Mendelova univerzita v Brně
Provozně ekonomická fakulta

Knihovna pro práci se soubory formátu Office Open XML

Diplomová práce

Vedoucí práce:

doc. Ing. Oldřich Trenz, Ph.D. **Bc. Jaroslav Košťál**

Brno 2015

Chtěl bych na tomto místě poděkovat vedoucímu diplomové práce doc. Ing. Oldřichu Trenzovi, Ph.D. za odborné vedení a cenné připomínky při zpracování diplomové práce.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Knihovna pro práci se soubory formátu Office Open XML**
vypracoval/a samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací.*

Jsem si vědom/a, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.

Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 21. května 2015 _____

Abstract

Košťál, J. *A Library for Working with Office Open XML Files.* Diploma Thesis. Brno, 2015.

The thesis deals with a library for working with Office Open XML Files – spreadsheet files –, with its analysis, suggested layout, and implementation. It includes a specification of a standard spreadsheet file and of ways of creating such files. Based on studied theory, a layout is suggested for a library for reading and recording data, and for implementation of such a library in the C# language, which offers interface suitable for working with this type of documents.

Keywords

Office Open XML, OOXML, .NET, C#, library, spreadsheet files, interface

Abstrakt

Košťál, J. *Knihovna pro práci se soubory formátu Office Open XML.* Diplomová práce. Brno: Mendelova univerzita v Brně, 2015.

Práce se zabývá analýzou, návrhem a implementací knihovny pro práci se soubory formátu Office Open XML, respektive soubory tabulkových dokumentů. Zahrnuje specifikaci standardu dokumentů tabulkových kalkulátorů a možností vytváření těchto dokumentů. Na základě nastudované teorie je postaven návrh vlastního řešení knihovny pro čtení a zápis a implementace knihovny v jazyce C#, poskytující rozhraní pro práci s tímto typem dokumentů.

Klíčová slova

Office Open XML, OOXML, .NET, C#, knihovna, tabulkové dokumenty, rozhraní

Obsah

1 Úvod

Motivací pro práci zabývající se studiem tabulkových dokumentů formátu Open Office XML a vytvořením knihovny pro práci s tímto formátem je především rozšíření formátu OOXML ve firemní sféře a kompatibilita s nejpoužívanějšími balíky kancelářských programů. Programové vytváření, čtení a zápis obsahu tabulkových dokumentů umožňuje například propojení knihovny s existujícím výrobním systémem a generování reportů ve vysoce čitelné a stylovatelné podobě, určené pro management podniku. Vypovídací hodnota vytvořených dokumentů pak umožňuje poměrně efektivní řízení procesů firmy propojených s daným výrobním systémem.

Výhodou formátu OOXML je jeho aplikační nezávislost, tedy přenositelnost mezi různými aplikacemi pro práci s kancelářskými dokumenty a možnost jeho zobrazení nebo editace bez nutnosti použití původní aplikace, pomocí které byl vytvořen. To je dané jeho zařazením do kategorie otevřených formátů, jejichž definice je veřejně přístupná softwarovým vývojářům a umožňuje tak zahrnout plnou podporu tohoto typu formátu do jejich aplikací. Práce s tímto typem formátu je komfortní z vývojářského hlediska také díky jeho vnitřní datové struktuře. Jak již jeho název napovídá, jedná se o formát postavený na technologii značkovacího jazyka XML a výsledný soubor dokumentu je vlastně sada XML souborů zkomprimovaná formátem ZIP.

Tato práce je rozdělena na teoretickou a praktickou část a to s ohledem na logické členění podle nastudované teorie zabývající se potřebnou problematikou a vlastním praktickým řešením a přínosem práce.

Teoretická část se zabývá standardem XML a především z něj vycházejícího standardu Office Open XML, důvody jeho vzniku a rozšíření. Dále především způsobem organizace dat pro tabulkové dokumenty podle tohoto standardu, tedy jejich strukturou a způsobem, jak je s těmito daty dále pracováno. Pro srovnání později vytvořené vlastní knihovny zahrnuje tato teoretická část přehled existujících knihoven určených pro práci s formátem OOXML, jejich funkce a práce s nimi a z toho plynoucí výhody a omezení. Zkoumána je uživatelská přívětivost a způsob generování souborů. Na teoretickém základu položeném předchozími kapitolami je vystavěn návrh řešení podoby vlastní knihovny. Tento krok zahrnuje objektový návrh a návrh rozhraní knihovny s ohledem na požadavky vytváření, čtení a zápisu souborů formátu OOXML tabulkových dokumentů.

Praktická část je zaměřena na samotnou implementaci knihovny na základně znalostí získaných v teoretické části a analýze možného řešení problému. Programovacím jazykem byl zvolen jazyk C#, který je přirozeným východiskem pro programovou realizaci knihovny podporující cílový formát OOXML. Tato část práce popisuje postup řešení praktické části a jednotlivé funkční části knihovny. Dále proces programového vytváření, čtení či zápisu souborů tabulkových dokumentů a možností jejich grafického stylování. Během vytváření knihovny bylo třeba kód optimalizovat z důvodů rychlejšího běhu, respektive zpracování nebo generování dokumentů, a proto je tento postup optimalizace

zahrnut v této práci spolu se srovnáním s existujícími knihovnami s ohledem na zvolené parametry. Výše popsané je zahrnuto v kapitole o testování.

Závěr práce je věnován zhodnocení dosaženého řešení a stanoveného cíle a diskuzi dalších možných rozšíření.

2 Cíl práce

Cílem práce je analýza, návrh a implementace knihovny pro práci se soubory formátu Office Open XML. Jejím výstupem je knihovna pracující s tabulkovým formátem Open Office XML dokumentů, implementovaná v jazyce C#, se zaměřením na tvorbu, čtení a zápis souborů v daném formátu. Knihovna bude poskytovat rozhraní pro přístup k prvkům dokumentu podle zvyklostí programu Microsoft® Excel (pracovní listy, sloupce, řádky, buňky) a umožňovat čtení jejich dat. Při vytváření nebo zápisu dokumentu bude knihovna umožňovat vkládání dat do výše zmíněných prvků a také jejich grafické stylování. Specifickým požadavkem je možnost šablonování, kdy se do dokumentu připraveného podle předepsaného formátu automaticky vygenerují data z předaných datových typů.

3 Analýza

Přirozená provázanost formátu Open Office XML, vycházejícího z technologie XML je motivací pro studium samotné technologie XML a to především principů formátu XML, čerpajících z [1] a [2], které budou dále použity pro hlubší pochopení principů použitých ve formátu OOXML [3]. Co bylo motivací pro vznik a vývoj standardu OOXML a jaká je struktura a organizace dat uvnitř souborů splňujících jeho definici, je rozebráno v následujících kapitolách této práce.

3.1 Formát XML

Formát XML (zkratka pro eXtensible Markup Language, tedy rozšířený značkovací jazyk) byl vyvinut pod záštitou World Wide Web Consorcium (W3C) v roce 1996 jako formát pro reprezentaci a přenos obecných dokumentů a dat. Formát vychází ze staršího standardu SGML (Standard Generalized Markup Language). Standard definuje syntaxi struktury dokumentu určenou pomocí značek jazyka XML, avšak význam značek nebo jejich sémantika definovány standardem nejsou. Definice významu a sémantiky značek použitých v dokumentu je až věcí aplikace zpracovávající tento dokument a formát se díky díky tomu stává nezávislý právě pro potřeby reprezentace a přenosu obecných dokumentů. Cíle, kterými se řídil návrh XML, jsou [2]:

- XML musí být použitelný v rámci internetu
- XML musí podporovat širokou škálu aplikací
- XML musí být kompatibilní s formátem SGML
- Musí být snadné psát programy, které zpracovávají XML dokumenty
- Počet volitelných funkcí v XML musí být minimální, ideálně nulový
- XML dokumenty by měly být pro člověka čitelné a jasné
- Konstrukce XML by měly být vytvořeny snadno
- Konstrukce XML musí být formální a stručné
- XML dokumenty musí být snadné na vytvoření
- Stručnost XML značkování má minimální význam

Tyto cíle vedly ke splnění požadavků na reprezentaci a přenos obecných dokumentů a dat, přičemž formát XML (případně formáty, které z něho vycházejí) je díky tomu v této oblasti značně rozšířený.

Dokument splňující definici formátu XML musí začínat hlavičkou definující použitou verzi XML (1.0 nebo 1.1) a kódování (utf-8 a podobně). Může obsahovat také položku specifikující požadavky na strukturu dokumentu neboli gramatiku

označovanou schématem dokumentu. Dokument je možné označit za validní pouze v případě, že odpovídá svému schématu. Za hlavičkou musí následovat kořenový element dokumentu, který je obsažen právě jednou, a dále již elementy značkující dokument.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <informations>
3   <customer id="27840">
4     <name>
5       <firstName>Jaroslav</firstName>
6       <lastName>Košťál</lastName>
7     </name>
8     <address>
9       ...
10    </address>
11  </customer>
12 </informations>
```

Obr. 1 Struktura jednoduchého XML dokumentu.

Příklad struktury jednoduchého XML dokumentu nabízí přiložený obrázek 1. Dokument obsahuje povinnou hlavičku a kořenový element pojmenovaný `informations` . Každý element v XML je obecně vymezen ostrými závorkami a je identifikován jménem uvedeným uvnitř vymezujících závorek. Obsah elementu je uveden mezi otevírací (start-tag) a uzavírací závorkou (end-tag) a může obsahovat další zanořené elementy a jiný obsah dokumentu. Například `<name>` je v příkladu start-tag a `</name>` end-tag, tedy element popisující jméno zákazníka, který dále tento popis obsahuje v podobě zanořených elementů `firstName` a `lastName`. Pokud je obsah elementu prázdný, jedná se o takzvaný prázdný element (empty-element) a zápis lze zkrátit na `<name/>`. Prázdný element může v dokumentu zastávat místo, kam se má později vložit další obsah a podobně. Formát XML umožňuje specifikovat elementy použitím takzvaných atributů. Atributy se vkládají do otevíracích závorek zápisem `název="hodnota"`, přičemž hodnota musí být vždy uvozena uvozovkami nebo apostrofy. Element `customer` popisující zákazníka obsahuje atribut identifikačního čísla. Jedná se o alternativní zápis této informace, stejně tak by se dalo identifikační číslo uvést přímo elementem `<id>27840</id>` uvnitř elementu `customer`. Při programovém zpracování dokumentu představuje přepis informace pomocí atributu nespornou výhodu. U všech elementů odpovídajících zákazníkům se vyhledá atribut konkrétního identifikačního čísla zákazníka a programově se nemusí zanořovat hlouběji do struktury pro nalezení požadované informace. Specifickými částmi vkládanými do dokumentů jsou XML komentáře, instrukce pro zpracování pro konkrétní aplikaci nebo sekce CDATA, která říká, že její obsah nemá být interpretován (například pokud je třeba vložit část, která by mohla být chápána jako část XML popisu, ale to není žádoucí).

V dalším textu bude popsána struktura a organizace dat formátu OOXML, a protože formát vychází z XML, obsahuje vlastní sadu značek, pojmenování elementů, jejich atributů a podobně. Pokud chceme v XML reprezentovat jistý druh informace, vytváříme právě vlastní sadu značek, takzvaný markup vocabulary (značkovací slovník). Pokud je v dokumentu třeba použít více nezávislých sad značek, respektive kombinovat schémata, může dojít ke konfliktu, protože použitá jména nemusí být jedinečná. Problém řeší samostatný standard (není zahrnut přímo ve standardu XML) XML namespaces (jmenný prostor XML). Řešení pomocí tohoto standardu spočívá v uvedení prostoru jmen s jedinečnou identifikací pomocí URI referencí do dokumentu a explicitním zařazením elementů do

```
<?xml version="1.0" encoding="utf-8"?>
<i:informations xmlns:i="http://schemas.informations.com/informations"
                xmlns:d="http://schemas.informations.com/data">
  <i:customer id="27840">
    <i:name>
      <i:firstName>Jaroslav</firstName>
      <i:lastName>Košťál</lastName>
    </i:name>
    <i:address>
      ...
    </i:address>
    <d:data>
      ...
    </d:data>
  </i:customer>
</i:informations>
```

Obr. 2 Struktura XML dokumentu kombinujícího více jmenných prostorů.

odpovídajících prostorů jmen uvedením unikátního prefixu ke jménu elementu. Na obrázku 2 je uveden příklad použití dvou schémat pro informace a pro data, identifikovaných jedinečným URI a prefixem. K informacím o zákazníkovi byla přidána další data, která by mohla obsahovat duplicitní jména použitá již ve schématu pro informace, ale která je třeba reprezentovat odlišně. Proto jsou elementy zařazené do jmenného prostoru informací rozšířeny o prefix `i:` a o prefix `d:` potom elementy zařazené do jmenného prostoru dat.

3.2 Vznik a vývoj standardu OOXML

Formát Office Open XML vznikl v roce 2006 jako důsledek stále vyšších požadavků na standardizaci formátů pro ukládání a reprezentaci dat, manipulaci s nimi a aplikační přenositelnost [5]. V rámci podpory standardizace byl formát označen jako otevřený a jeho specifikace vydána veřejně, což umožňuje využití formátu a zahrnutí podpory do vlastní aplikace bez nutnosti licencování ze strany tvůrce standardu.

OOXML byl vyvinut především firmou Microsoft Corporation, ale na jeho definici a vývoji se podílela také řada dalších společností (Apple, Intel, atd.). Firma Microsoft nový formát využila ve svém kancelářském balíku Microsoft® Office 2007 pro ukládání dokumentů. Do uvedení verze balíku Office 2007 byl používán starší typ ukládání a přenosu dokumentů ve formě binárních souborů, které měly problém se zpětnou kompatibilitou a obecně manipulací se souborem bez aplikace, ve které byly vytvořeny. Pro dokumenty kancelářského balíku Office byly typické přípony podle typu programu. Textové dokumenty aplikace MS Word byly identifikovány příponou `.doc`, tabulkové dokumenty aplikace MS Excel příponou `.xls` a dokumenty prezentací aplikace MS PowerPoint končily příponou `.ppt`. Nový formát počínající verzí MS Office 2007 již přináší přechod od binárního kódování obsahu souboru k popisu obsahu pomocí formátu XML [7]. Výsledný dokument je sadou XML souborů popisujících data dokumentu a tato sada je zkomprimována formátem ZIP do jediného souboru. Za soubory vytvořenými podle nového formátu dnes uživatel najde přípony `.docx` pro textové dokumenty, `.xlsx` pro tabulkové dokumenty a `.pptx` pro dokumenty prezentací.

Dokumenty vytvořené sadou Office byly a jsou značně rozšířené, což je dáno především rozsahem nasazení samotného operačního systému MS Windows v soukromém i firemním sektoru. Nový otevřený formát tak přináší pro velké množství uživatelů i vývojářů aplikací pracujících s dokumenty následující výhody:

- Nezávislost na platformě a aplikaci
- Snadná manipulace s dokumenty (sdílení, modifikace obsahu)
- Otevřená specifikace všech součástí (OOXML, XML, ZIP)
- Efektivnost (menší velikost souboru oproti binární verzi)
- Robustnost (odolnost vůči chybám)
- Bezpečnost (soubory jsou transparentní)
- Kompatibilita

Podporu pro tato tvrzení přináší následující kapitola, kde je blíže rozebrána struktura a organizace dat formátu OOXML.

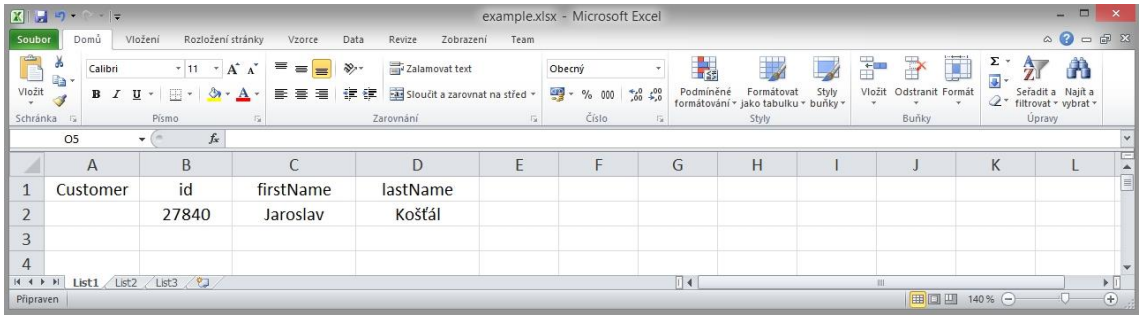
3.3 Struktura a organizace dat OOXML

Formát Office Open XML je specifikován pomocí standardu ECMA-376 [3]. Tento standard má několik tisíc stran textu a definuje vlastní sadu značek (markup vocabulary) specifickou pro formát OOXML (textové, tabulkové a prezentační dokumenty), dále strukturu dokumentů a způsob jejich zabalení, tedy vytvoření jediného souboru. Specifikuje také požadavky kladené na uživatele a vývojáře formátu. Části lze chápat také jako normativní a non-normativní (informativní),

tedy části popisující důsledně standard a části ilustrující na příkladech a diagramech praktické použití. Standard ECMA-376 podrobněji [4]:

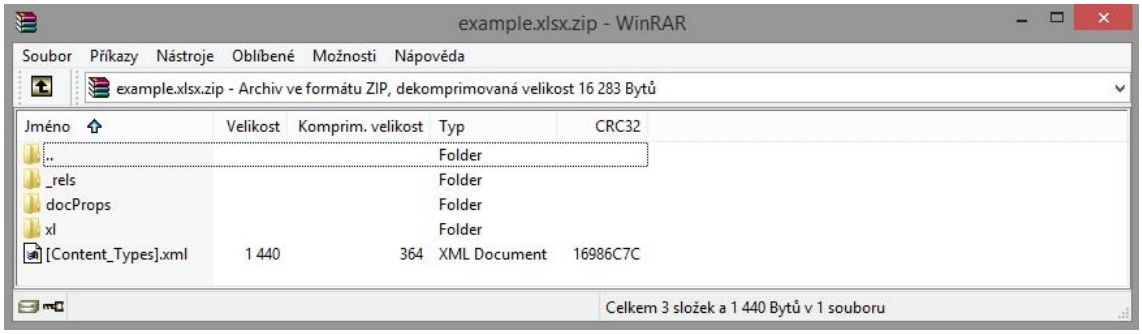
- definuje slovník, konvence značení a zkratky,
- shrnuje tři základní značkovací jazyky a podpůrné značkovací jazyky,
- stanovuje podmínky pro shodu se standardem a pokyny pro požadovanou platformní a aplikační nezávislost,
- popisuje omezení vztahující se ke každému dokumentu,
- definuje formát balení podle OPC (Open Packaging Conventions),
- popisuje funkce jednotlivých značkovacích jazyků, poskytuje jejich kontext,
- definuje každý element a atribut, jejich hierarchii rodič/potomek.

Jak bylo uvedeno, pro každý typ dokumentu standard popisuje základní značkovací jazyk a podpůrné značkovací jazyky. Každý základní značkovací jazyk zachovává základní strukturu, ale liší se v použitém značkování podle charakteristických vlastností daného typu. Textové dokumenty jsou popisovány značkovacím jazykem WordprocessingML a obsah je strukturován podle odstavců, tabulek a jiných vložených objektů. Tabulkové dokumenty popisované značkovacím jazykem SpreadsheetML jsou obsahově strukturovány podle pracovních listů a ty jsou na nižší úrovni strukturovány na řádky a sloupce popisující buňky pracovního listu. Dokumenty prezentací popsaných značkovacím jazykem PresentationML jsou strukturovány dle jednotlivých slajdů prezentace, na nižších úrovních popsané svým obsahem.
Tato práce se zaměřuje na práci s tabulkovým typem dokumentů, a proto bude podrobněji rozebrána struktura tohoto typu dokumentů a popsán značkovací jazyk SpreadsheetML. Popis a postup dále vychází z dokumentů [3]. Pro účely zkoumání struktury a popisu lze nejlépe využít reálného souboru tabulkového dokumentu, který byl vytvořen pomocí aplikace MS Excel kancelářského balíku MS Office 2010. Vizuální podobu souboru `example.xlsx` v prostředí aplikace znázorňuje obrázek 3.



Obr. 3 Tabulkový dokument aplikace MS Excel.

Při znalosti vytváření souborů podle standardu ECMA-376 je možné uvedený soubor upravit tak, aby byla jeho obálka pro uživatele transparentní a mohl přistupovat k vnitřní sadě XML souborů [6]. Toho je dosaženo jednoduchou záměnou přípony souboru. Na konec názvu souboru (za příponu `.xlsx`, případně přímo místo ní) se vepíše přípona `.zip` a soubor je operačním systémem (předpokládá se operační systém MS Windows) považován za archiv, který je možno extrahovat a prohlížet vhodným nástrojem (obrázek 4).



Obr. 4 Vnitřní struktura tabulkového dokumentu.

Typická struktura souboru jednoduchého tabulkového dokumentu včetně podadresářů je následující:

```
/[Content_Types].xml
/_rels/.rels
/docProps/app.xml
/docProps/core.xml
/xl/_rels/workbook.xml.rels
/xl/theme/theme1.xml
/xl/worksheets/sheet1.xml
/xl/worksheets/sheet2.xml
/xl/worksheets/sheet3.xml
/xl/sharedStrings.xml
/xl/styles.xml
/xl/workbook.xml
```

Z důvodu většího množství získaných dat, se obsahu popisovaných souborů věnuje až část Přílohy na konci práce. V textu je možné narazit na některé pojmy vztahující se k pojmenování částí tabulkového dokumentu, které standard definuje takto:

Buňka (cell)
Oblast, kde se protíná řádek a sloupec, obsahující textová nebo numerická data nebo vzorce. Buňka obsahuje řadu vlastností, jako formátování obsahu, zarovnání, font, barvu, okraje a podobně. Buňku je možné identifikovat kombinací jména sloupce a řádku, na jejichž průsečíku se nachází. Například buňka umístěná na druhém řádku druhého sloupce má označení `B2`.

Sloupec (column)
Vertikální skupina buněk. Každý sloupec je pojmenován abecedně, přičemž sekvence pojmenování pokračuje jako `A–Z`, `AA–AZ`, a tak dále.

Řádek (row)
Horizontální skupina buněk. Každý řádek je pojmenován numericky a sekvence pojmenování začíná číslem `1`.

Pracovní list (worksheet)
Dvoudimenzionální mřížka buněk organizovaných do řádků a sloupců.

Pracovní sešit (workbook)
Kolekce pracovních listů.

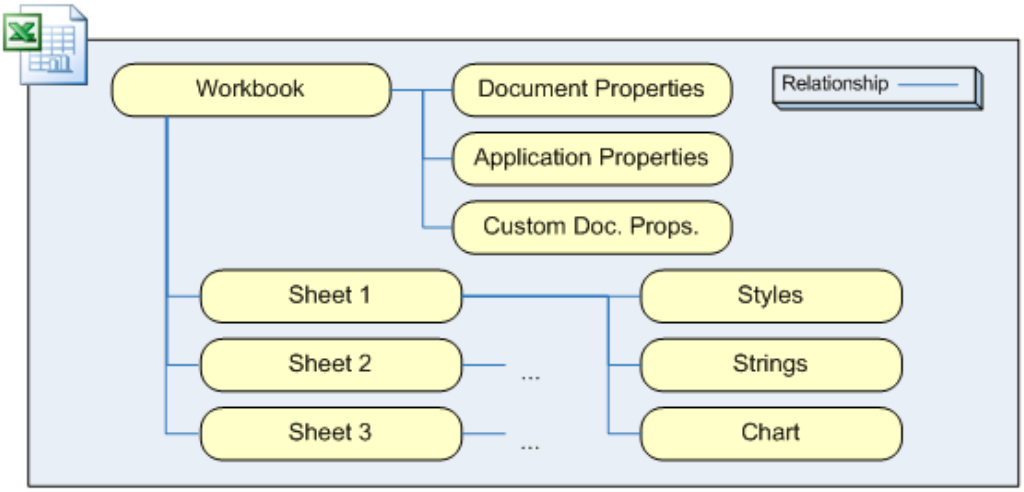
Kořen každého Open Office XML dokumentu musí povinně obsahovat soubor pojmenovaný `[Content_Types].xml`, který identifikuje všechny části obsažené v dokumentu (pracovní list, styly, a podobně) a uvádí k nim relativní cestu v rámci adresářové struktury. Každá část musí mít povinně definován svůj typ:

```
<Override PartName="/xl/worksheets/sheet1.xml" ContentType=
"application/vnd.openxmlformats-officedocument.spreadsheetml
.worksheet+xml"/>
```

Ve složkách `/_rels/` jsou obecně umístěny soubory končící příponou .rels a ty popisují vztahy mezi částmi struktury dokumentu (obrázek 5). To slouží k vyhledání potřebných součástí ve struktuře a vytvoření odpovídajících referencí. Složka `/_rels/` je vytvářena ve stejné složce, ve které je umístěna součást, pro niž je vztah definován, a jméno souboru je od této části odvozeno. V kořenové složce dokumentu je obsažen soubor s příponou `.rels` bez názvu. V úrovni adresáře

`/xl/`, kde soubor popisuje vztahy k sešitu (workbook), je pojmenován od odvozené části, tedy `workbook.xml.rels`.

```
<Relationship Id="rId1" Type="http://schemas.openxmlformats.
org/officeDocument/2006/relationships/officeDocument" Target
="xl/workbook.xml"/>
```



Obr. 5 Vazby mezi částmi tabulkového dokumentu.
Zdroj: Introducing the Office 2007 Open XML File Formats, 2006.

Vlastnosti dokumentu jsou uchovávány konzistentně na jednom místě ve složce `/docProps/`, nezávisle na typu dokumentu (textový, tabulkový, prezentace). Vlastnosti samotného dokumentu jsou pak logicky rozděleny do tří souborů. Výhodou takového přístupu je oddělení dat od ostatních dat dokumentu a jasné umístění ve struktuře. Soubor `core.xml` obsahuje informace identifikující dokument, jako jeho název, předmět, autor atd. Vlastní informace přidané uživatelem nebo vývojářem jsou uchovány v souboru `custom.xml`. Specifické vlastnosti pro daný dokument pak uchovává samostatný soubor `app.xml` a podle typu dokumentu obsahují informace například o počtu použitých pracovních listů nebo verzi aplikace.

Závislá na typu dokumentu, respektive použitém základním značkovacím jazyce, je poslední složka obsažena v kořenovém adresáři. Pro textové dokumenty se jedná o složku `/word/`, pro dokumenty prezentací `/ppt/`. Tabulkové dokumenty mají svoji strukturu a data uložené ve složce s názvem `/xl/`.

Soubor `/xl/workbook.xml` obsahuje informace definující pracovní sešit aplikace. Obsahuje například verzi souboru, pojmenování a identifikátory pracovních listů a definuje vlastnosti použité aplikací pro výpočty:

```
<sheet name="List1" sheetId="1" r:id="rId1"/>
```

Styly použité v dokumentu, jako fonty, barvy písma nebo buněk, jejich okraje a podobně, uchovává soubor `/xl/styles.xml`. Ten vepíše souhrnnou informaci o použitém stylu (případně více stylech) do XML elementů a jednotlivé části popisující font, výplň, okraje odkáže indexem do příslušných pod-elementů, které vytvoří k daným částem. Zápis popisující jeden z použitých stylů může vypadat následovně:

```
<xf numFmtId="0" fontId="0" fillId="1" borderId="0"/>
```

Soubor obsahuje mimo jiné element `fills` popisující výplň buněk, proto lze dohledat podle identifikátoru příslušnou výplň. V tomto případě šedou barvu podkladu `gray125` (jedná se o pojmenovanou barvu definovanou standardem):

```
<fills count="2">
   <fill><patternFill patternType="none"/></fill>
   <fill><patternFill patternType="gray125"/></fill>
</fills>
```

Podobně lze v souboru dohledat další položky použitého stylu zastoupené odpovídajícími identifikátory a princip je platný také u většiny dalších souborů odkazujících své položky přes identifikátor.

V rámci optimalizace načítání a ukládání dat dokumentu zavádí standard tabulku sdílených řetězců (shared strings). Struktura umístěná v souboru `sharedStrings.xml` představuje seznam unikátních řetězců použitých v pracovním sešitě. Každý řetězec zde má svoji pozici (index reprezentovaný pořadím v seznamu), přes který je odkazován na své umístění v pracovních listech a v seznamu je obsažen unikátně právě jednou. Výhoda přístupu spočívá v omezení duplicit v dokumentech s velkým množstvím stejných řetězců. Je vhodné podotknout, že čísla (pokud nejsou explicitně reprezentována jako řetězec) na tomto místě ukládána nejsou:

```
<si>
   <t>Customer</t>
</si>
<si>
   <t>id</t>
</si>
```

Obsah a význam adresáře `/xl/_rels/` popisujícího vztahy částí k pracovnímu sešitu byl popsán výše.

Aplikace MS Excel obsahuje v základu komplexní předdefinovanou sadu stylů v adresáři `/xl/theme/`, kterými lze obsah vzhledově upravovat. Tato sada je podobná datům souboru `/xl/styles.xml`, avšak zde definované sady stylů nemusí být nikde použity. Jedná se o předdefinované záznamy, které může

aplikace využít přímo, tedy odkázat se na jejich index, a až v případě jejich použití je informace zapsána do souboru stylu dokumentu. Význam této komplexní předdefinované sady stylů spočívá ve vytvoření konzistentní vizuální podoby celého dokumentu.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<worksheet xmlns="http://schemas.openxmlformats.org/spreadsheetml/2006/main"
xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006" mc:
Ignorable="x14ac" xmlns:x14ac="http://schemas.microsoft.com/office/
spreadsheetml/2009/9/ac">
  <dimension ref="A1:D2"/>
  <sheetViews>
    <sheetView tabSelected="1" workbookViewId="0">
      <selection activeCell="A18" sqref="A18"/>
    </sheetView>
  </sheetViews>
  <sheetFormatPr defaultRowHeight="15" x14ac:dyDescent="0.25"/>
  <cols>
    <col min="1" max="1" width="11" customWidth="1"/>
    <col min="2" max="2" width="10.7109375" customWidth="1"/>
    <col min="3" max="4" width="14.28515625" customWidth="1"/>
  </cols>
  <sheetData>
```

Obr. 6 Soubor obsahující data pracovního listu (část 1).

Prozatím byla ve struktuře identifikována a popisována data či metadata, která úzce souvisí s obsahem vytvořeného tabulkového dokumentu, ale nedefinují obsah jeho buněk. Taková data jsou uchována v souborech `sheetX.xml` (kde `X` představuje číslo pracovního listu) v adresáři `/xl/worksheets/`. Příklad (obrázek 3) definoval celkem tři pracovní listy, přičemž obsah posledních dvou je prázdný a je popsána struktura prvního listu `/xl/worksheets/sheet1.xml` (obrázek 6 a obrázek 7).

Soubor pracovního listu vždy obsahuje kořenový element `worksheet` uvozující další elementy. Element `dimension` informuje o rozsahu použitých buněk, v příkladu u prvního listu se jedná o obdélníkovou oblast začínající buňkou `A1` a končící buňkou `D2`, zkráceně zapsáno jako rozsah `A1:D2`. Element `sheetView` definuje pohled na pracovní list, zda je list vybrán jako aktivní po spuštění a kde je umístěna aktivní buňka. Výchozí formátování listu, například výchozí šířka řádků, je definována v `sheetFormatPr` a podobnou funkci zastávají také elementy `cols` a `pageMargins`. Popis formátování sloupců vyplívá poměrně jasně ze zápisu: První sloupec, tedy s pojmenováním `A`, má nastavenu vlastní šířku 11 pixelů. Stejně tak sloupce v rozsahu `C` – `D` mají vlastní šířku.

```
15   <sheetData>
16     <row r="1" spans="1:4" x14ac:dyDescent="0.25">
17       <c r="A1" t="s"><v>0</v></c>
18       <c r="B1" t="s"><v>1</v></c>
19       <c r="C1" t="s"><v>2</v></c>
20       <c r="D1" t="s"><v>3</v></c>
21     </row>
22     <row r="2" spans="1:4" x14ac:dyDescent="0.25">
23       <c r="B2"><v>27840</v></c>
24       <c r="C2" t="s"><v>4</v></c>
25       <c r="D2" t="s"><v>5</v></c>
26     </row>
27   </sheetData>
28   <pageMargins left="0.7" right="0.7" top="0.78740157499999996" bottom="0.
78740157499999996" header="0.3" footer="0.3"/>
29 </worksheet>
```

Obr. 7 Soubor obsahující data pracovního listu (část 2).

Dle definice je pracovní list „dvoudimenzionální mřížka buněk organizovaných do řádků a sloupců“. Zanoření elementů, respektive vztah rodič/potomek v zápisu pomocí XML s touto definicí koresponduje. Data listu jsou uvozena elementem `sheetData`, který je dále dělen elementy `row` na jednotlivé řádky dat. Řádek obsahuje element buňky `c` (cell), který lze však také chápat jako sloupec (column), takže na této úrovni je již přesně identifikována buňka (průsečík řádku a sloupce). Atributy elementu dále upřesňují buňku, její styl a obsah. Atribut `r` (reference) popisuje její přesné umístění v rámci mřížky. Atribut `s` (style) odkazuje na index použitého stylu v souboru stylů `/xl/styles.xml`. Použití atributu `t` (type) slouží k určení typu dat obsažených uvnitř buňky. Pokud není atribut uveden, je obsah považován za numerickou hodnotu a je s ní tak dále zacházeno. Protože do buňky lze zapsat mimo numerických hodnot také vzorce a řetězce, umožňuje atribut umístění tohoto typu dat explicitně definovat. V kombinaci s tabulkou sdílených řetězců je nutné rozlišovat, kdy je hodnota uvnitř buňky numerická a kdy se pouze jedná o index do tabulky řetězců. V příkladu je typ buňky specifikován zápisem `t="s"` představujícím typ sdíleného řetězce. Hodnota buňky se obecně uvozuje elementem `v` a pro buňku `A1` je její hodnota `0`. Hodnota není interpretována jako numerická, ale jako index do tabulky řetězců, odkazující na řetězec `"Customer"`. Buňka `B2` naopak obsahuje přímo numerickou hodnotu `27840`. Tímto způsobem je možné definovat obsah celého tabulkového dokumentu a při znalosti výše uvedeného obsahu struktury dokumentu tento dokument programově vytvořit, číst i zapisovat.

Výčet elementů a jejich atributů použitých v souborech struktury, stejně jako samotná struktura souborů a adresářů, není kompletní a úplný výčet lze nalézt v aktuální definici standardu ECMA-376.

3.4 Existující knihovny a jejich funkce

Otevřenost standardu OOXML, jeho vývojářská přívětivost, platformní a aplikační nezávislost a další výhody oproti staršímu formátu kancelářských dokumentů vedly ke vzniku poměrně velkého množství knihoven pro práci s tímto novým formátem dat. V dalším textu budou představeny ty nejčastěji používané a uživateli nejlépe hodnocené. Z důvodu různých implementačních jazyků těchto knihoven a zaměření vlastní vytvořené knihovny na práci s dokumenty tabulkových kalkulátorů jsou ve výčtu obsaženy především knihovny podporující platformu .NET (C#, VB.NET, ASP.NET atd.) a zaměřené na práci pouze s novým typem tabulkových dokumentů (`.xlsx`/`.xlsm`). Cílem tohoto výčtu je také možnost pozdějšího srovnání s vlastní vytvořenou knihovnou.
Mezi možnosti knihoven a jejich funkcionalitu nejčastěji patří:

- vytváření Excel reportů,
- importování tabulkových dat z datových struktur (datové tabulky apod.),
- čtení, zápis a úpravy existujících souborů podporovaných formátů,
- převod mezi formáty,
- vytváření grafů,
- grafické stylování,
- filtrování a třídění nad daty.

Shrnout výhody a nevýhody placených a volně dostupných knihoven (obecně jakéhokoliv softwaru) je obtížné a záleží na daném úhlu pohledu. U komerčního softwaru, který vytváří tým placených, tedy finančně motivovaných odborníků, se předpokládá vyšší kvalita kódu i širší množství nabízených funkcí. Další výhodou komerčního softwaru je technická podpora ze strany výrobce, nicméně za volně dostupným softwarem může být naopak vybudována silná uživatelská komunita. Podobně nelze jednoznačně říci, že placený software bude moderní, aktuální či inovativní. Výhodou volně dostupného softwaru bývá často dostupnost zdrojových kódů a možnost jejich modifikace. Z pohledu bezpečnosti je dostupnost zdrojových kódů v kombinaci se silnou uživatelskou komunitou výhodná při nacházení a odstraňování chyb. Pro potenciální útočníky zase poskytuje možnost nalézt a využít bezpečnostní chybu ve svůj prospěch. V komerčním sektoru se pro hledání a odstraňování chyb softwaru musí vyhradit určité kapacity zdrojů. Nevýhodami plynoucími z komerčního vývoje a následných úprav v průběhu životního cyklu produktu (pokud není přímo na zakázku pro uživatele) je minimální možnost ovlivňovat jeho funkcionalitu. Když bude v základu softwaru chybět určitá část funkcionality požadovaná uživatelem, uživatel musí buď dokoupit modul, který ji implementuje, nebo čekat na její zahrnutí do budoucích verzí. Také šíření placeného softwaru ve firemním prostředí vytváří určité omezení, protože je třeba správně kalkulovat počet a typ zakoupených licencí

a potenciál budoucího rozšíření. Například nasazení placené knihovny v rámci vlastního komerčního produktu může představovat další komplikace. Podobně se však dá nahlížet na volně dostupné knihovny s licencí GPL. Licence vyžaduje použití stejné licence GPL také u produktu, který tyto knihovny využívá [8].

3.4.1 Placené knihovny

Zde uvedené knihovny jsou komerčním softwarem a jsou dostupné pouze po zaplacení licenčního poplatku za jejich používání. Cena za pořízení knihovny se často liší podle počtu licencí, délky jejich platnosti, verze knihovny (respektive nabízené funkcionality), platformy (.NET, Java, atd.) a typu licence (developer/site, OEM/small business). Aby měl potenciální zájemce možnost jejího vyzkoušení před zakoupením licence, nabízí často firmy jejich stažení pod Trial nebo Demo licencí. U Trial verze knihovny je často časově omezena její funkcionalita, po uplynutí povolené doby je produkt zablokován a je nabídnuto zakoupení plné verze. Demo verze knihovny bývá omezena přímo poskytovanou funkcionalitou (omezení počtu vygenerovaných dat, omezená podpora formátů a podobně).

Aspose

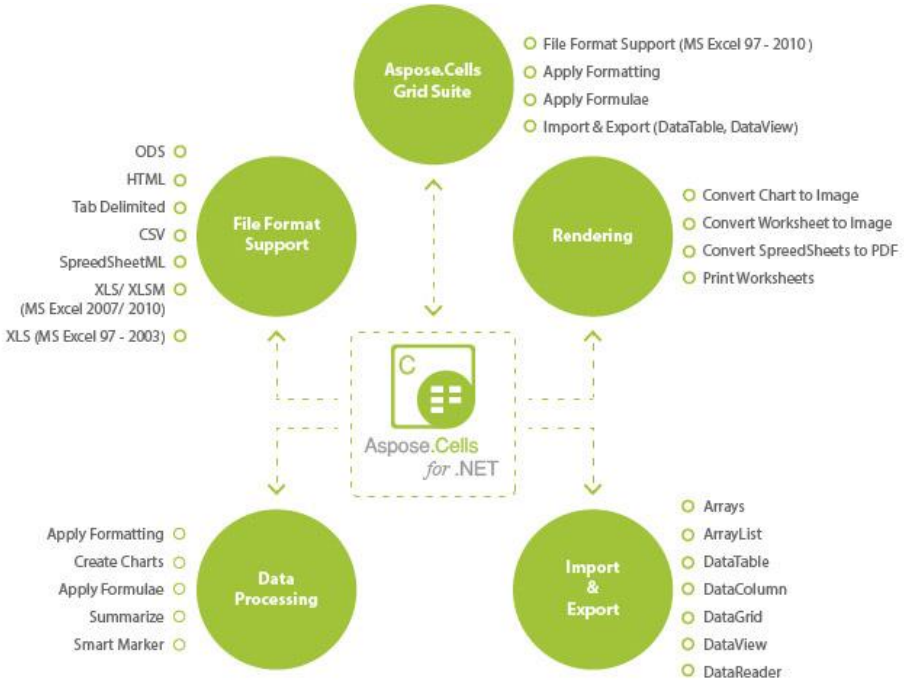
Firmy Aspose vyvíjí celou řadu produktů pro vývojáře na různých typech platforem, se zaměřením na velké množství typů souborových formátů, poskytujících API rozhraní pro práci s nimi. Hlavní platformy, pro které vyvíjí své produkty, jsou .NET, Java, Cloud, Android. Mezi zákazníky využívající software Aspose patří například společnosti 3M, Dell, JP Morgan, Nissan, DHL International, Oracle a mnohé další.

Knihovna Aspose.Total for .NET [9] zahrnuje komplexní balíček menších knihoven pracujících s různými typy souborových formátů a poskytuje tak komplexní řešení. Její součástí jsou knihovny pro práci s dokumenty kancelářského balíku MS Office: Aspose.Cells pro tabulkové dokumenty, Aspose.Words pro textové dokumenty, Aspose.Slides pro dokumenty prezentací. Podporuje navíc množství dalších formátů (PDF, generování čárových kódů a dalších). Nejlevnější licence pro malé podniky začíná na částce 2 499 $. Lze zakoupit také jednotlivé součásti pouze pro konkrétní typ formátu. Knihovna Aspose.Cells umožňuje vytvářet, číst, upravovat, konvertovat a tisknout tabulkové dokumenty bez nutnosti instalace aplikace MS Excel a její licenci lze zakoupit přibližně za částku 999 $. Aspose.Cells umí konkrétně pracovat s formáty XLS, XLSX, XLSM, XLTX/XLTM, HTML, CSV, ODS a PDF a programovacími jazyky platformy .NET (C#, VB.NET, ASP.NET, atd.) a jazyky PHP, Python a Mono. Množství funkcí je naznačeno na obrázku 8.

Produkty Aspose je možné vyzkoušet pod 30-ti denní dočasnou licencí (Trial), která po stanovenou dobu poskytuje plnou verzi produktu.

Jestliže jsou pominuty obecné výhody a nevýhody komerčního softwaru, pak výhodami knihovny jsou možnosti jejího nasazení a jednoduchá integrace, komplexnost, rychlost a nulová externí závislost. Žádná jiná knihovna neposkytuje

takové množství podporovaných formátů a funkcí. Ve srovnání s ostatními jí lze vytknout pouze vyšší cenu.



Obr. 8 Přehled funkcí knihovny Aspose.Cells
Zdroj: Oficiální stránky Aspose.Total for .NET [9].

GemBox

GemBox Software (Gemmeus d.o.o.) vyvíjí dva .NET produkty pro práci s kancelářskými dokumenty. Mezi jejich zákazníky patří Microsoft, NASA, Citigroup, Verizon, Swarovski a další. Knihovna GemBox.Document se zaměřuje na práci s textovými dokumenty a poskytuje možnost čtení, zápisu, konverze a tisku dokumentů. Podpora stejné funkcionality pro tabulkové dokumenty je implementována v knihovně GemBox.Spreadsheet [10] za cenu jedné vývojářské licence od 480 $. Podpora formátů knihovny opět obsahuje také starší verzi binárních souborů XLS a nové XLSX, dále ODS, CSV, HTML, PDF, XPS. Knihovnu je možné nasadit v .NET aplikacích s využitím jejího jednoduchého API rozhraní.

K dispozici jsou obě knihovny také ve volně dostupné verzi, přičemž konkrétně GemBox.Spreadsheet Free obsahuje omezení stanovující maximální počet řádků pracovního listu na 150 a maximální počet pracovních listů na pracovní sešit na 5.

Mezi výhody knihovny GemBox.Spreadsheet lze zařadit nasazení knihovny bez dalších licenčních poplatků (royalty-free deployment), tedy bez nutnosti platit poplatek za jakoukoliv vzniklou kopii aplikace využívající tuto knihovnu. Technickou podporu a rychlé odstraňování chyb má uživatel k dispozici po dobu 18 měsíců zdarma

Spire

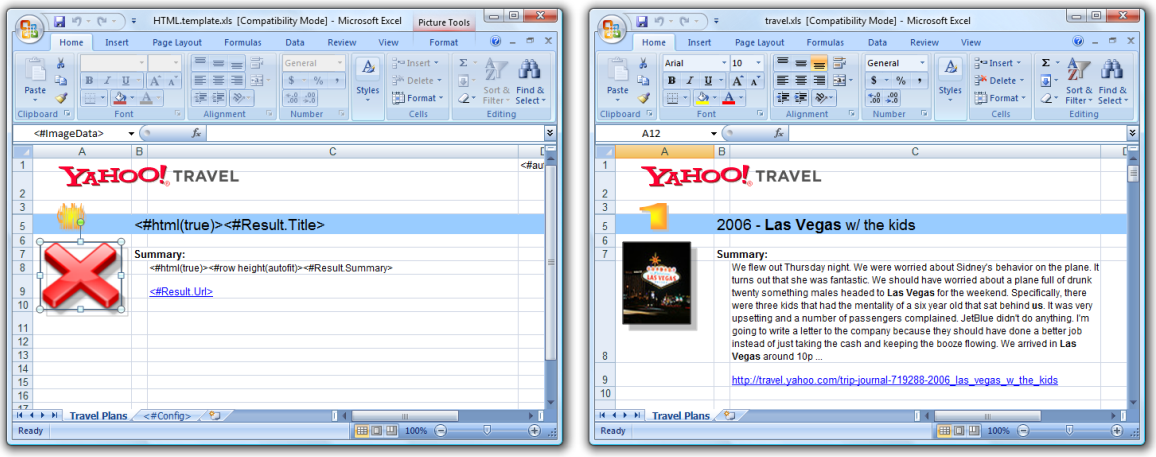
Spire je (podobně jako produkt Aspose) soubor knihoven určených pro nasazení na různých platformách, které poskytují jak komplexní funkcionalitu v podobě jednoho balíku pro větší množství formátů, tak možnost zakoupení a využití pouze některých z nich. Firma E-iceblue Co., Ltd., zabývající se jejím vývojem, obsluhuje zákazníky z řad velkých firem jako ABB, AT&T, IBM, Siemens, Philips a dalších.

Spire.Office for .NET (cena licence od 1899$) je balíček knihoven [11] pro práci s velkým množstvím souborových formátů a existuje také ve specifických verzích pro WPF (Windows Presentation Foundation) a Silverlight. Knihovna přímo určená pro práci s tabulkovými dokumenty se nazývá Spire.XLS for .NET a cena licence se pohybuje na částce 599 $. Knihovna podporuje všechny typy formátů zmíněné u předchozích a tvorbu aplikací od .NET frameworku 2.0 až 4.5. V nižších edicích však chybí podpora tisku a konverze formátů, například z XLSX do PDF.

Produkty Spire lze vyzkoušet pod 30-ti denní dočasnou licencí (Trial), která po stanovenou dobu poskytuje plnou verzi produktu.

TMS Flexcel Studio

Knihovna firmy TMS Software [12] pro práci s `.xls` a `.xlsx` soubory, podporuje aplikace na platformě .NET, Windows Phone, iOS, Android a export do formátů PDF, HTML, SVG. Obecně podpora formátů je u této knihovny podstatně menší, avšak knihovna obsahuje funkce pro práci s obsahem dokumentu srovnatelné s ostatními knihovnami. Obsažena je podpora šablonování (obrázek 9), která na základě dat v předané datové struktuře naplní připravenou šablonu.



Obr. 9 Příklad šablonování pomocí knihovny TMS Flexcel Studio
Zdroj: Oficiální stránky TMS Software [12].

Cena nejlevnější licence začíná na částce 175 € a zahrnuje všechny zdrojové kódy. Nevýhodou je prioritní zákaznická podpora pouze skrze email nebo fórum. Knihovna také nepodporuje standardní práci s dílčími objekty dokumentu (pracovní list, řádek apod.), ale metody se volají pouze nad hlavním objektem

knihovny a pracuje se s kontextem. Pro získání buňky jiného pracovního listu se tedy určenou metodou přepne vnitřní kontext na jiný list a z něj se následně získávají buňky.

3.4.2 Volně dostupné knihovny

Zde uvedené knihovny jsou volně dostupným softwarem a jsou dostupné bez zaplacení licenčního poplatku za jejich používání. Volná dostupnost knihovny však automaticky neznamená přístup k jejím zdrojovým kódům a záleží také na druhu licence vybrané knihovny, tedy za jakých podmínek může být nasazena v aplikaci. Níže uvedené knihovny mají své zdrojové kódy zveřejněny, čehož lze s výhodou využít při úpravách jejich funkcionality nebo zkoumání principu jejich vnitřní struktury, fungování a závislostí. Nedosahují takové komplexnosti co do počtu podporovaných formátů nebo poskytovaných funkcí, ale část, na kterou se omezují, je ve vybraných knihovnách zpracována kvalitně, případně uživatelskou komunitou vhodně opravena.

EPPlus
Knihovna EEPlus [13] vychází a rozšiřuje původní knihovnu ExcelPackage [14]. Původní ExcelPackage měla problém především při zpracování a generování velkých objemů dat, respektive časovou složitostí a licencí GPL. V inovované knihovně je zahrnuta podpora pouze formátu OOXML, konkrétně souborů `.xlsx`, ale celý projekt má silnou uživatelskou základnu a prochází stabilním vývojem. Změna licence na LGPL umožňuje její komerční nasazení. Podporuje stylování buněk, vkládání grafů, obrázků a komentářů do pracovního listu, validaci vkládaných dat a mnoho dalšího. Nevýhodou knihovny jsou časté drobné chyby při přechodu na vyšší verze, které se nicméně následně rychle opravují (nefungující automatické přizpůsobení obsahu sloupců jejich obsahu, práce s rozsahem buněk a podobně). S každou vyšší verzí klesají nároky knihovny na operační paměť při generování obsahu souboru, naproti tomu vzrůstá čas generování dokumentů.

ClosedXML
Téměř shodnou funkcionalitu a uživatelskou základnu jako EEPlus nabízí knihovna ClosedXML [15] včetně podpory jediného formátu `.xlsx` tabulkových dokumentů. Jejím komunitní sponzorem je firma E-iceblue, která vyvíjí komerční produkt Spire. Vývoj knihovny není v porovnání s EEPlus tak živý a rychlost zpracování a generování dokumentů je pomalejší. Licence MIT vyžaduje šíření licenčního textu v každé kopii softwaru a oproti GPL licenci je volnější, opět ji lze využít v komerční aplikaci. Knihovna umožňuje načíst, zpracovat a uložit soubor, který již obsahoval vložený obrázek, nicméně jednou z nevýhod je fakt, že obrázky není schopna sama do dokumentu vkládat nebo s nimi jakkoliv jinak pracovat. Výhodou je výborně zpracovaná a jednoduchá podpora šablonování z předaných datových struktur.

NetOffice

Tato knihovna (licence MIT) přistupuje k práci s formáty kancelářských dokumentů značně odlišně. Podporuje práci s dokumenty téměř všech aplikací kancelářské sady Office (Excel, Word, PowerPoint, Outlook, atd.) a to skrze knihovnu COM Interop komunikující s danou aplikací přes COM rozhraní [16]. Výhodou tohoto přístupu je možnost využití téměř všech funkcí poskytovaných samotnou aplikací a s tím související podpora formátů. Nevýhodami přístupu je nutnost instalace dané kancelářské sady a konkrétních aplikací. Za další slabou stránku této knihovny lze označit i její samotnou výkonnost. Problém výkonnosti skrze COM rozhraní řeší knihovna některými optimalizačními postupy (dynamic binding), ale syntaxe a sémantika příkazů pro práci skrze toto rozhraní je shodná.

SpreadsheetLight

SpreadsheetLight představuje projekt, u kterého autor preferuje čistotu, jednoduchost a rychlost kódu a MIT licenci [17]. Podnětem pro vytvoření knihovny bylo množství menších chyb a doba jejich odstraňování v ostatních knihovnách (konkrétně EPPlus a ClosedXML). Podporuje pouze nový formát souborů `.xlsx` a je postavena nad Open XML SDK verze 2.0 pro práci s formátem OOXML od firmy Microsoft®. Nevýhodou knihovny je podpora pouze verze SDK 2.0, při vyšších verzích (SDK 2.5) vykazuje problémy s některými (deprecated) funkcemi. Poslední verze knihovny 3.4 je již z roku 2013 a není jistý její další vývoj.

4 Teoretická východiska, metodika a návrh řešení

4.1 Návrh knihovny pro práci s formátem OOXML

Motivací pro návrh a implementaci vlastní knihovny pro práci s formátem Open Office XML je několik:

- otevřenost formátu OOXML, jeho aplikační a platformní nezávislost,
- rozšíření kancelářského balíku MS Office ve firemní sféře,
- programové vytváření, čtení a zápis souborů s pomocí knihovny,
- znalost funkcionality a kódu vlastního řešení,
- možnost propojení s informačním systémem firmy.

Obecné výhody otevřenosti formátů, respektive jejich standardů byly zmíněny v předchozí kapitole. Aplikační a platformní nezávislost hrají v dnešní době podstatou roli také při sdílení dat. Dokument, vytvořený sadou kancelářského balíku nebo samostatně programově, lze šířit dále bez nutnosti zajišťování kompatibility s jinou aplikací či platformou.

Přestože se lze ve firmách setkat s infrastrukturou postavenou na operačních systémem typy Unix/Linux (především servery), na koncových stanicích je stále nejrozšířenějším operačním systémem MS Windows [18]. Komplexní řešení firem (nejčastěji se statusem Microsoft Certified Partner), je postavení infrastruktury na produktech a službách firmy Microsoft®, takže i serverová část běží často na operačním systému MS Windows Server. S tímto faktem souvisí další bod motivace k tvorbě vlastní knihovny, a sice rozšíření kancelářského balíku MS Office. Je tak zajištěna komunita potenciálních uživatelů těchto aplikací, kteří nechtějí nebo nemohou dokumenty vytvářet programově, ale chtějí jejich výslednou podobu prohlížet či editovat v grafickém uživatelském prostředí. Na práci s těmito aplikacemi jsou zvyklí a tak nástroj (v tomto případě knihovna), který by jim poskytl náhled na data z jiného prostředí, stylovaná v dobře známé podobě s možností jejich případného dalšího zpracování, představuje přínos. Jiným prostředím lze chápat například data generovaná na serveru a zobrazovaná skrze webový prohlížeč formou HTML kódu a podobně.

Programová manipulace se soubory skrze knihovnu umožňuje propojení a rozšíření funkcionality celého systému. Za předpokladu, že knihovna neobsahuje další programové závislosti (spolupráce se sadou MS Office skrze volání COM rozhraní), je možné její nasazení v desktopovém i serverovém prostředí, bez předinstalované kancelářské sady aplikací. To ve svém důsledku představuje nezávislost samotné knihovny a omezení jejího nasazení vychází pouze ze strany použité implementační technologie a požadované funkcionality.

Znalost funkcionality a kódu vlastního řešení je zásadní. Například dle zkušeností inženýrsko-dodavatelské společnosti COMPAS automatizace [19], může být využití placených i volně dostupných knihoven (obecně knihoven třetích stran) problémové. Typické výhody a nevýhody placeného či volně dostupného softwaru byly popsány v kapitole 3.4. Pro konkrétní firmu představuje cena za zakoupení licence nezanedbatelnou položku, dále mohou pro firemní využití být problémové i podmínky užívání a omezené možností přizpůsobení za účelem konkrétní aplikace. Volně dostupné knihovny omezují své využití použitými typy licencí. Negativním případem z praxe je například netriviální začlenění knihovny (určené na generování souborů formátu PDF) do informačního systému, přičemž v řádu několika měsíců byla podpora knihovny ukončena a produkt změněn z volně dostupného na komerční. Ani jedna ze skupin placených nebo volně dostupných knihoven nemusí pokrývat specifické nároky na ně kladené a naopak mohou obsahovat množství nevyužitelné funkcionality.

Práce primárně cílí na nasazení knihovny v průmyslové firemní sféře, konkrétně na jejím nasazení jako součásti výrobního informačního systému. Její možné využití je ve výrobním informačním systému v úrovni MES [20], například v systému COMES® zmíněné firmy COMPAS automatizace.

4.1.1 Výběr platformy a programovacího jazyka

Jestliže se potenciální nasazení vytvořené knihovny předpokládá na firemní infrastruktuře částečně či plně tvořené produkty firmy Microsoft®, pak logickým východiskem při volbě programové platformy a programovacího jazyka, na kterých bude knihovna implementována, jsou také produkty této firmy. Výhodu představuje především práce v konzistentním prostředí, vývojovém nástroji a jazyce, který je s prostředím integrován.

Pro výše uvedené důvody byla pro následnou implementaci knihovny zvolena platforma .NET Framework [21]. Ta je určena pro vývoj a nasazení aplikací nejen na desktop systémech a mobilních zařízeních, ale také na webu (dynamické webové stránky) nebo v databázích , dále tato platforma nabízí výhody v podobě objektové orientace a jazykové nezávislosti. Nepředepisuje použití konkrétního programovacího jazyka a kód navíc prochází několika fázemi kompilace do společného kódu (Intermediate Language). Nejčastěji používané jazyky jsou C#, Visual Basic nebo C++/CLI.

Společně s platformou .NET byl společností Microsoft® vyvíjen a standardizován (standard ECMA-334) jazyk C# [22]. Jedná se o moderní objektově orientovaný programovací jazyk kombinující některé vlastnosti ostatních jazyků, například garbage collector pro automatickou správu paměti nebo labda výrazy. Přirozená kombinace jazyka C# s platformou .NET umožňuje vývojářům využití celého jejího potenciálu.

Za účelem vývoje aplikací na platformě .NET existuje produkt MS Visual Studio [23], který nabízí vývojové prostředí (IDE) podporující téměř libovolný programovací jazyk (podporu lze doinstalovat pomocí jazykových služeb). Visual Studio umožňuje tvorbu konzolových aplikací i aplikací s grafickým rozhraním

(podpora vizuálního stylování pomocí designeru) a poskytuje sadu nástrojů pro komfortní editaci zdrojového kódu, například nástroj IntelliSense pro automatické napovídání/dokončování.

4.2 Přístup ke struktuře dokumentu

Dle kapitoly 3.3, jež se věnovala popisu struktury a organizaci dat formátu OOXML, byl soubor upraven takovým způsobem, aby byla jeho obálka transparentní pro uživatele a dala se prozkoumat jeho vnitřní struktura. Výstup, který ze zkoumání vyplývá, zobrazuje obsah souboru obsah souboru jako soubor XML dokumentů obsahujících data nebo metadata kancelářského dokumentu.

Pro účely práce s formátem je potřeba k obsahu jednotlivých XML souborů přistupovat vhodným způsobem, samotný obsah pak číst, vytvářet či měnit. Přístup musí být zároveň realizován způsobem splňujícím definici formátu OOXML zachovávajícím kompatibilitu s aplikacemi kancelářského balíku MS Office. Mezi možné přístupy, které budou dále blíže popsány, patří využití:

- Excel application COM object model,
- SharePoint Excel Automation Services,
- Open XML SDK,
- knihoven třetích stran,
- XML DOM,
- SAX.

U některých z přístupů je uveden příklad programového vytváření souboru (vkládání dat) do podoby souboru `example.xlsx` z kapitoly 3.3 (obrázek 3). Tento původní příklad bude dále v textu označován jako referenční a každý z příkladů bude mít za cíl vložit stejná data do odpovídajícího umístění (buňky) v upravovaném tabulkovém dokumentu.

Excel application COM object model
Jedná se o přístup, který není příliš podporován a například ani doporučován pro nasazení na serverových aplikacích [24]. Přes COM rozhraní dané aplikace (v tomto případě MS Excel) se přímo volají její funkce, například pro čtení dat z pracovního listu a podobně. Komunikace zajišťuje knihovna `COM Interop` a výhodou řešení je množství funkcí (téměř identické s funkcemi poskytovanými samotnou aplikací). Zásadní nevýhodu ovšem představuje nutnost instalace samotné aplikace a výkonnostní problém, protože se jedná o volání COM objektů. Také syntaxe některých příkazů (například pro stylování dokumentu) je poměrně dlouhá a složitá. Příklad programového vytvoření tabulkového dokumentu `com.xlsx` naplněného daty je na obrázku 10.

```
Microsoft.Office.Interop.Excel.Application app = new Microsoft.Office.Interop.Excel.Application();
Microsoft.Office.Interop.Excel.Workbook workbook;
Microsoft.Office.Interop.Excel.Worksheet worksheet;
workbook = app.Workbooks.Add();
worksheet = (Excel.Worksheet)workbook.Worksheets.get_Item(1);
worksheet.Cells[1, 1] = "Customer"; worksheet.Cells[1, 2] = "id"; worksheet.Cells[1, 3] = "firstName";
worksheet.Cells[1, 4] = "lastName"; worksheet.Cells[2, 2] = 27840; worksheet.Cells[2, 3] = "Jaroslav";
worksheet.Cells[2, 4] = "Košťál";
workbook.SaveAs("com.xlsx", Excel.XlFileFormat.xlWorkbookDefault);
workbook.Close(true);
app.Quit();
System.Runtime.InteropServices.Marshal.ReleaseComObject(worksheet);
System.Runtime.InteropServices.Marshal.ReleaseComObject(workbook);
System.Runtime.InteropServices.Marshal.ReleaseComObject(app);
```

Obr. 10 Programová manipulace se souborem pomocí Excel COM rozhraní.

SharePoint Excel Automation Services

Platforma SharePoint [25] je dalším z produktů společnosti Microsoft® a v podstatě se jedná o webový framework, který umožňuje vytváření webů organizací a práci s informacemi. Prostředí je úzce provázáno se sadou nástrojů MS Office. Excel Automation Services je placená komponenta, která běží pod součástí SharePoint Server. Přístup odpovídá předchozímu modelu Excel application COM object model, ale toto řešení je určeno čistě pro nasazení na serverech. Byl navržen jako doplněk k Microsoft Office Open XML SDK (popsanému dále). Pro některé specifické případy, kdy je realizace pomocí Open XML SDK obtížná, se nabízí možnost oba přístupy vhodně kombinovat [26]. Nevýhodou řešení je nutnost nasazení SharePoint platformy, placení licence, zaměření pouze na serverové aplikace a pro některé případy chybějící funkcionalita. Výhodu lze naopak vidět v ucelenějším přístupu k manipulaci se soubory oproti Open XML SDK (obtížnější vytváření nevalidních souborů), v syntaxi příkazů a v podpoře konverze do jiných formátů (PDF atd.). Příklad naplnění souboru `services.xlsx` daty podle referenčního příkladu nabízí obrázek 11.

```
ES.ExcelService excelService = new ES.ExcelService();
ES.Status[] state;
excelService.Credentials = System.Net.CredentialCache.DefaultCredentials;
string sessionId = excelService.OpenWorkbook("services.xlsx", String.Empty, String.Empty, out state);
excelService.SetCellA1(sessionId, "List1", "A1", "Customer");
excelService.SetCellA1(sessionId, "List1", "B1", "id");
excelService.SetCellA1(sessionId, "List1", "C1", "firstName");
excelService.SetCellA1(sessionId, "List1", "D1", "lastName");
excelService.SetCellA1(sessionId, "List1", "B2", 27840);
excelService.SetCellA1(sessionId, "List1", "C2", "Jaroslav");
excelService.SetCellA1(sessionId, "List1", "D2", "Košťál");
excelService.CloseWorkbook(sessionId);
```

Obr. 11 Programová manipulace se souborem pomocí Excel Automation Services.

Microsoft Office Open XML SDK

Toto SDK (Software Development Kit) [27], postavené nad .NET API Packaging (viz kapitola 4.2.1), poskytuje silně typované třídy pro manipulaci se soubory kancelářských dokumentů. V době psaní této práce lze využít nejnovější verzi Open XML SDK 2.5 (vystavěné nad .NET 4.0) [28], která oproti starším verzím (SDK 1.0

a 2.0) zahrnuje podporu pro práci se sadou MS Office 2013 a rozšiřuje a opravuje funkcionalitu. Výhoda práce s Open XML SDK spočívá v reprezentaci struktury a přístupu k jednotlivých prvků dokumentu přesně podle definice standardu. Při vkládání obsahu je třeba postupně vytvořit pracovní list, naplnit tabulku sdílených řetězců (pokud některá z dat představují řetězce) a vložit na pracovní list buňku s hodnotou (příklad na obrázku 12). Nevýhodou je zpřístupnění API na poměrně nízké úrovni a možnost vytvářet nevalidní dokumenty, které neodpovídají standardu OOXML.

```
SpreadsheetDocument doc = SpreadsheetDocument.Create("sdk.xlsx", SpreadsheetDocumentType.Workbook);
WorkbookPart workbook = doc.AddWorkbookPart(); workbook.Workbook = new Workbook();
WorksheetPart worksheet = workbook.AddNewPart<WorksheetPart>();
worksheet.Worksheet = new Worksheet(new SheetData());
Sheets sheets = doc.WorkbookPart.Workbook.AppendChild<Sheets>(new Sheets());
Sheet sheet = new Sheet() { Id = doc.WorkbookPart.GetIdOfPart(worksheet), SheetId = 1, Name = "List 1" };
sheets.Append(sheet);
SharedStringTablePart sharedString = doc.WorkbookPart.AddNewPart<SharedStringTablePart>();
int index = insertSharedString("Customer", sharedString);
Cell cell = insertCell("A", 1, worksheet);
cell.CellValue = new CellValue(index.ToString());
cell.DataType = new EnumValue<CellValues>(CellValues.SharedString);
//...
worksheet.Worksheet.Save(); workbook.Workbook.Save();
doc.Close();
```

Obr. 12 Programová manipulace se souborem pomocí Open XML SDK v2.5.

Knihovny třetích stran

Výhody a nevýhody knihoven třetích stran, jejich funkce a omezení, stejně jako příklady jejich použití, byly uvedeny v kapitole 3.4.

XML DOM

Jedná se o objektový model (Document Object Model) reprezentující strukturu XML dokumentu jako strom uzlů [1]. Standard byl vyvíjen společně s ostatními značkovacími jazyky (XML, HTML, apod.) pro jejich manipulaci a standardizován konsorciem W3C. Přístup k XML souborům pomocí DOM poskytuje jazykově nezávislé a objektově orientované rozhraní, které je uvnitř paměti reprezentováno jako strom. Každý uzel (označený například proměnnou `node`), poskytuje metody pro přístup a modifikaci. Získání atributu uzlu může být realizováno voláním metody `node.getAttribute()`, přidání uzlu potomka voláním metody `node.AppendChild()` a podobně. Přístup je výhodný při dynamické modifikaci stromu a dotazování nad daty závisejícími na struktuře stromu. Nevýhodou může být výkonnost, tedy rychlost přístupu a spotřeba paměti pro uložení struktury. Přesto lze v kombinaci se znalostí definice formátu OOXML dosáhnout robustního přístupu k práci (především modifikaci) s tabulkovými dokumenty a k nástrojům platformy .NET, zároveň není třeba instalovat další volitelné součásti (nutné u Open XML SDK a podobně).

SAX
Rozhraní SAX [1] pro zpracování XML dokumentů je založeno na událostech, které při sekvenčním čtení dokumentu vyvolá význačný prvek značkovacího jazyka. Může se jednat o počátek dokumentu, značku elementu, jeho atribut a další. K daným událostem je třeba implementovat handlery (obslužné funkce) zpracovávající danou událost a její obsah. Z principu fungování se jedná o sekvenční přístup k obsahu dokumentu a zpracování probíhá pouze na jeho části, přičemž zbytek dokumentu není pro aktuální zpracovávání důležitý. Oproti XML DOM se jedná o rychlý a paměťově efektivní způsob, který přirozeně podporuje proudové zpracování. Nevýhodou je omezení pouze na čtení dat a to pouze ve směru vpřed (nelze se strukturou dokumentu vracet), modifikace tímto způsobem není podporována. Pro účely knihovny určené zároveň pro čtení i zápis se SAX jeví jako nevhodný. Je však možné v knihovně oba přístupy (SAX a XML DOM) kombinovat a při využití výhod obou přístupů eliminovat jejich nedostatky.

4.2.1 Komprimace a dekomprimace souborů

Rozhraní XML DOM a SAX se zaměřují přímo na práci s obecným formátem XML a oproti ostatním výše zmíněným přístupům, je třeba zajistit jejich funkcionalitu ještě skrze komprimační ZIP obálku formátu Open Office XML. Platforma .NET podporuje několik způsobů, jak komprimovat či dekomprimovat soubory a manipulovat tak s vnitřní strukturou dokumentu OOXML.

Packaging
Namespace System.IO.Packaging jako součást .NET poskytuje od verze 3.0 třídy pro ukládání datových objektů do jediného kontejneru. Podporuje ZIP kontejner a samotné třídy jsou pak určeny především pro datové objekty formátu Open Office XML. Při komprimaci tak automaticky v kontejneru vytváří `[Content_Types].xml`, který popisuje a identifikuje ostatní obsažené soubory a soubory v adresářích `/_rels/` a popisuje vazby mezi částmi struktury (více v kapitole 3.3). Pokud ale některé z těchto povinných částí v kontejneru chybí, je přístup ke struktuře při dekompresi neúspěšný.

SharpZipLib
SharpZipLib představuje knihovnu, která je oproti Packaging robustnější a podporuje širší škálu funkcí; například kompresní algoritmy ZIP, GZIP, TAR, BZIP2, podporu platformy Silverlight nebo ukládání prázdných adresářů do kontejneru. Neomezuje se primárně na podporu formátu OOXML a nabízí univerzálnější řešení. Každý objekt ve struktuře je reprezentován objektem `ZipEntry`, který uchovává informace o vlastnostech původního objektu (soubor/složka) a poskytuje množství metod pro další zpracování.

Přístup ke struktuře dokumentu formátu Open Office XML bude na základě zkoumaných možností realizován s pomocí knihovny SharpZipLib a zpracování XML souborů vnitřní struktury pomocí rozhraní SAX a XML DOM.

4.3 Čtení a zápis struktury dokumentu

Při konzultacích s firmou COMPAS automatizace, která vyvíjí výrobní informační systém COMES®, byla diskutována očekávaná a využitelná funkcionalita knihovny potenciálně rozšiřující systém. Na základě této zpětné vazby byly stanoveny užitečné funkce vlastní knihovny podle hierarchického dělení dokumentu:

Pracovní sešit (workbook)

- parsování tabulkového dokumentu formátu Open Office XML (.xlsx/.xlsm) do struktury dalších objektů (listy, sloupce, řádky, buňky, styly)
- práce s pozičním systémem mřížky dat (převody mezi indexy a názvy sloupců/buněk)
- konverze hodnot typických pro MS Excel (převody data a času, formátovací masky)
- kopie dokumentu (klonování)
- práce s vlastností dokumentu (název, počet pracovních listů)
- získání/vytváření pracovních listů
- ukládání dat

Pracovní list (worksheet)

- práce s vlastnostmi listu (název, počet řádků/sloupců, viditelnost, uzamčení)
- získání/vytváření řádků/sloupců/buněk
- porovnání obsahu všech řádků dvou listů
- získání/nastavení ukotvení/filtrů
- šablonování (vkládání dat z předaných datových struktur podle předpisu)

Sloupec (column)

- práce s vlastnostmi sloupce (index, název, viditelnost, šířka)
- získání/vytváření buněk

Řádek (row)

- práce s vlastnostmi řádku (index, viditelnost, počet buněk)
- získání/vytváření buněk
- porovnání obsahu dvou řádků

Buňka (cell)

- získání/vytvoření hodnoty/vzorce/komentáře/stylu
- příznak sloučení buňky
- získání/vytvoření horizontálního/vertikálního sloučení několika buněk

Styl buňky (cell style)

- získání/vytvoření formátovací masky
- získání příznaku příslušnosti formátovací masky k datu a času
- porovnání dvou stylů buněk
- kopie stylu (klonování)
- získání/vytvoření stylu podle konvencí jazyka CSS (tučnost, kurzíva, velikost textu, font, dekorace textu, barva textu, zarovnání textu, zalamování slov, okraje, barva pozadí)

V kapitole 3.3 byla stručně popsána vnitřní struktura dokumentů podle OOXML. Ke kterým ze souborů struktury je však nutné přistupovat, kde je v jejich značkovaném textu umístěna konkrétní informace nebo kam ji umístit? Tato kapitola definuje u každé knihovní funkce její vazbu na soubory a elementy nebo atributy vnitřního značkovacího jazyka XML. Protože vyšší části (celky) dokumentu často odkazují funkcionalitou na ty nižší (například pracovní list musí umožňovat vkládání buněk), je při dalším popisu postupováno zdola nahoru, aby byly potřebné pojmy nižších celků již vysvětleny. Získané poznatky lze poté již převést do kódu zvoleného programovacího jazyka a knihovnu prakticky realizovat. Pro podporu štábní kultury zdrojového kódu a smysluplného API rozhraní je zahrnuta také kapitola 4.4 zabývající se objektovým návrhem a rozhraním knihovny.

4.3.1 Styl buňky (cell style)

Funkce stylu buňky předpokládají existující soubor `/xl/styles.xml`, který uchovává globálně styly pro celý pracovní sešit dokumentu. V tomto dokumentu funkce následně vyhledají nebo vytvoří odpovídající elementy a atributy popisující styl obecné buňky a při spojení s konkrétní existující buňkou listu (tedy nastavení jejího stylu) vytvoří reference mezi zápisem stylu a buňky. Obsah souboru stylů je uvozen kořenovým elementem `styleSheet` a definuje zvlášť části stylu (podstyly) pro formátovací masku, písmo, výplně a okraje, které jednotlivě sdružuje v elementu `xf` podle pořadí (indexu) jejich definice v jednotlivých podstylech:

/xl/styles.xml

```
<styleSheet …>
   <numFmts …>…</numFmts>
   <fonts …>…</fonts>
   <fills …>… </fills>
   <borders …>…</borders>
   <cellXfs …>
      …
      <xf numFmtId="0" fontId="0" fillId="0" borderId="0"
       xfId="0" />
      …
   </cellXfs>
</styleSheet>
```

Získání formátovací masky představuje vyhledání elementu `numFmt` identifikátoru uloženého v atributu `numFmtId` a přečtení hodnoty atributu `formatCode`. Pokud atribut `formatCode` obsahuje zápis hodnoty typický pro datum a čas, pak má zároveň maska nastaven příznak příslušnosti k datu a času. Zápis formátovací masky vytváří nový element `numFmt` s unikátním identifikátorem. Také je třeba inkrementovat atribut `count` nadřazeného elementu `numFmts`, který informuje o počtu definovaných formátovacích masek a index nového elementu vložit do nového stylu elementu `xf` (viz dále). Podporované formátovací masky jsou součástí specifikace formátu:

```
<numFmts count="3">
   …
   <numFmt numFmtId="302" formatCode="d.m.yyyy h:mm:ss" />
   …
</numFmts>
```

Čtení nebo zápis stylu musí zpřístupňovat vlastnosti písma, výplně, okrajů a zarovnání, části těchto definic lze identifikovat v jednotlivých elementech podstylů. Nastavení písma obsahuje element `font` a definuje ho pomocí vložení potomků: Element `b` pro tučnost, `i` pro kurzívu a element `u` pro podtržení. Velikost písma (v jednotkách pixel) obsahuje atribut `val` elementu `sz` a hodnota stejného atributu popisuje v elementu `name` použitý font písma. Element `color` a jeho atribut `rgb` obsahuje zápis hexadecimální hodnoty barvy zapsané formátem ARGB (Alfa Red Green Blue):

```
<fonts count="5">
   …
   <font>
      <b/>
      <i/>
      <sz val="11"/>
      <name val="Calibri"/>
      <u/>
      <color rgb="FFFF0000"/>
   <font>
   …
<fonts>
```

Informace o způsobu vertikálního nebo horizontálního zarovnání a zalamování slov je uložena jako potomek elementu `xf`. `Alignment` může obsahovat atributy `vertical` a `horizontal`, které již podle názvu specifikují svůj účel a mohou nabývat hodnot `"left"`, `"right"`, `"center"` nebo `"justify"`. Zalamování slov lze nastavit atributem `wrapText="1"`:

```
<xf …>
   <alignment vertical="bottom" horizontal="left"
    wrapText="0" />
</xf>
```

Každý formát okraje uložený v elementu `border` se skládá z popisu dílčích okrajů buňky a je definován svojí barvou a druhem použité čáry. Pro konzistentní vzhled buňky je tak třeba nastavit styl všech jejích okrajů (vrchní, levý, pravý, spodní) stejně, nicméně je podporován také přístup pouze ke konkrétnímu okraji. Element nazvaný podle pozice okraje (`top`, `right`, `bottom`, `left`) obsahuje atribut stylu použité čáry a až jeho potomek `color` popisuje také jeho barvu:

```
<borders count="4">
   …
   <border>
      <left style="dotted">
         <color rgb="FF00FF00" />
      </left>
      <right style="dotted">
         <color rgb="FF00FF00" />
      </right>
      …
   </border>
   …
</borders>
```

Podobně jako okraje jsou ve struktuře uchovány informace o použité výplni buňky. Element `patternFill` obsahuje atribut `patternType="solid"`, který popisuje plnou výplň barvy potomka:

```
<fills count="5">
   …
   <patternFill patternType="solid">
      <fgColor rgb="FF000000"/>
   </patternFill>
   …
</fills>
```

Obsah všech výše uvedených elementů definoval vzhled podstylů buňky. Aby však bylo možné informaci o celkovém stylu buňky použít najednou, je třeba získat pozici odpovídajících elementů podstylů a sjednotit je na jedno konkrétní místo. Toto místo představuje element `xf`, který odkazuje identifikátorem `numFmtId` na použitou masku, číslem pozice v atributu `fontId` na pořadí stylu v rámci elementu `fonts` a podobně také pro styl okraje a výplň. Uvedením povolujících atributů (s hodnotou `"1"`) se potvrzuje použití těchto stylů, kdy styl zarovnání je obsažen jako samostatný potomek:

```
<cellXfs count="8">
   …
   <xf numFmtId="302" fontId="6" fillId="14" borderId="4"
    xfId="0" applyNumberFormat="1" applyFont="1"
    applyFill="1" applyBorder="1" applyAlignment="1">
      <alignment vertical="center" horizontal="center" />
   </xf>
   …
</cellXfs>
```

Každý ze stylů sdružených do elementu `xf` má pozicí v rámci struktury elementu `cellXfs` určen svůj unikátní index `[1…n]`. Skrze index je odkazováno v souboru pracovního listu, právě na tento styl. Pokud by například element `cellXfs` obsahoval celkem 3 popisy stylů a buňka by měla být stylována podle posledního, pak zápis buňky bude obsahovat atribut s indexem `"3"`.

Kopie stylu (klonování) je programovou záležitostí, kdy je vytvořena kopie objektu popisujícího styl bez závislosti na původním objektu a možnost změny jejího obsahu, tedy bez změn projevujících se v objektu původním. Bližší popis a význam bude uveden v kapitole 5.

Porovnání dvou stylů buněk v triviálním případě znamená srovnat shodu jejich indexů odkazujících na styl. V dokumentu a jeho vnitřní struktuře však může být uloženo více shodných stylů, které se liší některými zanedbatelnými detaily.

Porovnání stylu buněk má ale probíhat nad vlastnostmi, se kterými lze v rámci knihovny manipulovat. Proto je třeba projít shodu nastavených masek a jednotlivých vlastností (textu, okrajů, atd.).

4.3.2 Buňka (cell)

Buňka obsahuje nejčastěji svoji hodnotu ve formě čísla nebo řetězce. Pokud je čten obsah buňky, musí být nejprve identifikován typ dat, aby byl obsah správně interpretován. Atribut `t` popisuje data jako pravdivostní hodnotu (`="b"`), řetězec (`="s"`) nebo číslo (`="n"`). Existující hodnota buňky je uložena uvnitř elementu `v`, případný existující vzorec pak uvnitř elementu `f`. Pro případ čtení dat řetězce se využije hodnota elementu `v` jako index `[0…n]` do tabulky sdílených řetězců umístěné v souboru `/xl/sharedStrings.xml` a na odpovídající pozici se vyhledá daný řetězec. Teprve to je hodnota dané buňky:

/xl/worksheets/sheet1.xml

```
<c r="A2" t="s">
   <v>0</v>
</c>
```

/xl/sharedStrings.xml

```
<sst …>
   <si><t>první</t></si>
   <si><t>druhý</t></si>
   …
</sst>
```

Zápis hodnoty nebo vzorce představuje nutnost identifikovat typ vkládaných dat a zapsat ho do atributu typu. V případě zápisu čísla, pravdivostní hodnoty nebo formule se vytvoří konkrétní elementy a jejich obsah:

```
<c r="B2" t="n" s="2">
   <v>3.102</v>
   <f>SUM(B2:D2)</f>
</c>
```

Pro řetězec je postup složitější: tabulka sdílených řetězců se musí prohledat, protože řetězec už může existovat, pak je jeho index polohy vrácen. Pokud není řetězec ve struktuře nalezen, je přidán na konec před end-tag `</sst>` jako nový záznam `si` s indexem odpovídajícím celkovému počtu všech sdílených řetězců. Index existujícího nebo nově vytvořeného záznamu je následně zapsán do hodnoty buňky.

Odkaz na styl buňky je uložen v atributu `s` a jeho předpis je vyhledán nebo vytvořen postupem uvedeným v kapitole 4.3.1.

Čtení a zápis volitelného komentáře buňky pracuje s několika soubory vnitřní struktury. Čtení komentáře je ve srovnání se zápisem jednoduché a jedna z možností zahrnuje prohledání souboru vazeb příslušejícího listu `/xl/worksheets/_rels/sheetX.xml.rels` na přítomnost elementu `Relationship` s `Target` atributem odkazujícím na soubor komentářů. Další možností je prohledání adresáře `/xl/` na přítomnost souborů `commentsX.xml` (kde `X` může být číslováno postupně `[1…n]`). Číslo `X` odkazuje do souboru `/xl/workbook.xml` na atribut `sheetId` jednoho z elementů `sheet`. Tím je určena vazba mezi souborem komentářů a pracovního listu:

/xl/workbook.xml

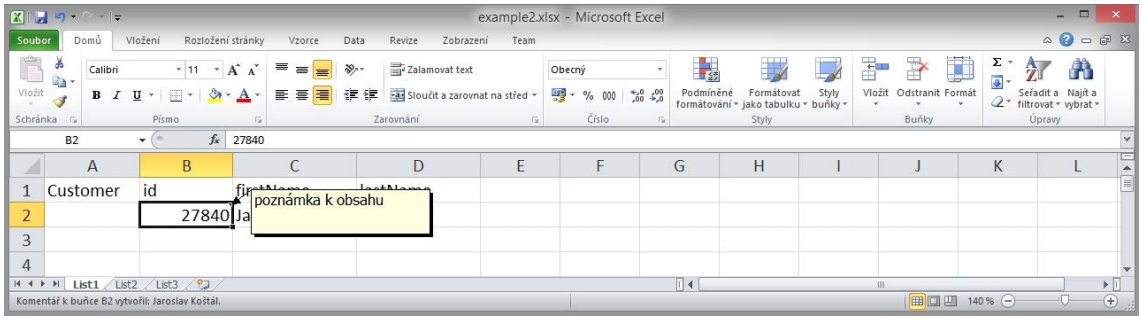
```
<sheets>
   <sheet name="List1" sheetId="1" r:id="rId1" />
   …
</sheets>
```

Ve struktuře souboru komentářů lze podle atributu `ref` elementu `comment` dohledat konkrétní ovlivněnou buňku a obsah jejího komentáře jako hodnoty uvnitř `t`:

/xl/comments1.xml

```
<commentList>
   …
   <comment ref="F11">
      <text><t>celkove soucty sloupcu</t></text>
   </comment>
   …
</commentList>
```

Zápis komentáře musí ověřit, zda soubor komentářů příslušný k pracovnímu listu, na kterém je umístěna buňka, existuje. V kladném případě se vyhledává komentář příslušný buňce a jeho stará hodnota se přepíše novou. Pokud pro buňku komentář v souboru komentářů neexistuje, pak je odpovídající obsah elementů vytvořen a vložen. Neexistence samotného souboru komentářů pro list představuje potřebu jeho vytvoření a vložení minimální kostry. Pojmenování (číslo na konci názvu) je odvozeno od `sheetId` pracovního listu. Objekt komentáře se v aplikaci MS Excel chová jako vyskakovací okno, které se objeví při najetí kurzoru nad buňku obsahující komentář (obrázek 13).



Obr. 13 Okno komentáře v aplikaci MS Excel.

Definice okna komentáře (v sekci Přílohy) musí být také uložena uvnitř struktury dokumentu a do adresáře `/xl/drawings/` je za tímto účelem vytvořen soubor `vmlDrawingX.vml`. Přípona souboru odkazuje na formát VML (Vector Markup Language), který vychází z formátu XML a zaměřuje se na popis dvoudimenzionální vektorové grafiky, přičemž definice samotného jazyka je součástí standardu Open Office XML. Číslo `X` v názvu se musí shodovat s číslem v názvu souboru komentářů. Záznamy o vytvoření je třeba vložit do `/[Content_Types].xml`, část pro komentáře do elementu `Override` a část o použití VML do elementu `Default`:

`/[Content_Types].xml`

```
<Types …>
   …
   <Default Extension="vml" ContentType="application
    /vnd.openxmlformats-officedocument.vmlDrawing" />
   …
   <Override PartName="/xl/comments1.xml" ContentType="
    application/vnd.openxmlformats-officedocument.
    spreadsheetml.comments+xml" />
   …
</Types>
```

Provázání pracovního listu `X` s novými záznamy znamená uložení jejich názvů a identifikátorů do souboru vazeb `sheetX.xml.rels` a vložení objektu kreslení přímo do pracovního listu `sheetX.xml`:

`/xl/worksheets/_rels/sheet1.xml.rels`

```
<Relationships …>
   <Relationship Id="rId1" Type="…"
    Target="../comments1.xml" />
   <Relationship Id="rId2" Type="…"
    Target="../drawings/vmlDrawing1.vml" />
</Relationships>
```

/xl/worksheets/sheet1.xml

```
<worksheet …>
   …
   <legacyDrawing r:id="rId2" />
</worksheet>
```

Získání či vytvoření vertikálního/horizontálního sloučení několika buněk, stejně jako příznak sloučení (daná buňka je součástí sloučení), lze nalézt, případně vytvořit, v souboru pracovního listu `sheetX.xml` pomocí elementu `mergeCells` obsahujícího rozsahy sloučených buněk:

/xl/worksheets/sheet1.xml

```
<worksheet …>
   …
   <mergeCells count="2">
      <mergeCell ref="A1:A2" />
      <mergeCell ref="B2:D5" />
   </mergeCells>
   …
</worksheet>
```

4.3.3 Řádek (row)

K vlastnostem řádku, tedy indexu, viditelnosti a počtu jeho buněk lze přistupovat skrze soubor pracovního listu. Index identifikuje přímo atribut `r`, stejně tak viditelnost atribut `hidden`. Počet buněk je zjistitelný součtem všech potomků daného řádku:

/xl/worksheets/sheet1.xml

```
<sheetData>
   …
   <row r="2" hidden="0">
      …
      <c …>…</c>
      <c …>…</c>
      …
   </row>
   …
</sheetData>
```

Vytvoření nové buňky listu v dvourozměrné mřížce dat znamená specifikovat název nebo index sloupce, ve kterém se má nacházet. Index řádku je již známý a tak je určením sloupce přesně určena také pozice buňky. Tím, že je funkce aplikována nad řádkem, se předpokládá jeho existence. Například přidání buňky

na druhý sloupec druhého řádku představuje vygenerování elementu a jeho atributů a přidání jako potomka do elementu `row` následovně:

```
<row r="2">
   …
   <c r="B2"/>
   …
</row>
```

Porovnání obsahu dvou řádků je průnikem všech buněk těchto řádků a vzájemné porovnání jejich obsahu a stylu. Musí se shodovat pozice na sloupcích, atributy (typ dat, styl) a obsah potomků (hodnoty a vzorce) každé z buněk.

4.3.4 Sloupec (column)

Sloupec není ve struktuře pracovního listu (mimo popisu jeho stylu) nikde fyzicky obsažen a pro účely knihovny je vytvářen uměle jako průsečík všech řádků a buněk se shodnou hodnotou atributu `r`, konkrétně jeho části pro sloupec (abecední část). Jedná se ve své podstatě o jiný pohled na strukturu dat.

Název sloupce je znám hned při vytváření, neboť podle jeho jména se agregují buňky v něm obsažené. Index lze potom přepočítat dle pravidla `A` = 0, `B` = 1 a tak dále.

Součástí obsahu listu mohou být volitelné definice vzhledu sloupců. Jedná se o nastavení viditelnosti a vlastní šířky. Element `col` obsahuje atributy `min` a `max`, které udávají, jakého rozsahu indexů sloupců se nastavení týká. Vlastní šířku atributu `width` je třeba explicitně povolit pomocí `customWidth="1"`. Zajímavostí je, že hodnota šířky podle standardu OOXML se udává v přibližném počtu znaků, které se mají do sloupce vejít, aniž by se text zalomil nebo odřádkoval, a to při standardním nastavení fontu. Nastavení atributu `hidden` skrývá sloupec a všechny buňky v něm obsažené:

/xl/worksheets/sheet1.xml

```
<cols>
   …
   <col min="2" max="2" width="9" customWidth="1" hidden="1"
   />
   …
</cols>
```

Vytvoření nové buňky listu znamená (podobně jako u řádku) specifikovat název nebo index řádku, ve kterém se má nacházet. Index sloupce je známý, nicméně další postup se liší. Sloupce totiž nejsou rodičovským potomkem buněk (tím jsou pouze řádky), jestliže řádek daného indexu neexistuje, je třeba ho ve struktuře pracovního listu vytvořit a teprve potom vložit také strukturu buňky. Postup vytváření řádků obsahuje kapitola 4.3.3.

4.3.5 Pracovní list (worksheet)

Název pracovního listu je uložen v souboru `/xl/workbook.xml` v atributu `name` odpovídajícího elementu `sheet`. Viditelnost definuje volitelný atribut `state` a konkrétní hodnota `"visible"` nebo `"veryHidden"`.

/xl/workbook.xml
```
<sheets>
   …
   <sheet name="List1" sheetId="1" r:id="rId1"
    state="veryHidden" />
   …
</sheets>
```

Počet řádků pracovního listu lze zjistit v souboru `/xl/worksheets/sheetX.xml` (kde `X` je číslo pracovního listu) v elementu `sheetData` na základě počtu potomků s názvem `row`. Počet sloupců odpovídá největšímu indexu sloupce uloženému v tagu `r` potomků `c` (cell) všech elementů `row`. V elementu řádku totiž mohou být uloženy pouze buňky obsahující hodnotu a je třeba zjistit poslední použitý sloupec v rámci všech řádků. Jednodušší postup pro výpočet počtu řádků a sloupců pracuje s nepovinným (optimalizačním) elementem `dimension` a jeho atributem `ref`, který informuje o rozsahu dat na pracovním listu. Hodnota `ref="A1:E3"` představuje rozsah 3 řádky (`1` - `3`) a 5 sloupců (`A` - `E`).

Přístup k řádku nebo jeho vytvoření pracuje s elementem `row` a unikátním indexem `r`. Jestliže řádek ve struktuře neexistuje, musí být vytvořena minimální kostra:

/xl/worksheets/sheet1.xml
```
<sheetData>
   …
   <row r="3"/>
   …
</sheetData>
```

Práce se sloupcem se odkazuje na jeho uměle vytvořenou strukturu agregací příslušných buněk shodného indexu nebo názvu sloupce. Přímo v XML struktuře lze nalézt pouze zápis stylu sloupců v elementu `cols`. Přístup ke sloupci skrze index představuje nalezení tohoto indexu v rozsahu atributů `[min – max]`. Přístup pomocí jména sloupce vyžaduje přepočet názvu na index a shodný přístup. Vytvoření neexistujícího sloupce `D` znamená minimální zápis:

```
<cols>
   …
   <col min="4" max="4" />
   …
</cols>
```

Pro načtení informací o buňce pracovního listu musí být specifikován index řádku a index případně název sloupce. Údaj o listu, na kterém se buňka vyhledává, je již znám. Zápis do struktury musí rozlišovat, zda daný řádek existuje. Neexistující řádek se nejprve vytvoří postupem uvedeným výše. Pokud řádek existuje, pak lze jako jeho potomka zapsat element buňky `c` s atributem `r` odpovídajícího umístění (kombinace pozice sloupce a řádku).

```
<row r="3">
   …
   <c r="C3"/>
   …
</row>
```

Uzamčení pracovního listu znamená možnost otevření pouze pro čtení, v aplikaci MS Excel se zápis do uzamčeného listu musí explicitně povolit. Informace o uzamčení obsahuje element `sheetProtection`, který upřesňuje objekty dokumentu, kterých se uzamčení týká:

```
<worksheet …>
   …
   <sheetProtection sheet="1" objects="1" />
   …
</worksheet>
```

Ukotvení v pracovním listu představuje stanovení horního počtu řádků (začínající prvním řádkem), které budou zobrazeny staticky. Při rolování aplikací ve vertikálním směru (posunu zobrazení dokumentu směrem dolů) budou ukotvené řádky stále viditelné jako hlavička sloupců. Struktura zápisu je složená z elementu `sheetView`, který obsahuje atributy určující první buňku (tedy zároveň řádek a sloupec), na kterých začíná ukotvení a nastavuje jeho povolení. Potomkem je element `pane`. Ten říká, která část pracovního listu je pohyblivá, a tento první nezávislý řádek a sloupec specifikuje pozicí první umístěné buňky. Příklad představuje horizontální ukotvení prvního řádku pracovního listu, druhý řádek je již nezávislý a roluje společně s obsahem:

```
<worksheet …>
   …
   <sheetView topLeftCell="A1" workbookViewId="0"
    tabSelected="1">
      <pane ySplit="1" topLeftCell="A2"
       activePane="bottomLeft"
    state="frozen" />
   </sheetView>
   …
</worksheet>
```

Mezi užitečné funkce v tabulkových dokumentech patří bezesporu také filtrování dat. V případě velkého množství záznamů lze pohled na data podle stanovených kritérií omezit. V aplikaci MS Excel je filtrování pro jednotlivé sloupce listu nastavováno ve vyskakovacím okně rozhraní. Filtr lze použít nad daty, jejichž rozsah je umístěn v elementu `autoFilter` a jeho atributu `ref`. Do souboru `/xl/workbook.xml` musí být zapsán předpis pro filtrování. Tento předpis definuje vazbu na konkrétní pracovní list podle indexu `localSheetId`, ve kterém mají být filtrovaná data zobrazena. Hodnota uvnitř elementu upřesňuje rozsah dat (jejich umístění v rámci pracovních listů a konkrétních buněk), která mají být pro potřeby filtru zpracována automatickou filtrovací funkcí konkrétní aplikace:

/xl/worksheets/sheet1.xml

```
<worksheet …>
   …
   <autoFilter ref="A2:D4" />
   …
</worksheet>
```

/xl/workbook.xml

```
<sheets>
   <sheet name="List1" sheetId="1" r:id="rId1" />
   <sheet name="List2" sheetId="2" r:id="rId2" />
</sheets>
…
<definedNames>
   <definedName name="_xlnm._FilterDatabase"
    localSheetId="0">
      List1!$A$2:$D$4
   </definedName>
   …
</definedNames>
```

Porovnání řádků dvou listů je programovou záležitostí, kdy se prochází struktura řádek po řádku a řádky se vzájemně porovnávají na shodu indexů a obsahu. Funkce vrací indexy řádků, které se neshodují.

Podpora šablonování spočívá ve vyhledání buňky s předpisem pro šablonu a provázání s daty odpovídajícího datového typu (tabulka, kolekce tabulek). Předpis pro šablonu musí mít specifickou syntaxi, aby nebyla pravděpodobná jeho shoda s běžně vkládanými daty a tak vzniklou následnou chybnou interpretací. Předaný datový typ pak musí obsahovat popis vnitřní struktury, pomocí které je možné jeho data přiřadit. Při plnění šablony probíhá vytváření buněk podle množství vkládaných dat na místě a ve směru jejich předpisu. Kopírován je kompletně také nastavený styl buněk.

4.3.6 Pracovní sešit (workbook)

Parsování dokumentu do struktury objektů je ve své podstatě programovou záležitostí, kdy se do instancí navržených objektových tříd (reprezentujících sešit, listy, sloupce, řádky, buňky a styly dokumentu) vloží příslušná data jako jejich atributy. Atributy budou v rámci objektu reprezentovat důležité informace reflektující požadavky na znalost o konkrétní části dokumentu. Funkcionalita bude dostupná skrze volání metod objektů.

Převody mezi indexy a názvy sloupců nebo buněk jsou užitečné, protože při určování pozice dat je výhodnější používat číselné indexy pozice, kdežto standard jmenuje sloupce abecedně (`A–Z`, `AA–AZ`, atd.) a buňky jako kombinaci názvu sloupce a čísla řádku (`A1`, `B2`, atd.). Indexy sloupců jsou číslovány v rozsahu `[0…n]` a indexy řádků `[1…n]`. Buňce pojmenované `C4` odpovídá dvojice indexů `[2, 4]` a sloupci s názvem `H` index `7`.

Tabulkový dokument reprezentuje datum a čas jako číselnou hodnotu [29]. Celá část představuje den od data 1. 1. 1900 a desetinná část představuje čas. Hodnota `.0` odpovídá času 00:00:00 a hodnota `.99999` času 23:59:59. Například datum 11. 7. 2013 a čas 13:11:20 budou v dokumentu uloženy jako společná číselná hodnota `41466,5495474421` s příznakem stylu buňky, že se jedná o speciální hodnotu. Formátovací maska buňky souvisí také s výše uvedenou reprezentací data a času. Podobu zobrazované hodnoty lze pomocí masky měnit, například čas 13:11:20 pod maskou `"h:mm AM/PM"` bude zobrazen jako `"1:11:20 PM"`.

Podpora kopie dokumentu (klonování) znamená možnost práce nejenom s dokumentem jako souborem formátu OOXML, ale také s jeho programovou reprezentací, kdy jsou veškeré programové struktury inicializovány a naplněny daty. Při začátku standardního zpracování dokumentu je třeba jeho soubor nejprve dekomprimovat, vybrat postupně odpovídající soubory vnitřní struktury a ty zpracovat. Po této fázi je knihovna připravena poskytnout objekty dokumentu a jejich atributy a metody. V případě podpory klonování je v tuto chvíli možno celou programovou reprezentaci dokumentu uložit do binárního souboru a při příštím zpracování znovu načíst z binárního souboru. Binární soubor lze sdílet,

použít jako výchozí šablonu a je přitom ušetřena nutnost počáteční inicializace a parsování dokumentu. Programově se jedná o takzvanou serializaci, což je uložení stavu objektů do bytového proudu a následné uložení do paměti. Opačný postup je nazýván deserializací.

Název pracovního sešitu je názvem celého dokumentu, respektive souboru. Lze jej získat programově mimo přístup k vnitřní XML struktuře, obvykle při prvním přístupu ke čtení nebo zápisu.

Počet pracovních listů sešitu lze zjistit v souboru `/xl/workbook.xml` v elementu `sheets` na základě počtu potomků s názvem `sheet`.

Získání pracovního listu je možné podle jeho unikátního názvu (`"List1"`) nebo indexu v rozsahu `[0…n]`, kde index `0` označuje první pracovní list ve struktuře označovaný atributem `sheetId="1"`. Přístup vyžaduje soubor `/xl/workbook.xml`, ve kterém je vyhledán element `sheet` s atributem `sheetId` odpovídajícím přepočítanému indexu nebo atributem `name` odpovídajícím názvu listu. Atribut `r:id` odkazuje pomocí vlastního identifikátoru do souboru `/xl/_rels/workbook.xml.rels`. Element `Relationship` shodného obsahu atributu `Id` (v tomto případě `Id="rId1"`) určuje jednoznačně cestu k XML souboru požadovaného pracovního listu:

/xl/workbook.xml

```
<workbook …>
   …
   <sheets>
      <sheet name="List1" sheetId="1" r:id="rId1" />
      <sheet name="List2" sheetId="2" r:id="rId2" />
   </sheets>
   …
</workbook>
```

/xl/_rels/workbook.xml.rels

```
<Relationships …>
   …
   <Relationship Id="rId1" Type="…"
    Target="worksheets/sheet1.xml" />
   <Relationship Id="rId2" Type="…"
    Target="worksheets/sheet2.xml" />
   …
</Relationships>
```

Vytváření nového pracovního listu vyžaduje zápis nového elementu `sheet` do `/xl/workbook.xml` s atributy unikátního jména `name`, indexu `sheetId` a identifikátoru `r:id`. V souboru `/xl/_rels/workbook.xml.rels` musí být vložen záznam o vazbě identifikátoru `Id` na soubor v atributu `Target` a tento

soubor `/xl/worksheets/sheetX.xml` s odpovídající strukturou (alespoň minimální povinná kostra elementů a atributů) vytvořen:

`/xl/worksheets/sheet3.xml`

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<worksheet …>
   <sheetViews>
      <sheetView tabSelected="1" workbookViewId="0"/>
   </sheetViews>
   <sheetData/>
</worksheet>
```

Informace o nově vzniklém souboru struktury je třeba nakonec zapsat do souboru `/[Content_Types].xml`:

```
<Types …>
   …
   <Override PartName="/xl/worksheets/sheet3.xml"
    ContentType="application/vnd.openxmlformats-
    officedocument.spreadsheetml.worksheet+xml"/>
   …
</Types>
```

Ukládání dat předpokládá uložení původní, změněné nebo nové struktury dokumentu do souboru nebo datového streamu. Knihovna připraví obsah všech XML souborů a zapíše je podle definice standardu Open Office XML do struktury, kterou zkomprimuje jako soubor s příponou `.xlsx` nebo `.xlsm` (podpora maker).

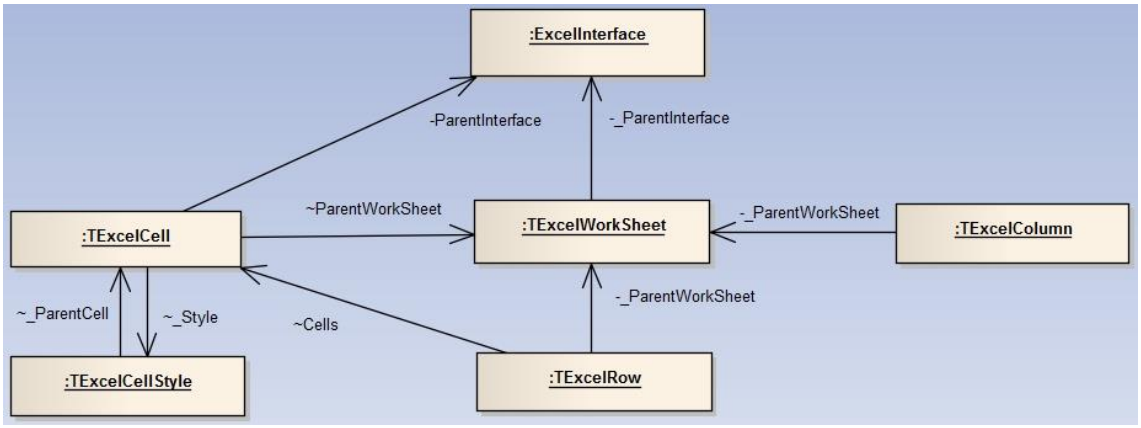
4.4 Objektový návrh a rozhraní knihovny

Definice standardu OOXML popisuje části tabulkových dokumentů, které se dále dělí na menší součásti a přirozeně tak vzniká hierarchie struktury. Mimo hierarchie XML souborů uvnitř obálky formátu pak lze identifikovat hierarchii objektů dokumentu, o jejichž dělení se zmiňovala předchozí kapitola.

Jednou z výhod objektově orientovaného programování je možnost reprezentace entit jako tříd (respektive objektů) uchovávajících svůj stav a poskytujících metody pro operace s nimi. Objekt je v tomto případě konkrétní instancí třídy.

Postupem shora dolů, od nejvyšší části dokumentu po tu nejnižší, při zachování jisté úrovně abstrakce, lze tabulkový dokument chápat jako: pracovní sešit (workbook) skládající se z pracovních listů (worksheets) a každý z pracovních listů skládající se z řádků (rows), které dále obsahují buňky (cells). Za účelem programové implementace knihovny se jeví jako nejvýhodnější

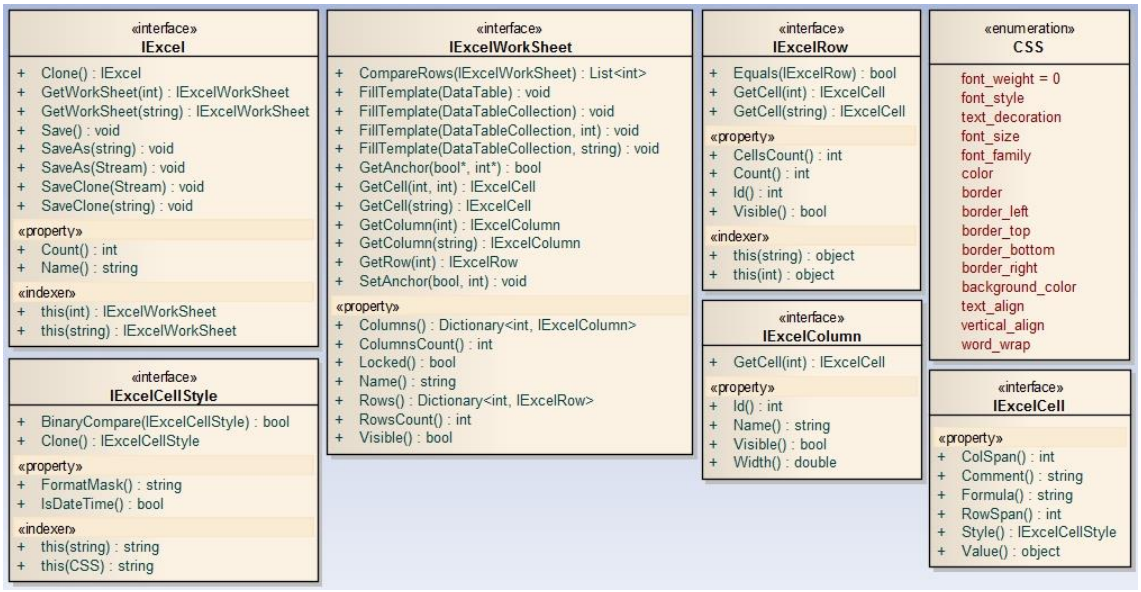
navrhnout třídy [30] dle tohoto dělení, které budou poskytovat atributy a metody diskutované v kapitole 4.3. Diagram tříd na obrázku 14 nezobrazuje atributy a metody poskytované třídami (z důvodu jejich velkého množství). Obsah tříd bude podrobně vysvětlen v kapitole 5.



Obr. 14 Diagram tříd a jejich vzájemných vazeb.

Třída `ExcelInterface` zastupuje entitu pracovního sešitu a implementuje rozhraní `IExcel`. Podobně lze z obrázku určit ostatní entity, k nim přidružené třídy a jejich vzájemné vazby. Aby bylo možné přiřadit každé buňce stylový vlastní předpis a také pro podporu nezávislé práce s buňkou a jejím stylem, je třída stylu `TExcelCellStyle` vyjmuta ze třídy `TExcelCell`.

Pro vývojáře využívající knihovnu je nejpodstatnější API rozhraní, které knihovna poskytuje pro práci s formátem Open Office XML. Na základě stanovené funkcionality jsou veřejně přístupné metody a atributy API rozhraní navrženy a znázorněny na obrázku 15.



Obr. 15 API rozhraní knihovny.

5 Implementace

Kapitola Implementace popisuje praktickou realizaci principů a navržených postupů pro knihovnu Open Office XML. Popisuje jako celek vytvořenou knihovnu a její jednotlivé funkční části na základě teoretického základu a navrženého postupu řešení.

Vývoj knihovny probíhal pod operačním systémem MS Windows 8.1 Professional 32-bit s využitím nástrojů platformy .NET verze 4.0. Jako vývojové prostředí byl použit program MS Visual Studio Professional 2010 s implementačním jazykem C# 4.0.

5.1 Popis postupu řešení

Programové zpracování struktury souboru OOXML využívá knihovnu SharpZipLib pro přístup k obsahu a následný přímý přístup k XML souborům skrze přímé čtení a modifikaci vnitřního značkovacího jazyka. Reference na potřebné knihovny byly vloženy do projektu MS Visual Studia a uvedením direktivy `using` pro každou z nich se zpřístupnily jejich jmenné prostory (třídy, hodnoty, datové typy, události a další). Knihovna využívá následujících jmenných prostorů:

- `System` - standardní jmenný prostor
- `System.Collections.Generic` - složitější datové typy (seznamy, slovníky, ...)
- `System.Text` - práce se znaky a řetězci
- `System.Text.RegularExpressions` - regulární výrazy
- `System.Data` - struktury ADO.NET (tabulky, kolekce tabulek, apod.)
- `ICSharpCode.SharpZipLib.Zip` - podpora komprese/dekomprese
- `System.IO` - čtení a zápis do souborů nebo datových streamů
- `System.Xml` - zpracování XML
- `System.Linq` - podpora LINQ (manipulace s daty)
- `System.Runtime.Serialization.Formatters.Binary` - serializace/deserializace objektů

Přístup ke zpracování dokumentu je obecně takový, že podstatné informace z jeho struktury jsou uchovávány ve vnitřních atributech (proměnných) příslušných objektů, které jsou realizovány složitějšími datovými typy (seznam, slovník uchovávající dvojice `klíč:hodnota` a podobně) nebo příznaky (pravdivostní hodnota). Knihovna nad objekty zpřístupňuje rozhraní veřejně

přístupných atributů a metod, které s vnitřními daty manipulují a vhodně je zpracovávají do výsledné podoby. Konkrétním příkladem může být vnitřní proměnná `Dictionary<string, int> WorkSheetsNames`, která je datového typu slovníku s klíčem datového typu řetězce a hodnotou datového typu celého čísla. Tato proměnná uchovává informace o názvu pracovního listu a jeho unikátním identifikátoru. Při čtení souboru `/xl/workbook.xml`, kde jsou tato data uložena, je proměnná naplněna daty. Pro získání konkrétního pracovního listu podle jeho názvu je pak nalezen odpovídající identifikátor, který přesně identifikuje daný list a je možné skrze něj přistoupit do indexované proměnné `List<TExcelWorksheet> _WorkSheets` uchovávající objekty pracovních listů.

Nad zvolenými datovými typy lze poměrně jednoduše iterovat a tento postup umožňuje využít jedné z optimalizací knihovny, kterou je sekvenční vytváření obsahu XML struktury zmíněné v kapitole 5.1.2 popisující zápis dat. Podobně sekvenční čtení umožňuje efektivní přístup prezentovaný dále.

Všechny objekty knihovny předávané funkcemi jako návratové hodnoty je nutné vždy kontrolovat na neexistující (`null`) hodnotu. Pokud funkce vrátí `null` objekt, pak daný objekt nebo vlastnost neexistuje a další práce s tímto objektem, jako je volání jeho metod nebo práce s veřejně přístupnými atributy, způsobí přístup k neinicializované paměti a vyvolá příslušnou výjimku.

Sloupce se číslují v rozsahu `[0…n]`, tedy od indexu `0` odpovídajícímu sloupci `"A"`. Řádky se číslují v rozsahu `[1…n]`, tedy od indexu `1` odpovídajícímu prvnímu řádku.

5.1.1 Čtení obsahu dokumentu

Čtení obsahu dokumentu představuje čtení a reprezentaci uložených informací, které jsou pro uživatele relevantní v rámci požadované funkcionality knihovny. Konstruktor knihovny očekává předaný dokument, který parsuje následujícím způsobem:

Tabulkový dokument je reprezentován jako datový stream, který dále zpracovává knihovna SharpZipLib. Ta dekomprimuje jeho obálku a všechny uvnitř obsažené soubory umístí do seznamu. Při znalosti struktury podle standardu OOXML jsou očekávány soubory pojmenované podle definice. Například je vyhledán soubor sdílených řetězců `/xl/sharedStrings.xml` a pokud existuje, jsou v něm umístěné řetězce rozděleny do dvou slovníků:

```
SharedStrings[index] = value;
ReverseSharedStrings[value] = index;
```

Kdykoliv je nutné později přistoupit k hodnotě řetězce reprezentované indexem v buňce, případně zjistit hodnotu indexu konkrétního řetězce, je tato informace uložena v jednom z vytvořených slovníků a k dispozici.

Čtení XML struktury zajišťuje třída `XmlReader` určená pro rychlé a pouze dopředné zpracování XML dat. V tomto ohledu se jedná o přístup SAX, jehož výhody a nevýhody byly popsány v kapitole 4.2, avšak v kontextu použití byly nevýhody eliminovány, a přístup je tak značně efektivní. `Reader`, jako instance třídy `XmlReader`, se metodou `Read()` v nekonečném cyklu, ukončeném zpracováním posledního uzlu, posunuje na zpracování dalšího uzlu. U aktuálního uzlu (pozice) je schopen vrátit hloubku zanoření, název uzlu, typ uzlu (`XmlNodeType.Element` pro start-tag, `XmlNodeType.EndElement` pro end-tag apod.), atributy a hodnotu. Předpis struktury každého z XML souborů podle definice formátu tak lze převést na zdrojový kód, kdy jsou na očekávaných místech, respektive místech splnění podmíněných příkazů, čtena požadovaná data. Příkladem zpracování může být tělo funkce `ReadWorkSheetsNames()`, která z `/xl/worksheet.xml` získává názvy, identifikátory a viditelnost pracovních listů:

/xl/workbook.xml

```
<workbook …>
   …
   <sheets>
      <sheet name="List1" sheetId="1" r:id="rId1" />
      <sheet name="List2" sheetId="2" r:id="rId2" />
   </sheets>
   …
</workbook>
```

ReadWorkSheetsNames()

```
while (reader.Read())
{
   if(reader.Depth == 2 && reader.HasAttributes &&
      reader.Name == "sheet" && reader.AttributeCount > 0)
   {
      WorkSheetsNames[reader["name"]] =
      int.Parse(reader["r:id"].Replace("rId", ""));
      WorkSheetsVisibility[reader["name"]] =
      reader["state"] ?? "visible";
   }
}
```

Kód očekává data v elementu `sheet` umístěného v hloubce zanoření 2, přičemž element musí obsahovat atributy. Jednoduchý kód získá požadovaná data a naplní jimi určené slovníky, přičemž se zápisem podmínky zároveň vyhne zpracování nežádoucích uzlů, které by však mohly vykazovat podobnost struktury či obsahu.

Obdobným postupem jsou zpracována ostatní data dokumentu a po skončení zpracování jsou vnitřní proměnné naplněny, takže k jejich obsahu může uživatel přistupovat skrze definované rozhraní.

5.1.2 Zápis obsahu dokumentu

Zápis obsahu probíhá na úrovni rozhraní pouze do vnitřních proměnných a teprve při ukládání metodou `Save()` nebo `SaveAs()` jsou změny reflektovány do samotného souboru dokumentu. Přístupem k určitému atributu objektu (přímo nebo pomocí metody) a vložením hodnoty je nastaven příznak modifikace tohoto objektu. Odložený zápis má výhodu především v počtu přístupů ke struktuře při ukládání. Pokud by byla každá změna ihned zpracována a uložena, znamenalo by to nutnost dekomprese, vyhledání příslušného souboru, vyhledání umístění v XML struktuře pro nová data, ošetření případných konfliktů a závislostí na již existujících datech, vložení a kompresi. Například v případě generování obsahu většího počtu buněk by byl takový postup časově značně neefektivní.

Odložený zápis knihovny probíhá následovně: Funkce `FindModifications()`, volaná při ukládání, vyhledá v objektech knihovny informace o jejich modifikaci a do seznamu souborů určených k úpravám vloží odpovídající záznamy. Jestliže byla v průběhu operací s knihovnou přidána buňka obsahující řetězec, je nastaven příznak `SharedStringsModifier` na hodnotu `true` a do seznamu upravovaných souborů přibude záznam:

```
Modifications.Add("xl/sharedStrings.xml");
```

Podobně pro styly, komentáře a mnohé jiné.

Další průběh zápisu zajišťuje tělo funkce `Modify()`. Ta přebírá množinu `Modifications` a do výstupního streamu nebo do souboru vkládá upravené nebo nově vytvořené XML. Nad proměnnými, které uchovávají čtená i změněná data, lze jednoduše iterovat, což poskytuje nástroj pro zrychlení zápisu. U existujících souborů by se muselo postupovat vyhledáním umístění v XML struktuře pro nová data a ošetřením případných konfliktů a závislostí na již existujících. Jestliže má však knihovna všechny informace potřebné pro vytvoření validního obsahu uloženy uvnitř proměnných, je výhodnější XML obsah vytvořit znovu s programovým ošetřením zmíněných konfliktů a závislostí. Využívá k tomu třídu `XmlWriter`, která metodami jako `WriteStartElement()`, `WriteAttributeString()`, `WriteEndElement()` a podobnými vytváří sekvenčně XML strukturu a plní ji daty předaných proměnných. Každý ze specifických souborů má vlastní funkci implementující vytvoření jeho struktury, například `ModifyWorkSheets()` pro data pracovního listu nebo `ModifySharedStrings()` pro data sdílených řetězců:

```
ModifySharedStrings()
foreach (string sharedString in SharedStrings.Values)
```

```
{
   Document.WriteStartElement("si");
   Document.WriteStartElement("t");
   Document.WriteValue(sharedString);
   Document.WriteEndElement();
   Document.WriteEndElement();
}
```

Uvedenému kódu předchází vytvoření instance `Document` třídy `XmlWriter`, zapsání hlavičky (deklarace verze a kódování) použitého XML metodou `WriteStartDocument()` a zapsání kořenového elementu `sst` s atributem `xmlns` použitého jmenného prostoru. Po uvedeném kódu následuje uzavření kořenového elementu. Vygenerovaný obsah může vypadat následovně:

```
/xl/sharedStrings.xml
<sst xmlns="http://schemas.openxmlformats.org/
 spreadsheetml/2006/main">
   <si><t>první</t></si>
   <si><t>druhý</t></si>
   …
</sst>
```

Sekvenční zápis je s výhodou využit u souborů, u kterých je očekáváno zapisování velkého množství dat, a je proto využit při generování obsahu pracovních listů a sdílených řetězců.

Je nutné zmínit, že sekvenční zápis se nehodí pro všechny uvažované scénáře použití. Například manipulace se souborem `[Content_Types].xml` při přidávání nové součásti vyžaduje zachování informací o obsahu a pouze vložení nové informace. Sekvenční přístup není možný, protože knihovna nezpracovává všechny možné typy obsahu, které soubor povoluje a ukládá, a nemá o nich tedy potřebné informace v proměnných. Východiskem je použití alternativní třídy `XmlDocument` reprezentující DOM přístup (kapitola 4.2). Nad stromem uzlů lze implementovat vyhledávání příslušného místa pro vložení elementu nebo atributu s požadovanou hodnotou a připojení k existující struktuře dat. Příklad vložení informace o nově vzniklém souboru řetězců do `[Content_Types].xml`:

```
sse = Document.CreateElement("Override",
      Document.DocumentElement.NamespaceURI);
sse.SetAttribute("PartName", "/xl/sharedStrings.xml");
sse.SetAttribute("ContentType",
                 "application/vnd.openxmlformats-
                  spreadsheetml.sharedStrings+xml");
RootElement.AppendChild(sse);
```

Proměnná `sse` je instancí třídy `XmlElement` reprezentující element a `RootElement` představuje kořenový element s názvem `Types`. Ze zápisu je patrné vytvoření elementu (uzlu) a vytvoření a naplnění jeho atributů. Nově vzniklý uzel je připojen jako potomek kořenového.

Na závěr postupu zápisu (nezávisle na zvoleném přístupu) jsou do kompresní obálky zahrnuty také nezměněné soubory původní struktury, aby byla zachována celistvost a vazby prvků dokumentu.

5.2 Popis tříd

V následujícím textu bude prezentován obsah a funkcionalita tříd, které tvoří funkční celek implementované knihovny. Z důvodu rozsahu jsou prezentovány pouze nejdůležitější funkční části, přičemž detailní přehled lze získat přímo nahlédnutím do zdrojových souborů knihovny přiložených k této práci. Zdrojové kódy jsou komentovány stylem XML komentářů, což umožňuje generování vývojářské dokumentace (nástroj SandCastle) a zároveň podporuje zobrazení nápovědy IntelliSense při najetí kurzoru nad komentovaný prvek (třídu, metodu, atribut, …) v prostředí MS Visual Studio.

5.2.1 ExcelInterface : IExcel

Hlavní třída knihovny, která skrze svůj konstruktor parsuje obsah předaného dokumentu do vnitřních proměnných a zpřístupňuje ho skrze implementované rozhraní. Představuje úroveň pracovního sešitu a v rámci hierarchie zpřístupňuje či vytváří podřízené pracovní listy. Třída dále poskytuje funkce pro převody pozičního systému mřížky dat (indexy a názvy sloupců) a data, času nebo formátovací masky. Vnitřní strukturu dat je možné uložit do souboru nebo datového streamu.

Veškeré informace, které je možné číst nebo ukládat globálně na nejvyšší úrovni pracovního sešitu a s podřízenými objekty jsou spojeny až následně referencemi (identifikátory), mají v této třídě zástupnou proměnnou. Jedná se například o seznamy pro filtry, komentáře, styly, sdílené řetězce a pracovní listy.

Klonování dokumentu, tedy podpora načítání nebo ukládání serializované podoby programových objektů, je implementováno s využitím třídy `BinaryFormatter` a metod `Serialize()` či `Deserialize()`. Před definicí všech tříd musí být uveden atribut `[Serializable]` takto:

```
[Serializable]
public class ExcelInterface : IExcel
{ … }
```

Dalším požadavkem kladeným pro úspěšnou serializaci je využití vnitřních datových typů, které podporují serializaci také. V podstatě musí být celý obsah knihovny serializovatelný.

Čtení, respektive parsování souborů dokumentu, zpřístupňuje skrze obálku třída `ZipFile` uvnitř `ReadDataFromZipStream()`. Dekomprimovaný obsah dále přebírají specifické funkce čtení XML struktury (`ReadSharedStrings()`, `ReadStyles()`, atd.), které v příslušných souborech vyhledají obsah a naplní proměnné knihovny. V ohledu zpracování pracuje poměrně netriviálně funkce `ReadStyles()`. Ta musí při čtení obsahu přeformátovat styly zapsané syntaxí OOXML do podoby CSS, aby byl přístup ke stylu pro uživatele jak při čtení, tak zápisu konzistentní, a zároveň se orientuje pomocí indexů, prostřednictvím kterých výsledné styly identifikuje a spojuje.

Při ukládání dokumentu jsou volány funkce `FindModifications()`, `AddFile()` a `Modify()`. Nejprve se na základě pravdivostních příznaků objektů určí upravené informace, které v dalších specifických funkcích (`ModifyWorkSheets()`, `ModifyComments()`, atd.) upraví a vloží do struktury XML souborů metoda `Modify()`. Neexistující soubory jsou vytvořeny a přidány do obálky metodou `AddFile()` a třídou `ZipOutputStream`.

Funkce `ComputeStyle()` je volána při ukládání dokumentu, aby přeformátovala styly zapsané syntaxí CSS do stylové podoby dané formátem OOXML. Jedná se především o odstranění jednotek ze zápisu (`"px"`), formátování hexadecimálních zápisů barev a sloučení nového stylu do společného předpisu indexů `xf` pro vložení do elementu `cellXfs` bez kolizí s existujícími styly.

5.2.2 TExcelWorksheet : IExcelWorksheet

Třída obsahuje proměnné a metody typické pro úroveň pracovního listu. Získáním instance této třídy z objektu pracovního sešitu je doplněn kontext pro mřížku dat a jsou tak zpřístupněny konkrétní řádky, sloupce a buňky dokumentu. Samotný objekt pracovního listu je vytvořen a naplněn již při čtení na úrovni sešitu, případně vytvořen a inicializován na výchozí hodnoty při vytvoření. Jestliže se objekt listu musí odkazovat na rodičovský sešit, má k dispozici referenci `ParentInterface` a může tak informaci o své změně delegovat výše. Toho je využito například při přejmenování pracovního listu, kdy informace o změně názvu musí být přenesena a reflektována do rodiče, respektive jeho struktur, odkazovaných na pracovní list právě názvem. Na této úrovni jsou dále uchovány informace o názvu listu, viditelnosti, uzamčení, ukotvení a filtrech. Metody `GetCell()`, `GetColumn()` a `GetRow()` poskytují pohyb po mřížce dat a vrací příslušné objekty. Neexistující objekt je při volání metod vytvořen, vhodně uložen do vnitřních proměnných a vrácen. Třída definuje proměnnou `Dictionary<int,IExcelRow> Rows`, která obsahuje seznam řádků podle jejich indexu. Touto strukturou je možné přistoupit ke konkrétnímu řádku a následně v rámci něho ke konkrétní buňce.

Šablonovaní umožňuje naplnit dokument daty z předaných datových struktur podle předpisu. Dokument je možné připravit ručně nebo naplnit pomocí funkcí knihovny (zapsání obsahu do vybraných buněk). Takto vytvořená šablona je po zavolání metody `FillTemplate()` parsována a všechny buňky

odpovídající syntaxi šablony a obsahu předaných tabulek jsou naplněny daty. Pro každý řádek a sloupec tabulky je vytvořena nová buňka na pozici podle předpisu a další buňky se generuji ve vertikálním směru (pokud není specifikováno jinak). Chybně zapsaná šablona způsobí, že chybný předpis zůstává v dokumentu, protože se může jednat o data dokumentu. Správný předpis je smazán a nahrazen obsahem tabulek. Předpis může byt syntakticky správně zapsaný, ale struktura s daty nemusí danou informaci obsahovat. V takových případech jsou vkládány prázdné hodnoty a kopíruje se pouze styl buněk, aby byl zachován kompaktní vzhled vygenerovaného dokumentu. Metoda `FillTemplate()` jako parametry očekává strukturu datové tabulky `DataTable` nebo kolekci datových tabulek `DataTableCollection`. Volitelně může být přidán v případě kolekce tabulek ještě parametr určující defaultní tabulku. Knihovna se chová v případě datové tabulky způsobem popsaným výše, tedy pro nalezené dvojice `předpis:data` generuje obsah a chybějící přeskakuje. Ignoruje také volitelné parametry určené pro kolekce tabulek. V případě kolekce tabulek však knihovna nejprve vyhledá všechny tabulky, které danou informaci obsahují, a konkrétní tabulka je pak zvolena explicitně buď předpisem v šabloně, parametrem defaultní tabulky nebo je vždy zvolena první nalezena tabulka a to v tomto poradí.

Pro definici šablony a její funkčnosti jsou implementované předpisy umožňující doplnit do dokumentu jména, nadpisy sloupců a obsahy sloupců předaných tabulek. Formát předpisu je case-sensitive a je nutné přesně dodržet danou syntaxi (tabulka 1).

Název tabulky
Předpis `_#TN` (Table Name) v dané buňce bude nahrazen názvem tabulky. Pokud je použit volitelný parametr `_#Ti` (Table index) a šablona pracuje s kolekci tabulek, pak je doplněn název tabulky daného indexu. V případě, že není tabulka specifikovaná nebo index tabulky neexistuje, je vložen název defaultní tabulky, nebo první v pořadí.

Nadpis sloupce
Předpis bude nahrazen nadpisem specifikovaného sloupce. Předpis pro nadpis sloupce `_#CC` (Column Caption) se používá ve spojení s předpisem pro jeho hodnotu `_#CV` (Column Value). V případě kolekce tabulek se konkrétní tabulka se sloupcem vybere podle tabulky buňky s předpisem `_#CV` a obě buňky spolu musí těsně sousedit. Pokud nemá sloupec nastaven nadpis, použije se jako nadpis identifikátor sloupce.

Hodnota sloupce
Buňka s předpisem a její styl budou kopírováný pro všechny řádky sloupce na další umístění v nastaveném směru generovaní. Defaultní směr generování je vertikálně vpřed `"VF"` (Vertical Forward) a lze ho měnit parametrem `_#D` (Direction), kde `"V"` je vertikální směr, `"H"` horizontální směr, `"F"` je směr vpřed a `"B"` směr

vzad. Pokud je použit volitelný parametr `_#Ti` a šablona pracuje s kolekcí tabulek, pak jsou doplněna data sloupce umístěného v tabulce daného indexu. Neexistující index tabulky způsobí, že je vložen obsah sloupce defaultní tabulky nebo první v pořadí, která daný sloupec s daty obsahuje. Alternativně k parametru `_#Ti` lze použít parametr `_#Tn` (Table name), který určuje tabulku podle názvu. Neexistující název tabulky způsobí, že je vložen obsah sloupce výchozí tabulky nebo první v pořadí, která daný sloupec s daty obsahuje. Při použití obou parametrů `_#Ti` a `_#Tn` má prioritu index tabulky `_#Ti`.

Tab. 1 Šablonovací příkazy

Formát zápisu	Popis
`_#TN`	název tabulky (Table Name)
Volitelné parametry	
`_#Ti:"index_tabulky"`	index tabulky (Table index) [0..n]
`_#CC:"jmeno_sloupce"`	nadpis sloupce (Column Caption)
`_#CV:"jmeno_sloupce"`	hodnota sloupce (Column Value)
Volitelné parametry	
`_#D:"smer_generovani"`	směr generování (Direction) `[V/H/F/B]`
`_#Ti:"index_tabulky"`	index tabulky (Table index) `[0..n]`
`_#Tn:"nazev_tabulky"`	název tabulky (Table name)

5.2.3 TExcelColumn : IExcelColumn

Reprezentace všech buněk pracovního listu ležících pod shodným vertikálním indexem skrze všechny řádky. Na úrovni sloupce jsou uchovány proměnné reprezentující šířku, viditelnost a vlastní index či odpovídající pojmenování. Metoda `GetCell()` je schopna získat nebo vytvořit konkrétní buňku na sloupci uvedením jeho další souřadnice, tedy řádku. Uchovaná reference `ParentWorkSheet` na rodičovský pracovní list je využita při zápisu informace o jakékoliv modifikaci sloupce pro pozdější ukládání a také právě při získávání buňky. Samotná struktura sloupců v zápisu pracovního listu neexistuje a buňky jsou uloženy na řádcích pouze s uvedením sloupce (nikoliv jeho fyzické přítomnosti jako elementu). Proto ani programová reprezentace neuchovává duplicitně data o buňkách v řádcích a sloupcích. Funkce `GetCell()` se namísto toho odkáže na rodičovský pracovní list a vyhledá řádek uvedeného indexu metodou `ParentWorkSheet.GetCell()`, přičemž index sloupce je znám z aktuálního kontextu.

5.2.4 TExcelRow : IExcelRow

Třída řádku uchovává především proměnnou typu pole buněk `TExcelCell[] Cells` a množinu `SortedSet<int> _CellsModifier` obsahující indexy

změněných. Dále proměnné standardní vlastnosti indexu a viditelnosti a metodu `GetCell()` pro získání buňky na indexu či názvu sloupce. Uvnitř těla této metody je opět, podobně jako v případě sloupce, odkazováno na metodu `ParentWorkSheet.GetCell()` rodičovského objektu. Ačkoliv zde není její využití zřejmé, jedná se o nutnost ošetření existence příslušných objektů a zařazení do datových struktur na úrovni pracovního listu. Právě funkce na vyšší úrovni hierarchie je toho schopna, protože má k dispozici potřebné struktury a metody bez dalšího odkazování.

5.2.5 TExcelCell : IExcelCell

Buňka pracovního listu, respektive její implementační třída, je z hlediska obsahu dokumentu nejdůležitější. Uchovává totiž užitečná data tabulkového dokumentu za účelem jejich zobrazení a dalšího zpracování. Svými vnitřními proměnnými odkazuje jak na nadřazené třídy (sešit a list), tak na třídu svého stylu `TExcelCellStyle`. Obsahuje informace o své pozici v souřadném systému (index řádku a sloupce), sloučení s více buňkami a text případného komentáře. Tyto informace jsou také přeneseny do pracovního listu, na jehož úrovni jsou ukládány do struktury dokumentu.

Jak bylo uvedeno v kapitole 4.3.2, buňka může obsahovat pravdivostní, numerickou či řetězcovou hodnotu a typ specifikuje, jak má být hodnota reprezentována. Společně s hodnotou povoluje ještě čtení či zápis vzorce pro automatické výpočty do proměnné `Formula`. Přímé čtení hodnoty probíhá odkazem na proměnnou `Value`. Zvláštní zpracování a netriviální přístup vyžaduje zápis hodnoty. Ten do vnitřní proměnné `_Value` vloží data bez zpracování, a při dotazu na hodnotu je vrátí. Obsah typu řetězce je tak vždy k dispozici, oproti jeho indexové hodnotě příslušné tabulce sdílených řetězců. Druhou vnitřní proměnnou je `_SavedValue`, která může obsahovat právě zmíněný index do tabulky řetězců v případě řetězce, a jedná se tak již o hodnotu, která je ve výsledku ukládána do elementu buňky. Zápis tedy znamená zjištění typu vkládaných dat a případný přepočet zahrnující další operace. U řetězců se jedná o vyhledání indexu existujícího v tabulce sdílených řetězců nebo vytvoření záznamu nového, u data přepočet na číslo (kapitola 4.3.6) a nastavení příslušné formátovací masky stylu a podobně.

Příznak modifikace je v objektu buňky bytový, přičemž jednotlivé bity jsou nastavovány funkcí `Modifier()` podle toho, co vše se v buňce měnilo. Bitový příznak `Value` označuje změnu hodnoty, příznak `Formula` změnu vzorce a `Style` změnu stylu. Příznak `Save` všechny změny potvrzuje a odkazem do rodičovských objektů nastaví také jejich příznaky. Tím je dosaženo optimalizace, protože se nemusí prohledávat všechny buňky struktury a kontrolovat jejich vnitřní příznak modifikace, ale stačí projít přímo seznam buněk modifikovaných.

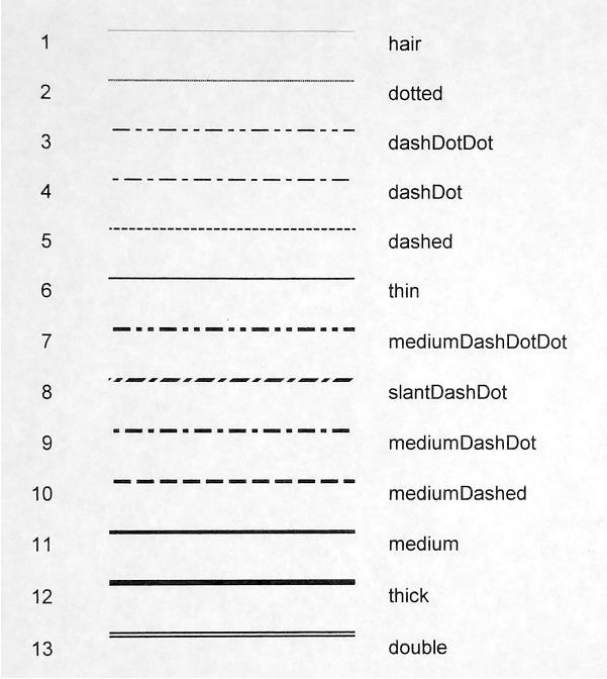
5.2.6 TExcelCellStyle : IExcelCellStyle

Třída stylu buňky byla pro přehlednost a lepší programovou manipulaci oddělena od samotné třídy buňky. Její obsah lze prezentovat jako množství proměnných definujících grafický vzhled otevřeného dokumentu v rámci aplikace kancelářského balíku. Mimo formátovací masku se jedná o styly odpovídající syntaxi a částečně sémantice CSS. Pří čtení souboru stylů jsou na úrovni pracovního sešitu tyto hodnoty ze zápisu formátu OOXML přepočítány a přeformátovány do CSS a uloženy podle indexů elementů `xf` do seznamu `List<TExcelCellStyle> _CellStyles`. Následně je při čtení elementů buněk získán také index stylu z atributu `s` a pomocí něho je do nově vzniklého objektu buňky přiřazen objekt stylu uložený na odpovídajícím indexu ze seznamu. Je patrné, že parsování stylů musí předcházet parsování buněk, aby byla potřebná data k dispozici. Třída obsahuje referenci na buňku, ke které přísluší, aby mohla v případě potřeby promítnout změny nastavením příznaku modifikace na vyšší úroveň.

Podporované stylování pomocí CSS, tedy předpisy a jejich povolené hodnoty (v hranatých závorkách) jsou následující:

```
font-weight = "[bold, normal]"
font-style = "[italic, normal]"
font-size = "[rozmer px]"
font-family = "[pismo]"
text-decoration = "[underline, none]"
color = "[#hexadecimalni_barva]"
text-align = "[left, right, center, justify]"
vertical-align = "[top, bottom, center]"
word-wrap = "[normal, break-word]"
border = "[(rozmer) styl barva]"
background-color = "[#hexadecimalni_barva]"
```

Syntaxe a sémantika příkazů odpovídá standardu CSS, kromě stylu čar, které jsou specifické pro formát OOXML a většina nemá ve standardu CSS odpovídající protějšek, proto je očekávaná hodnota stylu okraje podle obrázku 16.



Obr. 16 Specifické pojmenování stylu okrajů podle standardu OOXML
Zdroj: Stránky Kohei Yoshida's Webspace.

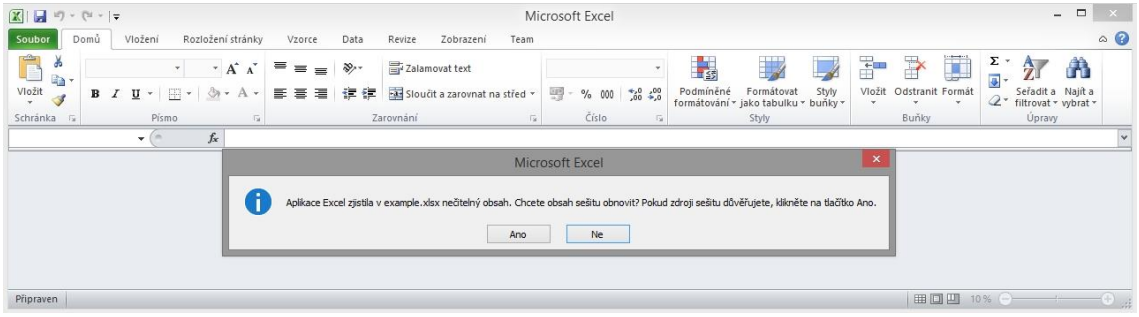
6 Testování

Tato část práce zahrnuje přehled a popis testování funkčnosti implementované knihovny. Zkoumána byla validita čtených a zapisovaných dat a ověření správnosti provedených změn při různém nastavení atributů a metod skrze veřejně přístupné rozhraní knihovny. Nad knihovnou byla také provedena série výkonnostních a zátěžových testů, která měla za cíl určit její limity a poskytnou uživateli náhled na výkonost implementace oproti jejím detailům zmíněným v předchozích kapitolách. Další část kapitoly srovnává z pohledu výkonu vlastní řešení s řešením ostatních placených či neplacených knihoven.

6.1 Vlastní knihovna

Testování knihovny se přirozeně prolínalo s implementační částí, během které byla ověřována funkcionalita právě realizovaných částí.

Validitu zapisovaných dat, a tedy splnění požadavků na strukturu definovaných formátem Open Office XML, bylo možné ověřit uložením do souboru a otevřením v aplikaci kancelářského balíku MS Excel. Pokud je soubor jakkoliv poškozen či jinak chybně zapsán a tedy neodpovídá standardu, hlásí aplikace Excel chybu podle obrázku 17. Aplikace upozorní na nečitelný obsah a vyzve uživatele k potvrzení pokusu o obnovení. Ten prakticky probíhá tak, že vnitřní struktura porušující definici je opravena, respektive je proveden pokus o její opravu. Může se jednat o vynechání chybných dat, rekonstrukci (pokud to související data dovolují) nebo dokument není možné opravit a okno nezobrazí jeho obsah.



Obr. 17 Chyba otevírání souboru a pokus o obnovu aplikace MS Excel.

Testování čtení dat probíhalo automatizovaně nad ručně vytvořeným dokumentem. V rámci ověření se procházel jeho obsah a nad objekty dokumentu se porovnávaly návratové hodnoty metod a hodnoty atributů s hodnotami očekávanými. V průběhu testů čtení a zápisu byla zjištěna následující omezení a specifika:

- povinná pořadí elementů,
- unikátní název pracovního listu,

- omezení kombinace `text-align` a `word-wrap`,
- odsazení indexů masek,
- výpočet dimenze,
- viditelnost prvků,
- překrývání spojených buněk.

Ve struktuře XML souborů je třeba zachovat pořadí elementů přesně stanovená standardem OOXML. Vložení elementu jako potomka kořenového uzlu (například v souboru pracovního listu) nestačí a je třeba dodržet také pořadí uvedených potomků. Element `sheetViews` uvedený v kořeni `worksheet` musí vždy předcházet elementům `sheetData`, `autoFilter` a `legacyDrawing` (kapitola 4.3.5). Shodně je vyžadováno řazení obsahu elementu `sheetData`, ve kterém jsou uložena data řádků, obsahující dále data buněk. Elementy `row` se řadí vzestupně podle hodnoty jejich atributu `r` (indexu řádku). Buňky uvnitř řádků reprezentované elementy `c` je třeba řadit vzestupně podle indexu sloupců v atributu `r`. Buňka s atributem `r="B2"` musí předcházet buňce s atributem `r="C2"` a tak dále, což vyžaduje přepočet názvu sloupce na index a odpovídající seřazení. V opačném případě je hlášen nečitelný obsah a dokument je třeba obnovit (aplikace seřadí elementy do správného pořadí).

Unikátnost názvu pracovního listu v rámci ostatních pracovních listů je z principu vazeb uvnitř dokumentu logická. Prvek se může odkazovat na pracovní list názvem a povolení duplicit by znamenalo nejednoznačnost ve vzájemném přiřazení. Obecně se také nedoporučuje změna názvu listu v dokumentech s podporou maker `.xlsm`, neboť kód makra může být vázán na název pracovního listu a kód se kvůli provedené změně může stát nefunkčním.

Aplikace MS Excel ve svém grafickém uživatelském rozhraní ani standard OOXML nepovolují uvnitř stylu buňky kombinaci hodnot přizpůsobení textu velikosti buňky `text-align="justify"` a zalamování slov podle šířky buňky `word-wrap="break-word"`.

Zápisy masek pro formátování obsahu buněk jsou uloženy v souboru `/xl/styles.xml` a elementech `numFmt` s atributem jejich indexu `numFmtId`. Ačkoliv by se logicky nabízelo volit index nové formátovací masky jako inkrementovanou hodnotu nejvyšší již existující hodnoty (například po elementu s `numFmtId="6"` přidat element s atributem `numFmtId="7"`), není to možné. Formát má totiž prostor rezervovaných identifikátorů pro předdefinované masky. Jestliže uživatel vkládá vlastní definici masky, musí její index začínat identifikátorem s hodnotou `301`.

Volitelným elementem XML zápisu pracovního listu je dimenze. `Dimension` svým atributem `ref` udává, v jakém rozsahu řádků a sloupců lze očekávat data. Tím zrychluje přístup k obsahu (aplikace ví, v jakém rozsahu buněk má načítat) a v případě poškození dovoluje lépe obnovit dokument, proto vlastní

knihovna tento element do struktury zahrnuje a v případě potřeby (změna rozsahu řádků nebo sloupců) přepočítává.

Zajímavý přístup volí aplikace MS Excel při skrývání prvků dokumentu (atributy `visible` u řádků a sloupců). Kromě viditelnosti pracovního listu, který je vždy skutečně skryt a není přes GUI přístupný, nastavuje jeden z rozměrů prvku na `0` (u řádků je to výška, u sloupců šířka). Prvek je tak fyzicky stále přítomen v mřížce dat a uživatel ho může posuvníkem a roztažením zviditelnit.

Při zápisu sloučení buněk (merge) není povoleno jejich překrývání. Pokud jsou například slučovány buňky v rozsahu `"A1:B3"` a `"A1:A3"`, pak je buňka `A1` konfliktní a po obnovení dokumentu nebude sloučení provedeno.

Oficiální stránky produktu MS Office [31] stanovují maximální limity pro data a vlastnosti dokumentu MS Excel podle tabulky 2. Nedodržení těchto limitů může vést k problémům se stabilitou a výkonností aplikace a nedoporučuje se je proto překračovat při programovém zpracování knihovnou.

Tab. 2 Limity vlastností aplikace MS Excel

Vlastnost	Maximální limit
Počet otevřených pracovních sešitů	Limitováno dostupnou pamětí a systémovými zdroji
Velikost pracovního listu	1 048 576 řádků a 16 384 sloupců
Šířka sloupce	255 znaků
Výška řádku	409 bodů
Celkový počet znaků, které může buňka obsahovat	32 767 znaků
Počet pracovních listů v pracovním sešitě	Limitováno dostupnou pamětí
Počet unikátních formátovacích předpisů nebo stylů buněk	64 000
Počet stylů výplní	256
Počet unikátních typů písma	1 024 globálně, 512 pro pracovní list
Počet číselných formátů v pracovním listu	200 – 250 (v závislosti na jazykové verzi aplikace MS Excel)
Číselná přesnost	15 desetinných míst
Celkový počet znaků vzorce	8 192
Rozsah vzorce (počet zahrnutých buněk)	2 048

Zdroj: Oficiální stránky produktu MS Office.

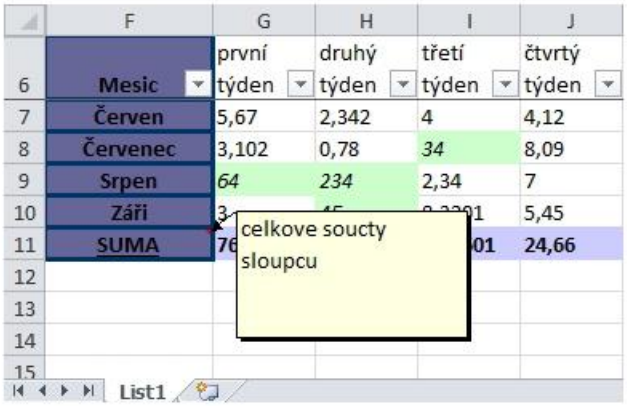
6.1.1 Příklad generování obsahu

Obrázek 18 uvádí ukázku kódu a výsledku generování obsahu dokumentu a uložení do souboru. Je vhodné na tomto místě upozornit, že uvedený příklad je minimální kostrou funkčního kódu a obecně je vždy nutné ošetřit návratovou

hodnotu metod na `null`, aby nad neexistujícím objektem nedocházelo k vyvolávání výjimek přístupu do paměti a podobně.

Konstruktor třídy pracovního sešitu, respektive knihovny `ExcelInterface()`, parsuje předaný zdrojový dokument a získá z něj metodou `GetWorkSheet()` první pracovní list. Tomu nastaví ukotvení prvního viditelného řádku a zároveň zapne filtr pro data. Metoda `GetColumn()` získá první viditelný sloupec a do atributu `Width` jeho objektu se přiřazuje nová šířka odpovídající 15 znakům. Metoda `GetRow()` získá řádek s indexem `6` a pomocí metody `GetCell()` s názvem sloupce je odkázána přímo buňka `F6`. Ta dostane přiřazen styl tučného písma a pozadí. Pro buňku `F11` zajišťuje předpis zapsání hodnoty řetězce `"SUMA"` a poznámky. Ve vynechaných částech jsou skryty řádky `1 – 5` a sloupce `A – E`, stylovány a plněny další buňky a vkládán vzorec součtu hodnot sloupce ve tvaru `"SUM(G7:G10)"`.

```
IExcel document = new ExcelInterface("template.xlsx", false);
IExcelWorkSheet worksheet = document.GetWorkSheet(0);
worksheet.SetAnchor(true, 1);
//...
IExcelColumn column = worksheet.GetColumn("F");
column.Width = 15;
//...
IExcelRow row = worksheet.GetRow(6);
//...
IExcelCell cell = row.GetCell("F");
cell.Style["font-weight"] = "bold";
cell.Style["background-color"] = "#666699";
//...
cell.Value = "SUMA";
cell.Comment = "celkove soucty sloupcu";
//...
document.SaveAs("output.xlsx");
```



Obr. 18 Ukázka programového kódu knihovny a výsledné podoby dokumentu.

Série měření závislosti času generování na počtu buněk se zohledněním přístupu po řádcích či po sloupcích a zahrnutí stylového předpisu je v tabulce 3 a výkonnost v tabulce 4.

Tab. 3 Výsledky měření času generování [ms] vlastní knihovny

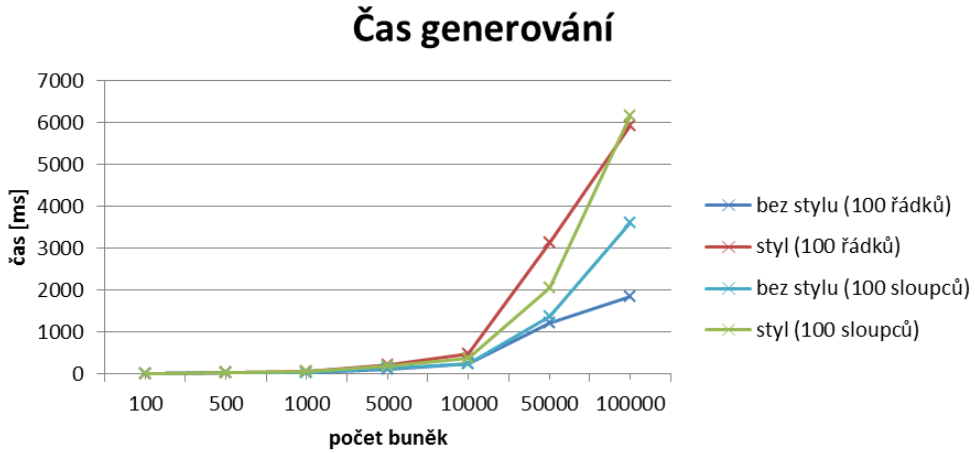
Počet buněk	Bez stylu (100 řádků)	Styl (100 řádků)	Bez stylu (100 sloupců)	Styl (100 sloupců)
100	16	16	12	14
500	22	30	20	22
1000	33	54	29	46
5000	97	205	130	180
10000	246	483	235	369

50000	1221	3141	1361	2061
100000	1846	5922	3601	6157
1000000	33036	122108	96946	138845

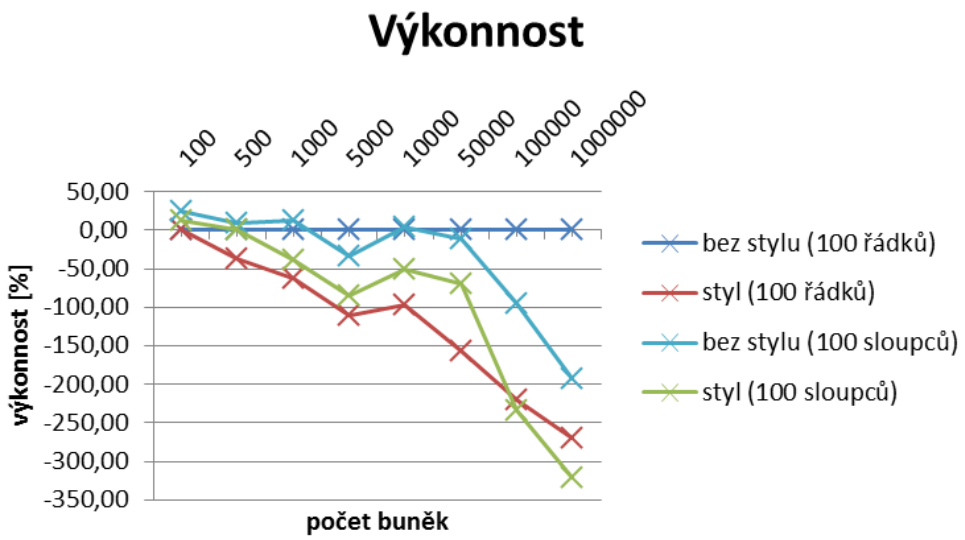
Tab. 4 Výsledky měření výkonnosti generování [%] vlastní knihovny

Počet buněk	Bez stylu (100 řádků)	Styl (100 řádků)	Bez stylu (100 sloupců)	Styl (100 sloupců)
100	0,00	0,00	25,00	12,50
500	0,00	-36,36	9,09	0,00
1000	0,00	-63,64	12,12	-39,39
5000	0,00	-111,34	-34,02	-85,57
10000	0,00	-96,34	4,47	-50,00
50000	0,00	-157,25	-11,47	-68,80
100000	0,00	-220,80	-95,07	-233,53
1000000	0,00	-269,62	-193,46	-320,28

Čas generování přirozeně kopíruje vnitřní datovou strukturu. Pokud je iterováno přes více sloupců (buněk) jednoho řádku, je přístup programově rychlejší než iterace nad větším množstvím řádků a menším množstvím obsažených sloupců (buněk). Objekt buňky je totiž potomkem řádku a je s ním tak zacházeno jak v proměnných, tak v XML struktuře.



Obr. 19 Graf výsledků času generování vlastní knihovny.



Obr. 20 Graf výsledků výkonnosti čtení vlastní knihovny.

Výsledky grafů ukazují očekávaný průběh času a výkonnosti. Iterace přes menší počet řádků je rychlejší a zahrnutí stylování do každého objektu buňky dále snižuje výkonost řešení. Pozitivním jevem je skutečnost, že nárůst není skokový a jedná se o přičtení časové konstanty.

6.2 Srovnání knihovny s existujícími knihovnami

Nad vlastní knihovnou byla provedena série testů a měření za účelem porovnání s existujícími knihovnami. Aby měly výsledky testů vypovídací hodnotu, byl navržen následující postup měření: programem MS Visual Studio, respektive jeho analytickými nástroji, byla měřena doba běhu knihovny od spuštění až po ukončení. V každém kroku byl stanoven výsledek, kterého musí knihovna zápisem zdrojového kódu (tedy metod a atributů poskytovaných skrze své rozhraní) dosáhnout. Kód knihovny byl následně spuštěn a měřen čas jeho běhu v interaci celkem deseti cyklů pro každý z případů. Zprůměrovaný výsledný čas je výsledkem provedených měření, které jsou prezentovány dále.

Vlastní knihovna je brána jako referenční vůči ostatním a v případě určení výkonnosti na základě měřených časů se postupuje vzorcem:

```
výkonnost = 100 - ( Y / X) * 100
```

- kde `Y` je čas srovnávané knihovny a `X` čas knihovny vlastní. Výsledky měření jsou vyneseny v přiložených tabulkách a grafech.

6.2.1 Zpracování a čtení dokumentu

Pseudokód testu pro měření rychlosti čtení dokumentu:

```
//rows = { 100 }
//cols = {1, 5, 10, 50, 100, 500, 1000}
foreach(col in cols){
   for(i = 0; i < ITERATIONS; i++){
      var dokument = new Library(FILEPATH);
      var worksheet = document.worksheet[0];
      for(r = 0; r < rows; r++){
         var val = worksheet.cell[row,0].Value;
         for(c = 1; c < col; c++){
            var cell = worksheet.cell[row,c];
            var val2 = cell.Value;
         }
      }
   }
}
```

Kód iteruje nad množstvím buněk a uvedeným počtem opakování měření (proměnná `ITERATIONS`) a vždy vytvoří objekt knihovny (rozparsuje předaný dokument v proměnné `FILEPATH`). Následně načítá objekt prvního pracovního listu, čte zvlášť hodnotu první buňky každého řádku a poté všechny ostatní buňky řádku.

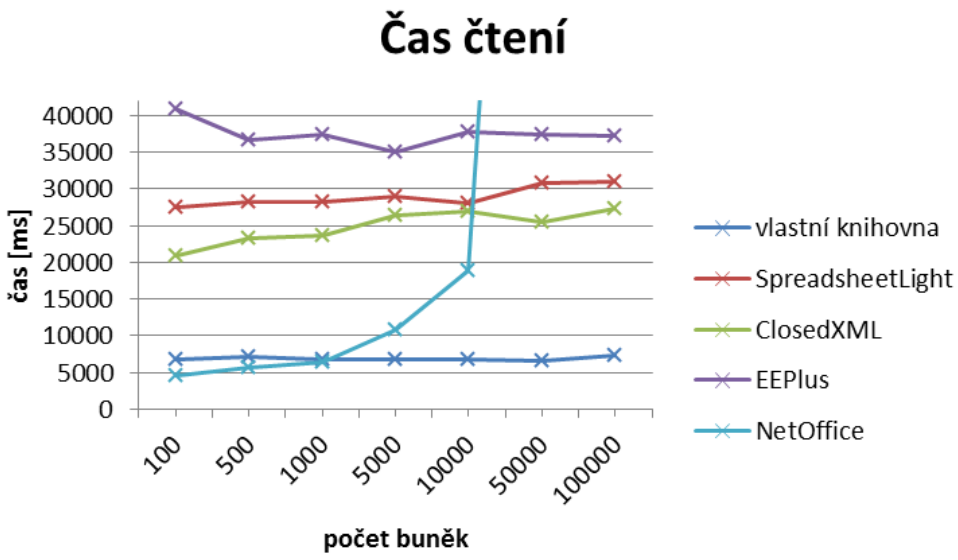
Tab. 5 Výsledky měření času čtení [ms] pro volně dostupné knihovny

Počet buněk	vlastní knihovna	SpreadsheetLight	NetOffice	ClosedXML	EEPlus
100	6842	27486	4636	20972	40083
500	7076	28316	5683	23365	36691
1000	6761	28167	6437	23700	37346
5000	6790	28903	10875	26421	35045
10000	6723	28007	18940	26881	37757
50000	6613	30848	167435	25512	37428
100000	7217	31085	832176	27355	37281

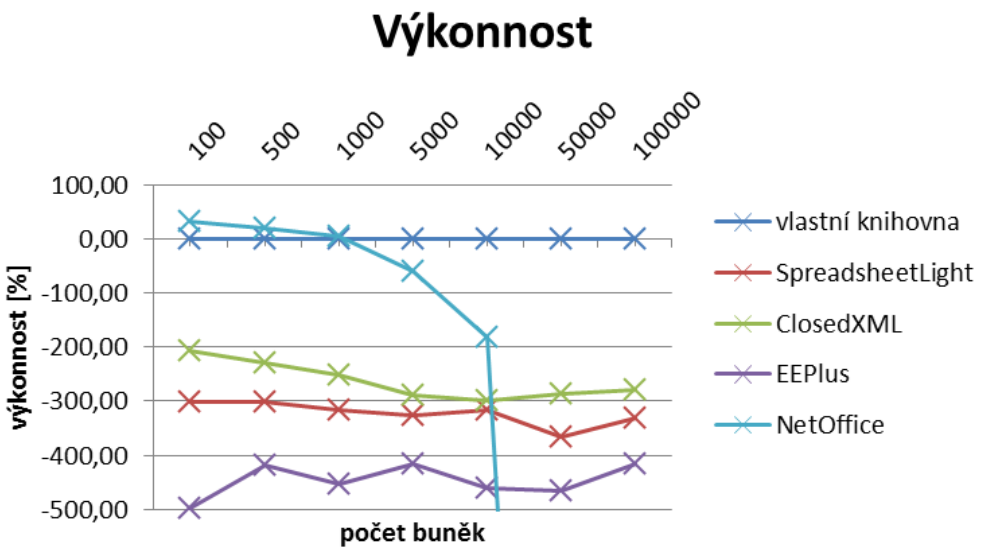
Tab. 6 Výsledky měření výkonnosti čtení [%] pro volně dostupné knihovny

Počet buněk	vlastní knihovna	SpreadsheetLight	NetOffice	ClosedXML	EEPlus
100	0,00	-301,72	32,24	-206,52	-497,68
500	0,00	-300,17	19,69	-230,20	-418,53
1000	0,00	-316,61	4,79	-250,54	-452,37
5000	0,00	-325,67	-60,16	-289,12	-416,13
10000	0,00	-316,58	-181,72	-299,84	-461,61
50000	0,00	-366,48	-2431,91	-285,79	-465,98
100000	0,00	-330,72	-11430,77	-279,04	-416,57

Čas zpracování a čtení 100 000 buněk je u vlastní knihovny průměrně 7,2 sekund, ostatní knihovny se pohybují v rozmezí 27 sekund – 13,9 minut, což u všech ostatních knihoven představuje horší výsledek o 279 – 417 % (v nejhorším případě dokonce o 11 431 %).



Obr. 21 Graf výsledků času čtení volně dostupných knihoven.



Obr. 22 Graf výsledků výkonnosti čtení volně dostupných knihoven.

Zajímavého průběhu patrného z grafů dosahuje oproti ostatním knihovna NetOffice. Ta neparsuje dokument do svých vnitřních datových struktur, ale přistupuje přímo k dokumentu skrze COM rozhraní aplikace Excel. Přístup do celkového počtu 1 000 buněk se jeví jako efektivní i oproti vlastní knihovně. Téměř nepoužitelnou se ovšem stává pro generování větších objemů, z důvodu její exponenciálního průběhu závislosti času na počtu generovaných buněk.

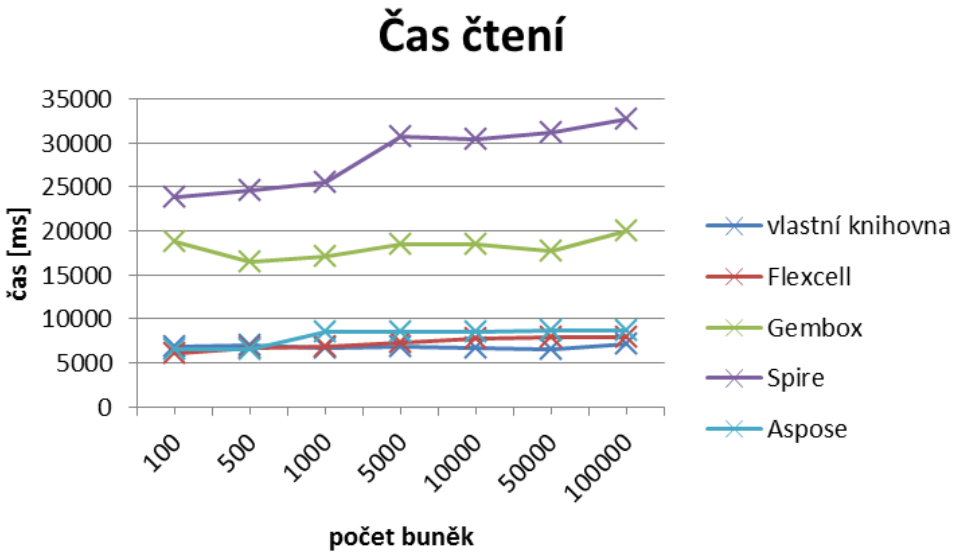
Tab. 7 Výsledky měření času čtení [ms] pro placené knihovny

Počet buněk	vlastní knihovna	Flexcell	Gembox	Spire	Aspose
100	6842	6065	18723	23904	6563
500	7076	6639	16525	24607	6564
1000	6761	6781	17084	25495	8505
5000	6790	7383	18535	30671	8563
10000	6723	7701	18464	30491	8537
50000	6613	7901	17661	31157	8619
100000	7217	7973	19996	32795	8695

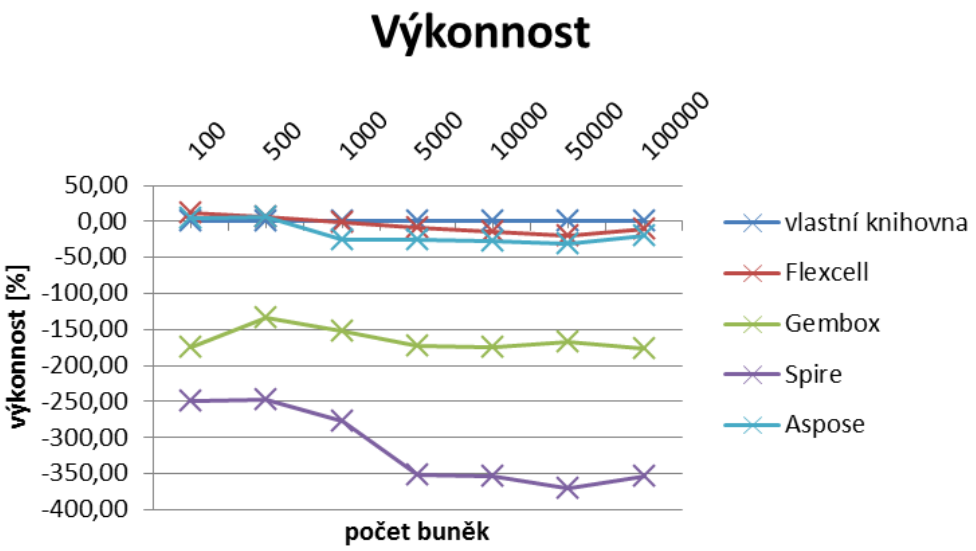
Tab. 8 Výsledky měření výkonnosti čtení [%] pro placené knihovny

Počet buněk	vlastní knihovna	Flexcell	Gembox	Spire	Aspose
100	0,00	11,36	-173,65	-249,37	4,08
500	0,00	6,18	-133,54	-247,75	7,24
1000	0,00	-0,30	-152,68	-277,09	-25,80
5000	0,00	-8,73	-172,97	-351,71	-26,11
10000	0,00	-14,55	-174,64	-353,53	-26,98
50000	0,00	-19,48	-167,06	-371,15	-30,33
100000	0,00	-10,48	-177,07	-354,41	-20,48

Čas zpracování a čtení 100 000 buněk je u vlastní knihovny průměrně 7,2 sekund, ostatní knihovny se pohybují v rozmezí 8 – 32,8 sekund, což u všech ostatních knihoven představuje horší výsledek o 10 – 354 %. Do počtu přibližně 700 buněk jsou nepatrně efektivnější knihovny Flexcell a Aspose.



Obr. 23 Graf výsledků času čtení placených knihoven.



Obr. 24 Graf výsledků výkonnosti čtení placených knihoven.

Z provedených měření volně dostupných a placených knihoven vyplývá, že čas parsování dokumentu, který vždy obsahoval 1 000 000 buněk, představuje největší část ze sledované doby běhu knihovny a následné čtení ovlivňuje čas v řádu sekund s rostoucím počtem čtených buněk. Knihovna vykazuje ve všech srovnáních zpracování a čtení dokumentu lepší výsledky než testované knihovny třetích stran.

6.2.2 Zápis dokumentu

Pseudokód testu pro měření rychlosti čtení dokumentu:

```
//rows = { 100 }
//cols = {1, 5, 10, 50, 100, 500, 1000}
foreach(col in cols){
   for(i = 0; i < ITERATIONS; i++){
      var dokument = new Library(FILEPATH);
      var worksheet = document.worksheet[0];
      worksheet.Name = "List_1";
      for(r = 0; r < rows; r++){
         worksheet.cell[row,0].Value = "id: "+row;
         for(c = 1; c < col; c++){
            var cell = worksheet.cell[row,c];
            cell.Value = c + row;
         }
      }
   }
```

```
}
```

Kód testu je téměř shodný s kódem testu pro čtení. Následně se načítá objekt prvního pracovního listu a tento list se přejmenovává. Probíhá vkládání řetězce do první buňky každého řádku a poté numerická hodnota součtu indexu řádku a sloupce do všech ostatních buněk řádku.

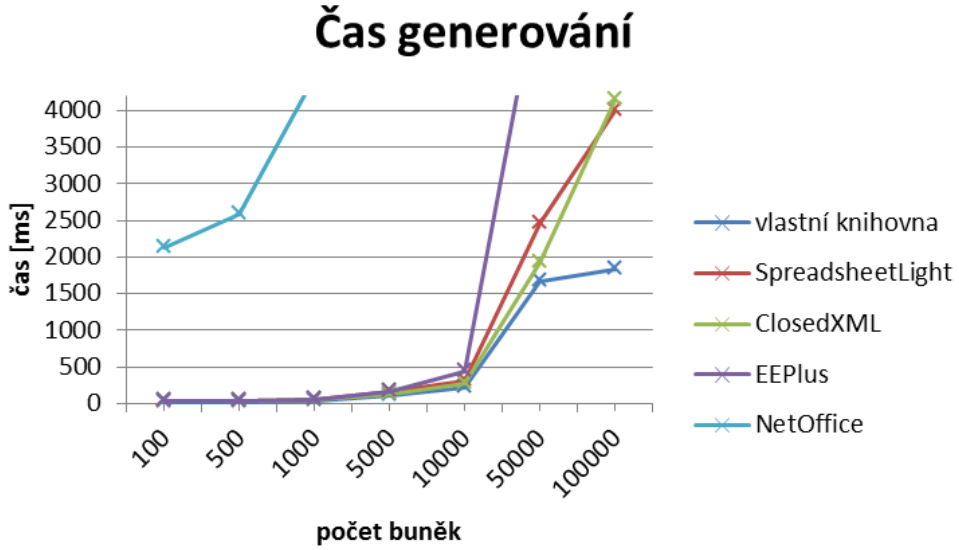
Tab. 9 Výsledky měření času zápisu [ms] pro volně dostupné knihovny

Počet buněk	vlastní knihovna	SpreadsheetLight	NetOffice	ClosedXML	EEPlus
100	19	35	2127	40	36
500	20	39	2578	41	28
1000	29	52	4452	43	42
5000	109	163	12364	126	166
10000	214	303	31330	280	443
50000	1671	2450	301656	1933	5874
100000	1834	4010	2159515	4154	22587

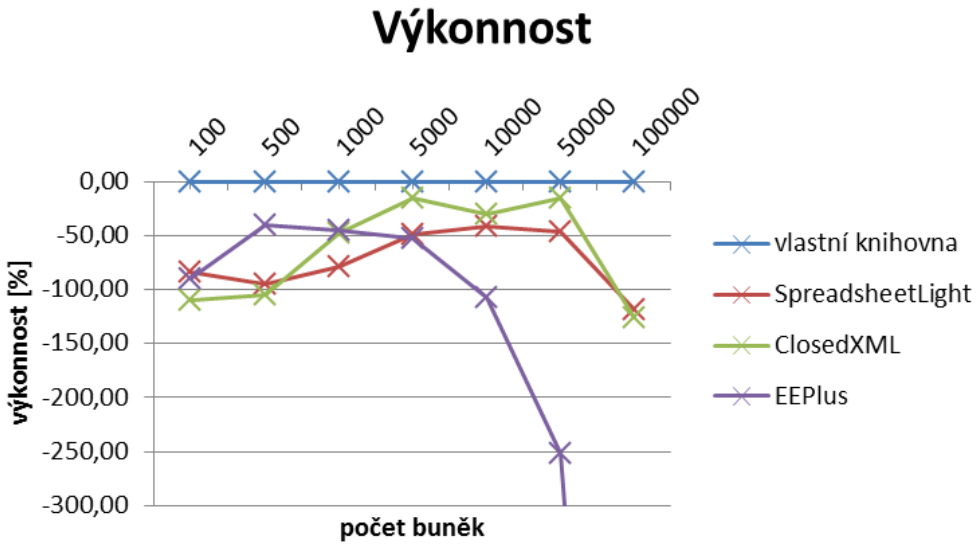
Tab. 10 Výsledky měření výkonnosti zápisu [%] pro volně dostupné knihovny

Počet buněk	vlastní knihovna	SpreadsheetLight	NetOffice	ClosedXML	EEPlus
100	0,00	-84,21	-11094,74	-110,53	-89,47
500	0,00	-95,00	-12790,00	-105,00	-40,00
1000	0,00	-79,31	-15251,72	-48,28	-44,83
5000	0,00	-49,54	-11243,12	-15,60	-52,29
10000	0,00	-41,59	-14540,19	-30,84	-107,01
50000	0,00	-46,62	-17952,42	-15,68	-251,53
100000	0,00	-118,65	-117648,91	-126,50	-1131,57

Generování 100 000 buněk trvá u vlastní knihovny průměrně 1,8 sekund, přičemž ostatní knihovny se pohybují v rozmezí 4 sekund – 36 minut, což u všech ostatních knihoven představuje horší výsledek o 119 – 1132 % (v nejhorším případě dokonce o 117 648 %).



Obr. 25 Graf výsledků času zápisu volně dostupných knihoven.



Obr. 26 Graf výsledků výkonnosti zápisu volně dostupných knihoven.

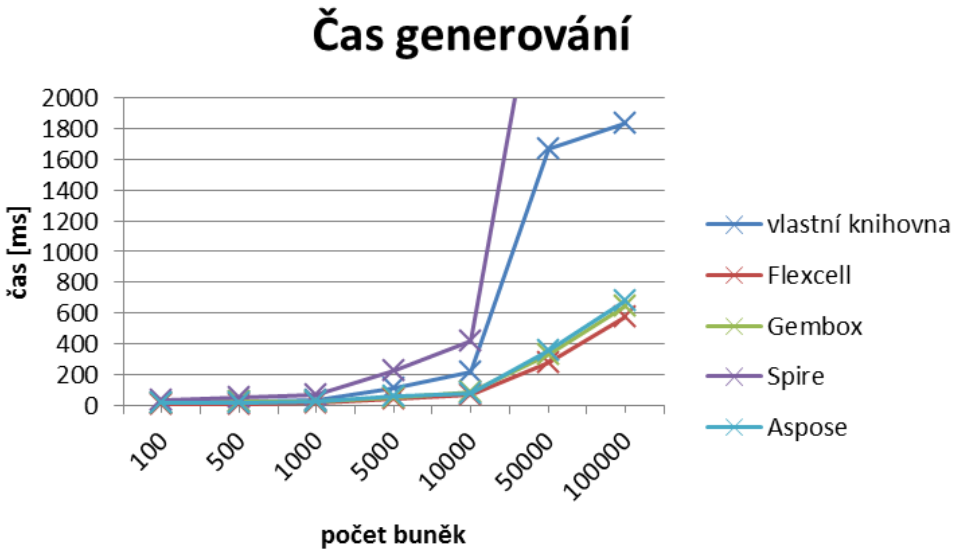
Z důvodu přehlednosti nebyla do grafu na obrázku 26 zahrnuta data knihovny NetOffice, jejíž výkonnost je mimo meze určené pro zobrazení osy grafu.

Tab. 11 Výsledky měření času zápisu [ms] pro placené knihovny

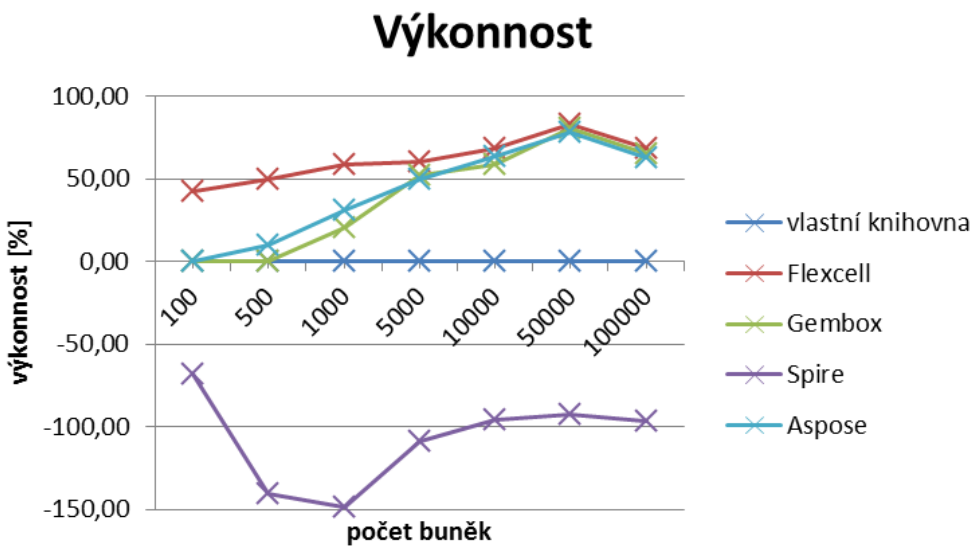
Počet buněk	vlastní knihovna	Flexcell	Gembox	Spire	Aspose
100	19	11	19	32	19
500	20	10	20	48	18
1000	29	12	23	72	20
5000	109	43	52	227	55
10000	214	68	89	418	78
50000	1671	279	330	3216	359
100000	1834	574	642	3598	681

Tab. 12 Výsledky měření výkonnosti zápisu [%] pro placené knihovny

Počet buněk	vlastní knihovna	Flexcell	Gembox	Spire	Aspose
100	0,00	42,11	0,00	-68,42	0,00
500	0,00	50,00	0,00	-140,00	10,00
1000	0,00	58,62	20,69	-148,28	31,03
5000	0,00	60,55	52,29	-108,26	49,54
10000	0,00	68,22	58,41	-95,33	63,55
50000	0,00	83,30	80,25	-92,46	78,52
100000	0,00	68,70	64,99	-96,18	62,87



Obr. 27 Graf výsledků času zápisu placených knihoven.



Obr. 28 Graf výsledků výkonnosti zápisu placených knihoven.

Na základě provedených měření volně dostupných knihoven lze říci, že knihovna vůči nim představuje rychlejší způsob práce s velkým množstvím dat. Oproti tomu u placených variant, dosahuje čas generování lepších výsledků. Pro celkový počet 100 000 buněk v rozmezí 65 – 69 %. Placená knihovna Spire však byla pomalejší celkem o 96 %.

7 Závěr

Cílem práce bylo navrhnout a implementovat knihovnu pracující s tabulkovým formátem Open Office XML dokumentů. Knihovna měla poskytovat rozhraní pro přístup k prvkům dokumentu podle zvyklostí programu Microsoft® Excel (pracovní listy, sloupce, řádky, buňky), jejich stylování a umožnit čtení i zápis dat do struktury.

Za účelem splnění stanoveného cíle byl položen teoretický základ formátu XML a z něj vycházejícího standardu Open Office XML, především motivace jeho vzniku a vnitřní struktury. Byly prozkoumány existující komerční i volně dostupné knihovny a diskutovány jejich výhody a nevýhody. Nad znalostí těchto kapitol mohl být následně vystavěn základ pro východiska, metodiku a návrh vlastní knihovny s cílem přínosného a inovativního řešení. Byla uvedena motivace pro vytvoření knihovny a možnosti nasazení v průmyslovém sektoru, společně s výběrem vhodné platformy a programovacího jazyka. Volbou kombinace přístupů ke struktuře dat dokumentu, jejich čtení a zápisu, se práce pokusila o optimální řešení problému, přičemž zároveň identifikovala klíčové informace obsažené ve struktuře a jejich vazby a závislosti. Nad těmito znalostmi byl postaven objektový návrh reflektující přirozenou hierarchii struktury tabulkových dokumentů a v kombinaci s požadovanou funkcionalitou také návrh veřejně přístupného rozhraní knihovny.

Realizační část práce popsala převod postupů do programové podoby zdrojového kódu s podstatnými detaily implementace tříd, jejich atributů a metod. S realizační částí se přirozeně prolínala část testování, během které byla funkcionalita ověřována. Nakonec byla provedena série testů s cílem potvrzení či vyvrácení správnosti optimálního návrhu a výkonnostní srovnání s ostatními knihovnami.

7.1 Zhodnocení řešení a možná rozšíření

Knihovna pro práci s formátem Open Office XML byla v rámci této práce úspěšně navržena, implementována a otestována. Programové rozhraní poskytuje sadu funkcí pro automatizované generování reportů a lze tak cílit na propojení s výrobním systémem, v rámci kterého představuje přidanou hodnotu řešení pro koncového zákazníka. Kombinací dvou přístupů ke struktuře XML dat a jejich přímého zpracování principy XML DOM a SAX dosahuje knihovna při čtení, zpracování a generování velkého množství dat výborných výsledků ve srovnání s již existujícími knihovnami. Podobně zavedením stylování buněk ve formátu CSS je dosaženo určitého stupně standardizace práce pro uživatele.

Testy generování 100 000 buněk provedla vlastní knihovna na základě měření průměrně za 1,8 sekund oproti 0,6 sekundám – 36 minutám u ostatních. Zde dosahovala nejlepších výsledků mezi volně dostupnými knihovnami, které dosahovaly horších výsledků o 119 – 1132 % (v nejhorším případě dokonce o 117 648 %). Placené varianty (kromě jedné) dosahovaly procentně vyšší

výkonnosti v rozmezí 63 – 69 %, pomalejší knihovna měla horší čas o 96 %. Čas zpracování a čtení 100 000 buněk je u vlastní knihovny průměrně 7,2 sekund, ostatní knihovny se pohybují v rozmezí 8,0 sekund – 13,9 minut, což u všech ostatních knihoven představuje horší výsledek o 10 – 417 % (v nejhorším případě dokonce o 11 431 %).

Nasazení knihovny bylo konzultováno a testováno přímo v průmyslovém prostředí výrobního informačního systému COMES® firmy Compas automatizace. Funkcionalita knihovny splňuje požadavky kladené na podobný produkt a díky použitým technologiím je možné ji snadno integrovat do stávajícího systému.

Možným rozšířením knihovny je podpora zobrazení grafiky, tedy především zahrnutí formátu VML pro reprezentaci vektorové grafiky uvnitř dokumentu. To vyžaduje studium struktury a vazeb na formát OOXML a rozšíření funkcionality především v oblasti validace a provázání s existující strukturou. Dalším rozšířením knihovny je pravidelná údržba, respektive zahrnutí aktualizací a nových verzí formátu OOXML.

8 Literatura

[1] MLÝNKOVÁ, Irena a Jaroslav POKORNÝ. *XML technologie: principy a aplikace v praxi*. 1. vyd. Praha: Grada, 2008, 267 s. Průvodce (Grada). ISBN 978-80-247-2725-7.

[2] BRAY, Tim. W3C. *Extensible Markup Language (XML) 1.0 (Fifth Edition)* [online]. 2008 [cit. 2015-04-19]. Dostupné z: http://www.w3.org/TR/xml/.

[3] ECMA. *Standard ECMA-376: Office Open XML File Formats* [online]. 2012 [cit. 2015-04-19]. Dostupné z: http://www.ecma-international.org/publications/standards/Ecma-376.htm.

[4] NGO, Tom. ECMA. *Office Open XML overview: ECMA TC45* [online]. 2006 [cit. 2015-04-20]. Dostupné z: http://www.ecma-international.org/news/TC45_current_work/OpenXML%20White%20Paper.pdf.

[5] Office Open XML, ECMA-376, and ISO/IEC 29500. *Office: Dev Center* [online]. 2011 [cit. 2015-04-20]. Dostupné z: https://msdn.microsoft.com/en-us/library/office/gg607163%28v=office.14%29.aspx.

[6] RICE, Frank. How to: Manipulate Office Open XML Formats Documents. *Microsoft: Developer Network* [online]. 2006 [cit. 2015-04-19]. Dostupné z: https://msdn.microsoft.com/en-us/library/aa982683.aspx.

[7] RICE, Frank. Introducing the Office (2007) Open XML File Formats. *Microsoft: Developer Network* [online]. 2006 [cit. 2015-04-20]. Dostupné z: https://msdn.microsoft.com/en-us/library/aa338205.aspx.

[8] FREE SOFTWARE FOUNDATION. *GNU General Public License* [online]. 2007 [cit. 2015-05-02]. Dostupné z: http://www.gnu.org/copyleft/gpl.html.

[9] Products: Aspose.Cells for .NET. *Aspose: Your File Format APIs* [online]. 2002 [cit. 2015-05-01]. Dostupné z: http://www.aspose.com/.net/excel-component.aspx.

[10] GemBox.Spreadsheet. *GemBox* [online]. 2008 [cit. 2015-05-01]. Dostupné z: http://www.gemboxsoftware.com/spreadsheet/overview.

[11] Spire.XLS for .NET. *E-ICEBLUE* [online]. 2011 [cit. 2015-05-01]. Dostupné z: http://www.e-iceblue.com/Introduce/excel-for-net-introduce.html.

[12] TMS Flexcel Studio for .NET. *Tmssoftware* [online]. 1995 [cit. 2015-05-02]. Dostupné z: http://www.tmssoftware.com/site/flexcelnet.asp.

[13] EEPlus: Create advance Excel spreadsheets on the server. *CodePlex: Project Hosting for Open Source Software* [online]. 2009 [cit. 2015-05-20]. Dostupné z: http://epplus.codeplex.com/.

[14] SpreadsheetLight. *SpreadsheetLight* [online]. 2011 [cit. 2015-04-28]. Dostupné z: http://spreadsheetlight.com/.

[15] ExcelPackage: Office Open XML Format file creation. *CodePlex: Project Hosting for Open Source Software* [online]. 2007 [cit. 2015-05-01]. Dostupné z: http://excelpackage.codeplex.com/.

[16] ClosedXML: The easy way to OpenXML. *CodePlex: Project Hosting for Open Source Software* [online]. 2010 [cit. 2015-05-02]. Dostupné z: http://closedxml.codeplex.com/.

[17] NetOffice: MS Office in .NET. *CodePlex: Project Hosting for Open Source Software* [online]. 2011 [cit. 2015-05-03]. Dostupné z: http://netoffice.codeplex.com/.

[18] Facts About Microsoft. *Microsoft: News Center* [online]. 2009 [cit. 2015-05-03]. Dostupné z: http://news.microsoft.com/facts-about-microsoft/.

[19] COMPAS AUTOMATIZACE. *Compas: Automatizace* [online]. 2015 [cit. 2015-05-10]. Dostupné z: http://www.compas.cz/.

[20] Výrobní informační systém COMES. *Compas: automatizace* [online]. 2015 [cit. 2015-05-20]. Dostupné z: http://www.compas.cz/produkty-a-sluzby/vyrobni-it.

[21] .NET Framework 4. *Microsoft: Developer Network* [online]. 2012 [cit. 2015-05-08]. Dostupné z: https://msdn.microsoft.com/en-us/library/w0x726c2%28v=vs.100%29.aspx.

[22] ALBAHARI, Joseph, Ben ALBAHARI a Peter DRAYTON.*C# 5.0 in a nutshell*. 5th ed. Sebastopol: O'Reilly, 2012, xvi, 1042 p. In a nutshell (O'Reilly. ISBN 978-144-9320-102.

[23] MICROSOFT. *Visual Studio* [online]. 1998 [cit. 2015-05-11]. Dostupné z: https://www.visualstudio.com/.

[24] Considerations for server-side Automation of Office. *Microsoft: Support* [online]. 2013 [cit. 2015-05-07]. Dostupné z: https://support.microsoft.com/en-us/kb/257757.

[25] Project Web App: Co je SharePoint? *Office* [online]. 2013 [cit. 2015-05-06]. Dostupné z: https://support.office.com/cs-cz/article/Co-je-SharePoint-97b915e6-651b-43b2-827d-fb25777f446f.

[26] Difference between 'Word Automation' and 'Word Automation Services'. *MSDN: Blogs* [online]. 2010 [cit. 2015-05-12]. Dostupné z: http://blogs.msdn.com/b/ericwhite/archive/2010/12/07/what-is-the-difference-between-word-automation-and-word-automation-services.aspx.

[27] Spreadsheets: Open XML SDK. *Office: Dev Center* [online]. 2012 [cit. 2015-05-06]. Dostupné z: https://msdn.microsoft.com/en-us/library/office/cc850837.aspx.

[28] Welcome to the Open XML SDK 2.5 for Office. *Office: Dev Center* [online]. 2013 [cit. 2015-05-06]. Dostupné z: https://msdn.microsoft.com/en-us/library/office/bb448854.aspx.

[29] How to use dates and times in Excel. *Microsoft: Support* [online]. 2006 [cit. 2015-05-03]. Dostupné z: https://support.microsoft.com/en-us/kb/214094

[30] ARLOW, Jim a Ila NEUSTADT. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky*. Vyd. 1. Brno: Computer Press, 2007, 567 s. ISBN 978-80-251-1503-9.

[31] Excel specifications and limits. *Office: Support* [online]. 2010 [cit. 2015-05-06]. Dostupné z: https://support.office.com/en-nz/article/Excel-specifications-and-limits-16c69c74-3d6a-4aaf-ba35-e6eb276e8eaa.

9 Seznam obrázků

10 Seznam tabulek

Přílohy

A Obsah CD

`/src/`	Zdrojové kódy knihovny.
`/doc/`	Programová dokumentace ve formátu HTML.
`DP_knihovna_OOXML.pdf`	Text diplomové práce ve formátu PDF.

B Obsah souborů z příkladu kapitoly 3.3

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Types xmlns="http://schemas.openxmlformats.org/package/2006/content-types">
  <Default Extension="rels" ContentType="application/vnd.openxmlformats-package.
relationships+xml"/>
  <Default Extension="xml" ContentType="application/xml"/>
  <Override PartName="/xl/workbook.xml" ContentType="application/vnd.
openxmlformats-officedocument.spreadsheetml.sheet.main+xml"/>
  <Override PartName="/xl/worksheets/sheet1.xml" ContentType="application/vnd.
openxmlformats-officedocument.spreadsheetml.worksheet+xml"/>
  <Override PartName="/xl/worksheets/sheet2.xml" ContentType="application/vnd.
openxmlformats-officedocument.spreadsheetml.worksheet+xml"/>
  <Override PartName="/xl/worksheets/sheet3.xml" ContentType="application/vnd.
openxmlformats-officedocument.spreadsheetml.worksheet+xml"/>
  <Override PartName="/xl/theme/theme1.xml" ContentType="application/vnd.
openxmlformats-officedocument.theme+xml"/>
  <Override PartName="/xl/styles.xml" ContentType="application/vnd.
openxmlformats-officedocument.spreadsheetml.styles+xml"/>
  <Override PartName="/xl/sharedStrings.xml" ContentType="application/vnd.
openxmlformats-officedocument.spreadsheetml.sharedStrings+xml"/>
  <Override PartName="/docProps/core.xml" ContentType="application/vnd.
openxmlformats-package.core-properties+xml"/>
  <Override PartName="/docProps/app.xml" ContentType="application/vnd.
openxmlformats-officedocument.extended-properties+xml"/>
</Types>
```

Obr. 29 Obsah souboru `[Content_Types].xml`.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/
relationships">
  <Relationship Id="rId3" Type="http://schemas.openxmlformats.org/
officeDocument/2006/relationships/extended-properties" Target="docProps/app.xml"
/>
  <Relationship Id="rId2" Type="http://schemas.openxmlformats.org/package/2006/
relationships/metadata/core-properties" Target="docProps/core.xml"/>
  <Relationship Id="rId1" Type="http://schemas.openxmlformats.org/
officeDocument/2006/relationships/officeDocument" Target="xl/workbook.xml"/>
</Relationships>
```

Obr. 30 Obsah souboru `/_rels/.rels`.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Properties xmlns="http://schemas.openxmlformats.org/officeDocument/2006/
extended-properties" xmlns:vt="http://schemas.openxmlformats.org/officeDocument/
2006/docPropsVTypes">
  <Application>Microsoft Excel</Application>
  <DocSecurity>0</DocSecurity>
  <ScaleCrop>false</ScaleCrop>
  <HeadingPairs>
    <vt:vector size="2" baseType="variant">
      <vt:variant>
        <vt:lpstr>listy</vt:lpstr>
      </vt:variant>
      <vt:variant>
        <vt:i4>3</vt:i4>
      </vt:variant>
    </vt:vector>
  </HeadingPairs>
  <TitlesOfParts>
    <vt:vector size="3" baseType="lpstr">
      <vt:lpstr>List1</vt:lpstr>
      <vt:lpstr>List2</vt:lpstr>
      <vt:lpstr>List3</vt:lpstr>
    </vt:vector>
  </TitlesOfParts>
  <LinksUpToDate>false</LinksUpToDate>
  <SharedDoc>false</SharedDoc>
  <HyperlinksChanged>false</HyperlinksChanged>
  <AppVersion>14.0300</AppVersion>
</Properties>
```

Obr. 31 Obsah souboru `/docsProps/app.xml`.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<cp:coreProperties xmlns:cp="http://schemas.openxmlformats.org/package/2006/
metadata/core-properties" xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:
dcterms="http://purl.org/dc/terms/" xmlns:dcmitype="http://purl.org/dc/dcmitype/
" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <dc:creator>Jaroslav Koštál</dc:creator>
  <cp:lastModifiedBy>Jaroslav Koštál</cp:lastModifiedBy>
  <dcterms:created xsi:type="dcterms:W3CDTF">2015-03-28T15:31:26Z</dcterms:
created>
  <dcterms:modified xsi:type="dcterms:W3CDTF">2015-03-28T15:36:10Z</dcterms:
modified>
</cp:coreProperties>
```

Obr. 32 Obsah souboru `/docsProps/core.xml`.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<workbook xmlns="http://schemas.openxmlformats.org/spreadsheetml/2006/main"
xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships">
  <fileVersion appName="xl" lastEdited="5" lowestEdited="5" rupBuild="9303"/>
  <workbookPr defaultThemeVersion="124226"/>
  <bookViews>
    <workbookView xWindow="240" yWindow="60" windowWidth="20115" windowHeight="
8010"/>
  </bookViews>
  <sheets>
    <sheet name="List1" sheetId="1" r:id="rId1"/>
    <sheet name="List2" sheetId="2" r:id="rId2"/>
    <sheet name="List3" sheetId="3" r:id="rId3"/>
  </sheets>
  <calcPr calcId="145621"/>
</workbook>
```

Obr. 33 Obsah souboru `/xl/workbook.xml`.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<styleSheet xmlns="http://schemas.openxmlformats.org/spreadsheetml/2006/main"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006" mc:
Ignorable="x14ac" xmlns:x14ac="http://schemas.microsoft.com/office/
spreadsheetml/2009/9/ac">
  <fonts count="1" x14ac:knownFonts="1">
    <font><sz val="11"/><color theme="1"/><name val="Calibri"/><family val="2"/>
<charset val="238"/><scheme val="minor"/></font>
  </fonts>
  <fills count="2">
    <fill><patternFill patternType="none"/></fill>
    <fill><patternFill patternType="gray125"/></fill>
  </fills>
  <borders count="1">
    <border><left/><right/><top/><bottom/><diagonal/></border>
  </borders>
  <cellStyleXfs count="1">
    <xf numFmtId="0" fontId="0" fillId="0" borderId="0"/>
  </cellStyleXfs>
  <cellXfs count="1">
    <xf numFmtId="0" fontId="0" fillId="0" borderId="0" xfId="0"/>
  </cellXfs>
  <cellStyles count="1">
    <cellStyle name="Normální" xfId="0" builtinId="0"/>
  </cellStyles>
  <dxfs count="0"/>
  <tableStyles count="0" defaultTableStyle="TableStyleMedium2"
defaultPivotStyle="PivotStyleLight16"/>
  <extLst>
    <ext uri="{EB79DEF2-80B8-43e5-95BD-54CBDDF9020C}" xmlns:x14="http://schemas.
microsoft.com/office/spreadsheetml/2009/9/main"><x14:slicerStyles
defaultSlicerStyle="SlicerStyleLight1"/></ext>
  </extLst>
</styleSheet>
```

Obr. 34 Obsah souboru `/xl/styles.xml`.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<sst xmlns="http://schemas.openxmlformats.org/spreadsheetml/2006/main" count="6"
uniqueCount="6">
  <si>
    <t>Customer</t>
  </si>
  <si>
    <t>id</t>
  </si>
  <si>
    <t>firstName</t>
  </si>
  <si>
    <t>lastName</t>
  </si>
  <si>
    <t>Jaroslav</t>
  </si>
  <si>
    <t>Košťál</t>
  </si>
</sst>
```

Obr. 35 Obsah souboru `/xl/sharedStrings.xml`.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/
relationships">
  <Relationship Id="rId3" Type="http://schemas.openxmlformats.org/
officeDocument/2006/relationships/worksheet" Target="worksheets/sheet3.xml"/>
  <Relationship Id="rId2" Type="http://schemas.openxmlformats.org/
officeDocument/2006/relationships/worksheet" Target="worksheets/sheet2.xml"/>
  <Relationship Id="rId1" Type="http://schemas.openxmlformats.org/
officeDocument/2006/relationships/worksheet" Target="worksheets/sheet1.xml"/>
  <Relationship Id="rId6" Type="http://schemas.openxmlformats.org/
officeDocument/2006/relationships/sharedStrings" Target="sharedStrings.xml"/>
  <Relationship Id="rId5" Type="http://schemas.openxmlformats.org/
officeDocument/2006/relationships/styles" Target="styles.xml"/>
  <Relationship Id="rId4" Type="http://schemas.openxmlformats.org/
officeDocument/2006/relationships/theme" Target="theme/theme1.xml"/>
</Relationships>
```

Obr. 36 Obsah souboru `/xl/_rels/workbook.xml.rels`.

```
108         <a:font script="Mong" typeface="Mongolian Baiti"/>
109         <a:font script="Viet" typeface="Arial"/>
110         <a:font script="Uigh" typeface="Microsoft Uighur"/>
111         <a:font script="Geor" typeface="Sylfaen"/>
112       </a:minorFont>
113     </a:fontScheme>
114     <a:fmtScheme name="Kancelář">
115       <a:fillStyleLst>
116         <a:solidFill>
117           <a:schemeClr val="phClr"/>
118         </a:solidFill>
119         <a:gradFill rotWithShape="1">
120           <a:gsLst>
121             <a:gs pos="0">
122               <a:schemeClr val="phClr">
123                 <a:tint val="50000"/>
124                 <a:satMod val="300000"/>
125               </a:schemeClr>
```

Obr. 37 Obsah souboru `/xl/theme/theme1.xml`.

C Obsah souboru z příkladu kapitoly 4.3.2

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xml xmlns:v="urn:schemas-microsoft-com:vml" xmlns:o="urn:schemas-microsoft-com:
office:office" xmlns:x="urn:schemas-microsoft-com:office:excel">
  <o:shapelayout v:ext="edit">
    <o:idmap v:ext="edit" data="1" />
  </o:shapelayout>
  <v:shapetype id="_x0000_t202" coordsize="21600,21600" o:spt="202" path="m,l,
21600r21600,l21600,xe">
    <v:stroke joinstyle="miter" />
    <v:path gradientshapeok="t" o:connecttype="rect" />
  </v:shapetype>
  <v:shape type="#_x0000_t202" style="position:absolute;margin-left:90pt;margin-
top:52.5pt;width:128pt;height:39.25pt;z-index:1;visibility:hidden" fillcolor="
#ffffe1" o:insetmode="auto" id="_x0000_s10510">
    <v:fill color2="#ffffe1" />
    <v:shadow on="t" color="black" obscured="t" />
    <v:path o:connecttype="none" />
    <v:textbox style="mso-direction-alt:auto">
      <div style="text-align:left" />
    </v:textbox>
    <x:ClientData ObjectType="Note">
      <x:MoveWithCells />
      <x:SizeWithCells />
      <x:Anchor>6,15,8,10,8,31,12,9</x:Anchor>
      <x:AutoFill>False</x:AutoFill>
      <x:Row>10</x:Row>
      <x:Column>5</x:Column>
    </x:ClientData>
  </v:shape>
</xml>
```

Obr. 38 Obsah souboru `/xl/drawings/wmlDrawing1.vml`.