného místa paměti.

REST (Representational State Transfer) – je architektura rozhraní navržená pro distribuované systémy.

SQL (Structured Query Language) – je standardizovaný strukturovaný dotazovací jazyk používaný v relačních databázových systémech.

UML (Unified Modeling Language) – je grafický jazyk pro vizualizaci, specifikaci, navrhování a dokumentaci programových systémů.

VBA (Visual Basic for Applications) – je programovací jazyk od společnosti Microsoft, který je používán v rámci balíčku Microsoft Office.

VSTO (Visual Studio Tools for Office) – je soubor vývojářských nástrojů v rámci produktu Microsoft Visual Studio, prostřednictvím kterých lze spouštět .NET Framework aplikace v prostředí produktů řady Microsoft Office.

VTL (Velocity Template Language) – je jazyk pro definici šablon v rámci Velocity Engine.

XML (Extensible Markup Language) – je obecný značkovací jazyk, který se používá pro přenos a serializaci dat.

ZIP – je souborový formát pro kompresi a archivaci dat.

Literatura

[1] *About The ANTLR Parser Generator* [online]. Parr, T.; Harwell, S., 2021. [cit. 2022/03/22]. Dostupné z: `https://www.antlr.org/about.html`.

[2] *ANTLR 4 Documentation* [online]. Parr, T.; Hamilton, B., 2019. [cit. 2022/03/23]. Dostupné z: `https://github.com/antlr/antlr4/blob/master/doc/index.md`.

[3] *ANTLR 4 License* [online]. Parr, T.; Harwell, S., 2012. [cit. 2022/03/22]. Dostupné z: `https://www.antlr.org/license.html`.

[4] *ANTLR v4 Runtime API* [online]. Parr, T.; Harwell, S., 2022. [cit. 2022/03/22]. Dostupné z: `https://www.antlr.org/api/`.

[5] *Apache Velocity Project* [online]. The Apache Software Foundation, 2020. [cit. 2022/03/02]. VTL Reference. Dostupné z: `https://velocity.apache.org/engine/devel/vtl-reference.html`.

[6] *Blockly in Stimulsoft products* [online]. Stimulsoft OÜ, 2022. [cit. 2022/02/26]. Dostupné z: `https://www.stimulsoft.com/images/articles/2021/using-blockly-in-the-stimulsoft-report-designer/designer-en.png`.

[7] Dilmegani, C. *Document Automation* [online]. AIMultiple, 2020. [cit. 2022/03/06]. The Ultimate Guide to Document Automation in 2022. Dostupné z: `https://research.aimultiple.com/document-automation/`.

[8] *Document automation software: how to get started* [online]. Juro.com, 2022. [cit. 2022/03/06]. What is document automation? Dostupné z: `https://juro.com/learn/document-automation`.

[9] *Figma* [online]. Figma, Inc., 2022. [cit. 2022/05/15]. Dostupné z: `https://www.figma.com/`.

[10] *Formstack* [online]. Network Solutions, LLC, 2019. [cit. 2022/02/27]. Dostupné z: `https://techpoint.org/tech-directory/formstack`.

[11] *Formstack Documents* [online]. Formstack, LLC, 2022. [cit. 2022/02/27]. Documents Plans and Pricing. Dostupné z: `https://www.formstack.com/pricing/documents`.

[12] *Formstack Documents* [online]. Formstack, LLC, 2022. [cit. 2022/02/27]. Take control of your document generation. Dostupné z: `https://www.formstack.com/documents/features`.

[13] *Formstack Documents* [online]. Formstack, LLC, 2022. [cit. 2022/05/14]. Introduction. Dostupné z: https://www.formstack.com/products/documents.

[14] *Microsoft Office Interop Word* [online]. Microsoft Corporation, 2022. [cit. 2022/03/23]. Definition. Dostupné z: https://docs.microsoft.com/en-us/dotnet/api/microsoft.office.interop.word._application.

[15] *Office solutions development overview (VSTO)* [online]. Microsoft Corporation, 2021. [cit. 2022/03/23]. Choose an Office project type. Dostupné z: https://docs.microsoft.com/en-us/visualstudio/vsto/office-solutions-development-overview-vsto?view=vs-2022.

[16] Parr, T. *The Definitive ANTLR 4 Reference, 1. vyd.* The Pragmatic Bookshelf, 2013. ISBN 978-1-93435-699-9.

[17] *Standardní řetězce formátu data a času* [online]. Microsoft Corporation, 2022. [cit. 2022/04/18]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/standard/base-types/standard-date-and-time-format-strings.

[18] *Stimulsoft BI Designer* [online]. Stimulsoft OÜ, 2022. [cit. 2022/02/25]. Dostupné z: https://www.stimulsoft.com/en/products/designer.

[19] *Stimulsoft BI Designer - licenses* [online]. Stimulsoft OÜ, 2022. [cit. 2022/02/26]. Dostupné z: https://www.stimulsoft.com/en/online-store#desktop/designer.

[20] *Templates Docs* [online]. Formstack, LLC, 2022. [cit. 2022/03/07]. Vendor Contract Document Template. Dostupné z: https://www.formstack.com/templates/vendor-contract-documents.

[21] *The Apache Velocity Project* [online]. The Apache Software Foundation, 2020. [cit. 2022/03/02]. What is Velocity? Dostupné z: https://velocity.apache.org/.

[22] *The Apache Velocity Project* [online]. The Apache Software Foundation, 2020. [cit. 2022/03/02]. Apache Velocity Projects. Dostupné z: https://velocity.apache.org/.

[23] *Unit testing C# with MSTest and .NET* [online]. Microsoft Corporation, 2022. [cit. 2022/05/14]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-mstest.

[24] *What is a VSTO add-in?* [online]. QuickAdviser, 2021. [cit. 2022/03/23]. Dostupné z: https://quick-adviser.com/what-is-a-vsto-add-in/.

[25] *Windward Core* [online]. Windward Studios, LLC, 2021. [cit. 2022/02/27]. Dostupné z: `https://www.windwardstudios.com/solution/windward-core-more-info`.

[26] *Windward Core Pricing* [online]. Windward Studios, LLC, 2022. [cit. 2022/02/27]. Dostupné z: `https://www.windwardstudios.com/price/core`.

[27] *Windward Core Software* [online]. Software Advice, Inc., 2015. [cit. 2022/02/27]. About Windward Core. Dostupné z: `https://www.softwareadvice.com/bi/windward-reporting-profile/`.

Seznam příloh

A Jazyk VTL

Ukázka kódu A.1: Příklady direktivy `set`; Zdroj: [5]

```
#  set( $a =   "  Velocity  " )
#  set( $monkey = $bill )
#  set( $monkey.Friend =   '  monica  ' )
#  set( $monkey.Blame = $whitehouse.Leak )
#  set( $monkey.Plan = $spindoctor.weave($web) )
#  set( $monkey.Number = 123 )
#  set( $monkey.Numbers = [1..3] )
#  set( $monkey.Say = [  "  Not  ", $my,   "  fault  "] )
#  set( $value = $foo + 1 )
#  set( $value = $bar - 1 )
#  set( $value = $foo * $bar )
#  set( $value = $foo / $bar )
#  set( $value = $foo % $bar )
```

Tabulka A.1: Operátory jazyka VTL; Zdroj: [5]

Název operátoru	**Symbol**	**Alternativní symbol**
Rovnost čísel	`==`	`eq`
Rovnost řetězců	`==`	`eq`
Ekvivalence objektů	`==`	`eq`
Nerovnost	`!=`	`ne`
Větší než	`>`	`gt`
Menší než	`<`	`lt`
Větší nebo rovno než	`>=`	`ge`
Menší nebo rovno než	`<=`	`le`
Negace	`!`	`not`

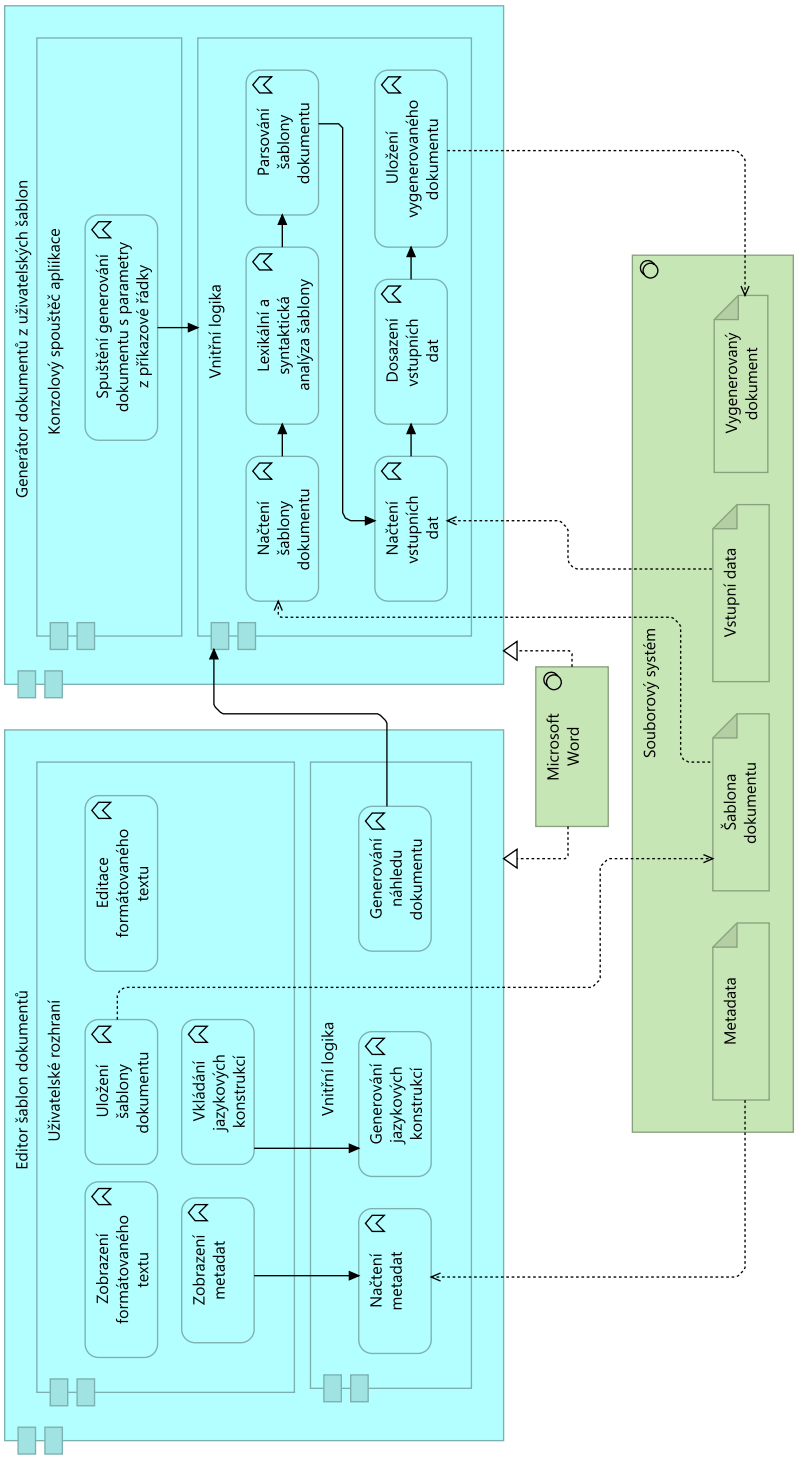
Ukázka kódu A.2: Příklady direktiv `if, elseif, else`;
Zdroj: vlastní zpracování dle [5]

```
## Priklad 1
#if($foo == $bar)
    ## Pravda - VTL kod
#end

## Priklad 2
#if($foo == $bar)
    ## Pravda - VTL kod
#  {  else  }
    ## Nepravda - VTL kod
#end

## Priklad 3
#if($foo < 10)
    ## VTL kod
#  elseif($foo == 10)
    ## VTL kod
#  else
    ## VTL kod
#end
```

Ukázka kódu A.3: Příklady direktivy `foreach`; Zdroj: [5]

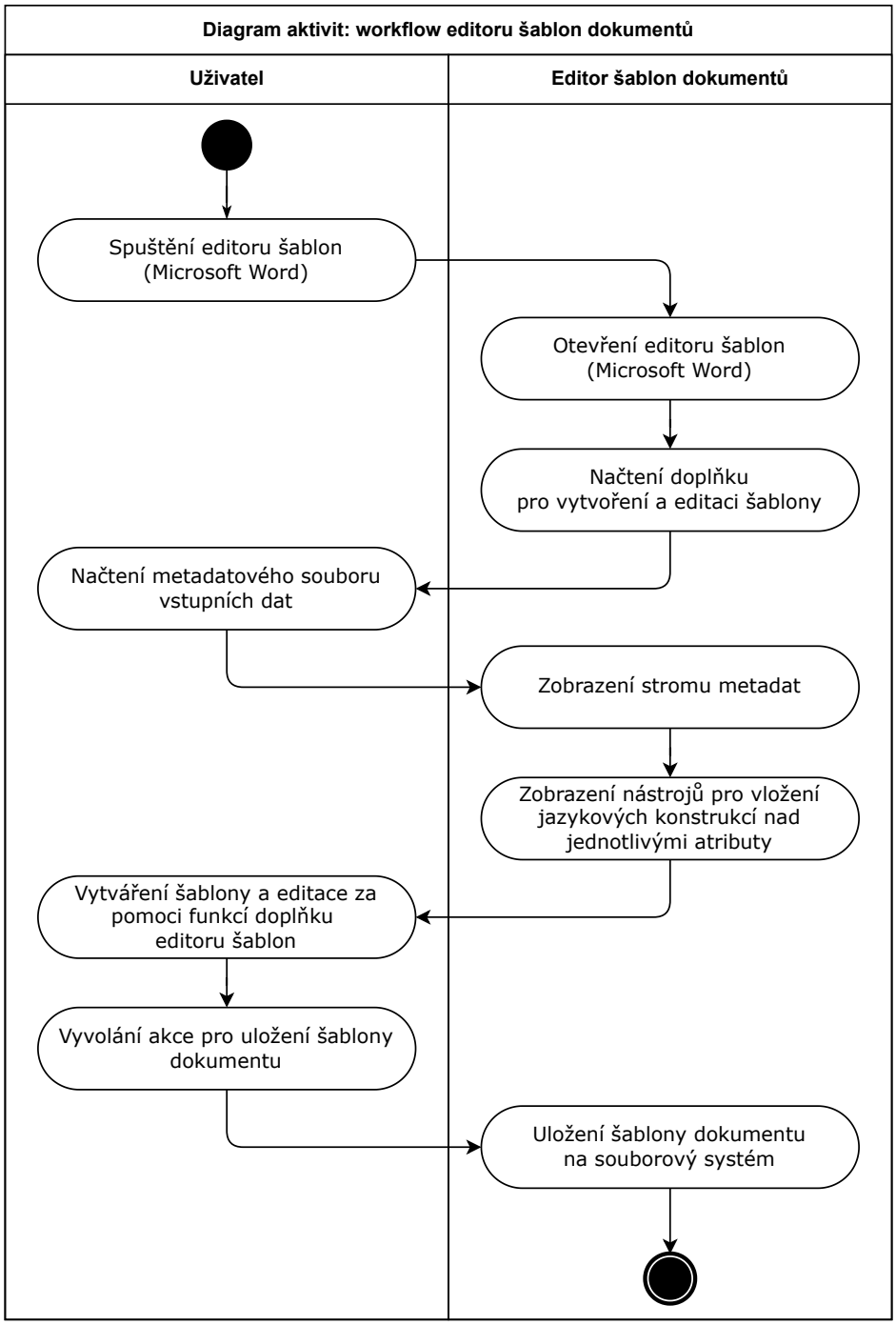
```
## Priklad 1
#foreach ($item in $items)
    ## Pokud je kolekce items validni
#  else
    ## Kolekce items neni validni
#end

## Priklad 2
#foreach ($item in [  "  Not  ", $my,   "  fault  "])
    ## Pokud je kolekce items validni
#end

## Priklad 3
#foreach ($item in [1..3])
    ## Pokud je kolekce items validni
#end
```

B Návrh architektury řešení

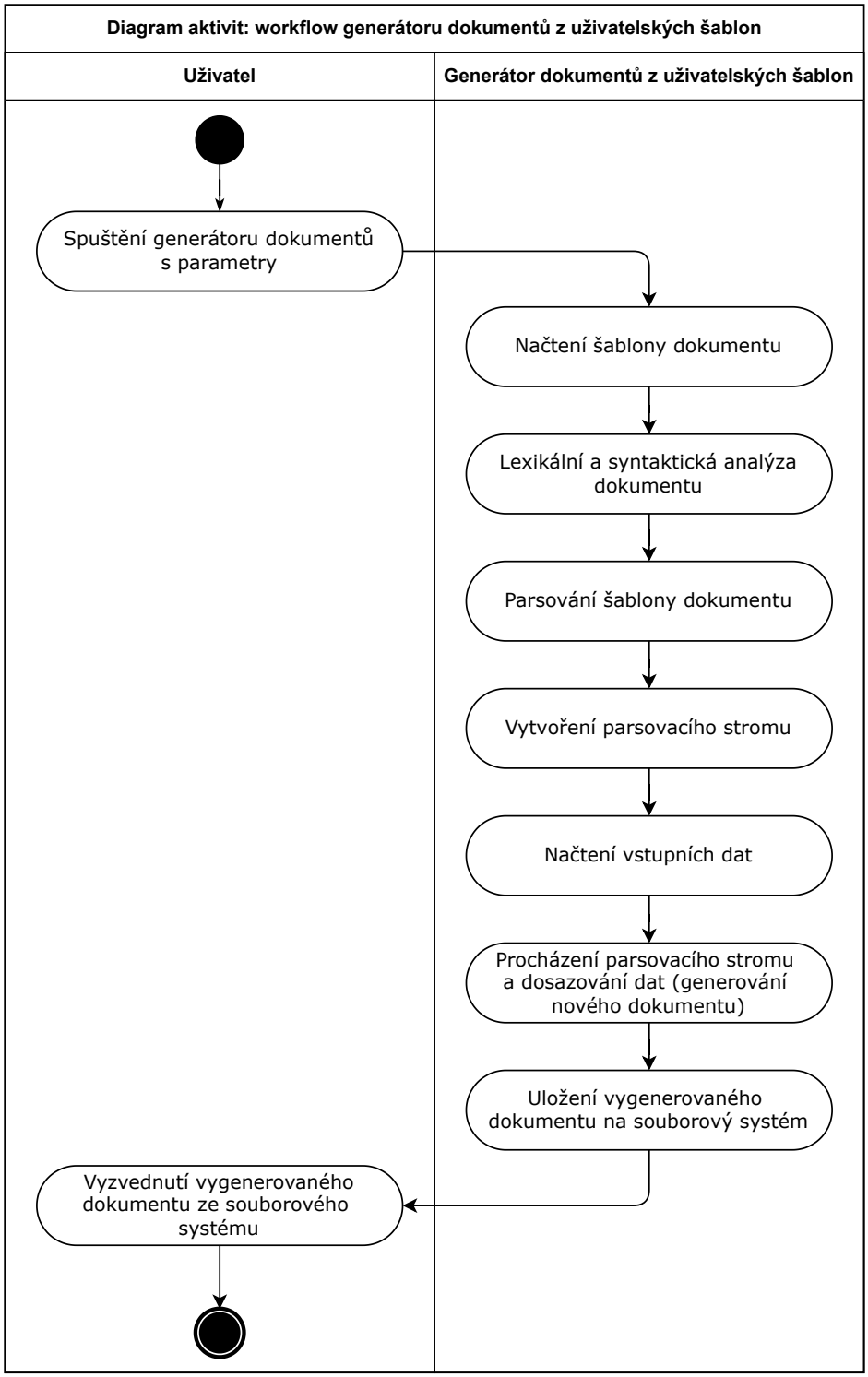


Obrázek B.1: Architektura softwarového řešení (ArchiMate);
Zdroj: vlastní zpracování

C Workflow řešení



Obrázek C.1: Workflow editoru šablon dokumentů;
Zdroj: vlastní zpracování



Obrázek C.2: Workflow generátoru dokumentů z uživatelských šablon; Zdroj: vlastní zpracování

D Struktura vstupních dat

Ukázka kódu D.1: Příklad použití objektu `DataObject`;
Zdroj: vlastní zpracování

```
{
 "name":"spis",
 "descendants":[
        {
       "name":"ucastnici",
       "descendants":[
              {
             "name":"ucastnik1",
             "attributes":[
                    {
                   ...
                    },
                    {
                   ...
                    }
             ]
              },
              {
             "name":"ucastnik2",
             "attributes":[
                    {
                   ...
                    },
                    {
                   ...
                    }
             ]
              }
       ]
        }
 ]
}
```

Ukázka kódu D.2: Příklad použití objektu `DataAttribute`;
Zdroj: vlastní zpracování

```
{
 "name":"ucastnik1",
 "attributes":[
        {
       "name":"id",
       "value":"1"
        },
        {
       "name":"role",
       "value":"manžel"
        },
        {
       "name":"jmeno",
       "value":"Evžen"
        },
        {
       "name":"prijmeni",
       "value":"Barelský"
        },
        {
       "name":"datumNarozeni",
       "value":"1976-10-29T00:00:00.000Z"
        }
 ]
}
```

E Struktura metadat

Ukázka kódu E.1: Příklad použití objektu `MetadataObject`; Zdroj: vlastní zpracování

```
{
 "name":"spis",
 "label":"Spis",
 "isArray":false,
 "descendants":[
        {
       "name":"ucastnici",
       "label":"Účastníci",
       "isArray":true,
       "descendants":[
              {
             "name":"ucastnik1",
             "label":"Účastník",
             "isArray":false,
             "attributes":[
                    {
                   ...
                    },
                    {
                   ...
                    }
             ]
              }
       ]
        }
 ]
}
```

Ukázka kódu E.2: Příklad použití objektu `MetadataAttribute`; Zdroj: vlastní zpracování

```
{
 "name":"ucastnik1",
 "label":"Účastník",
 "isArray":false,
 "attributes":[
        {
       "dataType":"integer",
       "name":"id",
       "label":"Id",
       "example":"1"
        },
        {
       "dataType":"string",
       "name":"role",
       "label":"Role",
       "example":"manžel"
        },
        {
       "dataType":"string",
       "name":"jmeno",
       "label":"Jméno",
       "example":"Jan"
        },
        {
       "dataType":"string",
       "name":"prijmeni",
       "label":"Příjmení",
       "example":"Barelský"
        },
        {
       "dataType":"datetime",
       "name":"datumNarozeni",
       "label":"Datum narození",
       "example":"1976-10-29T00:00:00.000Z"
        }
 ]
}
```

F Jazyk pro definici šablon

```
#foreach ($ucastnik in $spis.ucastnici)
${ucastnik.role}: ${ucastnik.titul} ${ucastnik.jmeno} ${ucastnik.prijmeni},
                  nar. ${ucastnik.datumNarozeni.formatDate("dd.MM.yyyy")}
                  bytem ${ucastnik.adresa}
#end
```

Obrázek F.1: Příklad použití direktivy `foreach` v jazyce šablon;
Zdroj: vlastní zpracování

#tableforeach ($ucastnik in $spis.ucastnici)

Role	**Titul**	**Jméno**	**Příjmení**
${ucastnik.role}	${ucastnik.titul}	${ucastnik.jmeno}	${ucastnik.prijmeni}

#end

Obrázek F.2: Příklad použití direktivy `tableforeach` v jazyce šablon;
Zdroj: vlastní zpracování

```
#include("C:\workspace\documents_generation\data\sub_template.docx")
```

Obrázek F.3: Příklad použití direktivy `include` v jazyce šablon;
Zdroj: vlastní zpracování

G Gramatika jazyka šablon v ANTLR 4

Ukázka kódu G.1: Gramatika jazyka šablon – lexer;
Zdroj: vlastní zpracování

```
lexer grammar DocsGenLexer;

tokens {
  OPAR, CPAR, OBRACK, CBRACK, OBRACE, CBRACE, STRING, INTEGER, ID, REFERENCE, DOT,
      FOREACH, TABLEFOREACH, INCLUDE, END, K_IN
}

ESCAPED_CHAR
 : '\\' .
 ;

START_DIRECTIVE
 : '#' -> skip, pushMode(DIR_)
 ;

DOLLAR_OBRACE
 : '$' '{' -> pushMode(FRM_)
 ;

DOLLAR
 : '$' -> pushMode(VAR_)
 ;

TEXT
 : .
 ;

// Formal mode
mode FRM_;

FRM_ID
 : ID -> type(ID)
 ;

FRM_DOT
 : '.' -> type(DOT)
 ;

FRM_OPAR
 : '(' -> type(OPAR), pushMode(CODE_)
 ;

FRM_CBRACE
 : '}' -> type(CBRACE), popMode
 ;


```

```
// Directive mode
mode DIR_;

DIR_FOREACH
 : ( 'foreach' | '{foreach}' ) SPACES? '(' -> type(FOREACH), popMode, pushMode(CODE_)
 ;

DIR_TABLEFOREACH
 : ( 'tableforeach' | '{tableforeach}' ) SPACES? '(' -> type(TABLEFOREACH), popMode,
     pushMode(CODE_)
 ;

DIR_INCLUDE
 : ( 'include' | '{include}' ) SPACES? '(' -> type(INCLUDE), popMode, pushMode(CODE_)
 ;

DIR_END
 : ( 'end' | '{end}' ) -> type(END), popMode
 ;

DIR_CUSTOM_CODE
 : ID SPACES? '(' -> type(ID), popMode, pushMode(CODE_)
 ;

DIR_CUSTOM
 : ID -> type(ID), popMode
 ;

// Variable mode
mode VAR_;

VAR_DOLLAR
 : '$'  -> type(DOLLAR)
 ;

VAR_DOLLAR_OBRACE
 : '$' '{' -> type(DOLLAR_OBRACE), popMode, pushMode(FRM_)
 ;

VAR_HASH
 : '#' -> skip, popMode, pushMode(DIR_)
 ;

VAR_ID
 : ID -> type(ID)
 ;

VAR_DOT
 : '.' -> type(DOT)
 ;

VAR_OPAR
 : '(' -> type(OPAR), pushMode(CODE_)
 ;

VAR_TEXT
 : . -> type(TEXT), popMode
 ;
// Code mode
mode CODE_;
```

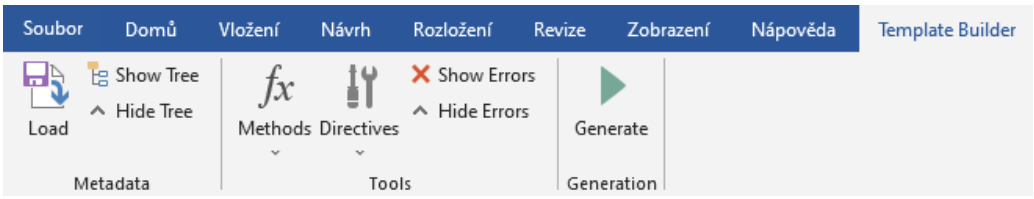
```
107
108 CODE_K_IN
109  : 'in' -> type(K_IN)
110  ;
111
112 CODE_ID
113  : ID -> type(ID)
114  ;
115
116 CODE_SPACES
117  : SPACES -> skip
118  ;
119
120 CODE_REFERENCE
121  : '$' ID -> type(REFERENCE)
122  ;
123
124 CODE_OPAR
125  : '(' -> type(OPAR), pushMode(CODE_)
126  ;
127
128 CODE_CPAR
129  : ')' -> type(CPAR), popMode
130  ;
131
132 CODE_DOT
133  : '.' -> type(DOT)
134  ;
135
136 CODE_INTEGER
137  : INTEGER -> type(INTEGER)
138  ;
139
140 CODE_STRING
141  : STRING -> type(STRING)
142  ;
143
144 CODE_OBRACE
145  : '{' -> type(OBRACE)
146  ;
147
148 CODE_CBRACE
149  : '}' -> type(CBRACE)
150  ;
151
152 fragment STRING
153  : STRING_DQ
154  | STRING_SQ
155  ;
156
157 fragment SPACES    : [ \t\r\n];
158 fragment ID        : [a-zA-Z] [a-zA-Z0-9_-]*;
159 fragment STRING_DQ : '"' ( ~[\r\n] | '""' )* '"';
160 fragment STRING_SQ : '\'' ( ~[\r\n] | '\'\'' )* '\'';
161 fragment INTEGER   : DIGIT+;
162 fragment DIGIT     : [0-9];
163 fragment EXPONENT  : [eE] [+-]? DIGIT+;
```

Ukázka kódu G.2: Gramatika jazyka šablon – parser;
Zdroj: vlastní zpracování

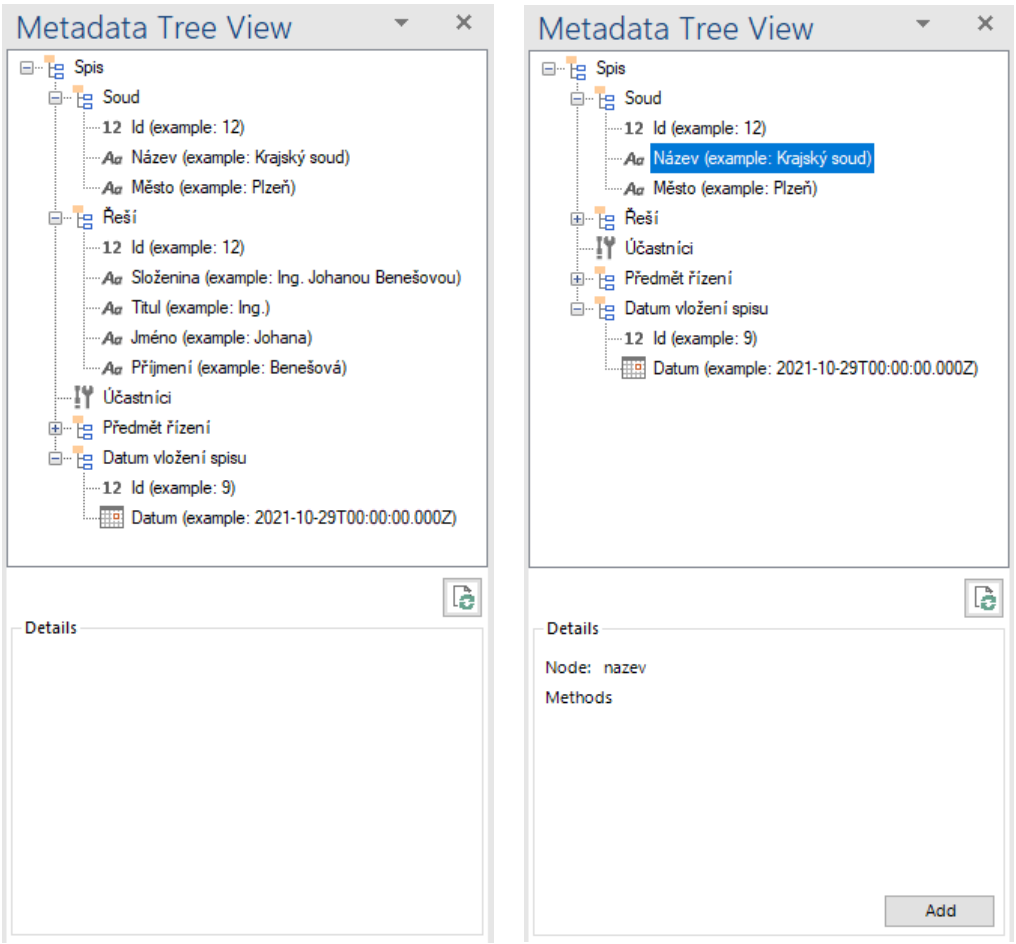
```
parser grammar DocsGenParser;

options {
  tokenVocab=DocsGenLexer;
}

parse
 : block EOF
 ;

block
 : atom*
 ;

atom
 : TEXT
 | ESCAPED_CHAR
 | variable
 | formal
 | property_or_method
 | directive
 ;

formal
 : DOLLAR_OBRACE formal_property_or_method CBRACE
 | DOLLAR_OBRACE id CBRACE
 ;

variable
 : DOLLAR id DOT?
 | REFERENCE DOT?
 ;

property_or_method
 : variable property_end+
 ;

formal_property_or_method
 : id property_end+
 ;

directive
 : foreach_directive
 | tableforeach_directive
 | include_directive
 ;

property_end
 : DOT ID
 | OPAR parameters? CPAR
 ;

parameters
 : parameter
 ;


```

```
58 foreach_directive
59  : FOREACH variable K_IN property_or_method CPAR block end
60  ;
61
62 tableforeach_directive
63  : TABLEFOREACH variable K_IN property_or_method CPAR block end
64  ;
65
66 include_directive
67  : INCLUDE STRING CPAR
68  ;
69
70 end
71  : END
72  ;
73
74 parameter
75  : STRING
76  | INTEGER
77  ;
78
79 id
80  : ID
81  | K_IN
82  ;
```

H Implementované GUI

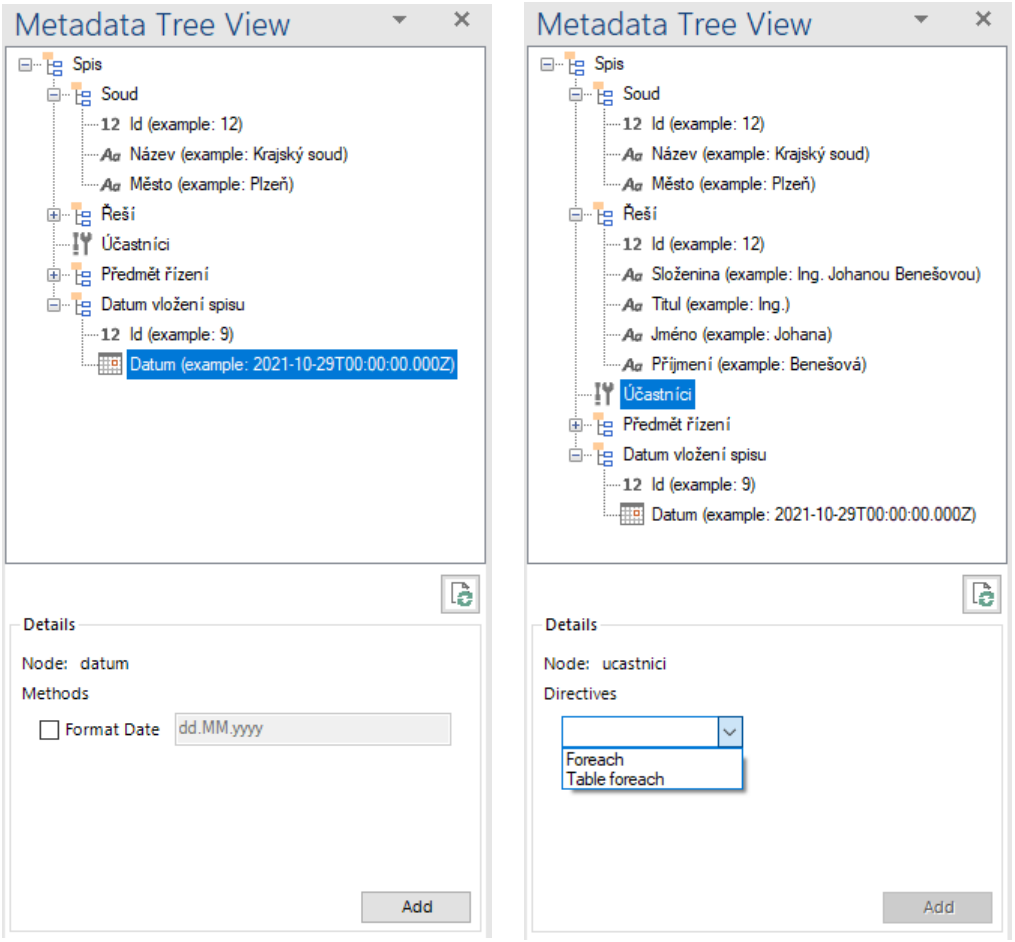


Obrázek H.1: GUI – lišta nástrojů; Zdroj: vlastní zpracování



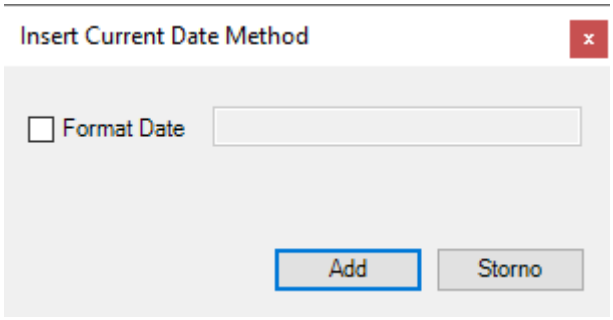
(a) Stav: nevybrán atribut ani objekt (b) Stav: vybrán atribut Název

Obrázek H.2: GUI – stromová struktura metadat, 1. část; Zdroj: vlastní zpracování

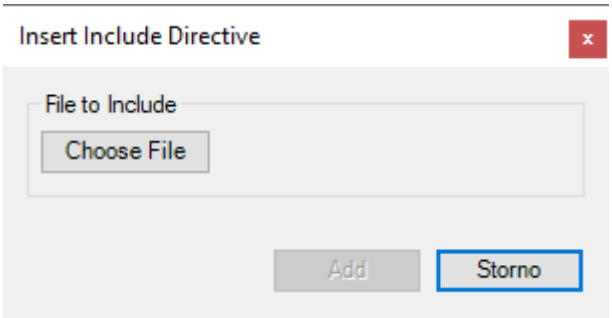


(a) Stav: vybrán atribut Datum (b) Stav: vybrán objekt Účastníci

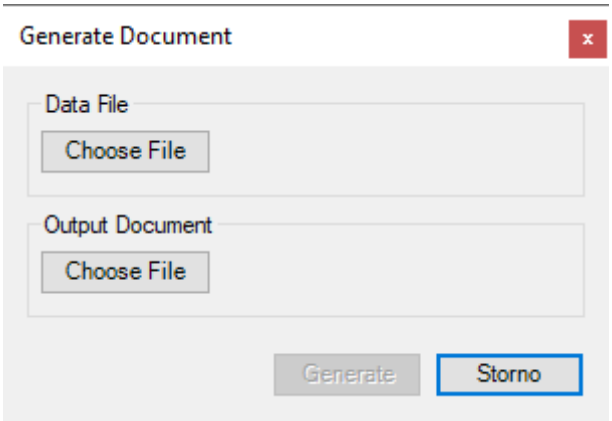
Obrázek H.3: GUI – stromová struktura metadat, 2. část;
Zdroj: vlastní zpracování



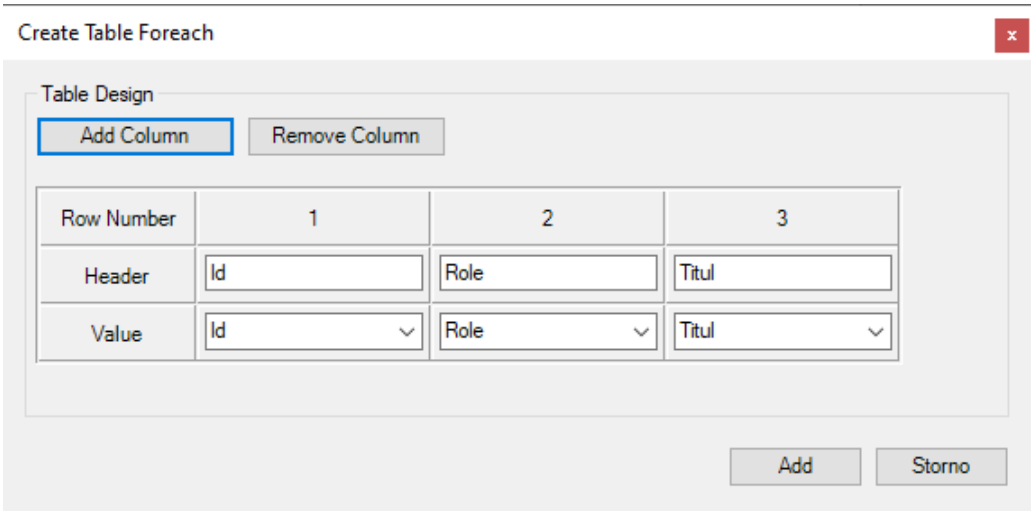
Obrázek H.4: GUI – formulářové okno pro vložení aktuálního data;
Zdroj: vlastní zpracování



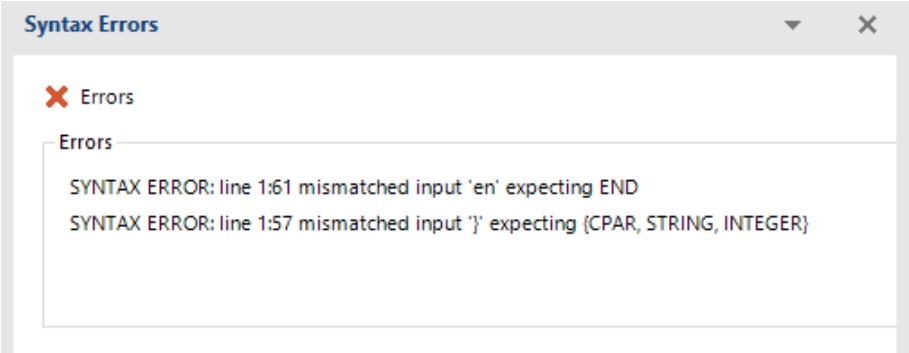
Obrázek H.5: GUI – formulářové okno pro vložení podšablony dokumentu;
Zdroj: vlastní zpracování



Obrázek H.6: GUI – formulářové okno pro vygenerování nového dokumentu;
Zdroj: vlastní zpracování



Obrázek H.7: GUI – formulářové okno pro vytvoření návrhu tabulky;
Zdroj: vlastní zpracování



Obrázek H.8: GUI – panel syntaktických chyb;
Zdroj: vlastní zpracování

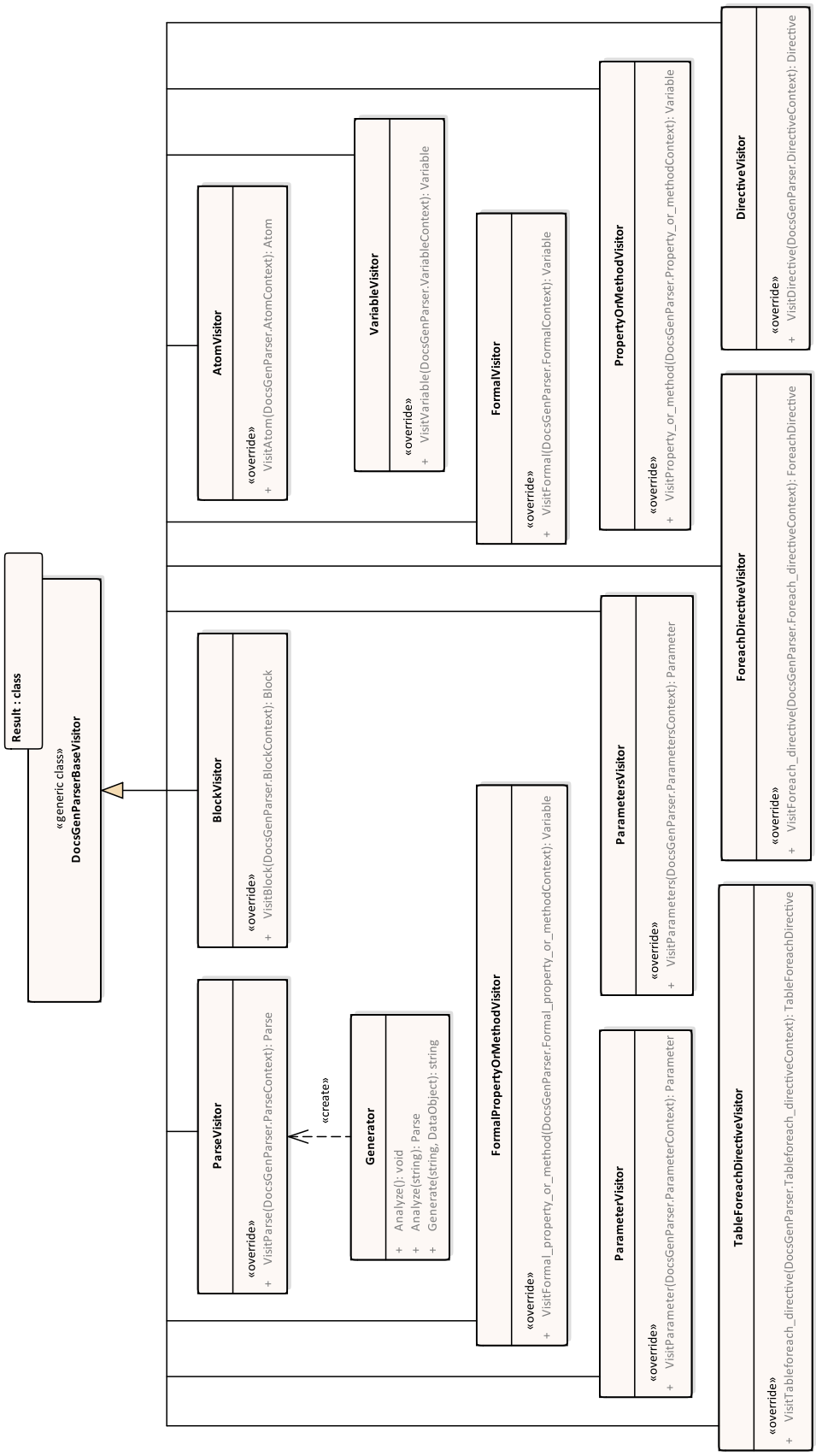
I Vytvoření parsovacího stromu

Ukázka kódu I.1: Implementace metody `Generate(...)`;
Zdroj: vlastní zpracování

```
public string Generate(string inputText, DataObject dataRoot)
{
    AntlrInputStream inputStream = new AntlrInputStream(inputText.ToString());
    DocsGenLexer docsGenLexer = new DocsGenLexer(inputStream);
    CommonTokenStream commonTokenStream = new CommonTokenStream(docsGenLexer);
    DocsGenParser docsGenParser = new DocsGenParser(commonTokenStream);

    docsGenParser.RemoveErrorListeners();
    docsGenParser.AddErrorListener(SyntaxErrorListener.Instance);

    DocsGenParser.ParseContext parseContext = docsGenParser.parse();
    Parse parse = new ParseVisitor().Visit(parseContext);

    return new DocumentBuilder(parse, dataRoot).BuildText();
}
```

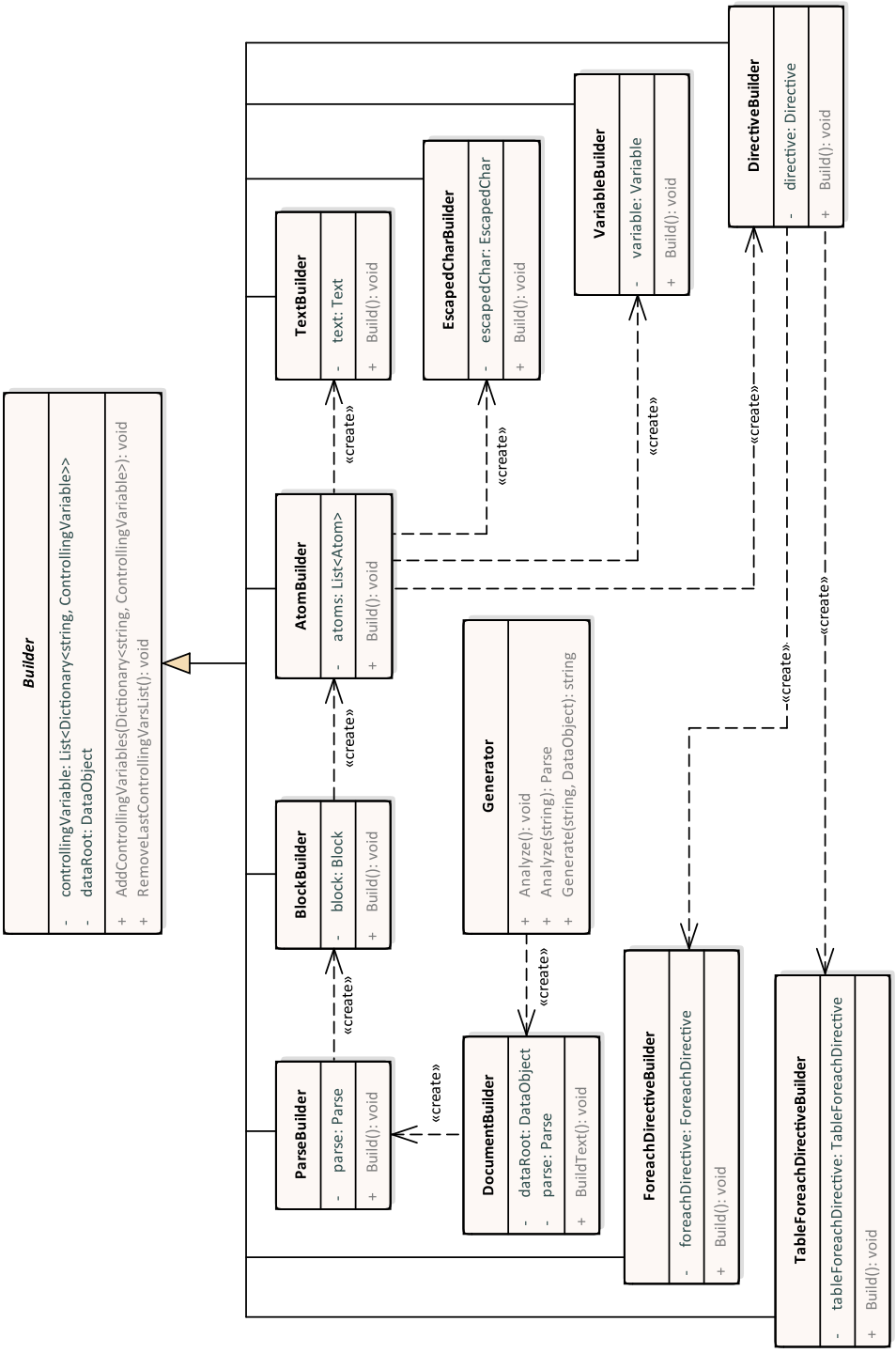
Ukázka kódu I.2: Implementace třídy `ParseVisitor`;
Zdroj: vlastní zpracování

```
using Antlr4.Runtime.Misc;
using GenLib.Entity;

namespace GenLib.Visitor
{
    class ParseVisitor : DocsGenParserBaseVisitor<Parse>
    {
        public override Parse VisitParse([NotNull] DocsGenParser.ParseContext context)
        {
            Block block = new BlockVisitor().Visit(context.block());

            return new Parse(block);
        }
    }
}
```



Obrázek I.1: UML diagram tříd – třídy `Visitor`; Zdroj: vlastní zpracování

J Procházení parsovacího stromu



Obrázek J.1: UML diagram tříd – třídy `Builder`; Zdroj: vlastní zpracování

K Volání generátoru dokumentů z externích aplikací

Ukázka kódu K.1: Příklad volání generátoru dokumentů z externích aplikací; Zdroj: vlastní zpracování

```
1  public static void CallGenLib(string templateFilePath, string documentFilePath, string
       dataFilePath)
2  {
3      DocumentManager documentManager = DocumentManager.Instance;
4
5      documentManager.LoadTempTemplateFromFile(templateFilePath);
6      documentManager.Preprocess();
7      documentManager.CreateDocument(documentFilePath);
8      documentManager.GenerateDocument(dataFilePath);
9
10     documentManager.Close();
11     documentManager.CloseWordProcesses();
12 }
```

L Soubory pro generování komplexního dokumentu

Ukázka kódu L.1: Vstupní data ve formátu JSON;
Zdroj: vlastní zpracování

```json
{
   "name":"spis",
   "descendants":[
      {
         "name":"soud",
         "attributes":[
            {
               "name":"id",
               "value":"12"
            },
            {
               "name":"nazev",
               "value":"Okresní soud Plzeň – město"
            },
            {
               "name":"mesto",
               "value":"Plzeň"
            }
         ]
      },
      {
         "name":"spisovaZnacka",
         "attributes":[
            {
               "name":"cisloSenatu",
               "value":"2"
            },
            {
               "name":"druhVec",
               "value":"C"
            },
            {
               "name":"bezneCislo",
               "value":"123"
            },
            {
               "name":"rocnik",
               "value":"2022"
            }
         ]
      },
      {
         "name":"resi",
         "attributes":[
            {
               "name":"id",
               "value":"12"
```

```
            },
            {
               "name":"slozenina7p",
               "value":"Ing. Johanou Benešovou"
            },
            {
               "name":"titul",
               "value":"Ing."
            },
            {
               "name":"jmeno",
               "value":"Johana"
            },
            {
               "name":"prijmeni",
               "value":"Benešová"
            }
         ]
      },
      {
         "name":"ucastnici",
         "descendants":[
            {
               "name":"ucastnik",
               "attributes":[
                  {
                     "name":"id",
                     "value":"1"
                  },
                  {
                     "name":"role",
                     "value":"manžel"
                  },
                  {
                     "name":"titul",
                     "value":"Ing."
                  },
                  {
                     "name":"jmeno",
                     "value":"Evžen"
                  },
                  {
                     "name":"prijmeni",
                     "value":"Barelský"
                  },
                  {
                     "name":"datumNarozeni",
                     "value":"1976-10-29T00:00:00.000Z"
                  },
                  {
                     "name":"adresa",
                     "value":"Mojestrasse 2/15, Praha 1 — Letňany"
                  }
               ]
            },
            {
               "name":"ucastnik",
               "attributes":[
                  {
                     "name":"id",
```

```
                    "value":"2"
                },
                {
                    "name":"role",
                    "value":"manželka"
                },
                {
                    "name":"titul",
                    "value":""
                },
                {
                    "name":"jmeno",
                    "value":"Anna"
                },
                {
                    "name":"prijmeni",
                    "value":"Haasová"
                },
                {
                    "name":"datumNarozeni",
                    "value":"1979-10-29T00:00:00.000Z"
                },
                {
                    "name":"adresa",
                    "value":"Švermova 872, Strakonice"
                }
            ]
        }
    ]
},
{
    "name":"predmetRizeni",
    "attributes":[
        {
            "name":"id",
            "value":"3"
        },
        {
            "name":"nazev",
            "value":"rozvod manželství 5 845 Kč"
        }
    ]
},
{
    "name":"datumVlozeni",
    "attributes":[
        {
            "name":"id",
            "value":"3"
        },
        {
            "name":"datum",
            "value":"2021-10-29T00:00:00.000Z"
        }
    ]
}
]
}
```

USNESENÍ

${spis.soud.nazev} rozhodl ${spis.resi.slozenina7p} ve věci

#foreach ($ucastnik in $spis.ucastnici)
${ucastnik.role}: **${ucastnik.titul} ${ucastnik.jmeno} ${ucastnik.prijmeni}**,
nar. ${ucastnik.datumNarozeni.formatDate("dd.MM.yyyy")}
bytem ${ucastnik.adresa}
#end
o: ${spis.predmetRizeni.nazev}

takto:

Odvolání manžela a manželky proti usnesení nadepsaného soudu o zastavení řízení čj. ${spis.spisovaZnacka.cisloSenatu} ${spis.spisovaZnacka.druhVec} ${spis.spisovaZnacka.be zneCislo}/${spis.spisovaZnacka.rocnik} ze dne 10.08.2021, které nabylo právní moci dne 21.8.2021, **není** pro opožděnost účinné (§ 96 odst. 5 zákona č. 99/1963 Sb., občanského soudního řádu, (dále jen „o. s. ř.").

Odůvodnění:

Odvoláním napadené rozhodnutí bylo doručeno manželovi a manželce dne. Patnáctidenní odvolací lhůta proti tomuto rozhodnutí uplynula dne. Odvolání proti shora uvedenému rozhodnutí podali manžel a manželka k poštovní přepravě (zaslali k soudu elektronicky). Předmětné rozhodnutí nabylo právní moci dne 21.8.2021. Návrh na zahájení řízení byl vzat zpět až poté, co rozhodnutí o věci již nabylo právní moci a zpětvzetí návrhu není proto účinné.

Poučení:

#include ("..\..\test_scenario\06_ComplexTemplate\correct\1\subtemplate.docx")

${spis.soud.mesto}

${spis.resi.titul} ${spis.resi.jmeno} ${spis.resi.prijmeni}

Obrázek L.1: Šablona dokumentu ve formátu docx;
Zdroj: vlastní zpracování

Proti tomuto usnesení je možno podat odvolání do 15 dnů ode dne jeho doručení ke Krajskému soudu v Plzni prostřednictvím ${spis.soud.nazev}.

Obrázek L.2: Podšablona dokumentu ve formátu docx;
Zdroj: vlastní zpracování

USNESENÍ

Okresní soud Plzeň - město rozhodl Ing. Johanou Benešovou ve věci

manžel: **Ing. Evžen Barelský**, nar. 29.10.1976
bytem Mojestrasse 2/15, Praha 1 - Letňany
manželka: **Anna Haasová**, nar. 29.10.1979
bytem Švermova 872, Strakonice
o: rozvod manželství 5 845 Kč

takto:

Odvolání manžela a manželky proti usnesení nadepsaného soudu o zastavení řízení čj. 2 C 123/2022 ze dne 10.08.2021, které nabylo právní moci dne 21.8.2021, **není** pro opožděnost účinné (§ 96 odst. 5 zákona č. 99/1963 Sb., občanského soudního řádu, (dále jen „o. s. ř.").

Odůvodnění:

Odvoláním napadené rozhodnutí bylo doručeno manželovi a manželce dne. Patnáctidenní odvolací lhůta proti tomuto rozhodnutí uplynula dne. Odvolání proti shora uvedenému rozhodnutí podali manžel a manželka k poštovní přepravě (zaslali k soudu elektronicky). Předmětné rozhodnutí nabylo právní moci dne 21.8.2021. Návrh na zahájení řízení byl vzat zpět až poté, co rozhodnutí o věci již nabylo právní moci a zpětvzetí návrhu není proto účinné.

Poučení:

Proti tomuto usnesení je možno podat odvolání do 15 dnů ode dne jeho doručení ke Krajskému soudu v Plzni prostřednictvím Okresní soud Plzeň - město.

Plzeň

Ing. Johana Benešová

Obrázek L.3: Očekávaný finální dokument ve formátu docx;
Zdroj: vlastní zpracování

USNESENÍ

Okresní soud Plzeň - město rozhodl Ing. Johanou Benešovou ve věci

manžel: **Ing. Evžen Barelský**, nar. 29.10.1976
bytem Mojestrasse 2/15, Praha 1 - Letňany
manželka: **Anna Haasová**, nar. 29.10.1979
bytem Švermova 872, Strakonice
o: rozvod manželství 5 845 Kč

takto:

Odvolání manžela a manželky proti usnesení nadepsaného soudu o zastavení řízení čj. 2 C 123/2022 ze dne 10.08.2021, které nabylo právní moci dne 21.8.2021, **není** pro opožděnost účinné (§ 96 odst. 5 zákona č. 99/1963 Sb., občanského soudního řádu, (dále jen „o. s. ř.").

Odůvodnění:

Odvoláním napadené rozhodnutí bylo doručeno manželovi a manželce dne. Patnáctidenní odvolací lhůta proti tomuto rozhodnutí uplynula dne. Odvolání proti shora uvedenému rozhodnutí podali manžel a manželka k poštovní přepravě (zaslali k soudu elektronicky). Předmětné rozhodnutí nabylo právní moci dne 21.8.2021. Návrh na zahájení řízení byl vzat zpět až poté, co rozhodnutí o věci již nabylo právní moci a zpětvzetí návrhu není proto účinné.

Poučení:

Proti tomuto usnesení je možno podat odvolání do 15 dnů ode dne jeho doručení ke Krajskému soudu v Plzni prostřednictvím Okresní soud Plzeň - město.

Plzeň

Ing. Johana Benešová

Obrázek L.4: Vygenerovaný finální dokument ve formátu docx;
Zdroj: vlastní zpracování

M Instalační dokumentace

V této kapitole popisuji sestavení zdrojových souborů všech částí softwarového produktu a následnou instalaci. Softwarový produkt je určen pro prostředí operačních systémů Microsoft Windows, proto uvádím postup pouze pro tyto operační systémy.

M.1 Sestavení zdrojových souborů

Zdrojové soubory softwarového řešení jsou rozděleny do *tří projektů*:

- `TemplateBuilder` – soubor projektu: `TemplateBuilder.csproj`,
- `GenLib` – soubor projektu: `GenLib.csproj`,
- `GenTrigger` – soubor projektu: `GenTrigger.csproj`,

Tyto projekty jsou součástí řešení (*solution*) `DocsGen.sln`. Umístění těchto souborů je uvedeno v příloze O.

Sestavení zdrojových souborů softwarového řešení lze realizovat dvěma způsoby:

- prostřednictvím produktu `Microsoft Visual Studio` (2019 nebo vyšší) – *doporučená varianta*,
- prostřednictvím samostatného nástroje `MSBuild` (2019 nebo vyšší).

V obou případech je nutné mít *nainstalované* a *správně nastavené* následující softwarové vybavení:

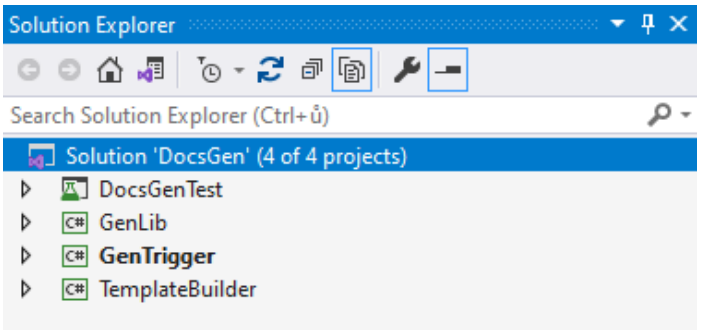
- `.NET Framework 4.7.2 Developer Pack`,
- `NuGet Package Manager 5.11.0` nebo vyšší,
- `MSBuild 2019` nebo vyšší.

V případě sestavení prostřednictvím produktu `Microsoft Visual Studio` se toto softwarové vybavení neinstaluje samostatně, ale přes `Microsoft Visual Studio Installer`.

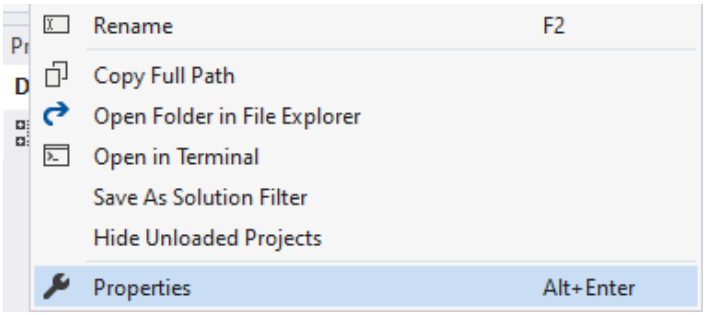
M.1.1 Sestavení pomocí `Microsoft Visual Studio`

Sestavení zdrojových souborů softwarového produktu lze jednoduše realizovat prostřednictvím vývojového prostředí `Microsoft Visual Studio`. Postup sestavení je následovný:

1. otevřít řešení `DocsGen.sln` v `Microsoft Visual Studio`,
2. v sekci `Solution Explorer` (obrázek M.1) kliknout pravým tlačítkem na řešení `DocsGen`, dále kliknout na tlačítko `Properties` (obrázek M.2),
3. nastavit konfiguraci všech projektů na `Release` (obrázek M.3) a potvrdit,
4. v sekci `Solution Explorer` kliknout pravým tlačítkem na řešení `DocsGen`, dále kliknout na tlačítko `Build Solution` (obrázek M.4).



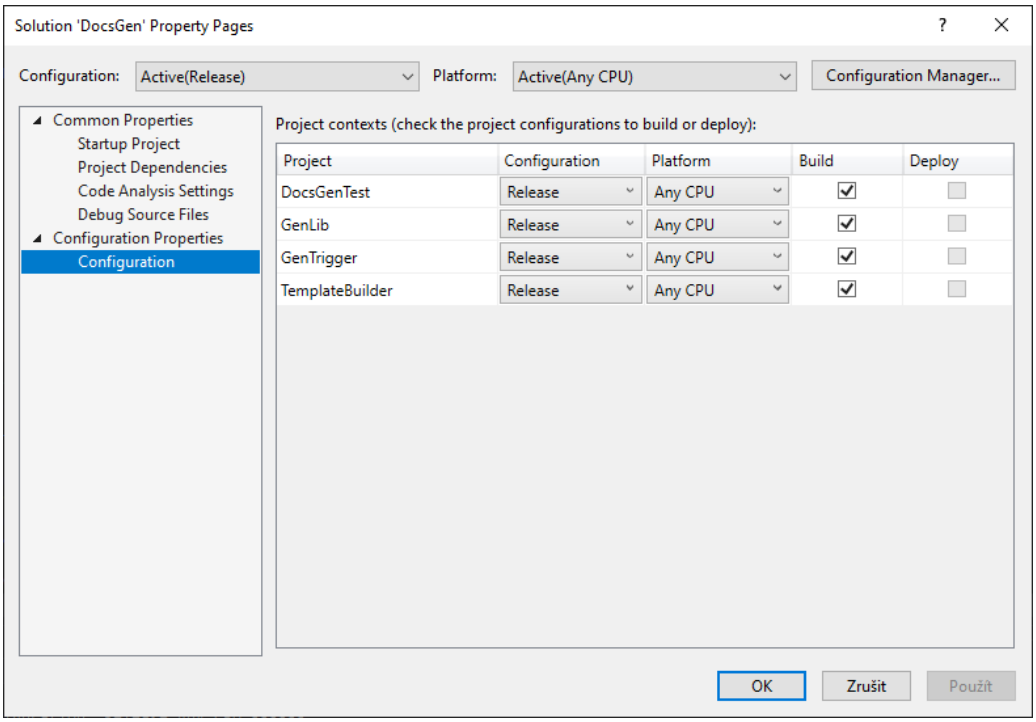
Obrázek M.1: Visual Studio – `Solution Explorer`



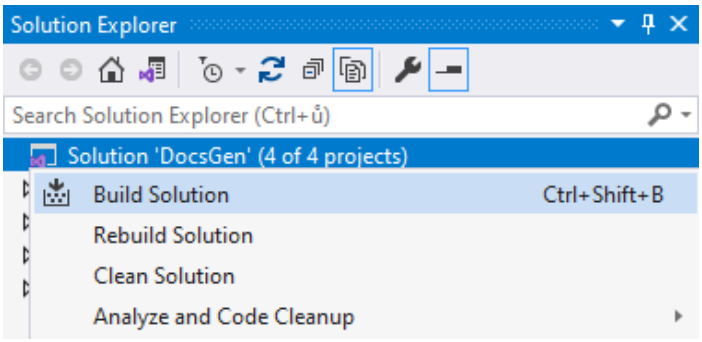
Obrázek M.2: Visual Studio – `Solution Explorer` (`Properties`)

Jednotlivé sestavené projekty se uloží do umístění:

`DocsGen\<nazev_projektu>\bin\Release`.



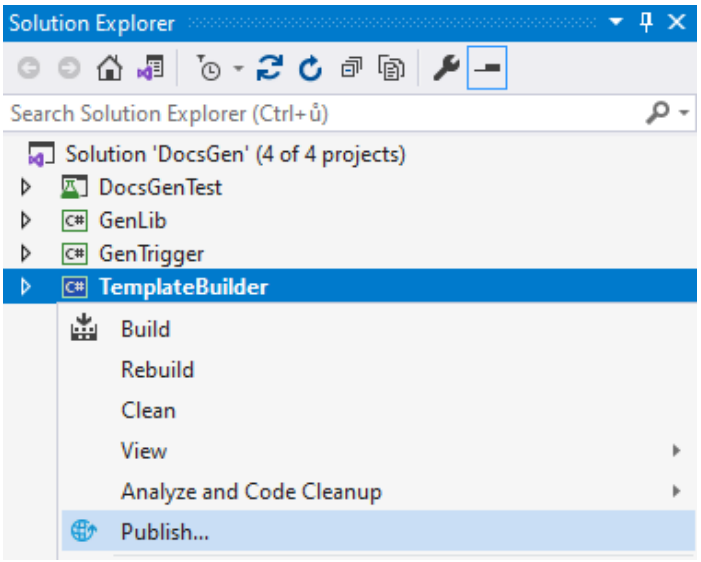
Obrázek M.3: Visual Studio – `Solution 'DocsGen' Property Pages`



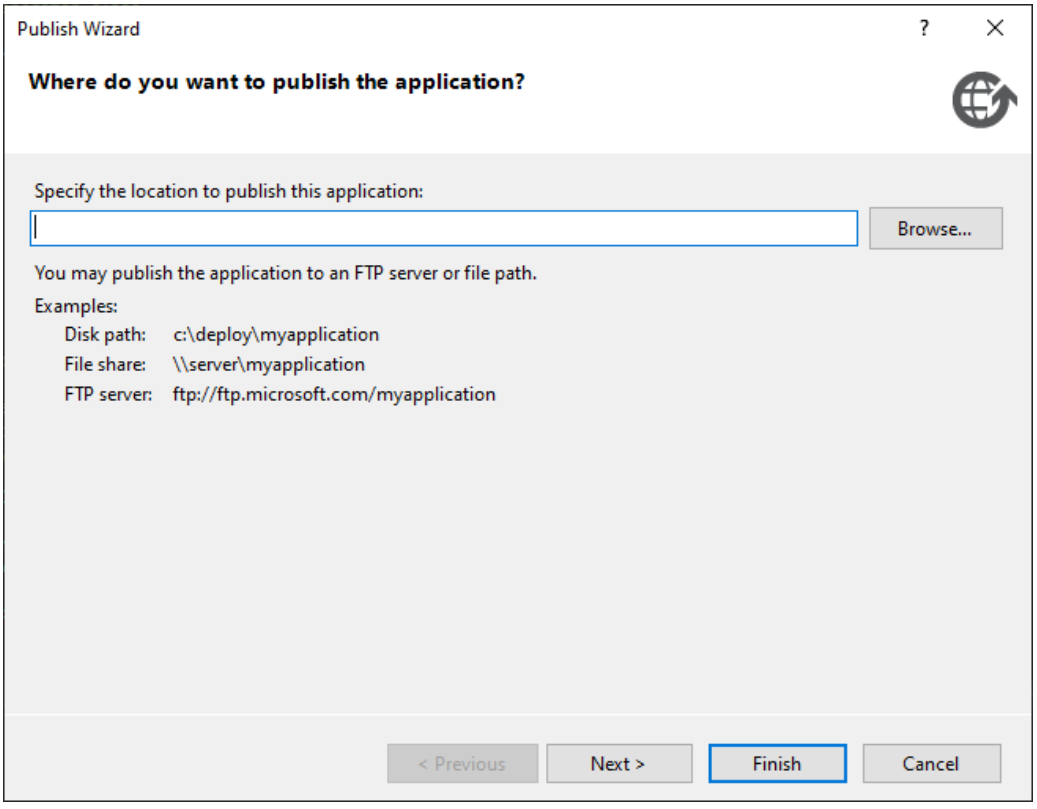
Obrázek M.4: Visual Studio – `Solution Explorer` (`Build Solution`)

V případě projektu `TemplateBuilder` (rozšíření produktu Microsoft Word) je vhodné vytvořit instalátor. Postup je následovný:

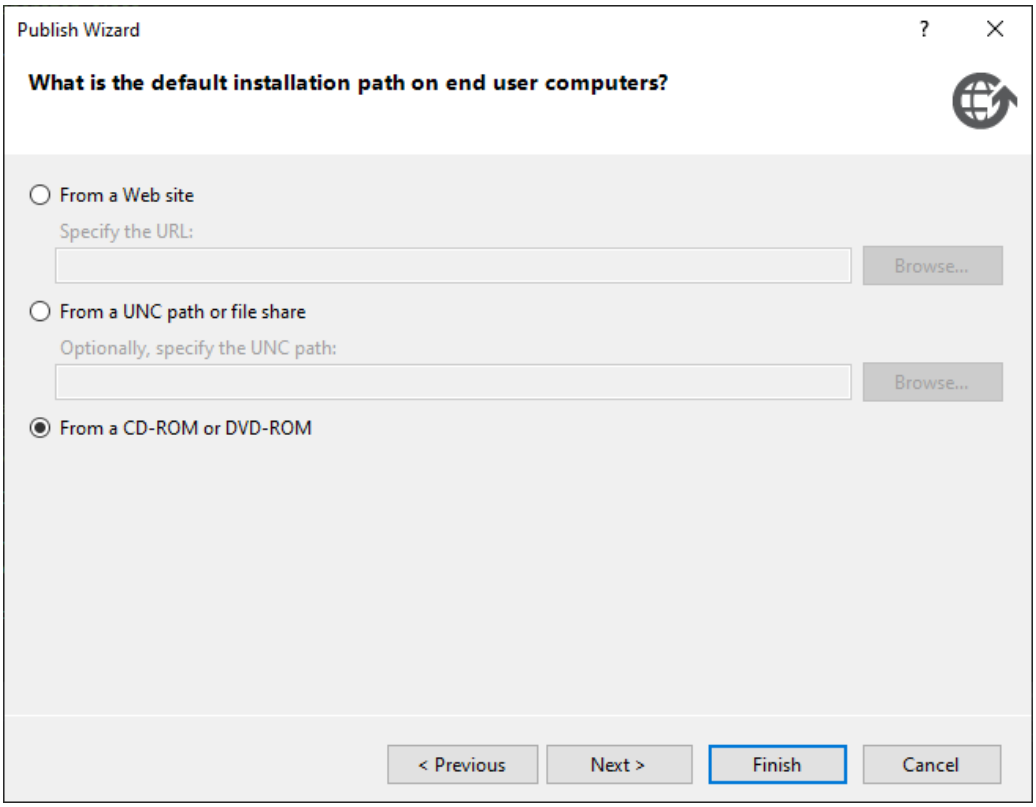
1. v sekci `Solution Explorer` kliknout pravým tlačítkem na projekt `TemplateBuilder`, dále kliknout na tlačítko `Publish` (obrázek M.5),
2. v okně `Publish Wizard` specifikovat cestu, kam má být instalátor uložen, dále kliknout na tlačítko `Next` (obrázek M.6),
3. v okně `Publish Wizard` zvolit možnost `From a CD-ROM or DVD-ROM` a kliknout na tlačítko `Finish` (obrázek M.7).



Obrázek M.5: Visual Studio – `Solution Explorer` (`Publish`)



Obrázek M.6: Visual Studio – `Publish Wizard`



Obrázek M.7: Visual Studio – `Publish Wizard`

M.1.2 Sestavení pomocí samostatného nástroje `MSBuild`

Sestavení zdrojových souborů softwarového produktu lze realizovat také prostřednictvím samostatného nástroje `MSBuild`. Postup sestavení je následovný:

1. stáhnout `NuGet` balíčky pro projekty v rámci řešení `DocsGen.sln`:
   - zadat příkaz v `Příkazovém řádku` pro každý projekt:

     ```
     <cesta_k_instalaci_nuget.exe> install
     <cesta_do_adresare_projektu>\packages.config
     -o packages\
     ```

   - například pro projekt `TemplateBuilder` a pro umístění instalace `nuget.exe` v cestě `C:\nuget.exe` může být příkaz následovný:

     ```
     C:\nuget.exe install .\TemplateBuilder\
     packages.config -o packages\
     ```

2. sestavit projekty řešení prostřednictvím nástroje `MSBuild`:
   - zadat příkaz v `Příkazovém řádku` pro řešení `DocsGen.sln`:

```
<cesta_k_instalaci_MSBuild.exe>
<cesta_k_souboru_DocsGen.sln> /t:Rebuild
/p:Configuration=Release /p:Platform="Any
CPU"
```
,

- příkaz může být následovný:

```
"C:\Program Files\BuildTools\MSBuild\Current\
Bin\MSBuild.exe" DocsGen.sln /t:Rebuild
/p:BuildProjectReferences=true
/p:Configuration=Release /p:Platform="Any CPU"
```
,

Jednotlivé sestavené projekty se i v tomto případě uloží do umístění:

`DocsGen\<nazev_projektu>\bin\Release`.

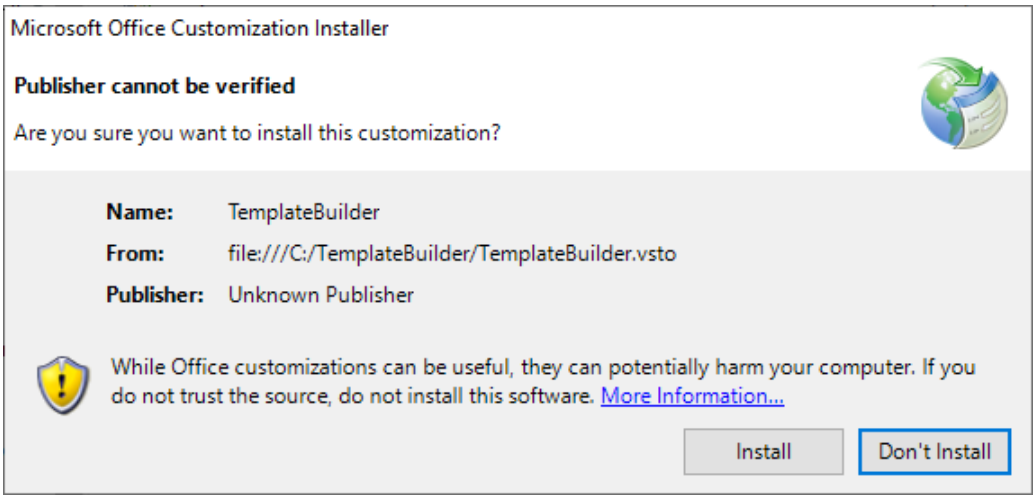
M.2 Instalace softwarového produktu

Po sestavení zdrojových souborů softwarového řešení je nutné nainstalovat rozšíření produktu Microsoft Word (výstup projektu `TemplateBuilder`). Ostatní části řešení se neinstalují.

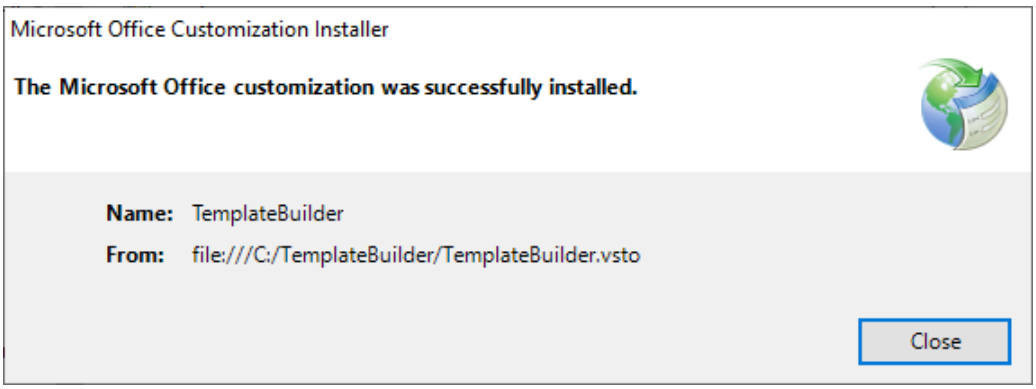
V případě vytvoření instalátoru prostřednictvím produktu `Microsoft Visual Studio` se k instalaci použije soubor `setup.exe` (umístění dle specifikované cesty při vytvoření instalátoru). V případě sestavení prostřednictvím nástroje `MSBuild` se použije soubor `TemplateBuilder.vsto`. Postup instalace pro obě tyto varianty je následovný:

1. spustit soubor `setup.exe` (případně `TemplateBuilder.vsto`),
2. v okně `Microsoft Office Customization Installer` kliknout na tlačítko `Install` (obrázek M.8).

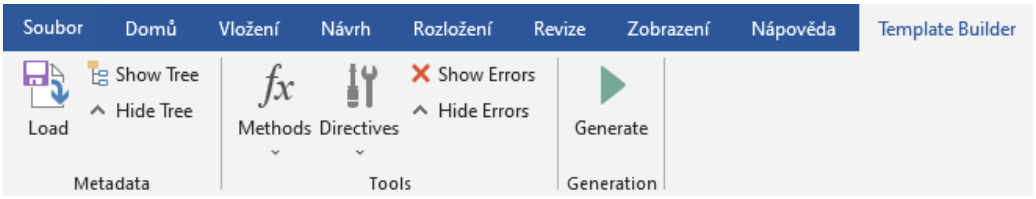
Rozšíření se automaticky nainstaluje (obrázek M.9) a přidá do navigační lišty produktu Microsoft Word (obrázek M.10).



Obrázek M.8: Instalace rozšíření – `Microsoft Office Customization Installer`



Obrázek M.9: Instalace rozšíření – `Microsoft Office Customization Installer`



Obrázek M.10: Instalace rozšíření – `Microsoft Word`

N Uživatelská dokumentace

V této kapitole popisuji spuštění a ovládání softwarového produktu. Předpokladem je dokončené sestavení a instalace softwarového produktu. Opět uvádím postup pouze pro operační systémy Microsoft Windows.

N.1 Generátor dokumentů z uživatelských šablon

Generátor dokumentů z uživatelských šablon je realizován jako DLL knihovna (projekt `GenLib`), která je spouštěna prostřednictvím konzolového spouštěče (projekt `GenTrigger`).

Konzolový spouštěč se spouští prostřednictvím `Příkazového řádku`, konkrétně se jedná o spustitelný soubor `GenTrigger.exe`. Při spuštění je nutné zadat tři parametry, kterými jsou:

- absolutní cesta k šabloně dokumentu (ve formátu `doc` či `docx`),
- absolutní cesta k souboru vstupních dat (ve formátu `JSON`),
- absolutní cesta, na kterou bude vygenerován nový dokument (ve formátu `doc` či `docx`).

Spuštění v `Příkazovém řádku` může vypadat následovně:

```
GenTrigger.exe C:\template.docx C:\input_data.json
C:\generated_document.docx
```

.

V případě, že vstupní parametry jsou správně zadány a všechny předané soubory jsou validní, dojde k vygenerování nového dokumentu.

N.2 Editor šablon dokumentů

Editor šablon dokumentů je realizován jako rozšíření produktu Microsoft Word. Toto rozšíření se spouští automaticky při spuštění Microsoft Word.

N.2.1 Ovládací a grafické prvky rozšíření

Po spuštění aplikace Microsoft Word je možné zobrazit *ovládací a grafické prvky rozšíření* kliknutím na záložku `TemplateBuilder` v navigační liště produktu Microsoft Word. Všechny ovládací a grafické prvky jsou ilustrovány na obrázcích v příloze H, a to konkrétně:

- lišta nástrojů (obrázek H.1),
- stromová struktura metadat (obrázky H.2 a H.3),
- formulářová okna (obrázky H.4, H.5, H.6 a H.7),
- panel syntaktických chyb (obrázek H.8).

N.2.2 Vytvoření a editace šablony dokumentu

Šablonou dokumentu je soubor ve formátu Microsoft Word, tudíž vytvoření souboru šablony odpovídá vytvoření nového souboru formátu Microsoft Word.

Editace šablony se realizuje:

- prostřednictvím *funkcí produktu Microsoft Word* (formátování textu, přidání tabulek, přidání obrázků atd.),
- prostřednictvím *funkcí implementovaného rozšíření* (načtení metadat, přidání jazykových konstrukcí, přidání metod).

Načtení metadat

Postup načtení metadat je následovný:

1. v liště nástrojů kliknout na tlačítko `Load` v sekci `Metadata` – zobrazí se formulářové okno,
2. vybrat příslušný soubor metadat ve formátu `JSON`,
3. potvrdit výběr.

V případně, že jsou metadata validní, načtou se. Jejich zobrazení ve *stromové struktuře metadat* se provádí kliknutím na tlačítko `Show Tree` v sekci `Metadata` lišty nástrojů.

Přidání jazykových konstrukcí

V závislosti na charakteru jazykové konstrukce ji lze přidat buď:

- prostřednictvím prvků v sekci `Tools` lišty nástrojů – pro jazykové konstrukce, které nejsou závislé na metadatech, tj. metoda `Current Date` (přidání aktuálního data) a direktiva `include` (vložení podšablony),
- nebo přes stromovou strukturu metadat (`Metadata Tree View`) – pro jazykové konstrukce, které jsou závislé na metadatech, tj. elementární dosazení dat, metoda `Format Date` (formátování data) a direktivy `foreach` (opakovací blok) a `tableforeach` (opakovací blok s tabulkou).

Postup přidání jazykové konstrukce pro metodu `Current Date` je následovný:

1. v sekci `Tools` lišty nástrojů kliknout na galerii `Methods`,
2. ve výběru kliknout na tlačítko `Current Date` – zobrazí se formulářové okno,
3. v případě potřeby jiného formátu data, než je výchozí hodnota (`dd.MM.yyyy`), zaškrtnout políčko `Format Date` a specifikovat formát data,
4. potvrdit tlačítkem `Add`.

Pro přidání jazykové konstrukce pro direktivu `include` se postupuje následovně:

1. v sekci `Tools` lišty nástrojů kliknout na galerii `Directives`,
2. ve výběru kliknout na tlačítko `Include Template` – zobrazí se formulářové okno,
3. kliknout na tlačítko `Choose File` pro výběr podšablony dokumentu,
4. vybrat příslušný soubor podšablony ve formátu Microsoft Word (`doc` či `docx`),
5. potvrdit výběr souboru,
6. potvrdit přidání direktivy tlačítkem `Add`.

U jazykových konstrukcí přidávaných přes stromovou strukturu metatadat (`Metadata Tree View`) je postup následovný:

1. ve stromové struktuře metadat zvolit příslušný atribut/objekt – v sekci `Details` se zobrazí detaily a možné metody či direktivy, které lze použít,
2. pro přidání jazykové konstrukce pro atribut (elementární dosazení dat) bez dalších metod kliknout na tlačítko `Add`,
3. v případě atributu typu `datetime` lze zvolit formát data následovně:
   (a) zaškrtnout políčko `Format Date` v sekci `Details`,
   (b) specifikovat formát data,
   (c) potvrdit tlačítkem `Add`,
4. v případě objektu typu `kolekce` lze přidat direktivu `foreach` následovně:
   (a) v seznamu direktiv vybrat `Foreach` v sekci `Details`,
   (b) zvolit název řídící proměnné (`Controlling variable`),
   (c) potvrdit tlačítkem `Add`,
5. v případě objektu typu `kolekce` lze přidat direktivu `tableforeach` následovně:
   (a) v seznamu direktiv vybrat `Table Foreach` v sekci `Details`,
   (b) zvolit název řídící proměnné (`Controlling variable`),
   (c) potvrdit tlačítkem `Add` – zobrazí se formulářové okno s konfigurací inicializační tabulky,
   (d) zvolit hlavičku inicializační tabulky,
   (e) zvolit hodnoty inicializační tabulky,
   (f) potvrdit tlačítkem `Add`.

Pokud se v šabloně nachází direktiva `foreach` a uživatel přesune kurzor v šabloně dokumentu do těla této direktivy, přegeneruje se stromová struktura metadat tak, aby obsahovala pouze ty atributy/objekty, které je možné do těla této direktivy vložit.

Uživatel má také možnost sledovat aktuální syntaktické chyby v šabloně dokumentu. Pro tento účel slouží tlačítko `Show Errors` v sekci `Tools` lišty nástrojů. Po kliknutí na toto tlačítko se zobrazí *panel syntaktických chyb*.

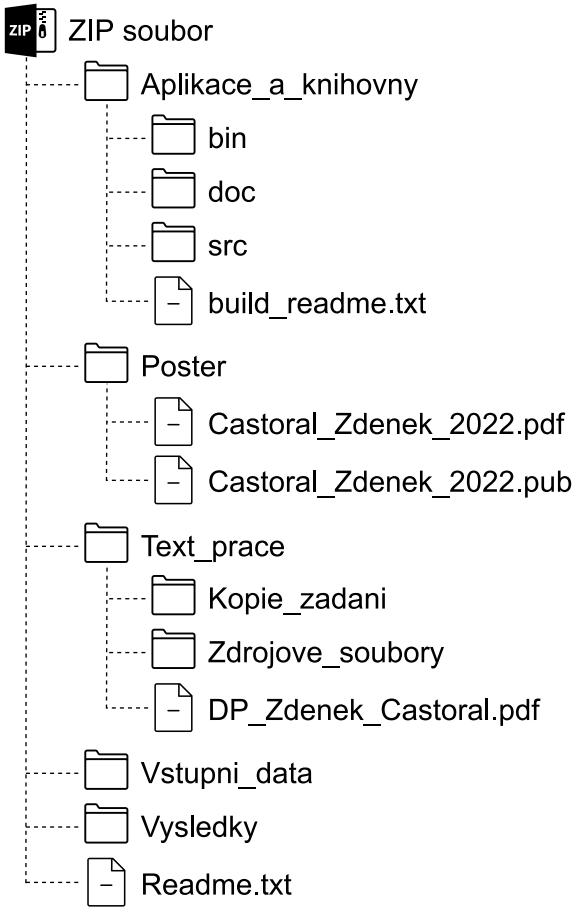
N.2.3 Generování nového dokumentu

Uživatel má možnost spustit generování nového dokumentu z aktuálně otevřené šablony. Postup je následovný:

1. kliknout na tlačítko `Generate` v sekci `Generation` lišty nástrojů – zobrazí se formulářové okno,
2. kliknout na tlačítko `Choose File` v sekci `Data File` pro načtení souboru vstupních dat,
3. vybrat příslušný soubor vstupních dat ve formátu `JSON`,
4. potvrdit výběr souboru,
5. kliknout na tlačítko `Choose File` v sekci `Output Document` pro zvolení cesty uložení vygenerovaného dokumentu,
6. vybrat příslušnou cestu pro uložení vygenerovaného dokumentu ve formátu `docx`,
7. potvrdit výběr cesty,
8. vygenerovat nový dokument tlačítkem `Generate`.

O Obsah ZIP souboru

Na obrázku O.1 je uvedena základní struktura obsahu odevzdávaného ZIP souboru. V této struktuře nejsou uvedeny všechny soubory a adresáře, ale pouze ty hlavní.



Obrázek O.1: Základní struktura ZIP souboru

O.1 Popis obsahu ZIP souboru

- Adresář `Aplikace_a_knihovny` – obsahuje zdrojové soubory, konfigurační soubory, spustitelné soubory softwarového produktu, knihovny třetích stran, vygenerovanou dokumentaci ze zdrojového kódu a návod na sestavení a spuštění. Je strukturován následovně:

  - adresář `bin` – obsahuje spustitelné soubory softwarového produktu,
  - adresář `doc` – obsahuje dokumentaci vygenerovanou ze zdrojového kódu,
  - adresář `src` – obsahuje zdrojové soubory (včetně souborů řešení `DocsGen.sln` a jednotlivých projektů: `GenLib`, `GenTrigger`, `TemplateBuilder` a `DocsGenTest`), konfigurační soubory, soubory použité k testování a knihovny třetích stran,
  - soubor `build_readme.txt` – textový soubor s popisem sestavení a spuštění.
- Adresář `Poster` – obsahuje soubory posteru ve formátech `PDF` a `PUB`.
- Adresář `Text_prace` – obsahuje text práce včetně všech zdrojových souborů textu a kopii zadání diplomové práce. Je strukturován následovně:
  - adresář `Kopie_zadani` – obsahuje kopii zadání diplomové práce,
  - adresář `Zdrojove_soubory` – obsahuje zdrojové soubory textu diplomové práce,
  - soubor `DP_Zdenek_Castoral.pdf` – výsledný soubor diplomové práce.
- Adresář `Vstupni_data` – obsahuje ukázkové šablony dokumentů, ukázkový soubor vstupních dat a metadat.
- Adresář `Vysledky` – obsahuje vygenerované dokumenty z ukázkových šablon dokumentů a vstupních dat.
- Soubor `Readme.txt` – je textový soubor s popisem obsahu ZIP souboru.